Ministère de l'Enseignement Supérieur

Et de la Recherche Scientifique

CENTRE UNIVERSITAIRE CHERIF BOUCHOUCHA AFLOU

Faculté de Mathématiques et de l'Informatique

Département de l'Informatique

Cours

Système d'exploitation

GUIDE DE L'ÉTUDIANT

Présenté Par : Dr. AMMAR. Sabrina

Cet ouvrage offre une présentation pédagogique des cours et des exercices exemplaires, synthétisant les concepts fondamentaux des systèmes d'exploitation.

Pour plus d'informations ou des précisions supplémentaires,
Veuillez contacter l'auteur :
Sabrina AMMAR Maître de conférence au centre universitaire cherif bouchoucha-Aflou
s.ammar@cu-aflou.edu.dz

Remercîment

La réalisation de cette polycopie à diriger des recherches n'aurait pas été possible sans le soutien et la collaboration de nombreuses personnes et institutions, à qui je souhaite exprimer ma profonde gratitude.

Tout d'abord, je tiens à remercier chaleureusement, mon directeur, pour sa confiance, ses précieux conseils et son encadrement tout au long de ce parcours. Son expertise et sa rigueur scientifique ont été déterminantes dans l'aboutissement de ce travail.

Je remercie également les membres du jury pour avoir accepté de juger ce travail et pour leurs remarques constructives et enrichissantes. Leur bienveillance et leur exigence ont été un véritable moteur dans l'amélioration de ce manuscrit.

Je suis reconnaissant à l'ensemble des collègues et amis chercheurs pour leurs soutiens, leurs échanges scientifiques stimulants et leurs encouragements constants. Leur amitié et leur solidarité ont grandement contribué à la richesse de cette expérience.

Mes pensées les plus sincères vont à mes étudiants et doctorants, passés et présents, dont l'enthousiasme et la curiosité intellectuelle ont été une source constante de motivation et d'inspiration. Leur engagement et leur travail acharné ont été des éléments essentiels à la réussite de ce projet.

Je tiens également à exprimer ma gratitude à ma famille et à mes proches pour leur patience, leur soutien indéfectible et leur amour. Leur présence à mes côtés a été un pilier fondamental tout au long de ce parcours exigeant.

Enfin, je souhaite remercier toutes les personnes et institutions, de près ou de loin, qui ont contribué à la réalisation de cette habilitation, que ce soit par leur aide technique, leurs discussions enrichissantes ou leur soutien moral.

Avant-propos

Chers étudiants,

Bienvenue dans le module "Système d'exploitation" de votre cursus de LMD 2ème licence en Informatique. Ce polycopié a été conçu pour vous fournir une base théorique essentielle, et les connaissances approfondies nécessaires à la compréhension et à la maîtrise des systèmes d'exploitation, une composante clé de tout environnement informatique.

L'objectif principal de ce document est de vous fournir une introduction complète et détaillée aux concepts fondamentaux des systèmes d'exploitation. Vous y trouverez des explications approfondies sur les rôles et les fonctionnalités clés d'un système d'exploitation, ainsi que sur les différentes architectures et modèles de conception existants.

Chaque chapitre de ce polycopié est structuré de manière à faciliter l'apprentissage progressif. Des exemples concrets, et des études de cas vous permettront d'appliquer les concepts théoriques à des situations réelles. Nous vous encourageons à travailler régulièrement sur les exercices et à participer activement aux discussions en classe pour maximiser votre compréhension.

Nous espérons que ce cahier sera un outil précieux dans votre parcours académique et qu'il vous aidera à développer les compétences nécessaires pour réussir dans le domaine de l'informatique. Nous vous souhaitons une étude fructueuse et enrichissante. N'hésitez pas à me solliciter si vous avez la moindre question ou difficulté.

Bonne lecture et bon courage.

SYLLABUS

Unit d'enseignement : **UEF412 Systèmes Informatiques (SI)**

Matière : **Système d'exploitation 1**

Domaine/ Filière : **2er année Licence Informatique académique.**

Semestre : **02**

Crédit : **05**

Volume Horaire Hebdomadaire Total : **63h**

Cours : **1h 30**

Travaux dirigés : **1h30**

Enseignant responsable : **Dr. AMMAR Sabrina**

E-mail : **s.ammar@cu-aflou.edu.dz**

Table des matières

CHAPITRE-IV- GESTION DES ENTREES-SORTIES

CHAPITRE-V- GESTION DES PROCESSUS

CHAPITRE-VI- ORDONNANCEMENT DES PROCESSUS (SCHEDULING DE L'UC)

CHAPITRE-VII- COMMUNICATION INTERPROCESSUS ET SYNCHRONISATION

CHAPITRE-VIII- GESTION DE MEMOIRE

Liste des figures

Liste des tableaux

Liste des abréviations

API : Application Programming Interface

APIC : Advanced Programmable Interrupt Controller

BIOS : Basic Input/Output System

CAS : Compare-and-Swap

CD/DVD : Compact Disc/ Digital Versatile Disc

CLI : Command Line Interface

CO : Compteur Ordinal

CPU : Central Processing Unit

CS : Chip Select

C-SCAN : Circular SCAN

DMA : Direct Memory Acces.

DRAM : Dynamic RAM

DRAM, : Dynamic Random Access Memory

EEPROM : Electrically Erasable Programmable ROM

EFI : Extensible Firmware Interface**.**

EPROM : Erasable Programmable ROM

FAT : File Allocation Table

FCFS : First-Come, First-Served

FCFS : First Come First Served

FIFO : First In, First Out

GUI : Graphical User Interface

HDD : Hard Disk Drive

I/O : Input/Output

IA : Intelligence Artificielle

IBM : International Business Machines Corporation

INIT : Initialisation ; est le premier programme informatique exécuté sur les systèmes d'exploitation basés Unix.

IoT : Internet of Things

IPC : Inter-Process Communication

ISA : Industry Standard Architectur

LAN : Local Area Network

LFU : Least Frequently Used

Liste des abréviations.

LRU : Least Recently Used

MFT : Master File Table

MFU : Most Frequently Used

MIC : Many Integrated Core

MINIX : MINI uniX

MMU : Memory Management Unit

MS-DOS : Microsoft Disk Operating System

NFU : Not Frequently Used

NTFS : New Technology File System

NUMA : Non-Uniform Memory Access

OPT : Optimal Page Replacement

OS : Operating System

PC : Program Counter

PCB : Process Control Bloc

PCI : Peripheral Component Interconnect

PIC : Contrôleurs d'interruption programmable

PID : Process Identifier

PPID : Parent Process Identifier

PROM : Programmable ROM

QNX : Quantum unix

R/W : Read/Write

RAM : Random Access Memory

Rit : Routine d'Interruption

ROM : Read-Only Memory

RR : Round Robin

SCAN : Elevator Algorithm.

SE : Systèmes d'Exploitation

SGF : Système de Gestion des Fichiers

SJF : Shortest Job First

SMP : Symmetric MultiProcessing

SRAM : Static RAM

SRT : Shortest Remaining Time

SRTF : Shortest Remaining Time First

SSD : Solid State Drive

SSTF : Shortest Seek Time First

SVC : SuperVisor Call

TCB : Thread Control Block

TCP/IP : Internet Protocol/ Transmission Control Protocol

TLB : Translation Lookaside Buffer

UC : Unité Centrale

UEFI : Unified Extensible Firmware Interface

UFS : Unix File System

UNIVAC : UNIVersal Automatic Computer

USB : Universal Serial Bus

VFS : Virtual File System

VI : vecteur d'interruption

WAN : Wide Area Network

Préface:

Le module de Systèmes d'Exploitation **(SE)** est une composante fondamentale de la formation en informatique, offrant une plongée au cœur du fonctionnement des ordinateurs modernes. Ce cours explore l'interface cruciale entre le matériel informatique et les applications logicielles, révélant les mécanismes complexes qui permettent l'utilisation efficace et sécurisée des ressources informatiques.

Les systèmes d'exploitation jouent un rôle central dans notre interaction quotidienne avec la technologie, qu'il s'agisse d'ordinateurs personnels, de smartphones, de serveurs ou de systèmes embarqués. Ils orchestrent l'exécution des programmes, gèrent les ressources matérielles, assurent la sécurité et fournissent une interface utilisateur conviviale.

Ce module couvre plusieurs aspects clés :

1. Principes fondamentaux : Introduction aux concepts de base, à l'architecture et aux fonctions des SE.
2. Systèmes de fichiers : Analyse des méthodes de stockage, d'organisation et d'accès aux données.
3. Gestion des périphériques : Compréhension de l'interaction entre le SE et le matériel.
4. Gestion des processus : Étude des mécanismes d'exécution, d'ordonnancement et de synchronisation des tâches.
5. Gestion de la mémoire : Exploration des techniques d'allocation, de pagination et de mémoire virtuelle.

L'objectif de ce module est de fournir une compréhension approfondie du fonctionnement interne des SE, essentielle pour tout professionnel de l'informatique. Ces connaissances sont cruciales pour le développement de logiciels efficaces, la gestion de systèmes et la résolution de problèmes complexes.

À travers ces cours, les étudiants développeront non seulement des connaissances théoriques, mais aussi des compétences pratiques via des travaux dirigés et des projets. Cette approche équilibrée prépare les futurs professionnels à relever les défis technologiques dans un domaine en constante évolution.

En somme, ce module de Systèmes d'Exploitation est une pierre angulaire de la formation en informatique, ouvrant la voie à une compréhension approfondie de l'infrastructure logicielle qui sous-tend notre monde numérique.

CHAPITRE-I- INTRODUCTION AUX SYSTEMES D'EXPLOITATION

I.1 Introduction :

Les systèmes d'exploitation **(SE)** sont des ensembles de programmes logiciels essentiels au fonctionnement des ordinateurs et autres appareils électroniques. Ils jouent un rôle central en fournissant une interface entre l'utilisateur, les applications et le matériel informatique.

I.2 Objectif :

L'objectif est de fournir une compréhension de base de ce qu'est un système d'exploitation **(SE)**, de son rôle essentiel et de son fonctionnement.

I.3 Système informatique :

I.3.1 Définition :

Un système informatique peut être défini comme un ensemble coordonné de composants matériels et logiciels interconnectés qui travaillent ensemble pour traiter, stocker et transmettre des données en vue d'atteindre un objectif spécifique, tels que la gestion des données, la communication, le divertissement, le contrôle de processus industriels, etc.

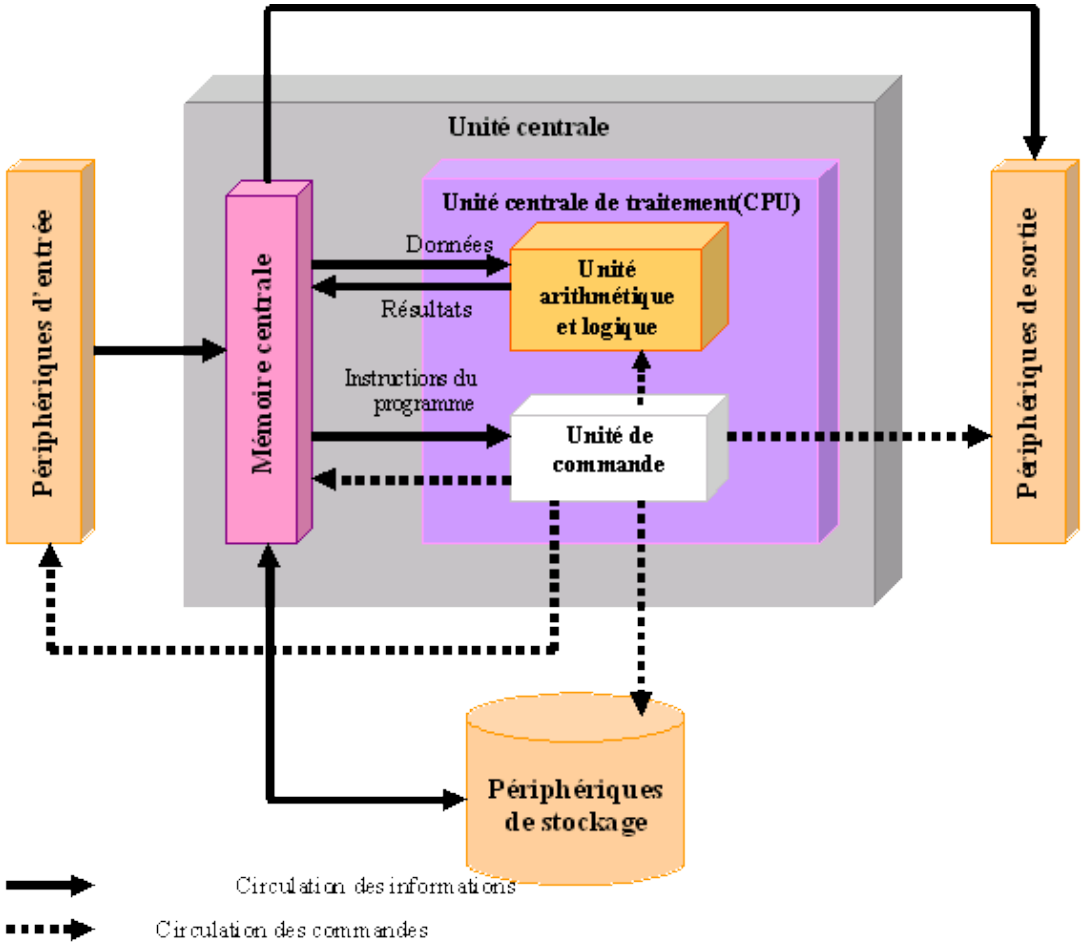


Figure I. 1: Architecture d'un ordinateur.

I.3.2 Relation entre matériel et logiciel :

- **Composants matériels (Hardware) :** Ce sont les éléments physiques tangibles qui constituent un système informatique, tels que les processeurs, la mémoire, le stockage (disques durs, **SSD**), les périphériques d'entrée/sortie (claviers, souris, écrans, imprimantes, etc.) et les composants réseau (cartes réseau, routeurs, etc.).
- **Composants logiciels (Software) :** Il s'agit des programmes informatiques qui contrôlent et coordonnent les opérations du système. Cela inclut le système d'exploitation, qui fournit une interface entre le matériel et les logiciels applicatifs, ainsi que les applications logicielles elles-mêmes, telles que les navigateurs web, les suites bureautiques, les logiciels de gestion, etc.

Le **"Software"** et le **"Hardware"** sont complémentaires :

- Le **"Hardware"** a besoin du software pour être piloté.
- Le **"Software"** a besoin du hardware pour être exécuté.

- **Interconnexion :** Les composants matériels et logiciels d'un système informatique sont interconnectés par divers moyens de communication, tels que les bus internes, les réseaux locaux (**LAN**), les réseaux étendus (**WAN**), et les connexions sans fil.
- **Traitement, stockage et transmission de données :** Les systèmes informatiques traitent les données en effectuant des opérations sur celles-ci, les stockent pour une utilisation future et les transmettent selon les besoins, que ce soit localement sur le système ou à travers des réseaux.

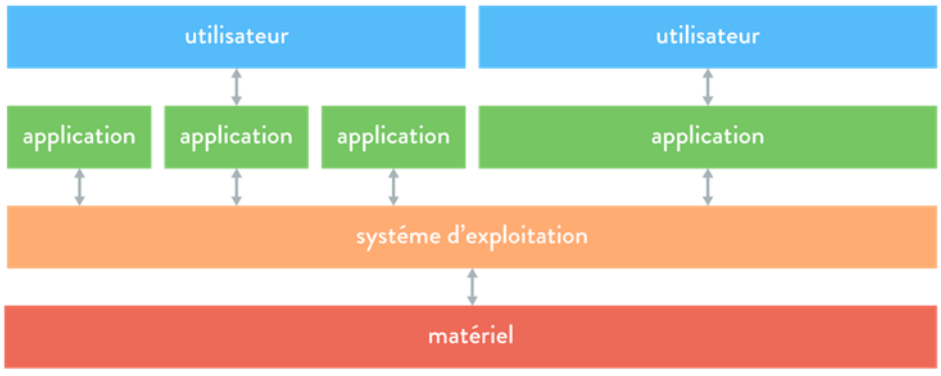


Figure I. 2: Système informatique matériel et logiciel.

I.3.3 Organisation d'un Système informatique :

L'organisation d'un système informatique se réfère à la manière dont ses différents composants matériels et logiciels sont structurés et interagissent pour assurer le fonctionnement global du système.

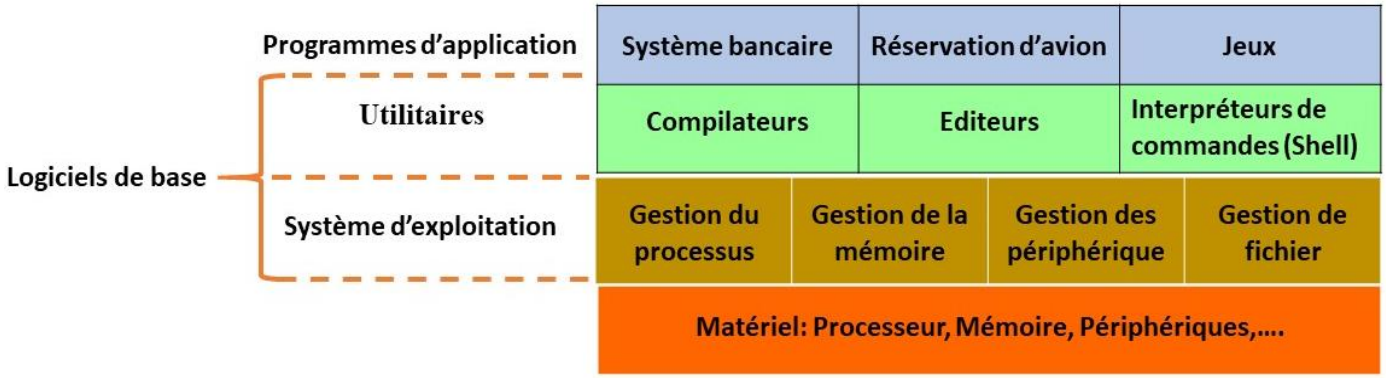


Figure I. 3: Organisation d'un système informatique.

I.4 Système d'exploitation :

I.4.1 Définition :

Système d'exploitation (**SE**, en anglais **Operating System** ou **OS**) est :

- Un logiciel qui agit comme une interface entre l'utilisateur, le matériel informatique et les applications logicielles.
- Un ensemble des logiciels effectuant la gestion optimale des ressources d'un système informatique.
- Une couche logicielle intercalée entre l'ordinateur et ses applications lancées par ses utilisateurs au démarrage.

Le système d'exploitation est une couche de logiciel a pour fonction de masquer la complexité du matériel et de proposer des instructions plus simples à l'utilisateur. Facilite la communication entre ces différents éléments et permet de gérer les ressources système.

Il existe différents types de **SE** :

- Les systèmes d'exploitation de bureau (comme **Windows**, **mac OS** et **Linux**) ;
- Les systèmes d'exploitation mobiles (comme **Android** et **iOS**) ;
- Les systèmes d'exploitation consol (**Orbis OS**) ;
- Les systèmes d'exploitation en temps réel (utilisés dans les systèmes embarqués) et les systèmes d'exploitation distribués.

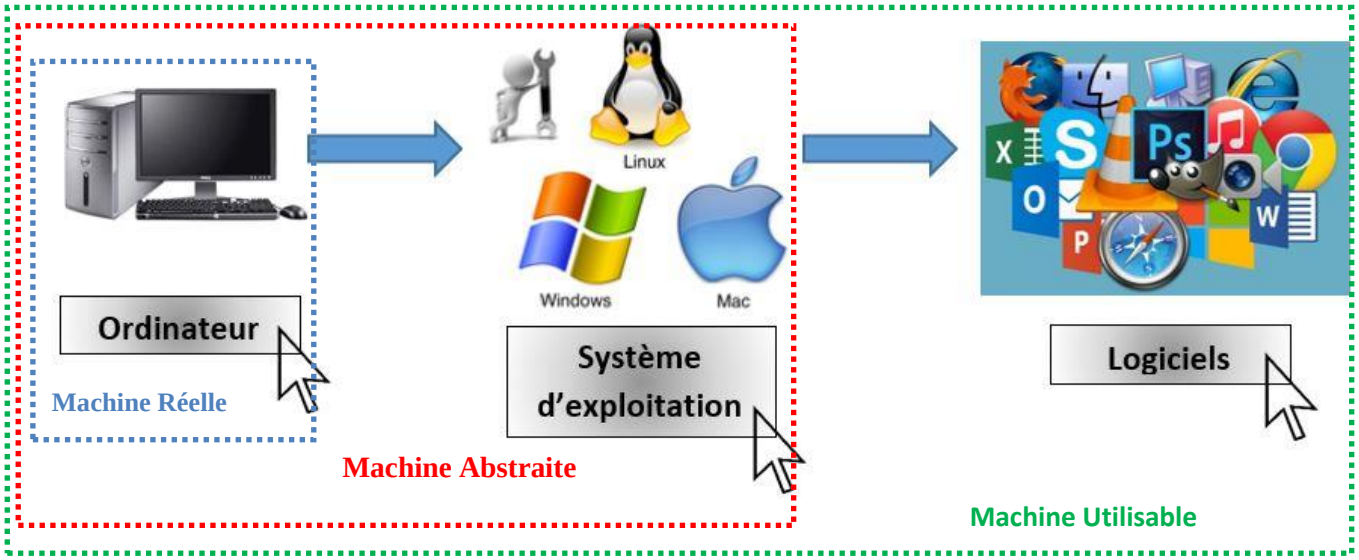


Figure I. 4: Machine Utilisable.

Machine Réelle = Unité centrale + Périphériques (**CPU**, Mémoire, **I/O**) ;

Machine Abstraite = Machine Réelle + Système d'exploitation ;

Machine Utilisable = Machine Abstraite + Application.

I.4.2 Rôle principal :

Le rôle principal d'un **SE** est de coordonner et de contrôler les ressources matérielles de l'ordinateur, telles que le processeur, la mémoire, le stockage et les périphériques d'entrée/sortie. Il permet également d'exécuter des programmes et d'assurer la sécurité du système.

I.4.3 Composants d'un système d'exploitation :

Les principaux composants d'un **SE** incluent le noyau (kernel), les utilitaires systèmes, les interfaces utilisateur et les pilotes de périphériques.

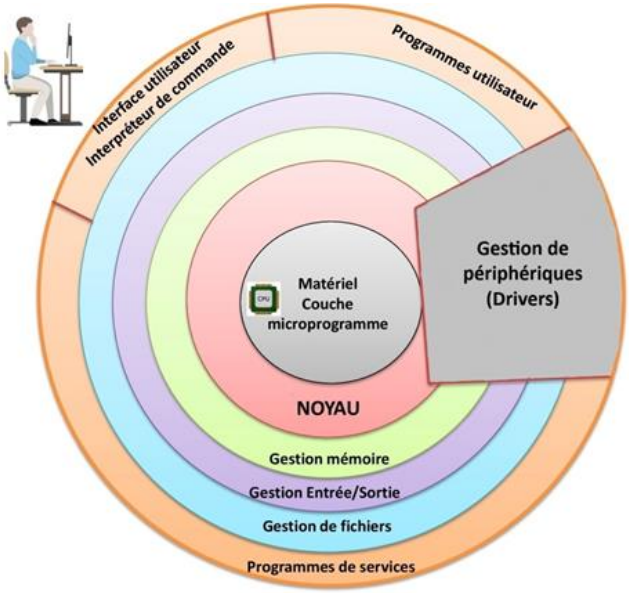


Figure I. 5: Couches du système d'exploitation.

- **Le noyau (en anglais Kernel) :**

C'est la partie la plus importante d'un système d'exploitation. Le noyau est un ensemble de programmes qui réside constamment en mémoire centrale. Il représente les fonctions fondamentales d'un système d'exploitation telles que la gestion du matériel (mémoire, processeur, périphériques), la gestion des processus (ordonnancement), et les fonctionnalités de communication entre les composants matériels et logiciels d'un ordinateur **[1]**.

- **L'interpréteur de commandes (en anglais Shell) :**

C'est l'interface entre l'utilisateur et le système d'exploitation. Il permet à l'utilisateur de communiquer avec le système d'exploitation par l'intermédiaire d'un langage de commandes. Ces commandes permettent à l'utilisateur de réaliser des tâches d'administration **(ex. user add person avec Unix)** de manipuler des fichiers ou des répertoires **(ex. mkdir répertoire avec Unix, md répertoire avec MSDOS)**, de lancer des applications, de contrôler les périphériques **(ex. mount floppy avec Unix)** ou d'agir sur la configuration, etc.

- **Le système de fichiers (en anglais filesystem) :**

C'est une structure de données permettant de stocker des informations et de les organiser dans des fichiers. Il fournit à l'utilisateur une vue abstraite des données enregistrées sous formes d'entités virtuelles. L'utilisateur peut ainsi enregistrer, lire, supprimer et localiser ces entités à partir d'un chemin d'accès.

I.4.4. Principe et fonctionnement d'un système d'exploitation :

Les différents programmes du système d'exploitation sont stockés sur le disque dur. L'ordinateur dispose donc d'un microprogramme lancé au démarrage, capable de monter la partition du système d'exploitation et de lancer son premier processus.

L'**UEFI (Unified Extensible Firmware Interface)** est le micrologiciel présent sur l'ordinateur, stocké dans une mémoire flash de la carte mère. Il succède au **BIOS** et en améliore la sécurité. Il peut avoir d'éventuelles extensions sur la première partition du disque dur. On peut accéder à sa configuration au démarrage via une touche fonction. Cette configuration permet notamment de configurer le mode de démarrage du **PC**, que ce soit sur disque dur ou en réseau.

Voyons comment l'**UEFI** charge le système d'exploitation en mémoire et démarre le premier processus du système d'exploitation. À la mise sous tension de l'ordinateur, l'**UEFI** ou le **BIOS** charge une configuration minimale (disque dur, carte réseau pour un éventuel démarrage depuis le réseau, écran, clavier, souris). Si le démarrage ne se fait pas paramétrer depuis le réseau, l'**UEFI** accède alors à la petite partition **EFI** du disque dur et ainsi à des applications et drivers complémentaires. S'il n'y a pas de configuration, l'**UEFI** lance le bootloader. Celui-ci peut proposer le choix entre plusieurs systèmes d'exploitation présents sur plusieurs partitions.

Une fois le choix effectué, le bootloader charge le noyau en mémoire et lance son exécution. Le kernel crée le processus de **PID 0**, le noyau. Ce processus lance alors le processus avec le **PID 1** pour monter le système de fichiers et passer en mode utilisateur. Le processus Thread de **PID** *2* est ensuite lancé pour gérer les processus du noyau. Le processus **INIT** de **PID 1** poursuit parallèlement son action en terminant de démarrer complètement le système d'exploitation **[8]**.

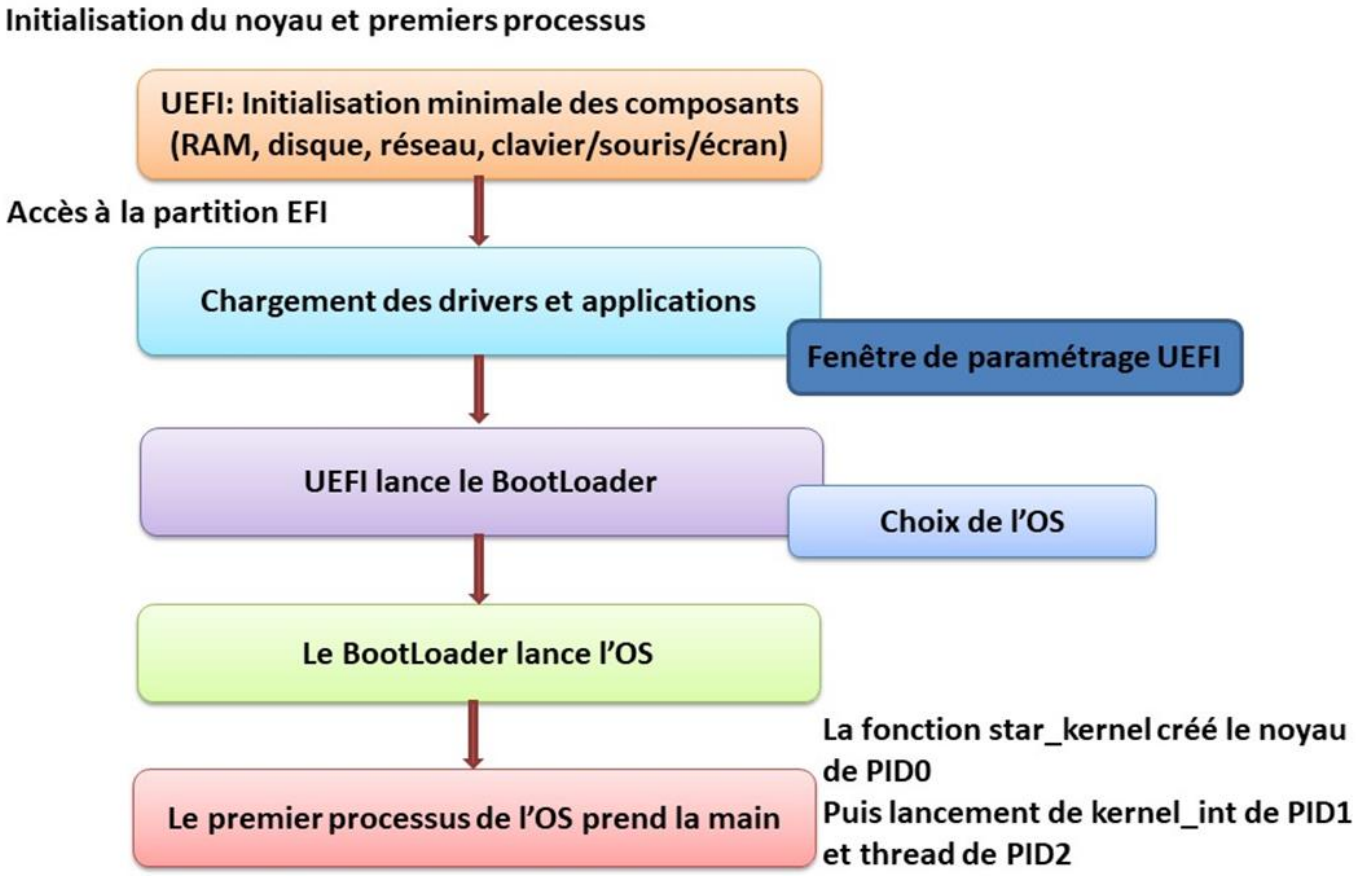


Figure I. 6: Démarrage du système.

I.4.5 Appels Système :

Les appels système, également connus sous le nom d'appels au noyau ou d'appels au système d'exploitation, sont des mécanismes utilisés par les programmes informatiques pour accéder aux fonctionnalités fournies par le système d'exploitation (**SE**). Lorsqu'un programme souhaite effectuer une opération qui nécessite des privilèges ou des ressources qui ne lui sont pas directement accessibles, il peut faire appel au noyau du système d'exploitation en utilisant un appel système.

En d'autres termes, Un appel système est une fonction fournie par le noyau **(kernel)** d'un **SE** et utilisée par les programmes s'exécutant dans l'espace utilisateur.

Les appels système permettent aux programmes utilisateur d'effectuer diverses opérations, telles que la gestion des fichiers, la communication entre processus, la gestion des entrées-sorties, la création de processus, la gestion de la mémoire, etc.

Quelques appels systèmes classiques :

- **Open**, **read**, **write** et **close** qui permettent les manipulations sur les systèmes de fichiers ;
- **Alloc**, **free** pour allouer et désallouer de la mémoire.

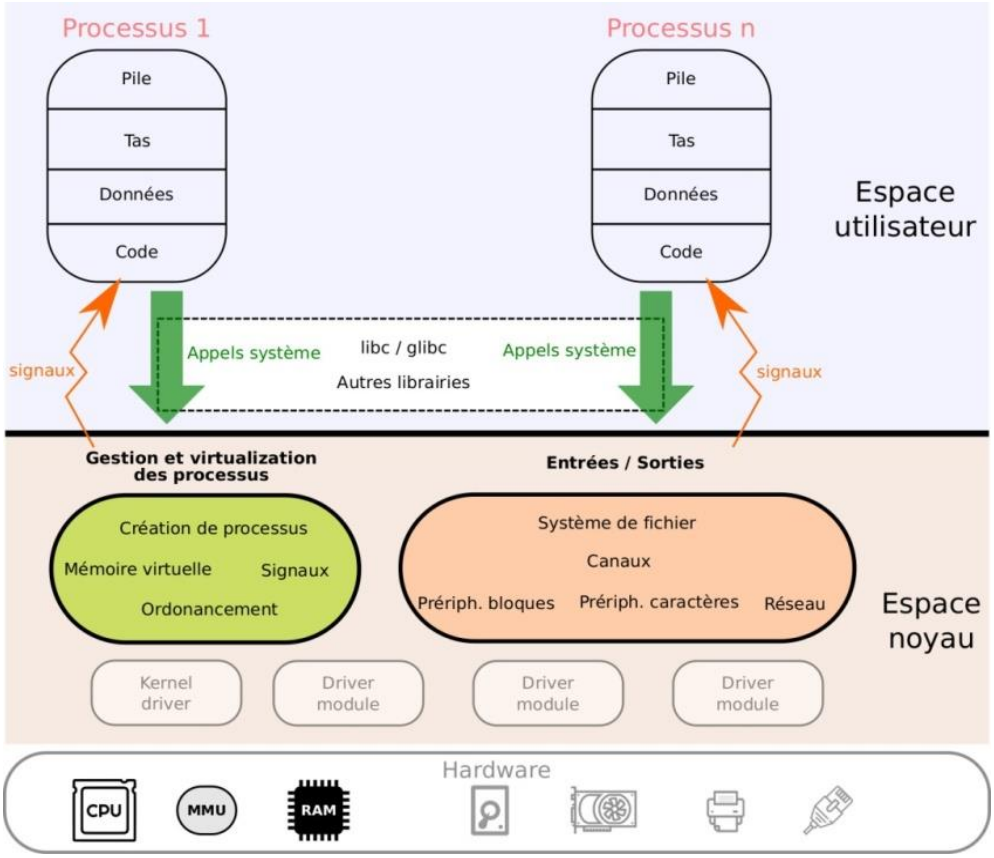


Figure I. 7: Illustration-appels système.

I.4.6 Chronologie des systèmes d'exploitation :

L'évolution des systèmes d'exploitation (**SE**) est un domaine vaste et fascinant, marqué par des avancées technologiques majeures et des changements significatifs dans la manière dont les ordinateurs sont utilisés et gérés. Voici un aperçu général de l'évolution des systèmes d'exploitation :

- **Premiers systèmes d'exploitation (années 1950-1960) :**

L'apparition des premiers systèmes d'exploitation, comme **GM-NAA I/O** pour le système **UNIVAC I** (1958) et le système d'exploitation Batch Monitor (1956) pour l'**IBM 704**, a permis de gérer plus efficacement les ressources et les tâches informatiques.

- **Ère des systèmes à temps partagé (années 1960-1970) :**

L'apparition du système d'exploitation **UNIX** dans les années 1970 a été particulièrement influente, jetant les bases de nombreux concepts modernes des systèmes d'exploitation, tels que les interfaces en ligne de commande, les systèmes de fichiers hiérarchiques et la gestion des processus.

- **Ère des ordinateurs personnels (années 1980-1990) :**

L'avènement des ordinateurs personnels dans les années 1980 a donné naissance à des systèmes d'exploitation populaires tels que **MS-DOS**, développé par Microsoft, et Macintosh System Software pour les ordinateurs Apple Macintosh.

L'introduction de Windows **3.0** par Microsoft en 1990 a marqué le début d'une série de systèmes d'exploitation Windows qui ont dominé le marché des **PC** pendant des décennies.

- **Ère de l'Internet et des réseaux (années 1990-2000) :**

L'avènement de l'Internet et des réseaux a conduit au développement de systèmes d'exploitation axés sur la connectivité et la communication, tels que Linux, qui est devenu un système d'exploitation majeur pour les serveurs et les systèmes embarqués, ainsi que Windows **NT** et ses dérivés, qui ont introduit des fonctionnalités réseau avancées.

Les systèmes d'exploitation mobiles ont également émergé, avec des plateformes telles que Palm **OS**, Symbian **OS** et BlackBerry **OS**.

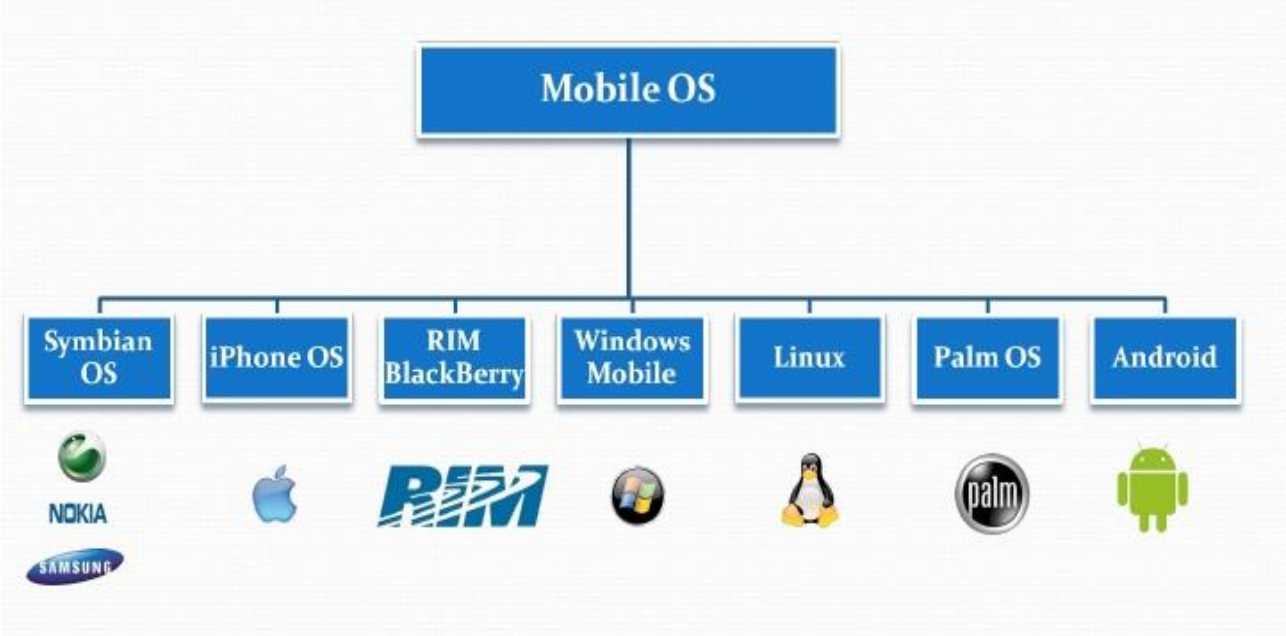


Figure I. 8: Chronologie systèmes d'exploitation Mobile.

- **Ère de la mobilité et du cloud (années 2000 à nos jours) :**

L'essor des smartphones et des tablettes a conduit à la domination des systèmes d'exploitation mobiles tels qu'i **OS** d'Apple et Android de Google.

- **Tendances actuelles et futures :**

Les développements récents se concentrent sur des aspects tels que la virtualisation, la conteneurisation, l'automatisation et la sécurité, avec des systèmes d'exploitation conçus pour répondre aux besoins des

environnements informatiques modernes, tels que les datacenters, l'Internet des objets (**IoT**) et l'intelligence artificielle (**IA**).

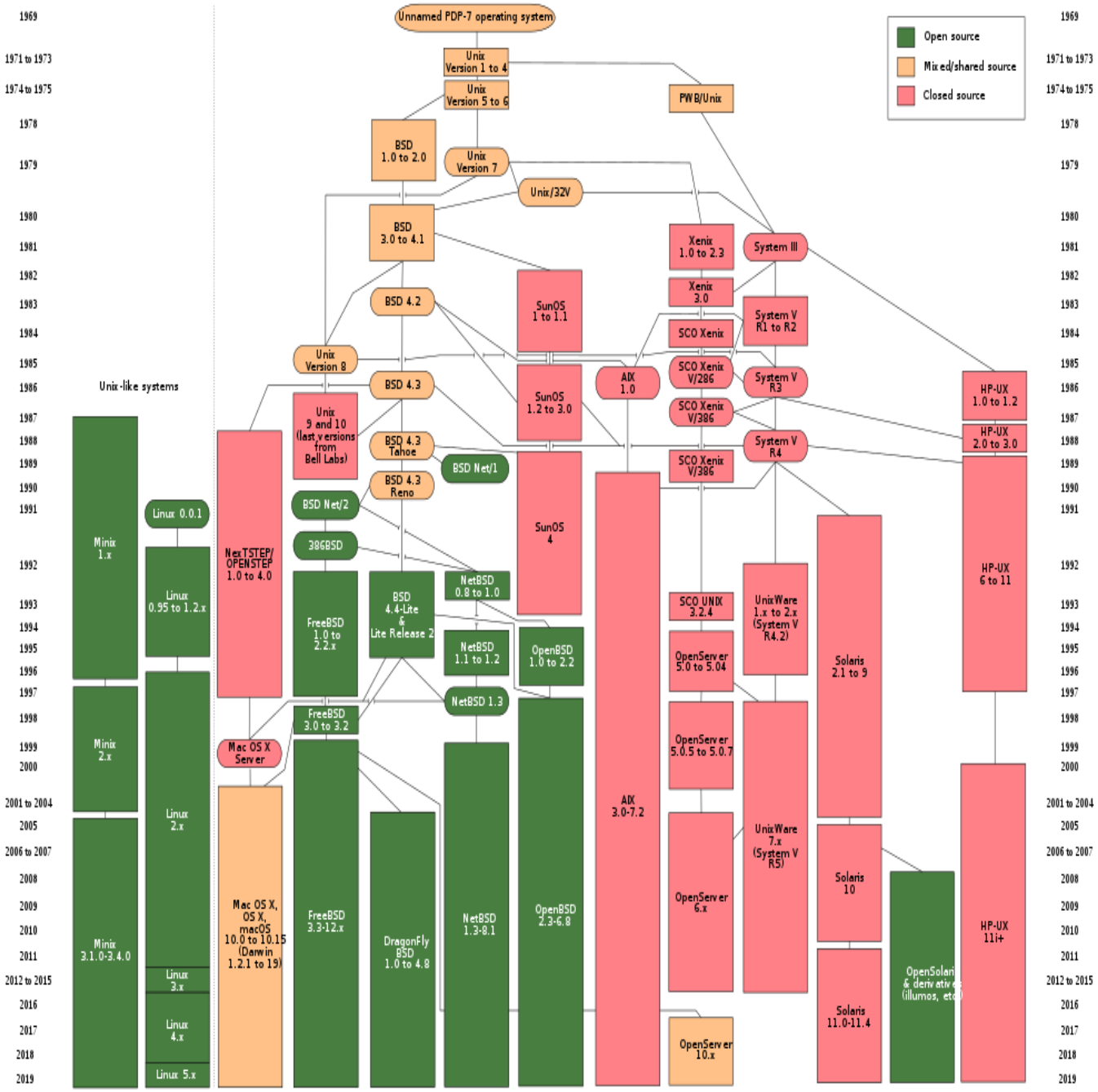


Figure I. 9: Chronologie systèmes d'exploitation UNIX.

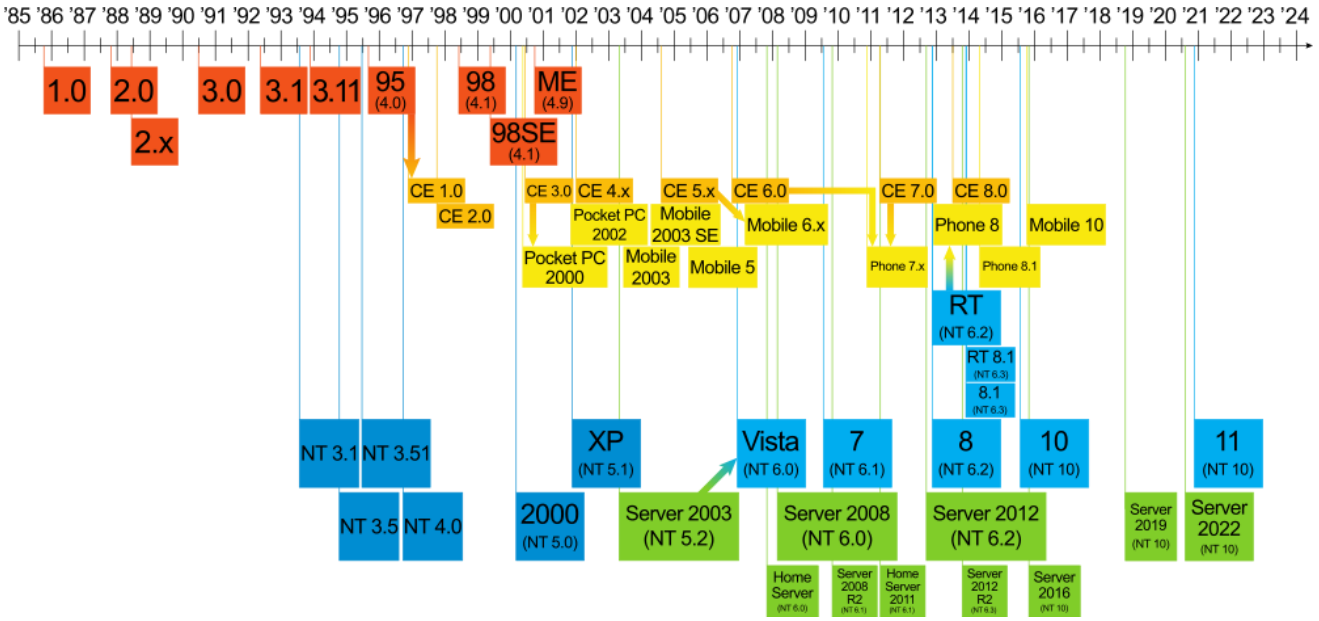


Figure I. 10: Chronologie systèmes d'exploitation Windows.

I.4.7 Taxonomie des systèmes d'exploitation :

I.4.7.1 Classification selon le type d'ordinateur :

a. **Systèmes d'exploitation pour ordinateurs personnels :** Conçus pour les ordinateurs personnels et de bureau, tels que **Windows**, **MacOs** et les distributions de **Linux** grand public comme **Ubuntu**.

b. **Systèmes d'exploitation pour serveurs :** Optimisés pour la gestion des ressources serveur, la sécurité et la fiabilité, comme Linux (avec des distributions spécialisées comme CentOS, RedHat Enterprise Linux) et Windows Server.

c. **Systèmes d'exploitation embarqués :** Destinés aux appareils et aux systèmes embarqués, tels que les systèmes utilisés dans les appareils mobiles, les systèmes de contrôle industriel, les appareils électroniques grand public, etc. Exemples : **Android**, **iOS**, **Windows Embedded**, **Linux embarqué.**

d. **Systèmes temps réel :** Sont des systèmes informatiques conçus pour répondre à des contraintes temporelles strictes. Contrairement aux systèmes classiques, où la performance est mesurée en termes de vitesse de traitement, les systèmes temps réel doivent garantir que certaines opérations soient effectuées dans des délais spécifiques, souvent très courts et souvent prédictibles. Ces systèmes sont utilisés dans une variété de domaines, tels que les systèmes embarqués dans les avions, les voitures, les satellites, les équipements médicaux, etc. Ils sont également utilisés dans les systèmes de contrôle industriels, les applications financières et les jeux vidéo, où le temps de réponse est critique.

I.4.7.2 Classification selon la structure :

a. **Traitement par lots :** Connu sous le nom de traitement par lots ou batch processing en anglais, est un mode de traitement informatique où un ensemble de tâches similaires est regroupé et exécuté en bloc, Dans le traitement par lots, les tâches sont généralement regroupées dans des fichiers de lot (batch files) ou des scripts, qui contiennent une séquence d'instructions à exécuter par l'ordinateur. Ces fichiers de lot peuvent inclure des commandes pour exécuter des programmes, traiter des données, ou effectuer d'autres opérations informatiques.

b. **Monolithique :** Le noyau **(kernel)** du système d'exploitation et la plupart des services système fonctionnent en mode noyau **(kernel mode),** et les pilotes et les services utilisateurs fonctionnent en mode utilisateur **(user mode).** Exemples : **Linux**, **Windows**.

c. **Micro-noyau :** Un noyau minimaliste qui fournit uniquement les services essentiels, et la plupart des fonctionnalités sont implémentées en tant que processus utilisateurs. Exemples : **QNX**, **MINIX**.

d. **Hybride :** Combinant des éléments des architectures monolithique et **micro-noyau**. Exemple : **MacOs**.

I.4.7.3 Classification selon la méthode de gestion des processus :

a. **Systèmes mono-utilisateur :** Conçus pour prendre en charge un seul utilisateur à la fois.
Exemple : **MS-DOS.**

b. **Systèmes multi-utilisateurs :** Permettent à plusieurs utilisateurs d'accéder simultanément au système et d'utiliser ses ressources. Exemple : **UNIX**, **Linux**.

c. **Systèmes Multi-processeurs :** Système avec plusieurs processeurs Parallèle, possèdent plus d'un processeur en étroite communication, Les systèmes multiprocesseurs les plus courants

- **SMP (Symmetric Multiprocessing) :** Dans ce type de configuration, chaque processeur partage l'accès à la même mémoire centrale et exécute des tâches indépendamment.
- **NUMA (Non-Uniform Memory Access) :** Les systèmes **NUMA** sont similaires aux systèmes SMP, mais ils ont une architecture où l'accès à la mémoire n'est pas uniforme pour tous les processeurs.
- **Cluster** multiprocesseur : Dans un cluster multiprocesseur, plusieurs ordinateurs individuels, appelés nœuds, sont reliés entre eux par un réseau haute vitesse.
- Manycore et **MIC** (Many Integrated Core) : Ces systèmes comportent un grand nombre de cœurs de processeur sur une seule puce.

d. **Systèmes multi-tâches :** Capables d'exécuter plusieurs processus simultanément. Exemple : **Windows**, **MacOs**.

I.4.7.4 Classification selon l'interface utilisateur :

a. **Interface en ligne de commande (CLI) :** Les interactions avec le système se font en saisissant des commandes textuelles. Exemples : **Unix**, **Linux**, **MS-DOS**.

b. **Interface graphique utilisateur (GUI) :** Les interactions avec le système se font à l'aide d'éléments graphiques tels que les fenêtres, les icônes et les menus. Exemples : **Windows**, **MacOs**, **Ubuntu** (avec son environnement de bureau **GNOME**).

Figure I. 11: Interface graphique (Ex : Win 11).

I.5 Conclusion :

Cette évolution constante des systèmes d'exploitation reflète les progrès technologiques et les besoins changeants des utilisateurs, permettant aux ordinateurs et appareils électroniques de devenir de plus en plus puissants et polyvalents, assurant ainsi une utilisation optimale et sécurisée.

CHAPITRE-II- DEVELOPPEMENT D'UN PROGRAMME

II.1 Introduction :

Le développement d'un programme est le processus de création de logiciels informatiques qui résolvent des problèmes spécifiques ou exécutent des tâches déterminées. Ce processus implique plusieurs étapes clés, allant de la conception initiale à la mise en œuvre jusqu'à la maintenance du programme.

II.2 Objectif :

L'objectif d'un programme peut varier en fonction de nombreux facteurs, tels que le domaine d'application, les besoins des utilisateurs et les exigences spécifiques du projet, Il est important de définir clairement l'objectif de votre programme dès le début du processus de développement, car cela vous aidera à orienter vos décisions de conception et à garantir que le produit final répond aux besoins de vos utilisateurs.

L'objectif de ce cours est de fournir aux étudiants :

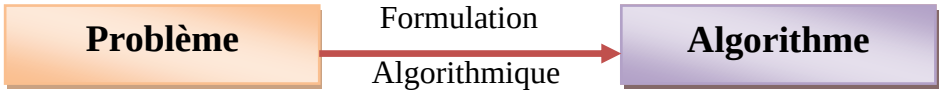
- Une compréhension complète des principes, des techniques et des outils nécessaires pour développer des programmes informatiques.
- Une connaissance approfondie des mécanismes fondamentaux qui régissent l'exécution des programmes informatiques.

II.3 Définition :

Le développement d'un programme est un mécanisme de base d'exécution des programmes, depuis l'analyse du problème jusqu'à sa mise au point, nécessite de nombreux outils logiciels qui constituent un environnement de programmation. Pour fonctionner, ces outils utilisent les services du système d'exploitation.

Pour qu'un problème puisse être interprété, exécuté et donne les résultats attendus par le processeur de la machine, il faut qu'il soit formulé dans un langage compréhensible par la machine et il faut que le système d'exploitation pilote son exécution.

Pour le faire, le programmeur commence au début d'écrire son problème d'une manière structurée sous forme d'un algorithme.



Cet algorithme est écrit en langage humain et pour le rendre exécutable par l'ordinateur, le programmeur fait le passer par plusieurs étapes tout en utilisant des outils de langage de programmation et des outils système.

II.4 Chaine de préparation d'un programme :

Le cheminement d'un programme dans un système d'exploitation peut être décrit de manière générale comme suit :

a) Lancement du programme : Le processus démarre lorsque l'utilisateur lance le programme à partir de l'interface utilisateur ou en utilisant une commande en ligne de commande.

b) **Chargement en mémoire** : Le système d'exploitation alloue de l'espace mémoire pour le programme et charge son code exécutable ainsi que les données nécessaires en mémoire principale (RAM). Cette opération peut également impliquer le chargement de bibliothèques partagées ou de fichiers de ressources.

c) **Initialisation :** Le système d'exploitation exécute le code d'initialisation du programme, qui peut inclure l'allocation de ressources, l'initialisation de variables, la configuration de l'environnement d'exécution, etc.

d) **Exécution du programme :** Le système d'exploitation commence à exécuter le code du programme, en suivant les instructions séquentiellement à moins qu'il ne rencontre des structures de contrôle telles que des boucles ou des branchements conditionnels.

e) **Interaction avec le système d'exploitation :** Le programme peut interagir avec le système d'exploitation en appelant des fonctions système pour effectuer des opérations telles que la lecture/écriture de fichiers, la communication réseau, l'allocation de mémoire, etc.

f) **Terminaison du programme :** Le programme se termine lorsque son code principal atteint la fin ou lorsqu'une instruction explicite de sortie est rencontrée. À ce stade, le système d'exploitation libère les ressources allouées au programme et récupère l'espace mémoire utilisé par celui-ci.

Ce processus peut varier légèrement en fonction du système d'exploitation spécifique et des caractéristiques du programme, mais ces étapes fournissent une vue générale du cheminement d'exécution d'un programme dans un système d'exploitation.

II.5 Cheminement d'un programme dans un système d'exploitation :

Le passage d'un programme de la forme externe à la forme interne se fait en plusieurs étapes, selon le cheminement suivant : Un programme est généralement écrit dans un langage évolué (**Pascal, C, VB, Java, etc.**), pour faire exécuter un programme par une machine.

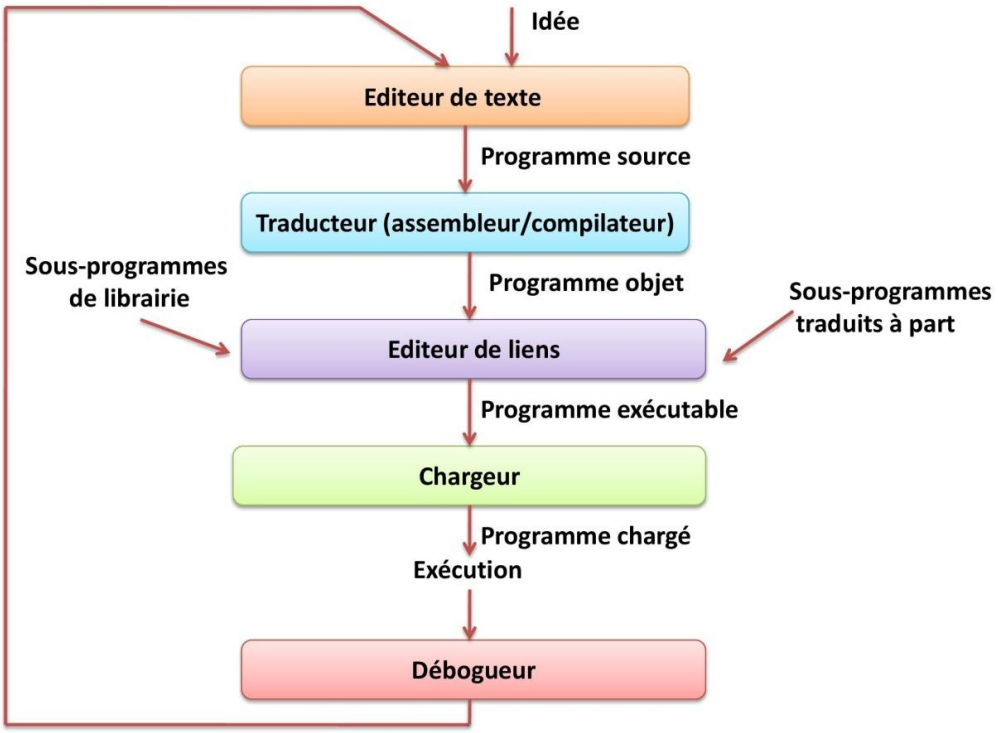


Figure II. 1: Etapes de préparation d'un programme.

II.6 Les étapes de cheminement d'un programme :

Ces étapes permettent de bien traduire le code source en code machine tout en permettant au programmeur de corrige les erreurs apparues dans chaque étape (voire **Figure II.1**).

II.6.1 L'éditeur de textes (L'édition de programme) :

Un éditeur de texte **(text editor)** est un logiciel interactif qui permet de saisir du texte à partir du clavier et de l'enregistrer dans un fichier.

C'est l'étape dont on saisit l'algorithme établit pour résoudre le problème tout en le traduisant à un programme écrit dans un langage évolué **(exemple : Java, Pascal, Delphi, C++, …)** ou dans le langage assembleur. L'éditeur de texte est l'outil qui nous permet de réaliser cette étape, c'est un logiciel interactif soit associé au système, soit associé au langage de programmation. Il nous permet d'appliquer toutes les opérations nécessaires sur un fichier nommé **« Code source ».** Ce code source à une extension spécifique tout dépend du langage utilisé **(exemple : *.pas, *.java, *.cpp, …).**

II.6.2 Traducteur (La traduction en langage machine) :

Le traducteur est un logiciel **(Compiler)** qui traduit un programme source écrit en langage de haut niveau en un programme objet. C'est un outil système qui permet de traduire les instructions écrites dans un langage de programmation vers des instructions écrites dans le langage machine (Binaire), on distingue deux types de traducteurs :

- **Le Compilateur** parcourt la suite de caractères, analyse les actions du programme et la traduit en une suite d'instructions dans le langage de la machine.
- **L'interpréteur** parcourt en permanence la suite de caractères, analyse les actions définies par le programme, exprimées dans le langage évolué, et exécute immédiatement la séquence d'action de la machine qui produit les mêmes effets.

II.6.2.1 Compilateur :

Un compilateur est un programme qui traduit des programmes écrits dans des langages évolués **(Pascal, C, Ada, Java, …etc.)** en programme binaires ou en langage machine, appelés aussi objets. Le langage machine est un ensemble de codes binaires directement décodables et exécutables par la machine.

Le Compilateur est un logiciel de complexité grandissante, permettant de traduire un programme source vers le langage machine en passant par :

- **L'analyse lexicale :** Le rôle de cette phase est la reconnaissance des unités lexicales **(tokens)**, en inspectant le code source caractère par caractère. Ces unités peuvent être des nombres, des identificateurs, des mots clés, des opérateurs, … Chaque unité est décrite par un type **(mot clé, identificateur, constante, opérateur, …)** et une valeur **(le nom qui figure dans le code source)**. Alors le résultat de cette étape :
  - La création de la table de symbole.
  - Elimination de blancs et de commentaires.
  - Signalisation des erreurs lexicales.
- **L'analyse syntaxique :** Vérifie que le code source à compiler respecte bien les règles syntaxiques du langage. Le résultat est la création de l'arbre syntaxique en se basant sur les unités lexicales et l'ensemble des règles syntaxiques du langage. Elle met à jour la table des symboles en ajoutant les informations manquantes comme les types des identificateurs, et elle signale les erreurs dues au non-respect de la syntaxe du langage.
- **L'analyse sémantique :** Qui vérifie la sémantique du programme. Cet analyseur utilise l'arbre syntaxique pour identifier les opérandes et les opérations et il utilise la table des symboles pour identifier les types des opérandes. Il signale les erreurs sémantiques qui peuvent être faites dans le code.

- **L'optimisation de code :** L'optimisation du code est une stratégie de modification du programme qui s'efforce d'améliorer le code intermédiaire, de sorte qu'un programme utilise le moins de mémoire possible, minimise son temps d'exécution et offre une vitesse élevée.

  - Il s'agit de la quatrième étape d'un compilateur, qui vous permet de choisir d'optimiser ou non votre code, ce qui la rend réellement optionnelle.

  - Il permet de réduire l'espace de stockage et d'augmenter la vitesse de compilation.

  - Il prend le code source en entrée et tente de produire un code optimal.

  - Le fonctionnement de l'optimisation est fastidieux ; il est préférable d'employer un optimiseur de code pour accomplir cette tâche.

- **Transformation en code objet :** C'est la partie qui réalise effectivement la traduction du code source en code objet.

Lors de ces étapes le compilateur signale des erreurs à corriger qui peuvent apparaître dans chaque étape. Le résultat de cette phase est un fichier nommé code objet a une extension spécifique selon le langage de programmation **(exemple : *.obj Pascal, *.class Java, …) [2].**

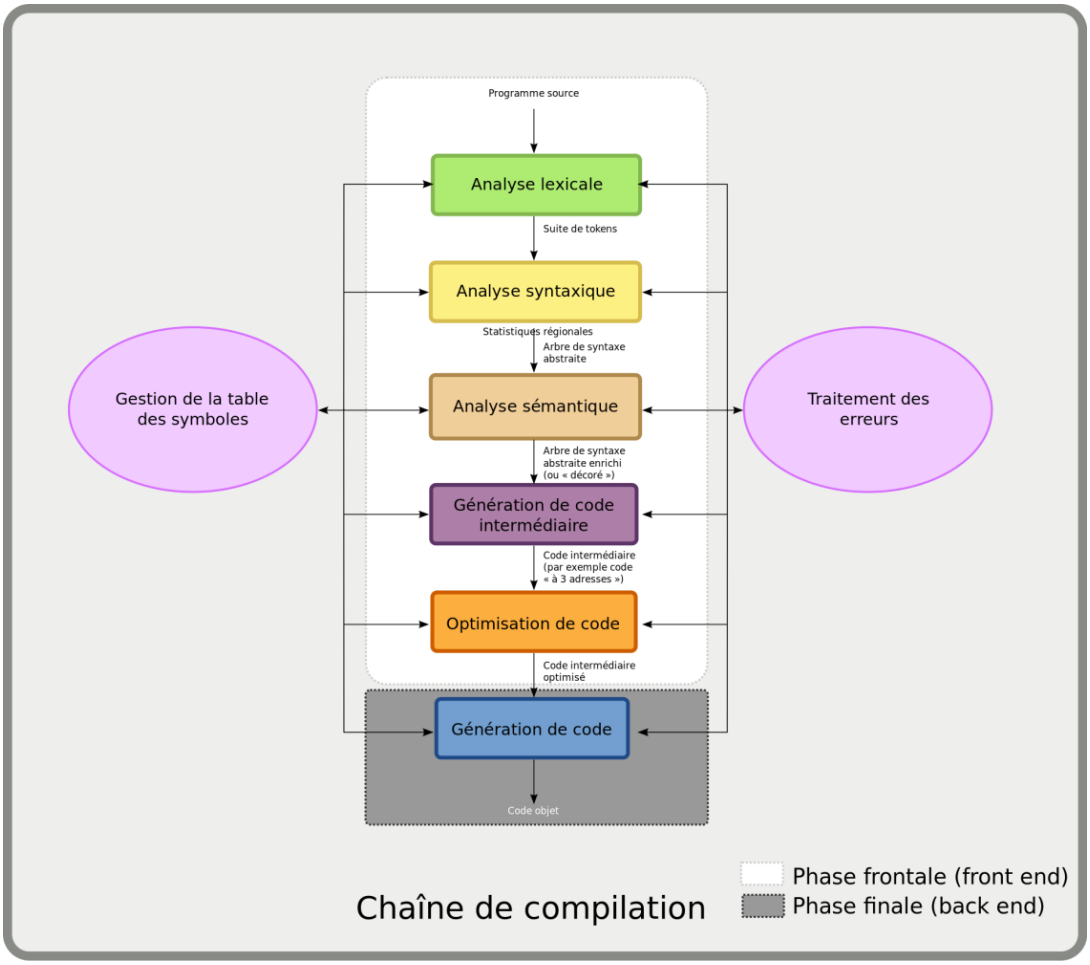


Figure II. 2: Chaine de compilation**.**

II.6.2.2 L'interpréteur :

L'interpréteur est un logiciel qui permet de traduire le programme vers le langage machine et de l'exécuter en même temps. Ce type de traducteur ne résulte pas un fichier contenant le code objet ce qui impose l'interprétation à chaque fois d'exécution **(exemple : l'interpréteur HTML).**

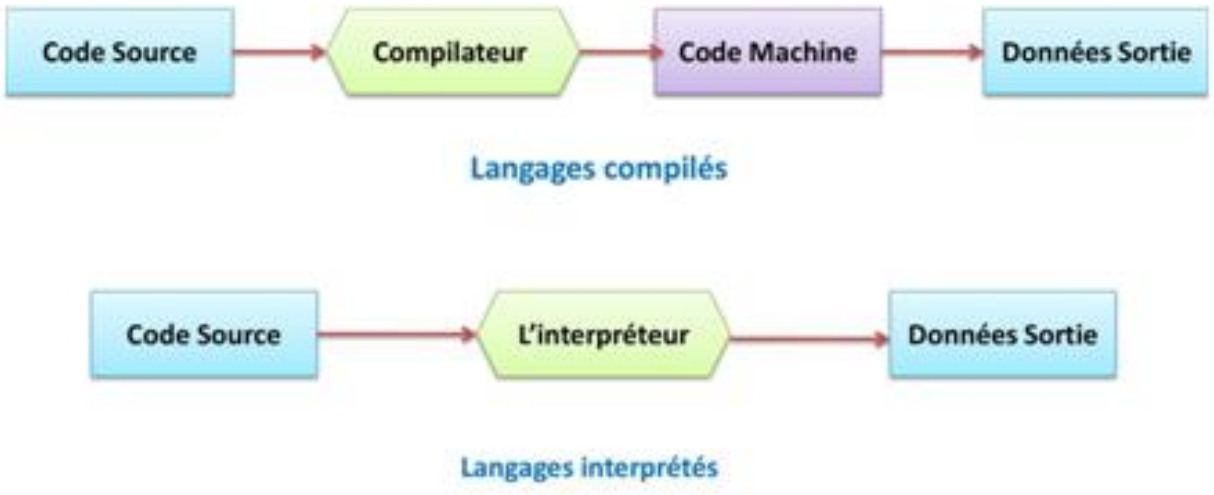


Figure II. 3: Différence entre langages interprétés et langages compilés.

N.B : le compilateur définit deux types de codes dans le code objet :

- Le code absolu qui a une adresse fixe dans la mémoire.
- Le code translatable qui peut se mettre dans n'importe quel emplacement dans la mémoire.

L'opération de translation consiste à ajouter à chaque emplacement qui contient une adresse, la valeur de l'adresse effective de l'implantation finale dans la mémoire

II.6.3 L'éditeur de liens :

Un éditeur de liens **(Linker)** est un logiciel qui permet de combiner plusieurs programmes objet en un seul programme exécutable.

Lors de la traduction d'un programme en code objet, le compilateur associe à chaque instruction son type, dont on peut avoir :

- Des données variables ;
- Des données constantes ;
- Des instructions ;
- Des appels à des procédures, des modules ou des variables externes.

Dans le dernier type, le programmeur peut référencer des structures ou des modules externes par rapport au module en question (il s'agit des références externes de la bibliothèque du langage ou personnelles). Alors le compilateur marque ces références sans les ramener et les ajouter au code objet.

Donc il associe à chaque référence soit :

- Un lien interne au programme, mais qui peut être accédé par d'autre modules → lien utilisable **(définition externe).**
- Un lien externe appartient à un autre module appelé dans ce programme → lien à satisfaire **(référence externe).**

L'éditeur de lien est un outil système qui a comme objectif d'accomplir la tâche du compilateur en ajoutant les codes objets de tous les liens à satisfaire dans le code objet principal. Alors l'éditeur de liens cherche l'origine de chaque référence externe et établit une table globale des modules contenant les informations : nom du module, taille et adresse d'implantation.

Le travail de l'éditeur de liens est basé sur la table de fonctions qui contient toutes les informations concernant la bibliothèque. On peut distinguer deux types des éditeurs de liens :

- **Editeur de lien statique :** C'est l'édition de liens qui s'établit une fois pour toute et elle engendre un fichier résultat liant toutes les références externes avec le programme principal, nommé le fichier exécutable. Ce type des éditeurs de liens ne se refait pas à chaque exécution.
  Exemple : L'éditeur de liens du langage Pascal → des fichiers *.exe.
- **Editeur de liens dynamiques :** C'est l'édition de liens qui s'établit à chaque demande d'exécution du programme, elle n'engendre pas un fichier exécutable.
  Exemple : l'éditeur de liens du langage Java.

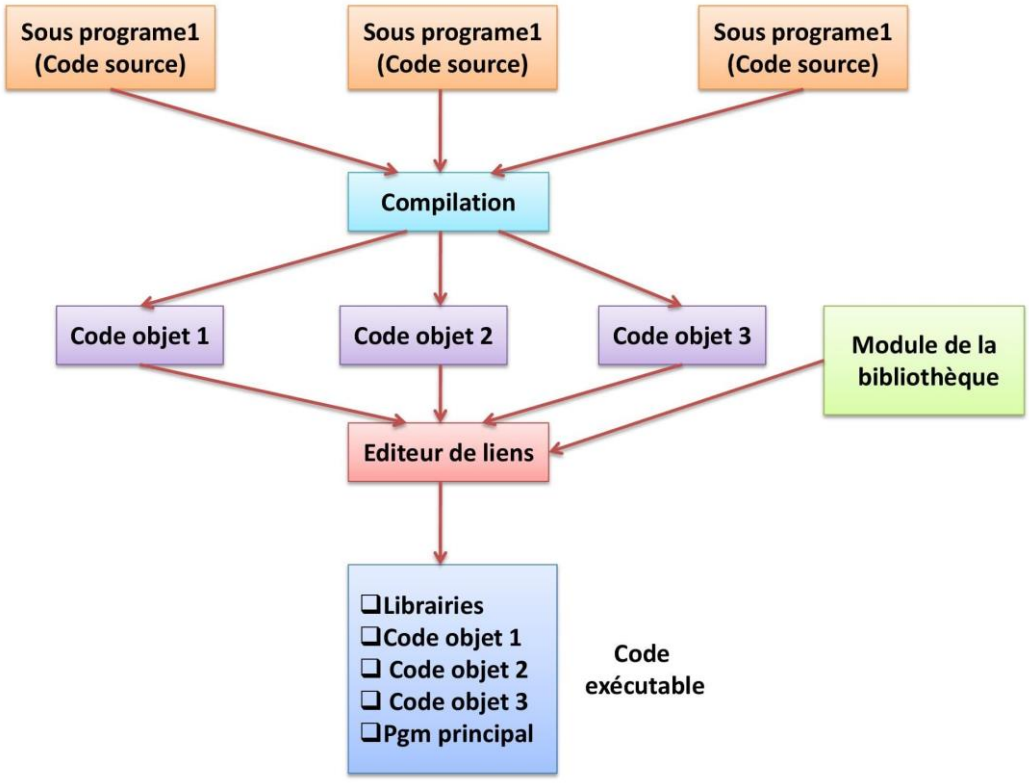


Figure II. 4: L'édition de liens.

II.6.4 Le Chargement :

Le programme exécutable, obtenu après l'édition de liens, doit être chargé en mémoire centrale pour être exécuté. Le chargeur **(Loader)** s'occupe de cette tâche.

Cette phase est faite par un outil système nommé le chargeur. Cette étape consiste à lire les instructions en mémoire secondaire, et les transférer en mémoire centrale tout en lisant au début l'adresse de chargement fournit par l'éditeur de liens. Dans ce cas le programme devient un processus.

On distingue deux types de chargement :

- **Le chargement absolu :** Le code chargé ne doit être pas met dans un autre bloc de données différent de celui indiqué dans le code objet, (d'une autre façon recopier le code).
- **Le chargement relogeable (translatable) :** Le chargement peut se réaliser à n'importe quelle adresse dans la mémoire centrale, la façon de translater les adresses est basée sur les informations fournies par l'éditeur de liens.

II.6.5 Débogueur :

Le débogueur **(Debugger)** est un logiciel qui facilite la mise au point de programmes **(détection et correction des erreurs, ou bugs)**. Il permet d'examiner le contenu de la mémoire ainsi que le contenu des différents registres. Ainsi, on peut suivre l'exécution pas à pas, c'est à dire instruction après instruction, et afficher les valeurs des variables à tout moment et mettre en place des points d'arrêts sur des conditions ou des lignes du programme. Il offre au programmeur la possibilité de contrôler l'exécution et de détecter l'origine des erreurs non corrigées.

II.7 Conclusion :

Comprendre ces étapes est crucial pour les développeurs, car elles permettent de diagnostiquer et de résoudre les problèmes liés à la performance, à la compatibilité et à la fiabilité des programmes. En maîtrisant chaque phase du cheminement d'un programme, les développeurs peuvent optimiser leurs codes et garantir une exécution fluide et efficace sur divers environnements et plateformes.

CHAPITRE-III- SYSTEME DE GESTION DES FICHIERS

III.1 Introduction :

Le volume des données traitées par les applications informatiques atteignant plusieurs giga et téraoctets, ces données ne peuvent pas être stockées dans la mémoire centrale. On souhaite également disposer d'un stockage à long terme qui ne disparaisse pas lorsqu'on éteint la machine.

Le principe consiste à stocker ces données dans des mémoires secondaires sous forme de fichiers, c'est-à-dire de **suites de blocs** (la plus petite unité que le périphérique de stockage est capable de gérer). Le contenu de ces blocs, simple suite de chiffres binaires, peut être interprété selon le format de fichier comme des caractères, des nombres entiers ou flottants, des codes d'opérations machines, des adresses mémoires, etc. L'échange entre les deux types de mémoires se fait ensuite par transfert de blocs.

III.2 Objectif :

L'objectif principal d'un système de gestion des fichiers **(SGF)** est de fournir une méthode efficace pour stocker, organiser, récupérer et protéger les fichiers sur un système informatique. Et de permettre l'accès au contenu du fichier (l'ouverture et la fermeture du fichier, sa création, lecture et écriture dans le fichier, sa recopie à un second emplacement ou sa suppression) à partir de son chemin d'accès, formé d'un nom précédé d'une liste de répertoires imbriqués.

En résumé, un **SGF** vise à simplifier la gestion des fichiers et à garantir la disponibilité, l'intégrité et la sécurité des données pour les utilisateurs et les applications sur un système informatique.

III.3 Définitions :

III.3.1 Le fichier (File) :

Un fichier est un ensemble d'informations structurées regroupées en vue de leur conservation (notion de persistance) et de leur réutilisation dans un système informatique.

- **Un fichier logique** : Est le regroupement de tous les enregistrements décrivant le même type d'entité.

Figure III. 1: Structure d'un fichier logique.

- **Un fichier physique** : Est un ensemble de blocs physiques composé chacun par **1** à **n** secteurs. Le secteur étant l'unité d'allocation du disque.

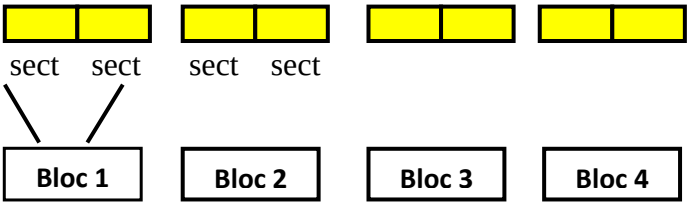


Figure III. 2: Structure d'un bloc.

Le système d'exploitation établit la correspondance entre le fichier et le système binaire utilisé lors du stockage de manière transparente pour les utilisateurs. Dans un fichier on peut écrire du texte, des images, des calculs, des programmes…etc **[3].**

Les fichiers sont généralement créés par les utilisateurs. Toutefois certains fichiers sont générés par le système ou certains outils comme les compilateurs.

Afin de différencier les fichiers entre eux, chaque fichier a un ensemble **d'attributs** qui le décrivent. Parmi ceux-ci on retrouve : **le nom**, **l'extension**, **la date et l'heure de sa création** ou de **sa dernière modification**, **la taille**, **la protection**. Certains attributs sont indiqués par l'utilisateur, d'autres sont complétés par le système d'exploitation.

III.3.2 Le système de fichiers (File System) :

C'est la partie du système d'exploitation qui se charge de gérer les fichiers. La gestion consiste en la création (identification, allocation d'espace sur disque), la suppression, les accès en lecture et en écriture, le partage de fichiers et leur protection en contrôlant les accès.

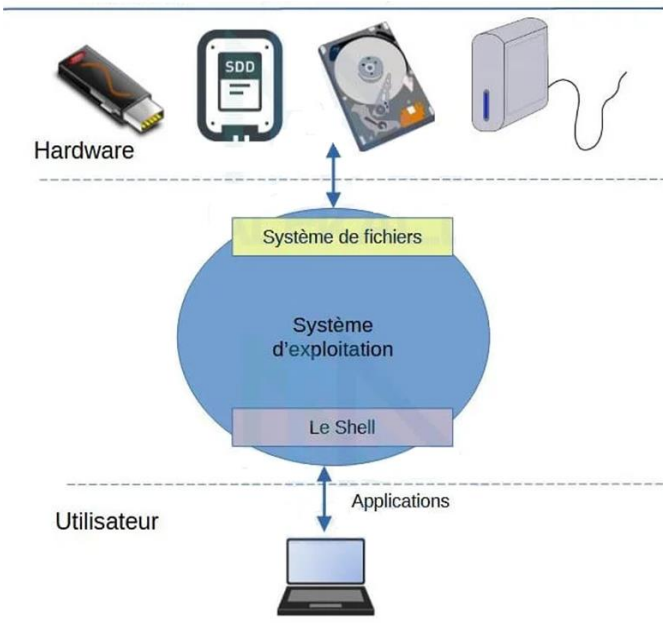


Figure III. 3: Le système de fichiers.

Les fichiers sont regroupés dans des répertoires. Un répertoire peut contenir soit des fichiers, soit d'autres répertoires.

III.3.3 Le répertoire :

Peut être considéré comme une table sur le support permettant de référencer tous les fichiers existants avec leurs noms et leurs caractéristiques principales qui peuvent varier d'un système d'exploitation à un autre. Parmi ces caractéristiques, on peut citer le nom, le type, la taille, le propriétaire, la protection, la date de création et la localisation sur disque (ou l'adresse du premier enregistrement) **[3]**.

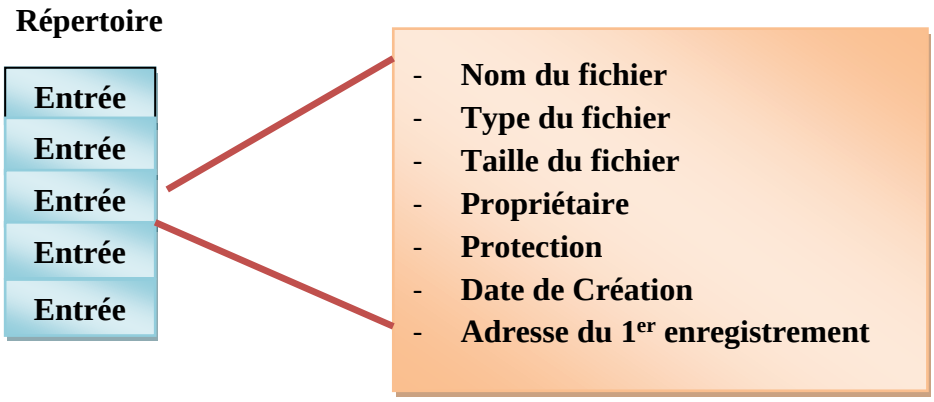


Figure III. 4: Structure d'un répertoire.

III.4 Un système de gestion de fichiers (SGF) :

C'est un ensemble de méthodes et de structures de données permettant de manipuler les données **(fichiers et répertoires)** stockées sur des mémoires secondaires. Les données sont regroupées par le **SGF** dans des fichiers localisés à partir d'un chemin d'accès. Le **SGF** lui-même est généralement composé de plusieurs niveaux différents. La figure suivante donne un exemple d'architecture de **SGF** :

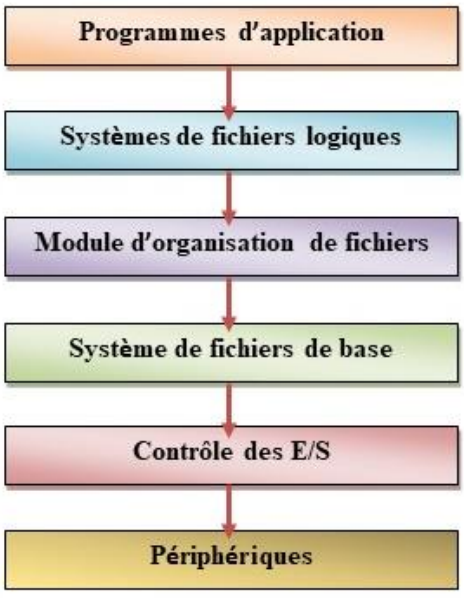


Figure III. 5: SGF en couches.

III.4.1 Les fonctionnalités d'un SGF :

- **Conservation permanente des fichiers.**
- **Manipulation des fichiers :** Le **SGF** fournie un certain nombre d'opérations pour la manipulation des fichiers et des répertoires (création, insertion, suppression, modification).
- **Allocation de zones mémoires :** Le **SGF** alloue à chaque fichier un nombre variable de blocs (suite d'octets de taille fixe).
- **Localisation des fichiers :** Chaque fichier est décrit par un ensemble d'informations qui permettent d'identifier son emplacement (nom, adresse, etc.).
- **Partage des fichiers :** Assurer une utilisation d'un même fichier / répertoire / disque par des utilisateurs différents.
- **Protection des fichiers :** Le **SGF** doit assurer la sécurité et la confidentialité des données surtout en cas de partage. Des droits d'accès sont ainsi associés à chaque fichier pour assurer une protection contre les accès interdits.

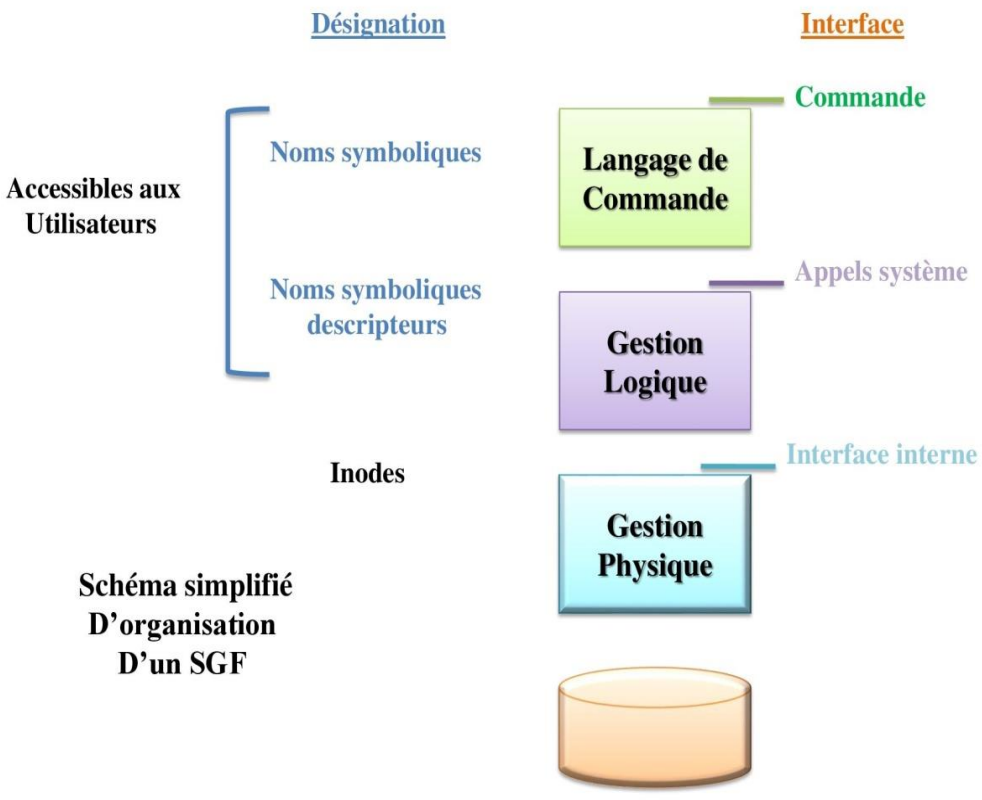


Figure III. 6: Organisation d'un SGF.

L'utilisateur possède une vue abstraite de l'organisation des fichiers qui sont désignés par des noms symboliques. Chaque système d'exploitation possède un système d'identification interne des fichiers. Le système d'exploitation **UNIX**, par exemple, identifie ses fichiers par ce qu'on appelle un numéro **d'inode** ou **i-Numbers**. La gestion logique concerne l'organisation des fichiers et des répertoires ainsi que toutes les fonctions qui vont avec comme la gestion des droits d'accès, le partage et la protection.

La gestion physique concerne l'allocation de l'espace disque (différentes stratégies d'allocation de l'espace disque sont définies en détail dans la section suivante).

III.5 Techniques d'allocation des blocs sur le disque :

Un fichier physique est constitué d'un ensemble de blocs physiques. La question qui se pose c'est comment allouer des blocs physiques pour les fichiers. Différents points doivent être pris en considération pour l'allocation de l'espace disque : l'optimisation du temps d'accès, le coût de stockage et l'évolution de la taille des fichiers (la fragmentation).

On distingue différentes méthodes pour allouer un fichier physique :

- Allocation contiguë (séquentielle simple) ;
- Allocation par blocs chaînés ;
- Allocation indexée ;
- Allocation par groupes ;
- Allocation dynamique.

III.5.1 Allocation contiguë :

La méthode de l'allocation contiguë demande que chaque fichier occupe un ensemble de blocs contigus sur le disque. Les blocs de données sont stockés de manière consécutive sur le disque. Cela signifie que chaque fichier occupe un bloc contigu de l'espace disque. Bien que cela simplifie l'accès aux données, cela peut entraîner de la fragmentation lorsque des fichiers sont supprimés ou modifiés. Les adresses disques définissent un ordre linéaire sur le disque.

L'allocation contiguë d'un fichier est définie par l'adresse disque et la longueur, en unités blocs, du premier bloc. Si le fichier est d'une longueur de **n** blocs et démarre à l'emplacement **b**, il occupe donc les blocs **b, b+1, b+2, …, b+n-1**. L'entrée du répertoire pour chaque fichier indique l'adresse du bloc de début et la longueur de la zone allouée au fichier. Ce qui montre l'exemple dans la **Figure III.6** suivante :

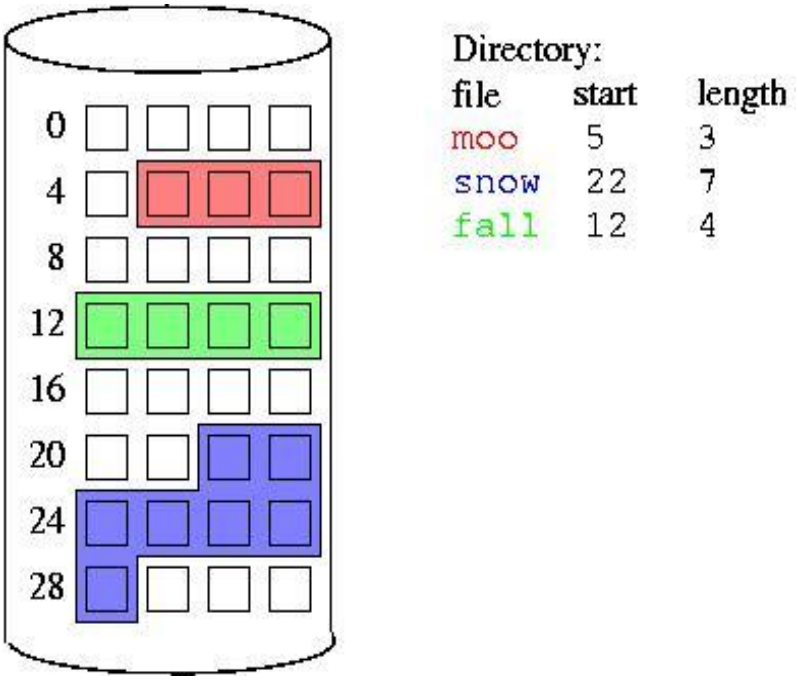


Figure III. 7: Méthode d'allocation contiguë.

- **Ajout d'un enregistrement :** L'enregistrement est ajouté à la suite du dernier bloc du fichier s'il y a suffisamment d'espace sinon il serait nécessaire de réaliser un déplacement du fichier dans un espace plus grand (réorganisation).
- **Suppression d'un fichier :** On parle de suppression logique. Un compteur est associé à chaque entrée fichier dans le répertoire correspondant. Ce compteur indique le nombre d'enregistrements du fichier. Dans le cas de la suppression, ce compteur est remis à zéro.
- **Avantage :**
  - Rapidité de l'accès (les blocs étant contigus, on limite les déplacements de la tête, lecture / écriture, coûteux en temps).
- **Inconvénient :**
  - Le dernier bloc a toutes chances d'être sous-utilisé et ainsi, on gaspille de la place ;
  - Il est difficile de prévoir la taille qu'il faut réserver au fichier ;
  - La perte d'espace sur le disque ;
  - Problème de fragmentation externe.

III.5.2 Allocation chaînée (non contigüe) :

La méthode d'allocation chaînée est proposée pour résoudre les problèmes de la méthode d'allocation contiguë. En effet, avec cette méthode, chaque fichier est une liste chaînée de blocs de disques. Ces blocs peuvent être dans n'importe quel emplacement sur le disque. Le répertoire contient un pointeur sur le premier et le dernier bloc du fichier. Chaque bloc, excepté le dernier, contient un pointeur vers le bloc

suivant. Le dernier bloc d'un fichier contient un pointeur vers le premier emplacement libre. Ce qui montre l'exemple dans la **Figure III.7** qui suite :

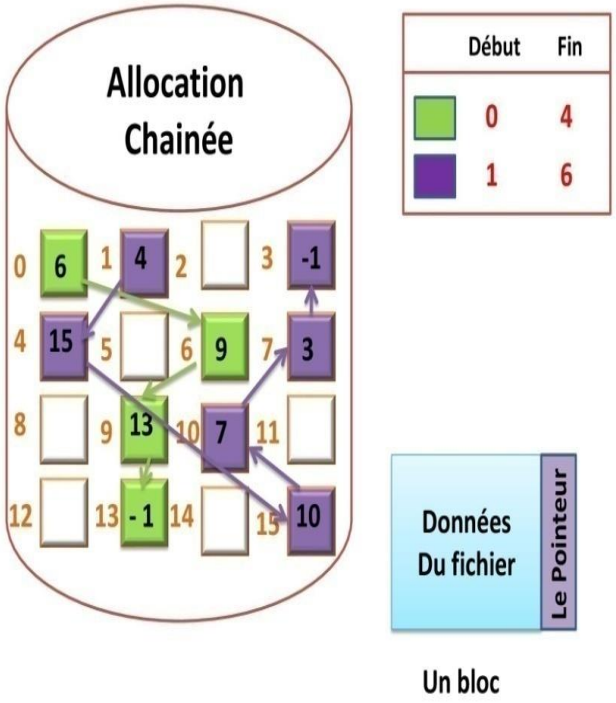


Figure III. 8: Méthode d'allocation chaînée.

- **Ajout d'un enregistrement :** L'ajout d'un enregistrement consiste à rechercher des blocs disponibles pour contenir l'enregistrement. A chaque fois qu'on alloue un bloc on le chaîne au dernier bloc trouvé.
- **Suppression d'un fichier :** La suppression suit le même principe de la suppression dans le cas d'allocation contiguë.
- **Avantage :**
  - Elimination du problème de fragmentation externe ;
  - Plus simple d'étendre un fichier (ajout d'un enregistrement) que dans le cas de la méthode contiguë. Également, grâce à l'allocation par bloc individuel, tout bloc libre peut être utilisé.
- **Inconvénients** :
  - L'accès au fichier est totalement séquentiel, on doit toujours commencer le parcours du fichier à partir du début ;
  - La perte d'un chaînage entraîne la perte de tout le reste du fichier. Pire encore, il suffit qu'une valeur soit modifiée dans un pointeur pour qu'on se retrouve dans une autre zone de la mémoire.

III.5.3 Allocation indexée :

L'allocation chaînée résout les problèmes de fragmentation, cependant elle ne peut pas supporter l'accès direct de façon efficace, car les pointeurs vers les blocs ainsi que les blocs eux-mêmes sont dispersés dans tout le disque et ils doivent être récupérés dans l'ordre. L'allocation indexée résout ce problème en rangeant tous les pointeurs dans un seul emplacement : le bloc d'index. Il suffit de retirer les pointeurs des blocs et de les placer dans une structure de données gardée en mémoire centrale, ainsi, les informations sur les numéros de blocs peuvent être obtenue à tout moment. Voire (**Figure III.8)**.

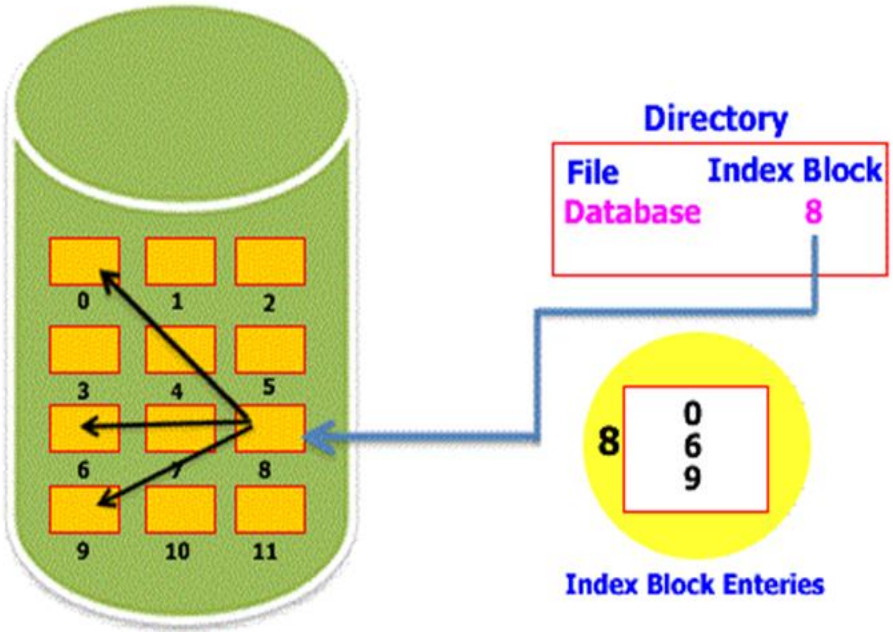


Figure III. 9: Méthode d'allocation indexée.

- **Avantages :**
  - Implémentation efficace de l'accès direct ;
  - Table des i-nodes de taille proportionnelle au nombre de fichiers (**Cf. FAT** : taille du disque).
- **Inconvénients :**
  - Problème de la taille des index. Besoin d'un espace supplémentaire pour l'index.

La plupart des systèmes actuels appliquent ce mode. **MS-DOS** utilise la **FAT (File Allocation Table)** pour y conserver les chaînages entre les blocs**. Windows NT** utilise la **MFT (Master File Table)** associé au système **NTFS (New Technology File System). UNIX**, **GNU/Linux** utilisent le **I-Node (Index node).**

a. FAT :

Les variantes incluent FAT12, FAT16 et FAT32, avec des différences dans la taille des blocs et la capacité totale de stockage prise en charge.

- **Le FAT16** est utilisé par **MS-DOS**. En FAT16, les numéros de blocs sont écrits sur 16 bits, la taille maximale adressables est alors **2 Go (216 x 32 Ko = 2097152 Ko = 2Go)**.
- **Le FAT32** est pris en charge par **Windows 95** et les versions qui ont suivis. Les numéros de blocs sont écrits sur **32** bits, la taille maximale adressable théoriquement est de **8 To (228 x 32 Ko = 8 To)**. Et Limites de taille de fichier **4 Go**. Toutefois, Microsoft la limite volontairement à **32 Go** sur les systèmes Windows **9x** afin de favoriser **NTFS**.

b. NTFS : Le système de fichiers NTFS **(New Technology File System)** est utilisé par **Windows 2000, NT, XP, Vista, 7, 8, 8,1** et **Windows 10** et **11**. Il utilise un système basé sur une structure appelée **MFT** (Master File Table), permettant de contenir des informations détaillées sur les fichiers, les numéros de blocs sont écrits sur **64 bits**, la limite physique d'un disque est de **2To**.

III.5.4 Allocation par groupes :

L'espace disque est divisé en groupes ou en blocs, et chaque groupe est alloué à un fichier. Cela peut réduire la fragmentation en allouant des blocs de manière prédictive, mais peut entraîner du gaspillage d'espace si les fichiers ne remplissent pas complètement les groupes.

- **Avantages :** Réduction de la fragmentation externe, gestion plus efficace de l'espace.
- **Inconvénients :** Complexité de gestion.

III.5.6 Allocation dynamique :

Les blocs de données sont alloués dynamiquement au fur et à mesure des besoins. Cela peut être combiné avec d'autres techniques d'allocation pour optimiser l'utilisation de l'espace disque et minimiser la fragmentation.

- **Avantages :** Utilisation efficace de l'espace, pas de fragmentation externe.
- **Inconvénients :** Complexité de gestion, accès potentiellement plus lent.

III.6 Gestions de l'espace libre :

Comme il n'existe qu'une quantité limitée d'espace disque, il est nécessaire de réutiliser l'espace des fichiers détruits pour les nouveaux fichiers. Pour cela, le système doit maintenir une liste d'espace libre. Cette liste mémorise tous les blocs libres du disque.

Pour créer un fichier, on recherche dans la liste d'espace libre la quantité requise d'espace et on alloue cet espace au nouveau fichier. Cet espace est ensuite supprimé de la liste. Quand un fichier est détruit, son espace disque est ajouté à la liste d'espace libre. Mais comment implémenter la liste d'espace libre ? Plusieurs méthodes existent dont : le vecteur binaire et la liste chaînée **[4]**.

III.6.1 Vecteur binaire :

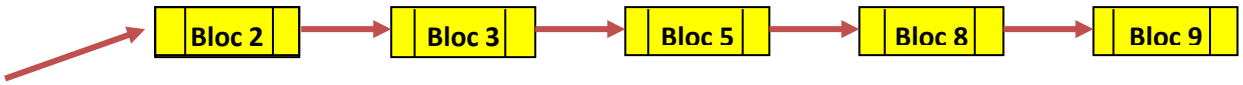
On implémente souvent la liste d'espace libre comme un tableau binaire. Chaque bloc est représenté par un bit. Si le bloc est libre, le bit est à **1**, s'il est alloué le bit est à **0**. Par exemple, si dans un disque les blocs **2, 3, 5, 8** et **9** sont libres, et les autres alloués, le vecteur représentant l'espace libre est alors :

N° bloc	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9 ….**
Bit	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**

- **Avantage :** Il est relativement facile de trouver le premier bloc libre ou les n blocs libres consécutifs.
- **Inconvénient :** La gestion du vecteur.

III.6.2 Liste chaînée :

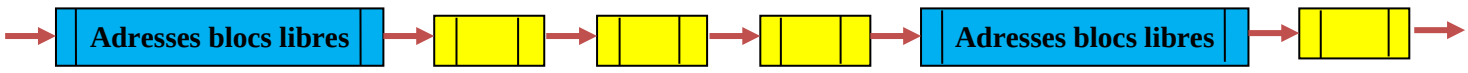
Il existe une autre approche pour représenter l'espace libre. Elle consiste à chaîner les blocs disques libres, en maintenant un pointeur sur le premier bloc libre dans un emplacement spécial du disque et en le mettant en mémoire cache.



- **Avantage :** La liste ne représente que les blocs libres.
- **Inconvénient :** Parcours de la liste.

III.6.3 Liste chaînée avec groupement :

Il existe une autre variante de la méthode de la liste chaînée : on stocke les **adresses** des **n blocs** libres dans le premier bloc libre. Les **n-1** premiers blocs sont réellement libres. Le dernier bloc contient les adresses de n autres blocs libres et ainsi de suite. L'importance de cette implémentation, c'est que l'on peut rapidement trouver les adresses d'un grand nombre de blocs libres, à la différence de la méthode chaînée standard.



III.6.4 Liste avec comptage :

La méthode de représentation d'espace libre avec comptage, profite du fait qu'il existe souvent plusieurs blocs libres contigus dispersés dans le disque. Alors, plutôt que de maintenir une liste de **n** adresses libres, on mémorise l'adresse du premier bloc libre et le nombre **x** de blocs contigus libres qui suivent le premier bloc. Chaque entrée dans la liste d'espace libre consiste donc en une adresse disque et un compteur.

III.7 Le système de gestion de fichiers d'UNIX :

Le système de gestion de fichiers **d'UNIX** est appelé **UFS** pour **Unix File System**. On distingue trois types de fichiers : les fichiers standards ou ordinaires (ex. texte), les répertoires et les fichiers spéciaux (ex. périphériques) **[15]**.

Les fichiers sont identifiés par un numéro unique le numéro **d'inode**. Un répertoire n'est qu'un fichier, identifié aussi par un **inode**, contenant une liste **d'inode** représentant chacun un fichier.

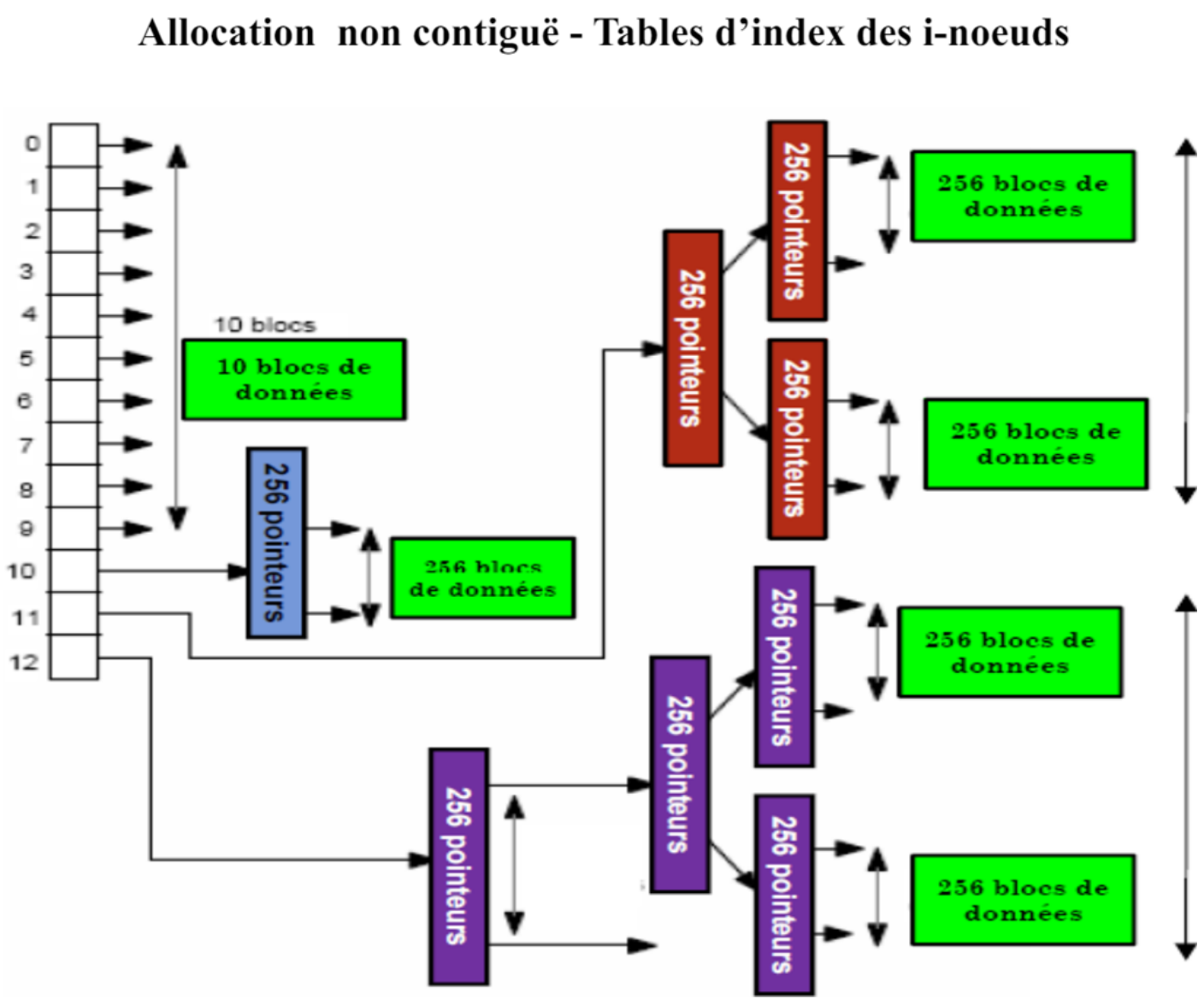


Figure III. 10: Allocation non contiguë - Tables d'index des i-nœuds.

Un bloc de **1024** octets pourra désigner jusqu'à **256** blocs de données, sachant que chacun d'eux est numéroté sur **4** octets

La taille maximale théorique d'un fichier décrit par un **i-noeud** est de :

(10x1k) + (256x1k) + (2562x1k) + (2563x1k) > 16 Go

Mais comme le champ taille du fichier dans un **i-noeud** est codé sur **32 bits**, la taille maximale effective pour un fichier est de **4 Go (2^{32}) [5].**

Chaque entrée de la table des **inodes** (ou **bloc d'index**) contient les informations sur les fichiers excepté le nom. Chaque **inode** contient un ensemble de pointeurs :

- Les premiers sont des pointeurs directs qui pointent vers les blocs de données de fichiers ;
- Les derniers sont des pointeurs indirects qui pointent vers les blocs d'index. Ces derniers contiennent à leur tour une nouvelle série de pointeurs directs et indirects.

Chaque fichier correspond un i-nœud contenant :

- Le type du fichier et les droits d'accès des différents utilisateurs ;
- L'identification du propriétaire du fichier ;
- La taille du fichier exprimée en nombre de caractères (pas de sens pour les fichiers spéciaux) ;
- Le nombre de liens physiques sur le fichier ;
- La date de dernière modification/consultation (écriture/lecture) du fichier ;
- La date de dernière modification du nœud (modification d'attributs) ;
- L'identification de la ressource associée (pour les fichiers spéciaux).

III.8 Les systèmes de fichiers virtuels :

Plusieurs systèmes de fichiers sont utilisés, souvent sur le même ordinateur, parfois avec un même système d'exploitation.

- Un système Windows peut gérer **NTFS** comme système de fichiers principal mais aussi un système **FAT-32** ou **FAT-16** par héritage.
- Un système **Linux** peut avoir un système **ext2** comme racine du système de fichiers, une partition **ext3** montée sur **/usr** et un second disque dur avec **RiserFS**.

Du point de vue de l'utilisateur, c'est un système de fichiers unique et hiérarchique. Ce qui se passe pour intégrer ces divers et incompatibles systèmes de fichiers est invisible aux utilisateurs et aux processus.

Les systèmes **UNIX** utilisent le concept de système de fichiers virtuels ou **VFS (Virtual File System)** pour tenter d'intégrer différents systèmes de fichier dans une structure ordonnée.

L'idée principale est d'extraire la partie du système de fichiers qui est commune à tous les systèmes de fichiers et de mettre le code dans une couche séparée qui **appelle les systèmes** de fichiers réels sous-jacent pour gérer les données **[5]**.

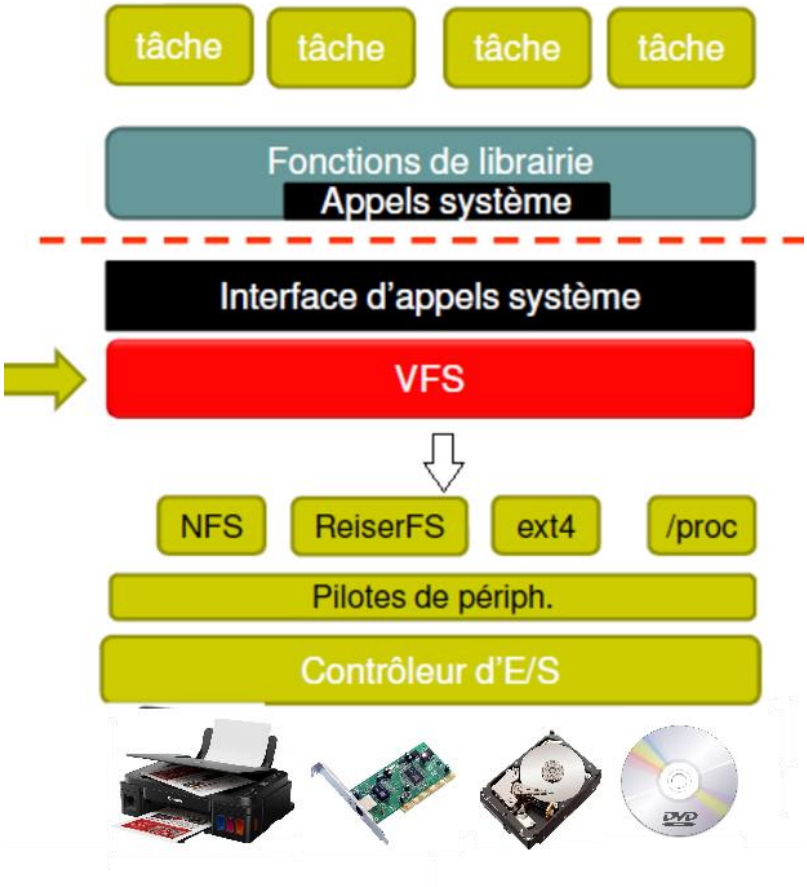


Figure III. 11: Systèmes de fichiers virtuels.

On effectue le montage d'un système de fichiers **FAT** avec la commande **shell** suivante :

#mount -t vfat /dev/hda1 /mnt/fatpartition.

III.9 Les performances d'un SGF :

Les performances d'un système de gestion de fichiers dépendent du compromis entre l'algorithme d'allocation et la fragmentation que l'on permet (optimisation physique / optimisation logique).

Les performances d'un système de gestion de fichiers (**SGF**) peuvent être évaluées selon plusieurs critères, notamment :

- **Vitesse d'accès aux fichiers** : Cela mesure la rapidité avec laquelle un fichier peut être ouvert, lu, écrit ou fermé. Les performances peuvent varier en fonction de la taille du fichier, de sa localisation sur le disque et de la méthode d'allocation des blocs physiques.
- **Débit** : Le débit représente la quantité de données qu'un **SGF** peut transférer par unité de temps. Un **SGF** performant devrait être capable de gérer de grandes quantités de données sans ralentir les opérations.

- **Latence** : La latence est le temps écoulé entre une demande d'accès à un fichier et le début de l'opération réelle. Une faible latence est souhaitable pour garantir une expérience utilisateur fluide et réactive.
- **Gestion de la concurrence :** Un bon **SGF** doit pouvoir gérer efficacement plusieurs demandes d'accès concurrentes à des fichiers, sans compromettre les performances ou la cohérence des données.
- **Tolérance aux pannes** : Un **SGF** robuste doit pouvoir récupérer rapidement après une panne, en minimisant la perte de données et en restaurant l'intégrité du système de fichiers.
- **Efficacité de l'espace disque :** Un **SGF** efficace utilise l'espace disque de manière optimale, en minimisant la fragmentation et en évitant le gaspillage d'espace.
- **Sécurité :** La sécurité des données est également un facteur important. Un **SGF** doit fournir des mécanismes robustes pour protéger les fichiers contre les accès non autorisés et les altérations malveillantes.
- **Extensibilité :** Les performances doivent rester stables même lorsque le volume de données ou le nombre d'utilisateurs augmente. Un **SGF** extensible peut s'adapter à ces changements sans compromettre les performances.

III.10 Conclusion :

L'importance des **SGF** ne cesse de croître avec l'augmentation exponentielle des données numériques. Les défis futurs incluent l'optimisation pour les nouvelles technologies de stockage (comme les **SSD NVMe**), l'amélioration de l'efficacité énergétique, et l'adaptation aux environnements de cloud computing et de stockage distribué.

CHAPITRE-IV- GESTION DES ENTREES-SORTIES

IV.1 Introduction :

L'ordinateur échange des données à travers des périphérique **E/S** externes (Électroniques : Mémoires, Magnétiques : Disque, Mécaniques : clavier, imprimante).

Pour dialoguer avec ces périphériques le microprocesseur a trois façons de communiquer avec ces derniers :

- En questionnant de façon continue le périphérique pour vérifier périodiquement que des données peuvent être lues ou écrites **(Polling)**. Scrutation (Polling) : **L'initiative est au programme**.
- En l'interrompant lorsqu'un périphérique est prêt à lire ou écrire des données **(interruption)** : **l'initiative est au périphérique**.
- En établissant une communication directe entre deux périphériques **(DMA : Direct memory acces).**

Lors d'une utilisation normale d'un ordinateur, il y a en permanence l'exécution d'instructions. Toutes ces instructions ne dépendent pas d'un même programme. En effet, il y a plusieurs programmes qui se partagent le **CPU**, les **entrées/sorties** et donc pour exécuter les instructions, il y a un procédé cyclique **d'interruption** / **sauvegarde** / **exécution** / **restauration [24]**.

IV.2 Objectifs spécifiques :

- Connaître l'utilité et l'organisation des périphériques d'entrée/sortie ;
- Connaître le principe de fonctionnement des périphériques d'entrée/sortie ;
- Comprendre le principe de communication entre **UC** et **E/S** ;
- Comprendre et maîtriser les algorithmes d'ordonnancement des disques de stockage.

IV.3 Définitions :

IV.3.1 Entrées Sorties (E/S) :

La fonction d'un ordinateur est le traitement de l'information (fonction réalisée au niveau de la **mémoire** et **l'UC**). L'ordinateur acquiert et restitue l'information au moyen **d'E/S.**

Entrées/Sorties **(E/S, Input/Output ou I/O en anglais)** désigne l'ensemble des transferts de données qui permettent au microprocesseur et à la mémoire de communiquer avec le reste

- **Entrée :** Une donnée allant du monde extérieur vers le microprocesseur.
- **Sortie :** Une donnée allant du microprocesseur vers l'extérieur.

Pour réaliser les **E/S, l'UC** doit communiquer avec les modules d'**E/S**, qu'il s'agisse d'un **périphérique** ou d'un **contrôleur** ou d'un **canal**, connectés sur des **bus**, commandé par des logiciels **driver**. Chaque module d'E/S contient un ou plusieurs registres servant à la communication avec le processeur **[6]**.

IV.3.2 Périphérique (Device) :

Un périphérique est un dispositif matériel permettant d'assurer les échanges d'informations en entrée et en sortie entre l'ordinateur et l'extérieur ou de stocker de manière permanente des informations. Appareil qui interagit avec microprocesseur et la mémoire. Certains périphériques sont branchés à l'intérieur de l'ordinateur (disques durs, carte graphique…), alors que d'autres sont branchés sur des interfaces externes de l'ordinateur (clavier, écrans, souris, etc.).

On distingue habituellement les catégories de périphériques suivantes :

- **Périphériques de sortie :** Ce sont des périphériques permettant à l'ordinateur d'émettre des informations vers l'extérieur, tels qu'un écran, une imprimante.
- **Périphériques d'entrée :** Ce sont des périphériques capables uniquement d'envoyer des informations à l'ordinateur, par exemple la souris, le clavier, etc.
- **Périphériques d'entrée-sortie :** Ce sont des périphériques capables d'envoyer des informations à l'ordinateur et permettant également à l'ordinateur démettre des informations vers l'extérieur, par exemple le modem, le disque dur **(périphériques de stockage).**

On considère qu'il y a deux grandes catégories de périphériques :

- **Les périphériques par blocs :** Dans un périphérique par blocs, on ne peut accéder à l'information que par blocs et chaque bloc possède une adresse **(**exemple : **le disque).**
- **Les périphériques par caractères** : Dans un périphérique par caractère, on accède bien sûr à l'information caractère par caractère (exemple : **le clavier**), mais on ne peut spécifier une adresse ni rechercher une information : on reçoit ou on envoie simplement un flux de caractères.

  - **La fonction de communication :**

  Les échanges d'informations entre les périphériques, le processeur central, la mémoire centrale.

La communication entre les modules du processeur et les périphériques contrôlés par un **coupleur** ou **contrôleur de périphérique [7]**. (**Figure IV.1)**.

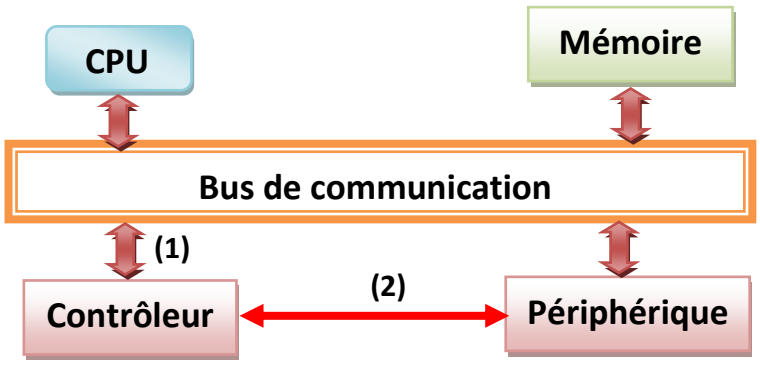


Figure IV. 1: Architecture "classique" de matériel.

IV.3.3 Contrôleur (coupleur) :

Le contrôleur sert d'interface entre le périphérique et le processeur : il reçoit les requêtes du processeur et les transforme en commandes pour le périphérique, et réciproquement, il envoie les requêtes du périphérique au processeur.

Par exemple : le module d'**E/S** servant d'interface entre l'**UC** et un disque dur sera appelé contrôleur de disque.

Les principales composantes d'un contrôleur d'**E/S** :

- La mémoire tampon pour les données (sous forme de registre) ;
- Une logique de contrôle pour décoder l'adresse et les lignes de contrôle ;
- Une ou plusieurs interfaces avec un ou plusieurs périphériques.

Les contrôleurs d'**E/S** ont plusieurs fonctions. En voici les principales :

- Lire ou écrire des données du périphérique ;
- Lire ou écrire des données de l'**UC**/Mémoire. Cela implique du décodage d'adresses, de données et de lignes de contrôle. Certains modules d'**E/S** doivent générer des interruptions ou accéder directement à la mémoire ;
- Communication avec la mémoire centrale et le microprocesseur à travers de bus dits bus d'extension (**ISA, USB, PCI, …**) ;
- Communication avec les périphériques avec pilotage ;
- Contrôler le périphérique et lui faire exécuter des séquences de tâches ;
- Tester le périphérique et détecter des erreurs ;
- Mettre certaines données du périphérique ou de l'**UC** en mémoire tampon afin d'ajuster les vitesses de communication.

Un contrôleur dispose, pour chaque périphérique qu'il gère, de trois types de registres :

- **Registres de données (Data Registers) :** Destinés à contenir les informations échangées avec le périphérique. Ils peuvent être lus (entrée) ou écrits (sortie).
- **Registre d'état (State Register) :** Qui permet de décrire l'état courant du coupleur (libre, en cours de transfert, erreur détectée,).
- **Registre de contrôle (Control Register) :** Qui sert à préciser au coupleur ce qu'il doit faire, et dans quelles conditions (vitesse, format des échanges).

Le contrôleur ou coupleur a la responsabilité de déplacer les données entre les unités périphériques qu'ils contrôlent et sa mémoire tampon ou buffer. La taille de ce buffer varie d'un contrôleur à un autre, selon le périphérique contrôlé et son unité de transfert **[8]**.

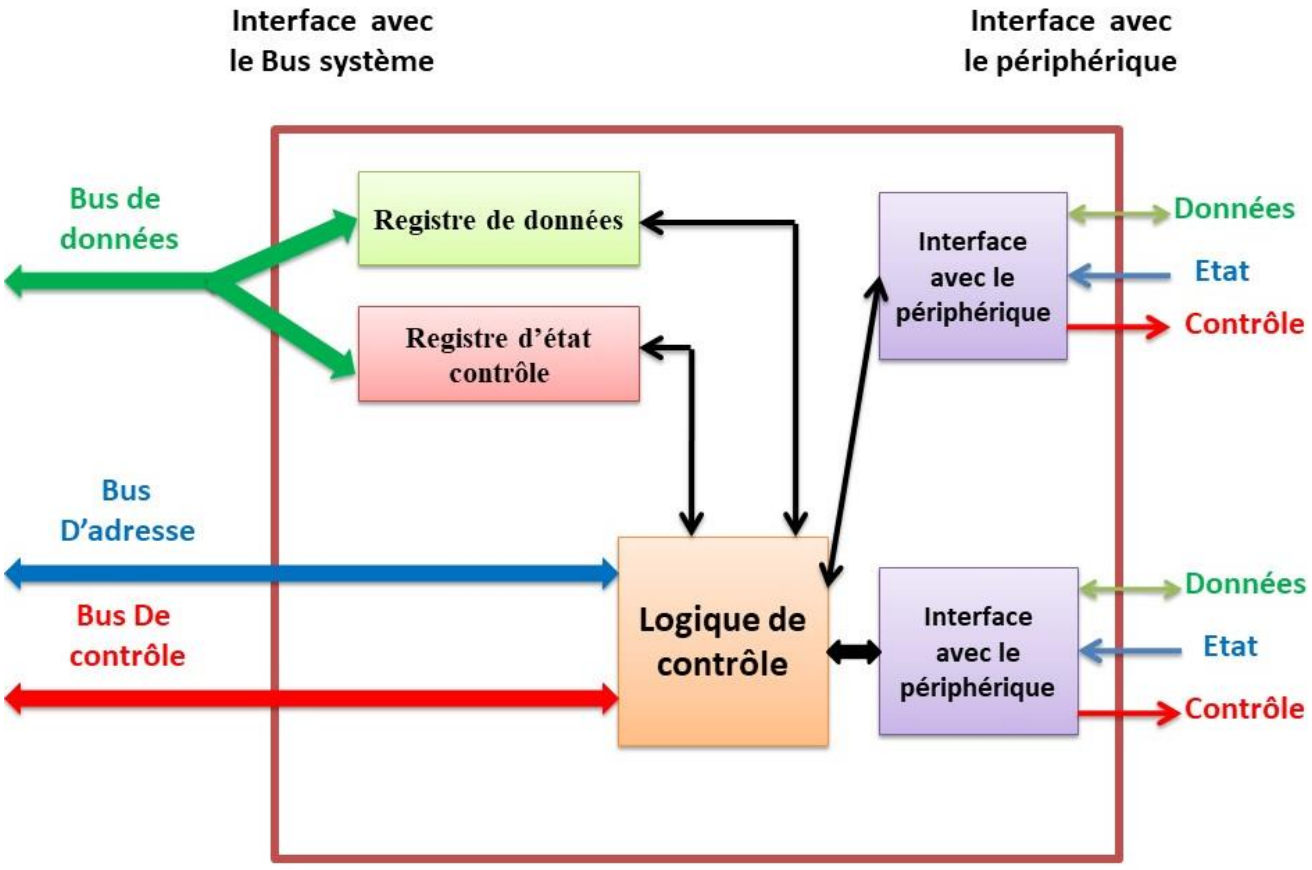


Figure IV. 2: Principaux composants d'un coupleur.

Le coupleur possède aussi une logique de contrôle pour décoder l'adresse et les lignes de contrôle (ou pour faire du **DMA**), et une ou plusieurs interfaces avec un ou plusieurs périphériques.

IV.3.4 Canaux :

Les canaux d'**E/S** (**I/O Channels**) est un processeur spécialisé dans les opérations d'**E/S**, son rôle est d'assuré la communication et la synchronisation entre le processeur et les contrôleurs.

Il ne peut pas être lancé que par un processeur central. Il peut interrompre l'**UC**. Et son rôle est de commander les contrôleurs et les périphériques.

IV.3.5 Bus :

Les contrôleurs d'**E/S** sont connectés sur des bus, reliés à d'autres bus par des contrôleurs de bus (souvent appelés **interfaces** ou **ponts**). Le processeur et la mémoire sont eux-mêmes sur des bus **[8]**.

Un bus est capable de véhiculer des signaux correspondant fondamentalement à trois types d'information (voire **Figure IV.3)** :

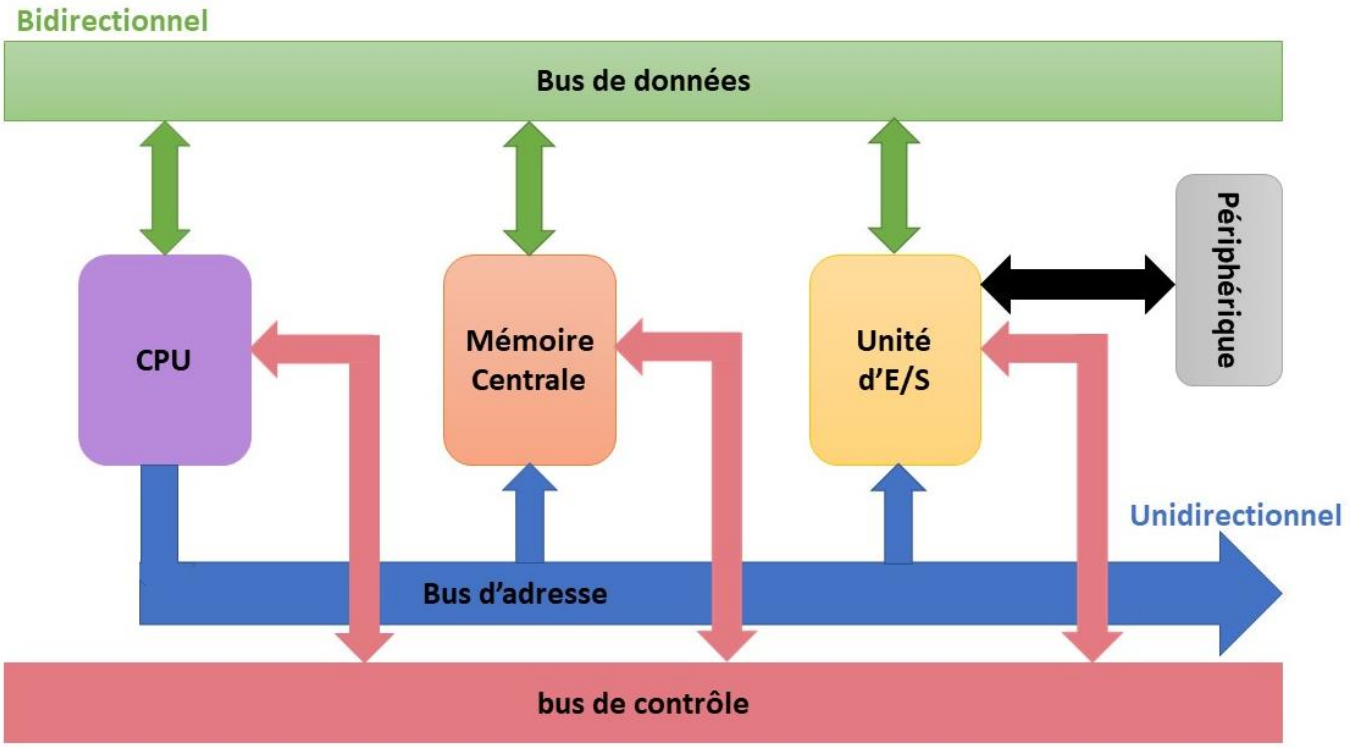


Figure IV. 3: Différentes catégories de bus.

- **Bus d'entrées/sorties des données :** Assure les transferts d'informations entre le microprocesseur et son environnement. Ces transferts sont **bidirectionnels**, c'est-à-dire que la transmission d'informations se fait dans les deux sens.
- **Bus d'adresses** : Permet la localisation des informations à traiter ou à utiliser dans l'espace mémoire. Il est **unidirectionnel**, c'est-à-dire que la transmission d'informations se fait dans un seul sens.
- **Bus de contrôle :** Assure la synchronisation des flux d'information sur les bus de données et d'adresses, permettant de faire correspondre les données de deux emplacements de stockage.

IV.3.6 Les drivers :

C'est la partie du système d'exploitation qui assure la communication entre l'utilisateur et le système. Un pilote (logiciel) fournie par le constructeur, qui facilite la communication entre un système d'exploitation et un périphérique matériel spécifique, en fournissant une interface standardisée et abstraite pour accéder aux fonctionnalités du périphérique.

En général, on organise les logiciels d'**E/S** suivant quatre couches :

Ce découpage :

- Séparer les problèmes liés au matériel.
- Regrouper les commandes pour la présentation à l'utilisateur.
- Permet d'abstraire les numéros de périphériques en noms logiques.

- Assure la généralité des opérations d'**E/S** pour satisfaire toutes les demandes.
- Permet de gérer les unités par type (disque banal).

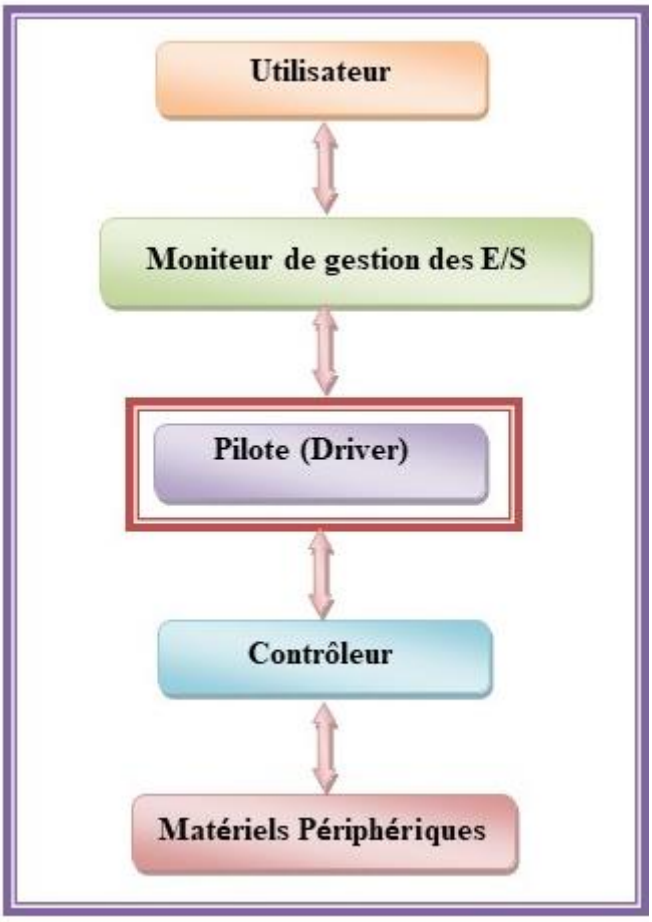


Figure IV. 4: Couche d'organisation logiciel d'E/S.

Note :

Un programme est un ensemble d'instructions et de données. Les instructions d'un programme sont chargées en langage machine (binaire) dans la mémoire afin d'être exécutées dans un ordre bien déterminée par le processeur. Un processus est un programme en cours d'exécution. Les instructions qui constituent un programme peuvent être classifiées en plusieurs catégories, et notre exemple dans ce chapitre, les Instructions **d'entrées/sorties** :

- Une instruction **d'entrée** ou **read** peut correspondre par exemple à la **lecture du disque**.
- Une instruction de **sortie** ou **write** peut correspondre par exemple à **l'affichage sur l'écran**.

IV.3 Communication entre UC et E/S :

La communication entre les modules d'**E/S** et l'**UC** suivent l'un des quatre protocoles suivants :

IV.3.1 Attente active :

L'UC émet une commande au module d'**E/S** pour lancer une opération d'**E/S**. L'**UC** entre dans une boucle pour vérifier si l'opération est achevée ou non et ensuite tester si l'opération est finie avec succès ou non. Les processus sont bloqués jusqu'à ce que l'**E/S** finisse.

IV3.2 La scrutation :

L'**UC** lance les opérations d'**E/S** puis retourne pour exécuter les processus du système. Après des délais périodiques, l'**UC** fait une scrutation des périphériques d'**E/S** pour voir si l'un d'eux a fini sa tâche ou non. Ce protocole est plus efficace que l'attente active, mais il est plus couteux.

IV.3.3 Interruption :

Dans ce cas, le périphérique d'**E/S** se charge d'informer l'**UC** de l'achèvement de l'**E/S** qu'il prépare. L'interruption utilisée par un module est généralement configurable et unique ce qui offre le parallélisme.

IV.3.4 Accès direct à la mémoire :

Les composants d'accès direct à la mémoire **(DMA : Direct Memory Access)** permettent aux modules d'**E/S** de lire ou écrire des données directement depuis le mémoire. Avec cette technique, les données ne doivent pas transiter par l'**UC**. Cet accès est utile pour des dispositifs tels que les disques **[9]**.

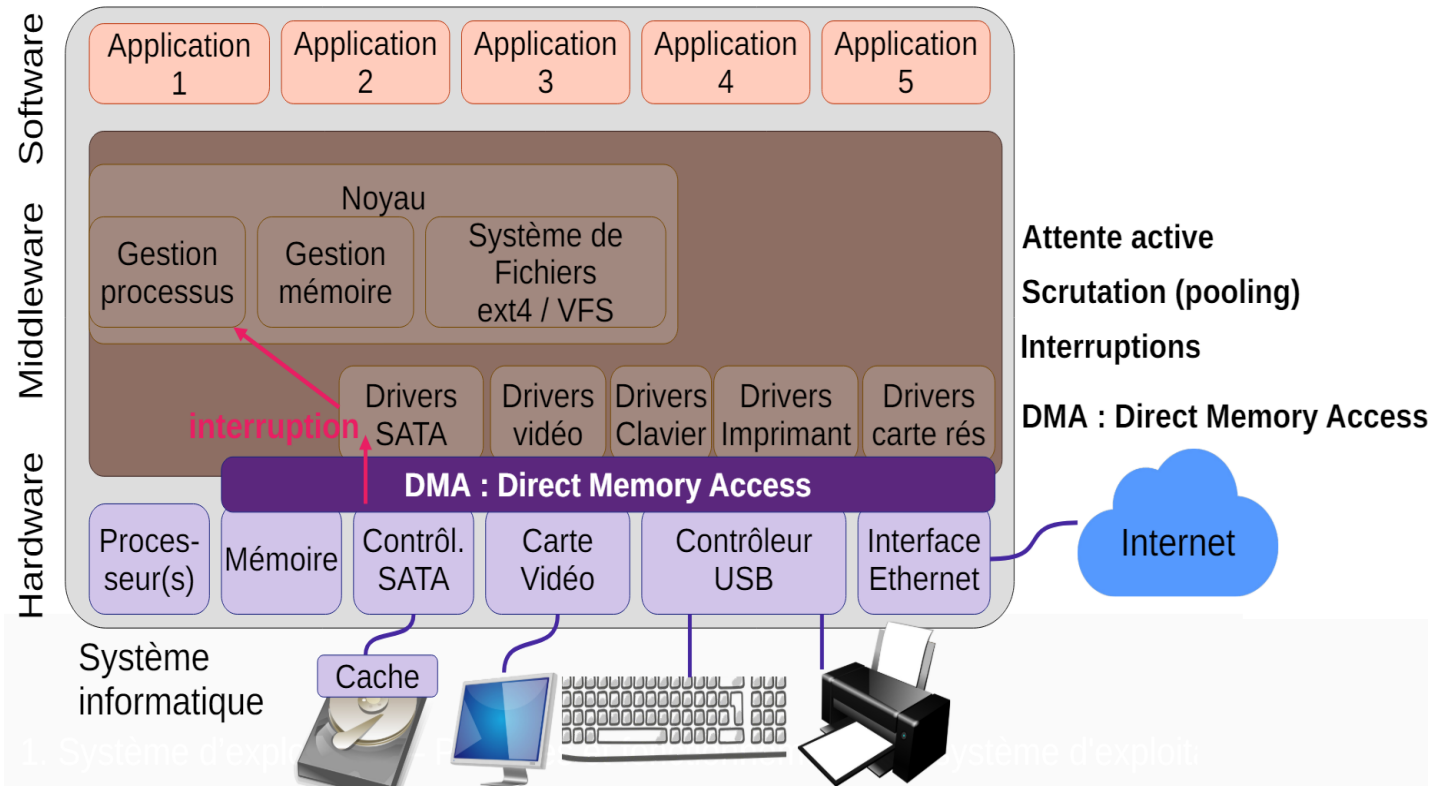


Figure IV. 5: Organisation des dispositifs entrées/sorties.

IV.4 Gestion des Entrées Sorties (E/S) :

La gestion des entrées/sorties (**E/S**) dans un système informatique est une composante essentielle qui permet aux périphériques de communiquer avec le processeur et la mémoire centrale.

Un aspect crucial de tout système informatique, assurant une communication efficace entre les périphériques et le reste du système. Elle implique des mécanismes tels que les interruptions, les buffers, les contrôleurs de périphériques, les pilotes et des techniques d'optimisation pour garantir des **E/S** fiables et performantes.

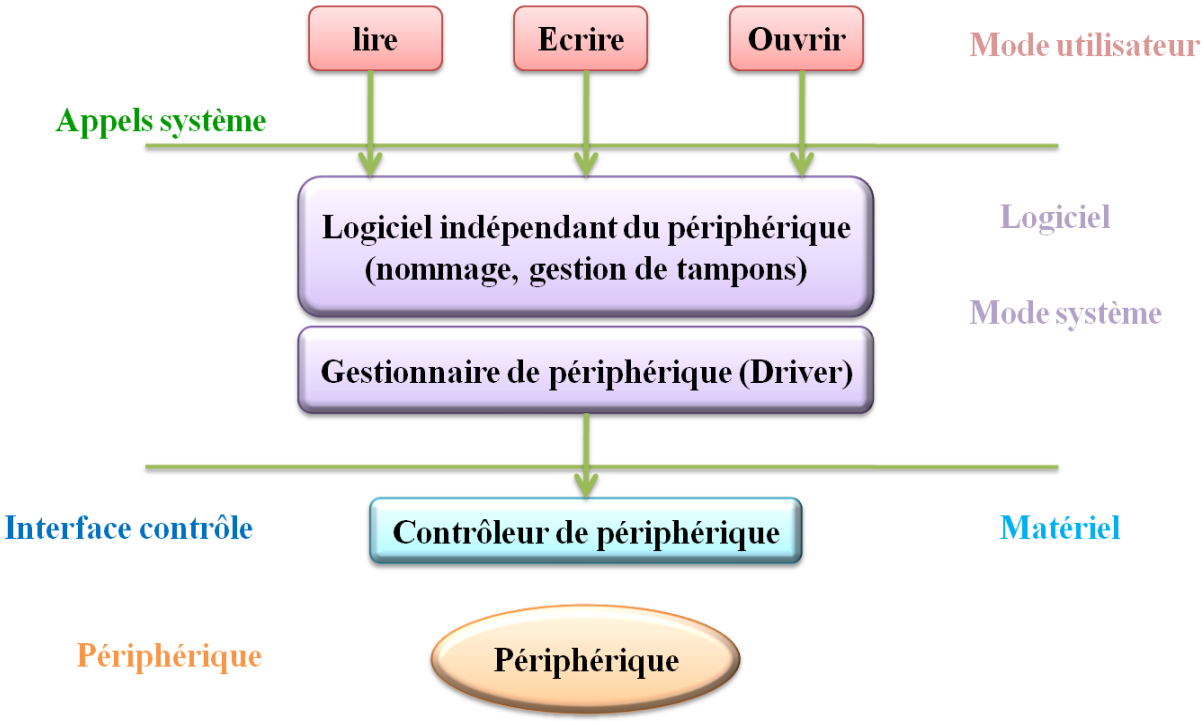


Figure IV. 6: Structure en couches d'un système d'E/S.

Plusieurs modes d'entrées-sorties ont été proposés dans les systèmes informatiques :

- La liaison programmée ;
- Les entrées-sorties pilotées par les interruptions ;
- L'utilisation d'un dispositif permettant des accès directs à la mémoire, **DMA** ;
- Les entrées-sorties avec processeur spécialisé.

IV.5 La liaison programmée :

Dans cette méthode Le microprocesseur exécute un programme pour interagir avec le périphérique, le contrôleur d'**E/S** est connecté via un bus à une paire de registres d'**E/S**, soit un registre pour les adresses et un autre pour les données. Les échanges avec les périphériques sont contrôlés et pilotés par le processeur.

La limite de la liaison programmée est qu'elle est très lente. Ce qui affecte donc les performances du système. De plus, le processeur n'attend que le périphérique soir libre en exécutant d'autres instructions.

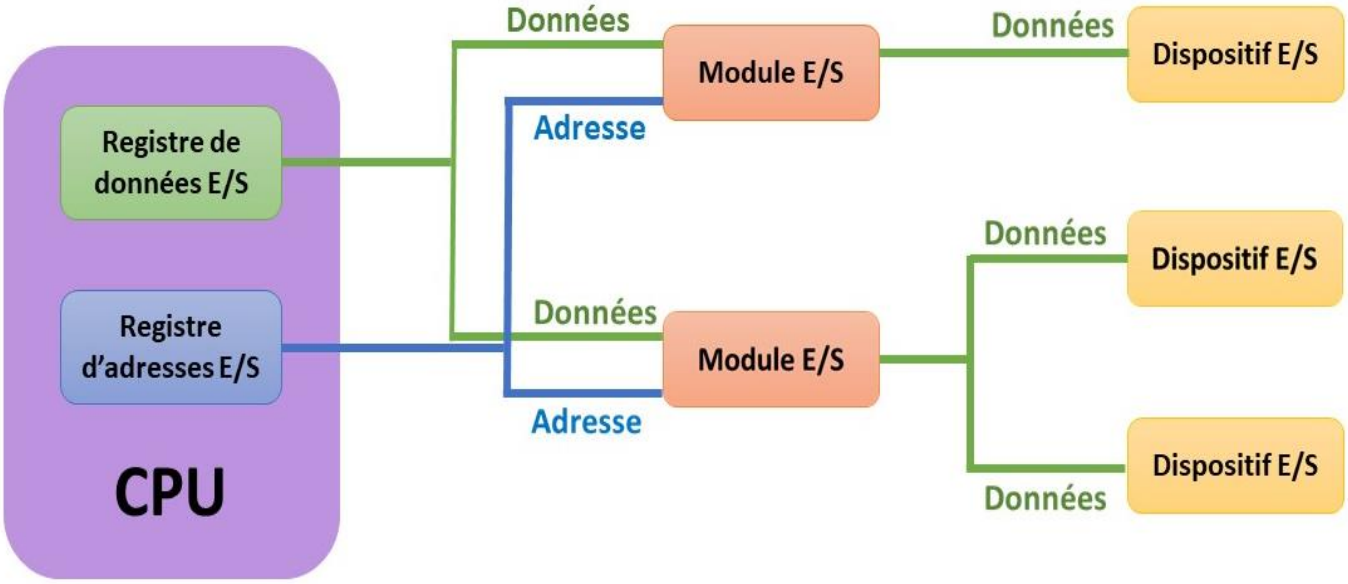


Figure IV. 7: La liaison programmée.

Le processeur central est totalement utilisé pour contrôler et piloter les échanges avec le périphérique (transfert d'un mot à la fois : le **CPU** reste bloqué durant toute la durée de l'échange).

IV.6 Entrées/sorties pilotées par les interruptions :

L'exécution d'un programme s'effectue instruction après instruction, une interruption est un mécanisme qui stoppe l'exécution d'un programme en cours afin d'aller exécuter une tâche jugée plus prioritaire, permettant de gérer efficacement des événements **asynchrones** et de prioriser les tâches en fonction de leur urgence. Elles assurent une réponse rapide aux événements critiques tout en permettant à l'exécution du programme principal de se poursuivre de manière cohérente.

Lorsqu'un programme est en cours d'exécution, plusieurs événements peuvent arriver :

a. Les événements synchrones qui sont liés à l'exécution du programme en cours, comme :

1. Division par zéro ;
2. Exécution d'une instruction inexistante ou interdite ;
3. Tentative d'accès à une zone protégée ;
4. Appel à une fonction du **S.E**.

b. Les événements asynchrones qui ne sont pas liés à l'exécution du programme en cours :

1. Fin d'opération d'**E/S** ;
2. Signal d'horloge.

IV.6.1 Définition (Interruption) :

Une **interruption** est une réponse à un événement qui interrompt l'exécution du programme en cours à un **point observable** (interruptible) du processeur central. Physiquement, l'interruption se traduit par un signal envoyé au processeur. Elle permet déforcer le processeur à **suspendre** l'exécution du programme en cours, et à déclencher l'exécution d'un programme prédéfini, spécifique à l'événement, appelé **routine d'interruption (Rit).**

Les interruptions peuvent d'être d'origines diverses, mais on les classe généralement en trois grands types :

- **Externes** (indépendantes du processus) interventions de l'opérateur, pannes, etc.
- **Déroutements** erreur interne du processeur, débordement, division par zéro, défaut de page (causes qui entraîne la réalisation d'une sauvegarde sur disque de l'image mémoire) , etc.
- **Appels systèmes**, comme les demandes d'entrées-sorties par exemple.

IV.6.2 Mécanismes de gestion des interruptions :

Les mécanismes de gestion des interruptions sont des systèmes utilisés dans les ordinateurs et d'autres appareils électroniques pour gérer les signaux d'interruption envoyés par des périphériques externes ou par d'autres parties du système.

Les interruptions peuvent être déclenchées par une variété de conditions dans un système informatique **[10] :**

1. Le système d'interruption est **actif** ;
2. L'**UC** est à un point observable (interruptible) ;
3. L'interruption est **armée** ;
4. L'interruption est **démasquée** ;
5. Gestion des conflits d'interruption.

1. Système d'interruption actif :

Cela signifie que le matériel et le logiciel nécessaires pour traiter les interruptions sont prêts et fonctionnels. Le système d'interruption est activé lorsque le processeur dispose d'un mécanisme d'activation/désactivation globale des interruptions, et capable de répondre aux signaux d'interruption envoyés par les périphériques ou d'autres parties du système, toute interruption est retardée à la prochaine activation du système d'interruption.

2. L'interruption est armée :

Lorsqu'une interruption est armée, cela signifie qu'elle est prête à être déclenchée. Cela se produit généralement après que le processeur ait été configuré pour surveiller une condition spécifique et qu'une

interruption peut être générée lorsque cette condition se produit. Par exemple, un périphérique peut armer une interruption pour signaler que des données sont prêtes à être lues.

3. L'interruption est démasquée :

Lorsqu'une interruption est démasquée, cela signifie que le processeur est autorisé à répondre à cette interruption si elle se produit. Lorsqu'une interruption est masquée, le processeur ignore les signaux d'interruption correspondants et ne les traite pas. Le démasquage d'une interruption peut se faire lorsque le système est prêt à la traiter ou lorsque la priorité de l'interruption est élevée et doit être gérée immédiatement.

Parfois, il est utile de protéger, contre certaines interruptions, l'exécution de certaines instructions (par exemple, les programmes d'interruption eux-mêmes). Une interruption masquée ne peut alors interrompre l'**UC**, mais toute demande d'interruption faite durant le masquage est retardée (mémorisée) pour être traitée à la levée du masquage.

IV.6.3 Type d'interruptions :

On distingue principalement deux types d'événements :

IV.6.3.1 Les interruptions externes ou matérielles :

Sont émises par les périphériques du processeur (fin d'écriture disques, plus de papier imprimante…). Ce sont les interruptions causées par des organes externes au processeur central, comme les horloges de temps, les périphériques d'**E/S**, ……etc.

Ces interruptions **asynchrones** (c'est-à-dire, peuvent arriver à tout moment indépendamment de l'exécution du programme en cours) sont dues à **[10]** :

- Périphérique prêt ;
- Erreur durant l'**E/S** ;
- Fin d'**E/S** ;
- Ecoulement d'un délai de garde (horloge) ;
- Réinitialisation du système…etc.

IV.6.3.2 Les interruptions internes ou logicielles :

Sont émises par le processeur lui-même lorsqu'il rencontre une erreur dans l'exécution du programme (division par zéro, accès mémoire illégal). Ce sont les **trappes**, Ces interruptions sont **synchrones** et se divisent en deux sous-classes **[10]**, voire (**Figure IV.8)** :

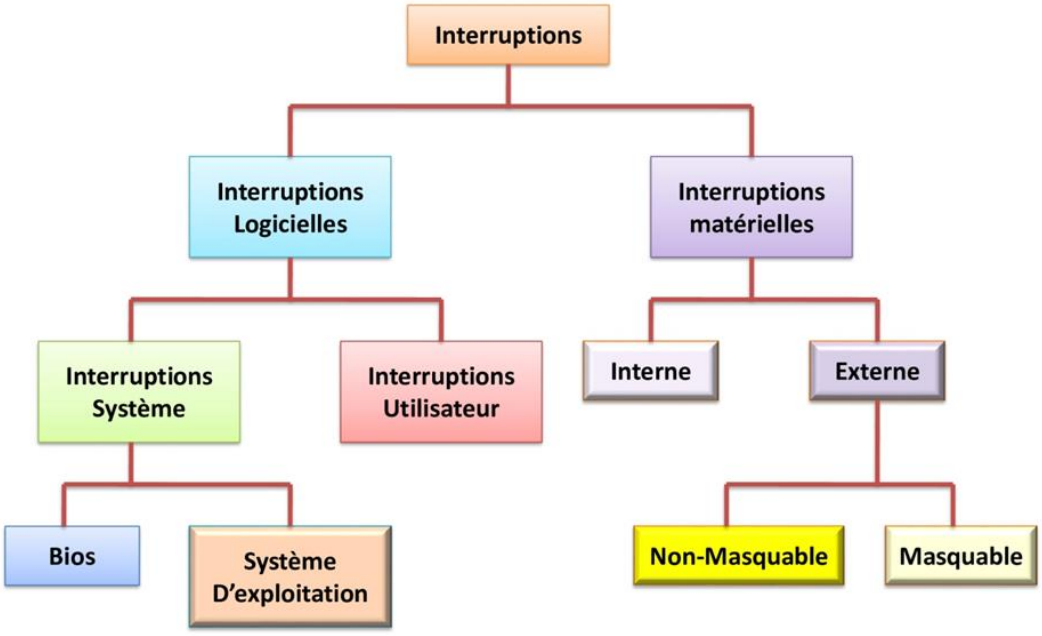


Figure IV. 8: Hiérarchie des interruptions.

a- **Les déroutements (trap** ou **exception) :** Qui sont dus à des erreurs lors de l'exécution d'un programme et en empêchent la poursuite de son exécution. Ces erreurs peuvent avoir diverses causes :

- Tentative d'exécution d'une opération interdite ou invalide ;
- Violation d'accès à une zone mémoire protégée ou inexistante ;
- Division par zéro ;
- Débordement arithmétique.…etc.

Un déroutement ne peut être masqué ou retardé, mais il peut être supprimé (comme pour les déroutements liés aux opérations arithmétiques).

b- **Les appels au superviseur (SuperVisor Call, SVC) :** Qui est une instruction permettant, à partir d'un programme utilisateur d'accéder à un service du **S.E**. (Ex. demande d'**E/S**, allocation dynamique de la mémoire, fin de programme, accès à un fichier, …etc.).

Cette façon de procéder permet au système de :

- Se protéger des usagers ;
- Vérifier les droits d'accès au service demandé.

IV.6.4 Priorité d'interruption :

La gestion des interruptions est une composante essentielle des systèmes informatiques, permettant au processeur de répondre rapidement aux événements externes. Dans un environnement où de multiples sources d'interruptions peuvent survenir simultanément, il est crucial d'établir des priorités d'interruption pour déterminer l'ordre dans lequel ces interruptions doivent être traitées.

IV.6.4.1 Nécessité de priorités d'interruption :

Lorsqu'un système reçoit plusieurs interruptions simultanément, il doit déterminer laquelle traiter en premier. Les priorités d'interruption aident à établir un ordre de traitement, garantissant que les interruptions critiques sont traitées avant les moins importantes **[23]**.

IV.6.4.2 Mise en œuvre des priorités d'interruption :

a) Vecteurs d'interruption :

Chaque type d'interruption est généralement associé à un vecteur d'interruption, qui est une adresse spécifique dans la table des vecteurs d'interruption. Cette adresse pointe vers le gestionnaire d'interruption approprié. Les vecteurs d'interruption peuvent également être associés à des niveaux de priorité.

Tableau IV. 1: Structure vecteur d'interruptions.

N° d'interruption	Adresse de la routine D'interruption
0	Adr0
1	Adr1
2	Adr2
3	Adr3
…..	…..
N	AdrN

Le vecteur d'interruption **(VI)** est une table qui contient les adresses des routines d'interruption. Les interruptions sont numérotées et ces numéros (allant de **0** à **255** dans un **PC**) servent d'index pour rechercher dans le vecteur d'interruption l'adresse de la routine à exécuter **[11]**.

b) Contrôleurs d'interruption programmable (PIC) :

Les **PIC**, comme le **8259A** dans les anciens systèmes, permettent de gérer les priorités des interruptions matérielles. Les **PIC** modernes intégrés dans les processeurs (comme le **APIC** dans les systèmes x86 modernes) offrent des fonctionnalités avancées pour la gestion des interruptions et des priorités **[12]**.

c) Niveaux de priorité :

Les niveaux de priorité peuvent être assignés de manière statique ou dynamique. Les systèmes avec des priorités statiques utilisent une configuration fixe où chaque type d'interruption a un niveau de priorité prédéterminé. Les systèmes avec des priorités dynamiques peuvent ajuster les priorités en fonction des conditions du système **[6]**.

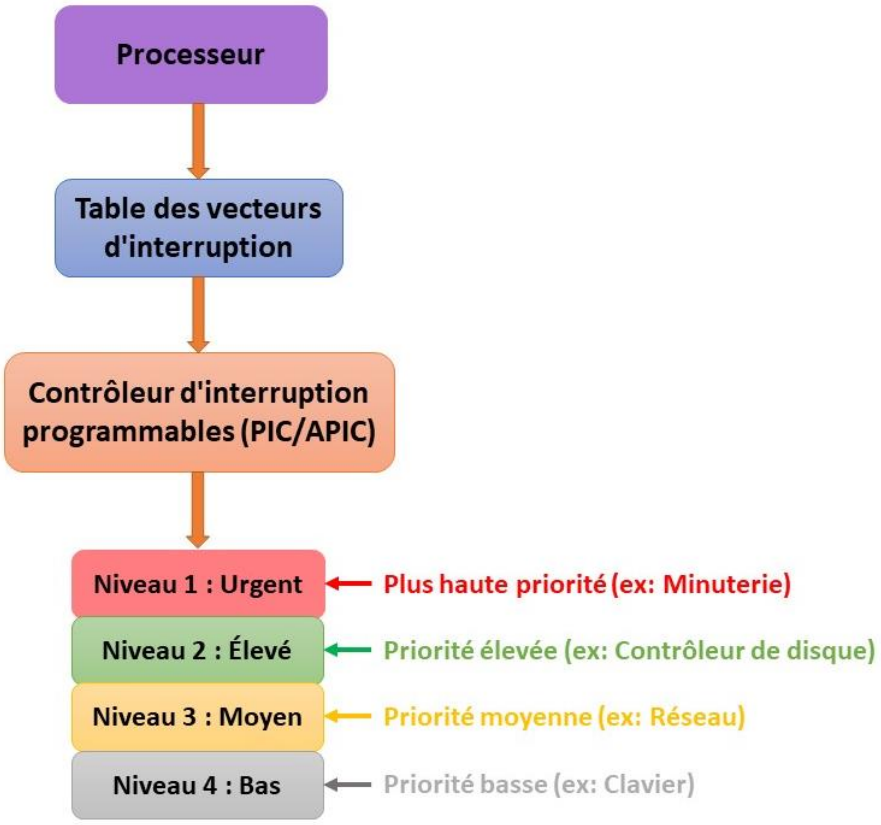


Figure IV. 9: Schéma de priorités d'interruption.

IV.7 L'utilisation d'un dispositif permettant des accès directs à la mémoire, DMA :

Pour que les interruptions soient gérées dans un délai raisonnable, le contrôleur d'E/S est doté d'un mécanisme d'accès direct à la mémoire. Ce mécanisme, implémenté sous forme d'un petit contrôleur dans le contrôleur d'**E/S**, s'appelle **contrôleur DMA** (**Direct Memory Access**).

Le contrôleur **DMA** ajoute un certain niveau d'intelligence au contrôleur d'E/S en prenant le contrôle du bus pour pouvoir envoyer des données directement à la mémoire centrale depuis les périphériques et inversement sans avoir le besoin de passer par le processeur.

Pour pouvoir contrôler les bus d'adresses et de données, le contrôleur **DMA** est doté d'un registre d'adresses qui relié au bus d'adresses, et un registre de données relié à la fois au périphérique et au bus de données. Le contrôleur **DMA** inclut également un compteur qui est utilisé pour la gestion des opérations de lecture de données par le contrôleur **DMA** et l'écriture dans la mémoire principale peut être résumée dans les points suivants :

1. Le processeur envoie l'adresse de la première case mémoire où les données doivent être stockées ;
2. L'adresse est stockée dans le registre d'adresses du contrôleur ;
3. Le nombre d'octets à transférer est chargé dans le compteur du contrôleur ;

4. La commande de lecture des données est envoyée au contrôleur par le processeur via le bus de contrôle ;
5. Le contrôleur récupère donc les données du périphérique et les stocke dans une mémoire interne ;
6. Le contrôleur procède au transfert des données à la mémoire principale en envoyant la première adresse sur le bus d'adresses avec la première donnée (premier octet par exemple) accompagné d'un signal d'écriture ;
7. Après l'écriture de la première donnée en mémoire, le registre d'adresses est incrémenté pour préparer le transfert de la prochaine donnée ;
8. Le compteur à son tour sera décrément après le transfert de la première donnée. Si le nombre d'octets à transférer est 4, alors la valeur du compteur devient 4-1 = 3 ;
9. Si le compteur est nul, le transfert est fini, sinon, la procédure continue avec la prochaine donnée ;
10. Une fois le transfert est complété, le contrôleur envoie un signal d'interruption au processeur pour l'aviser de la fin d'opération d'écriture.

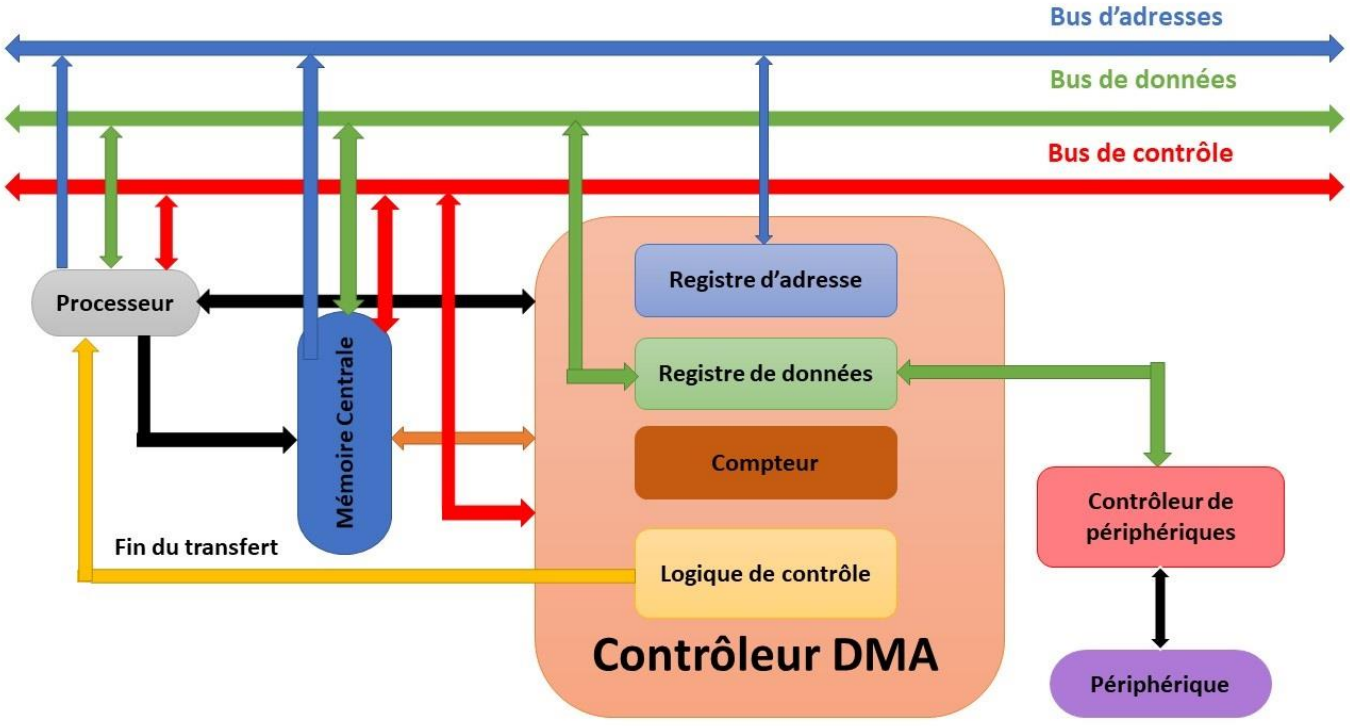


Figure IV. 10: Contrôleur DMA.

IV.8 Les entrées-sorties avec processeur spécialisé :

La gestion des **E/S** avec un processeur spécialisé est une stratégie efficace pour améliorer les performances des systèmes informatiques en déléguant les tâches d'**E/S** à un processeur dédié. Cela permet au **CPU** principal de se concentrer sur les calculs et les processus critiques, tout en assurant un traitement rapide et efficace des opérations d'**E/S**. Cette approche est particulièrement utile dans les

systèmes à forte intensité d'**E/S**, tels que les serveurs de bases de données, les systèmes de stockage de grande capacité et les environnements de calcul intensif.

IV.8.1 Qu'est-ce qu'un processeur d'E/S ?

Un processeur d'**E/S** est un processeur dédié exclusivement à la gestion des opérations d'**E/S**, soulageant ainsi le processeur principal (**CPU**) de ces tâches. Ce processeur spécialisé peut gérer des périphériques tels que des disques durs, des imprimantes, des claviers, des écrans, etc.

IV.8.2 Fonctionnement :

- **Délégation des tâches :** Le **CPU** principal délègue les opérations d'**E/S** au processeur spécialisé.
- **Autonomie :** Le processeur d'**E/S** gère les communications avec les périphériques de manière autonome, sans nécessiter l'intervention constante du **CPU** principal.
- **Signaux d'interruption :** Lorsqu'une opération d'E/S est terminée, le processeur d'**E/S** envoie un signal d'interruption au **CPU** principal pour l'informer que les données sont prêtes ou que l'opération est terminée.

IV.8.3 Architecture :

Voici une vue d'ensemble de l'architecture impliquant un processeur d'**E/S** :

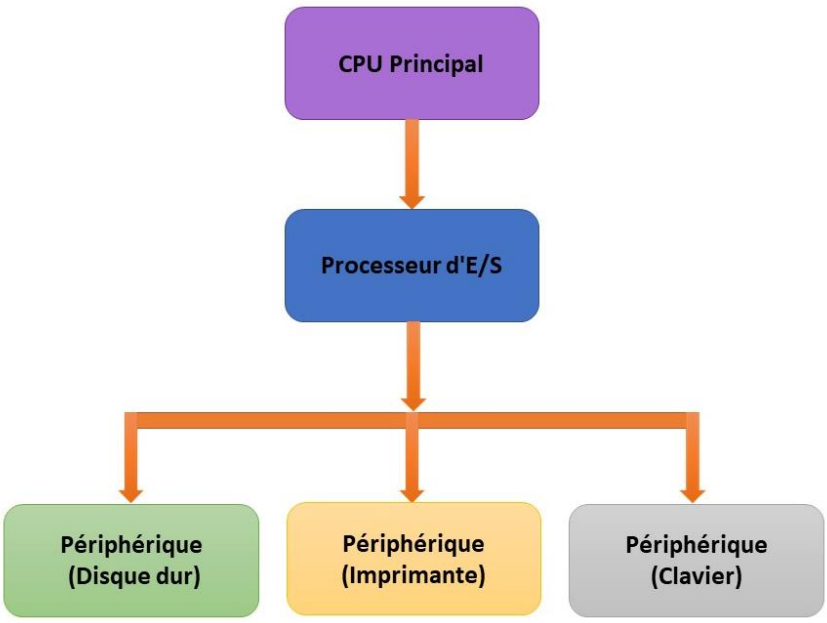


Figure IV. 11: Architecture processeur d'E/S.

IV.8.4 Avantages de l'utilisation d'un processeur d'E/S

IV.8.1.1 Performance améliorée :

- Le **CPU** principal est libéré des tâches d'**E/S**, ce qui lui permet de se concentrer sur les calculs et les processus critiques.

- Réduction de la charge sur le bus système, car le processeur d'**E/S** peut gérer les transferts de données directement avec les périphériques.

IV.8.1.2 Réactivité accrue :

- Les opérations d'**E/S** peuvent être traitées de manière plus rapide et plus efficace par un processeur dédié, améliorant ainsi la réactivité du système global.

IV.8.1.3 Parallélisme :

- Permet un traitement parallèle des opérations d'E/S et des calculs, augmentant l'efficacité globale du système.
- **Exemple d'utilisation : Prenons l'exemple d'un système avec un disque dur.**

1. **Lecture de données :**
   - Le **CPU** principal envoie une commande de lecture au processeur d'**E/S** ;
   - Le processeur d'**E/S** initie la lecture des données depuis le disque dur ;
   - Une fois les données lues, le processeur d'**E/S** les stocke dans une mémoire tampon et envoie un signal d'interruption au **CPU** principal ;
   - Le **CPU** principal récupère les données depuis la mémoire tampon pour les traiter.
2. **Écriture de données :**
   - Le **CPU** principal envoie une commande d'écriture avec les données au processeur d'**E/S** ;
   - Le processeur d'**E/S** transfère les données au disque dur ;
   - Une fois l'opération d'écriture terminée, le processeur d'**E/S** envoie un signal d'interruption au **CPU** principal pour l'informer que l'opération est terminée.

IV.9 Ordonnancements des requetés du disque :

L'ordonnancement du disque dur est une technique utilisée par les systèmes d'exploitation pour gérer les requêtes d'accès aux disques de manière efficace. L'objectif principal est de minimiser le temps de recherche (seek time) et la latence pour améliorer les performances globales du système **[13]**.

IV.9.1 Principaux algorithmes d'ordonnancement des requêtes :

1. **FCFS (First-Come, First-Served):**

Le premier arrivé, premier servi est l'algorithme le plus simple où les requêtes sont traitées dans l'ordre dans lequel elles arrivent.

- **Avantages :**
  - Simple à implémenter ;
  - Équitable car il traite les requêtes dans l'ordre d'arrivée.
- **Inconvénients :**

- Peut entraîner un temps de recherche élevé, surtout si les requêtes sont réparties de manière irrégulière sur le disque.

2. **SSTF (Shortest Seek Time First):**

Le plus court temps de recherche d'abord sélectionne la requête la plus proche de la position actuelle de la tête de lecture/écriture du disque.

- **Avantages :**
  - Réduit le temps de recherche total par rapport à **FCFS**.
- **Inconvénients :**
  - Peut entraîner une famine de certaines requêtes, car les requêtes proches peuvent être favorisées en continu.

3. **SCAN (Elevator Algorithm) :**

La tête de lecture/écriture se déplace dans une direction et traite toutes les requêtes jusqu'à la fin du disque, puis inverse la direction et traite les requêtes dans l'autre sens.

- **Avantages :**
  - Évite la famine ;
  - Fournit un temps de réponse plus prévisible.
- **Inconvénients :**
  - Peut-être inefficace si les requêtes sont principalement concentrées à une extrémité du disque.

4. **C-SCAN (Circular SCAN) :**

Semblable à SCAN, mais lorsque la tête atteint la fin du disque, elle revient au début sans traiter aucune requête. On recommence ensuite dans le même sens.

- **Avantages :**
  - Fournit une meilleure uniformité dans le temps de réponse par rapport à **SCAN**.
- **Inconvénients :**
  - Temps de retour au début du disque peut introduire un retard.

5. **LOOK et C-LOOK :**

Les algorithmes **LOOK** et **C-LOOK** sont des variantes des algorithmes **SCAN** et **C-SCAN**. La tête de lecture/écriture ne va que jusqu'à la requête la plus éloignée dans une direction avant de changer de direction.

- **Avantages :**
  - Réduit le temps de recherche par rapport à **SCAN** et **C-SCAN** en évitant de se déplacer jusqu'à la fin du disque si ce n'est pas nécessaire.

- **Inconvénients :**
  - Complexité de mise en œuvre légèrement plus élevée.

IV.9.2 Comparaison des Algorithmes :

Pour mieux comprendre les différences entre ces algorithmes, voici un exemple pratique :

Supposons que la tête de lecture/écriture se trouve à la position 50 et que les requêtes sont pour les positions suivantes : 10, 22, 20, 38, 45, 60, 70, 90.

1. **FCFS :**
   - Ordre de traitement : 10, 22, 20, 38, 45, 60, 70, 90.
   - Temps de recherche total élevé en raison de l'ordre de traitement non optimisé.
2. **SSTF :**
   - Ordre de traitement : 45, 38, 22, 20, 10, 60, 70, 90.
   - Réduit le temps de recherche total en traitant les requêtes les plus proches en premier.
3. **SCAN :**
   - Ordre de traitement : 45, 38, 22, 20, 10, puis revient et traite 60, 70, 90.
   - Temps de recherche équilibré.
4. **C-SCAN :**
   - Ordre de traitement : 45, 60, 70, 90, puis revient au début et traite 10, 20, 22, 38.
   - Temps de réponse plus uniforme, mais avec un délai de retour.
5. **LOOK :**
   - Ordre de traitement : 45, 38, 22, 20, 10, puis change de direction et traite 60, 70, 90.
   - Réduit le temps de recherche total en évitant de se déplacer jusqu'à la fin du disque.
6. **C-LOOK :**
   - Ordre de traitement : 45, 60, 70, 90, puis revient et traite 10, 20, 22, 38.
   - Combine les avantages de LOOK et C-SCAN pour un temps de réponse plus uniforme.

Chaque algorithme a ses propres avantages et inconvénients, et le choix de l'algorithme dépend des exigences spécifiques du système, telles que l'équité, le temps de réponse, et l'efficacité globale. Dans des systèmes modernes, des algorithmes plus sophistiqués et souvent hybrides peuvent être utilisés pour équilibrer ces différents aspects et offrir des performances optimales.

IV.10 Conclusion :

En conclusion, la gestion efficace des périphériques demeure un élément critique dans la conception et le fonctionnement des systèmes d'exploitation, jouant un rôle clé dans l'expérience utilisateur et les performances globales du système. Son évolution continue reflète la nature dynamique de l'informatique moderne et son adaptation constante aux nouvelles technologies et aux besoins des utilisateurs.

CHAPITRE-V- GESTION DES PROCESSUS

V.1 Introduction :

Le terme processus a été introduit dans les années 60 pour généraliser "'job concept"'. Le processus est une instance de programme binaire. Différents processus peuvent exécuter différentes instances du même programme **[11]**.

Les processus peuvent être créés par d'autres processus à l'aide de primitives système comme "**fork ()**" dans Unix/Linux. La gestion des processus inclut leur création, l'ordonnancement (**Scheduling**), la synchronisation, et la terminaison. Le système d'exploitation assure que les processus reçoivent une part équitable des ressources et peuvent communiquer entre eux de manière sécurisée.

V.2 Objectif :

- **Compréhension des processus** :
- Comprendre ce qu'est un processus dans le contexte d'un système d'exploitation ;
- Savoir comment un processus est créé, exécuté, suspendu et terminé.
- **Gestion des ressources** :
- Apprendre comment les ressources système telles que le **CPU**, la mémoire et les périphériques sont allouées aux processus ;
- Comprendre les mécanismes d'allocation des ressources et les problèmes liés à la concurrence et à la synchronisation.
- **Compréhension du modèle Fork-Join** :
- Comprendre le concept du modèle Fork-Join en programmation parallèle ;
- Savoir comment diviser un problème en sous-tâches indépendantes (fork) et les combiner ensuite (join).
- **Gestion des threads et Parallélisme** :
- Apprendre sur les threads et la différence entre les processus et les threads ;
- Comprendre comment les threads sont créés, planifiés et synchronisés ;
- Apprendre à exploiter le parallélisme pour améliorer les performances des applications ;
- Comprendre les avantages du modèle Fork-Join pour résoudre des problèmes pouvant être parallélisés.

V.3 Concepts Clés des Processus :

V.3.1 Processus :

Un processus est une instance en cours d'exécution d'un programme. Lorsqu'un programme est exécuté, il devient un processus. Un processus est une entité active avec un état d'exécution propre, et il possède les ressources nécessaires pour exécuter les instructions du programme. Voici les principales caractéristiques d'un processus :

- **Actif** : Un processus est en cours d'exécution et utilise le **CPU** et la mémoire.
- **Ressources** : Il possède ses propres ressources comme l'espace d'adressage mémoire, descripteurs de fichiers, et registres **CPU**.
- **État** : Un processus a différents états comme prêt, en exécution, bloqué, ou terminé.
- **Exemple** : Lorsque vous ouvrez une application comme un navigateur web, le programme du navigateur devient un ou plusieurs processus.

V.3.2 Programme :

Un programme est un ensemble de directives écrites dans un langage de programmation que l'ordinateur peut comprendre et exécuter. Il est une entité passive, stockée sur le disque dur et composée de code binaire ou de code source. Voici les principales caractéristiques d'un programme :

- **Passif** : Un programme est un fichier inactif qui contient des instructions, sans aucune activité.
- **Stockage** : Il est stocké sur un support de stockage tel que le disque dur ou un **SSD**.
- **Code Source/Binaire** : Il peut être du code source qui doit être compilé ou du code binaire déjà compilé prêt à être exécuté.
- **Exemple** : Un fichier exécutable (.exe) sur Windows ou un fichier binaire sur Linux.

Tableau V. 1: Différences Clés entre le processus et le programme.

Aspect	Programme	Processus
Nature	Entité passive (code inactif)	Entité active (en cours d'exécution)
Stockage	Stocké sur le disque	Réside en mémoire (RAM)
Exécution	Ne s'exécute pas par lui-même	Représente l'exécution d'un programme
Ressources	Aucun besoin de ressources d'exécution	Utilise le CPU, la mémoire, des fichiers.
Multitâche	Un programme peut créer plusieurs processus	Chaque processus est une instance distincte du programme
Exemple	Fichier exécutable .exe	Application en cours d'exécution

Un programme est un ensemble de directives passives écrites dans un langage de programmation, tandis qu'un processus est l'instance active d'un programme en cours d'exécution. La distinction est essentielle pour comprendre la gestion des ressources par le système d'exploitation, le multitâche et la sécurité des systèmes informatiques. En résumé, sans processus, les programmes ne peuvent pas être exécutés et resteront inactifs sur le disque dur.

V.3.3 Processus et démarrage :

Le kernel est chargé depuis le disque au démarrage du système. Le premier processus, nommée **INIT**, est lancé et reste actif jusqu'à l'arrêt du système. Tous les autres processus sont créés à partir d'init ou de ses descendants.

V.3.4 Espace d'adressage des processus :

Les processus partagent le **CPU**, la mémoire et les périphériques. Dans un système à temps partagé, les processus semblent s'exécuter simultanément. Chaque processus dispose de son propre espace d'adressage, incluant ses instructions et ses données, ainsi que les registres nécessaires à son fonctionnement.

V.3.5 Interaction entre processus et kernel :

Les interactions entre le kernel et les processus utilisateurs se font via l'**API** système et les appels système. Le kernel exécute les instructions pour les processus et intervient en cas d'actions interdites ou d'exceptions matérielles.

V.3.6 Espace d'adressage et kernel :

Une partie de l'espace d'adressage de chaque processus est dédiée au kernel, formant l'espace système. Il n'y a qu'une seule instance du kernel, partagée par tous les processus. Le kernel utilise deux objets pour gérer chaque processus :

- L’espace utilisateur (contenant des informations sur le processus) ;
- La pile kernel (gérant les fonctions lors des appels système).

V.4 Structure d'un processus :

Le système d'exploitation alloue à chaque processus une zone dans la mémoire centrale appelée espace d'adressage du processus pour s’exécuté et dans lequel il peut lire et écrire. L'espace d'adressage d'un processus comporte une zone de données qui contient les variables globales ou statiques du programme, une zone de code qui correspond aux instructions, en langage d'assemblage, du programme à exécuter et les appels de fonctions, avec leurs paramètres et leurs variables locales, empilées dans une pile d'exécution **[14]**.

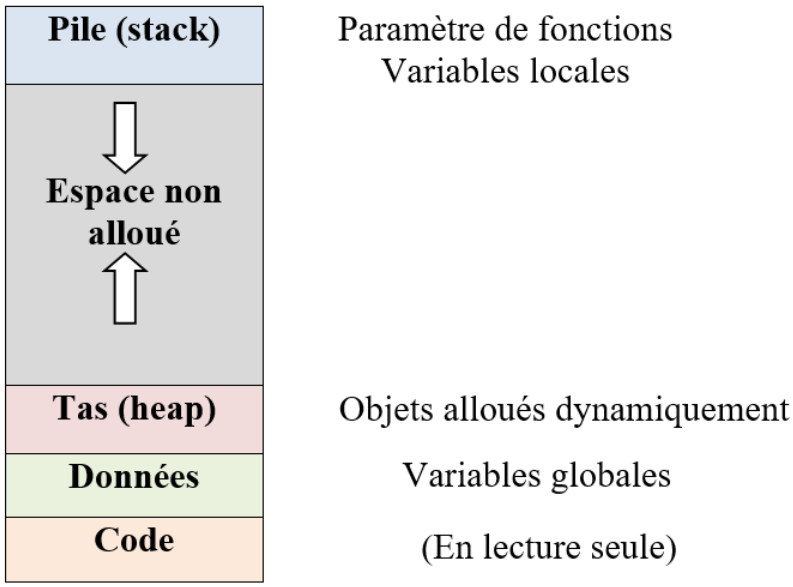


Figure V. 1: Schéma Illustratif d'un processus.

L'exécution d'un processus est une alternance entre des calculs effectués par le processeur et des requêtes d'entrée sortie effectuées par les périphériques. Un logiciel est organisé en un certain nombre de processus.

V.4.1 Structure de l'Espace Mémoire d'un Processus :

L'espace mémoire d'un processus est structuré de manière à organiser les différentes parties du programme et les données nécessaires à son exécution.

V.4.1.1 Segment de Code (Text Segment) :

Représente le programme à exécuter. Il est toujours placé dans des zones fixes de la mémoire, c'est-à-dire au début de la zone disponible (puisqu'il n'est pas amené à augmenter de taille durant l'exécution du processus).

- Contient le code exécutable du programme ;
- Chargé en mémoire lors de l'exécution du programme ;
- Souvent en lecture seule pour empêcher les modifications accidentelles ou malveillantes ;
- Inclut les instructions du programme compilé ;
- Souvent optimisé pour être en lecture seule et partageable entre processus exécutant le même programme.

V.4.1.2 Segment de Données (Data Segment) :

Au-dessus du segment de code se trouve le segment de données. Ce segment est traditionnellement composé d'un segment de données initialisées (les variables globales et statiques) et d'un segment de

données non initialisées qui est créé dynamiquement. Ce segment est amené à croître décroitre suite à l'allocation dynamique de variables.

- **Données Initialisées** : Contient les variables globales et statiques qui sont initialisées avant l'exécution, ont des valeurs initiales définies.
- **Données Non Initialisées BSS (Block Started by Symbol)** : Contient les variables globales et statiques qui ne sont pas initialisées explicitement. Par défaut, elles sont initialisées à zéro par le système.

V.4.1.3 Segment de Tas (Heap Segment) :

- Utilisé pour l'allocation dynamique de mémoire pendant l'exécution du programme.
- Les allocations et libérations de mémoire sont gérées via des appels comme **malloc**, **calloc**, **realloc** et free en **C.**
- La taille du tas peut augmenter ou diminuer au cours de l'exécution du programme, en fonction des besoins d'allocation de mémoire.
- Croît vers le haut, contrairement à la pile qui croît vers le bas.

V.4.1.4 Segment de Pile (Stack Segment) :

Pour stocker les données obtenues en cours d'exécution, le système utilise un segment pile. Ce segment est généralement situé en haut de l'espace d'adressage et il croît vers les adresses basses (pile renversée).

- Utilisé pour stocker les appels de fonctions, chaque appel de fonction crée une nouvelle "frame" sur la pile, contenant les variables locales, et les adresses de retour ;
- Croît vers le bas (des adresses hautes vers les adresses basses) ;
- Géré automatiquement par le compilateur et le système d'exécution.

V.4.1.5 Zone des Bibliothèques Partagées (Shared Libraries) :

- Contient les bibliothèques dynamiques partagées utilisées par le programme (.dll sous Windows, .so sous Linux) sont chargées ici ;
- Permet la réutilisation de code commun entre plusieurs processus, économisant ainsi de la mémoire.

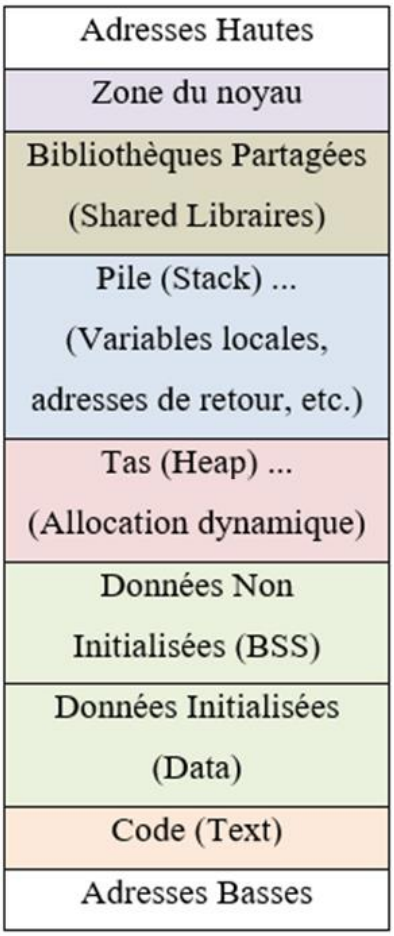


Figure V. 2: Schéma illustratif de l'espace mémoire d'un processus.

V.4.2 Structure de données pour la gestion des processus :

La gestion des processus par un système d'exploitation repose sur des structures de données spécifiques pour suivre et contrôler chaque processus. Ces structures contiennent toutes les informations nécessaires pour gérer l'exécution, l'ordonnancement et la synchronisation des processus.

Pour suivre son évolution, le **SE** maintient pour chaque processus une structure de données particulière appelée bloc de contrôle de processus **(PCB : Process Control Bloc)** et dont le rôle est de reconstituer tout le contexte du processus.

Les systèmes d'exploitation manipulent deux structures de données principales pour gérer les processus créés sur une machine : la table des processus et le bloc de contexte d'un processus

V.4.2.1 La table des processus :

La table des processus contient toutes les informations indispensables au système d'exploitation pour assurer une gestion cohérente des processus (informations sur le processus, son exécution, les fichiers qu'il manipule et son occupation mémoire). Elle est stockée dans l'espace mémoire du système d'exploitation, ce qui signifie que les processus ne peuvent pas y accéder.

V.4.2.2 Bloc de Contrôle de Processus (PCB) :

Le **PCB** est une structure de données clé utilisée par le système d'exploitation pour stocker toutes les informations nécessaires à la gestion d'un processus. Voici les principaux éléments qui composent un **PCB** :

1. **Identifiant du Processus (PID) :** Un identifiant unique attribué à chaque processus.
2. **État du Processus :** Indique l'état actuel du processus (prêt, en cours d'exécution, bloqué, terminé, etc.).
3. **Compteur de Programme (PC) :** Contient l'adresse de la prochaine instruction à exécuter pour ce processus.
4. **Registres du CPU :** Sauvegarde les registres du **CPU** utilisés par le processus, nécessaires pour restaurer l'état du processus lors des commutations de contexte.
5. **Informations de Gestion de la Mémoire :** Inclut des pointeurs vers les tables de pages, les tables de segments, ou d'autres structures de gestion de la mémoire utilisées par le processus.
6. **Informations de Comptabilité :** Temps **CPU** utilisé, temps d'exécution total, etc.
7. **Informations sur les E/S :** Liste des descripteurs de fichiers ouverts, périphériques utilisés, etc.
8. **Informations de Synchronisation :** Inclut les informations nécessaires pour la gestion de la synchronisation inter-processus, comme les sémaphores et les mutex.

Figure V. 3: Schéma Illustratif d'un PCB.

V.5 Les états du processus :

Les processus sont des programmes en cours d'exécution dans un système d'exploitation, et ils passent par différents états tout au long de leur cycle de vie. Comprendre ces états est crucial pour la gestion efficace des ressources du système et la planification des tâches. Voici une vue d'ensemble des principaux états d'un processus et de leurs transitions **[16]**.

- **Nouveau (New)** :

  Le processus est en cours de création. Les ressources nécessaires sont allouées et les structures de données internes sont initialisées.

- **Prêt (Ready)** :

  Le processus est prêt à être exécuté par le processeur. Il attend dans la file d'attente des processus prêts pour être assigné au processeur.

- **En cours d'exécution (Running)** :

  Le processus est actuellement exécuté par le processeur. Il utilise les ressources du **CPU** pour accomplir ses tâches.

- **Bloqué / En attente (Blocked / Waiting)** :

  Le processus ne peut pas continuer son exécution jusqu'à ce qu'un événement spécifique se produise (par exemple, une opération d'entrée/sortie se termine ou une ressource devienne disponible).

- **Terminé (Terminated)** :

  Le processus a fini son exécution et a été retiré de la mémoire. Toutes les ressources allouées sont libérées.

- **Suspendu (Suspended)** :

  Le processus est temporairement retiré de l'exécution et stocké sur le disque dur. Cela peut se produire pour libérer de la mémoire pour d'autres processus. Il peut être dans un état suspendu-prêt ou suspendu-bloqué selon qu'il attend d'être exécuté ou un événement particulier.

Ces états et transitions permettent au système d'exploitation de gérer efficacement les processus, d'optimiser l'utilisation du CPU et de maintenir la stabilité et la réactivité du système.

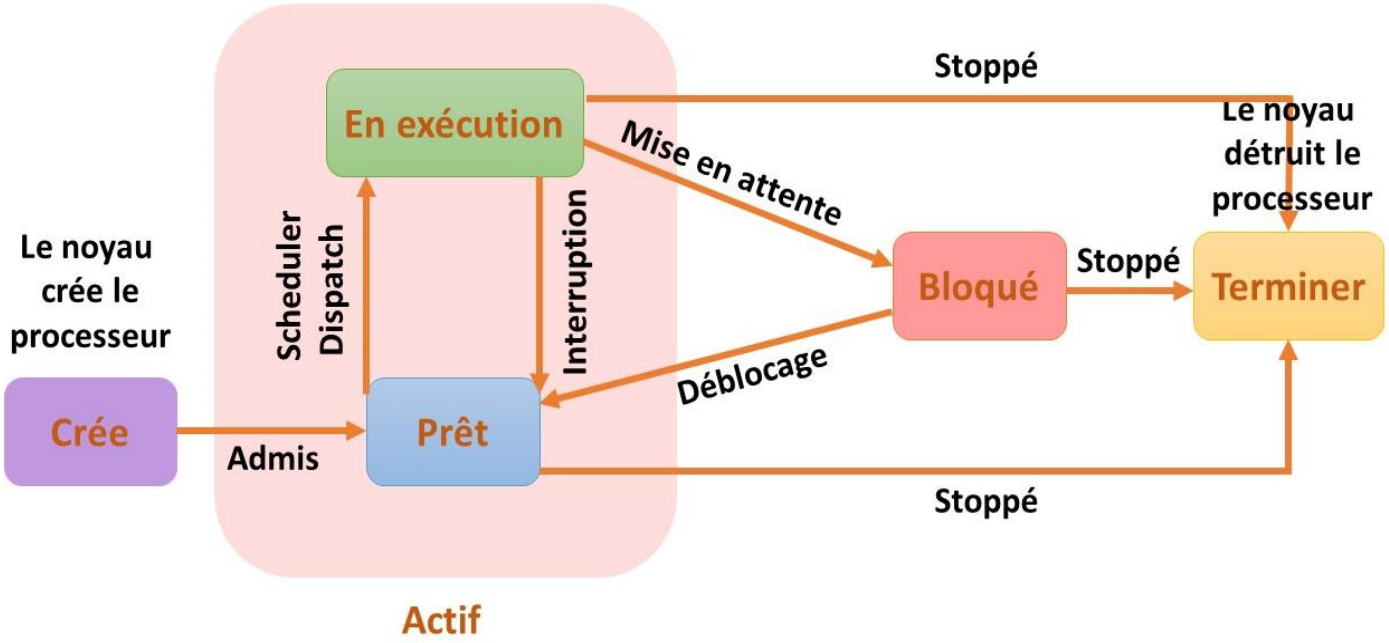


Figure V. 4: Modes d'exécution des processus (état du processus).

- Les transitions entre états provoquent le passage d'un état à un autre :

Tableau V. 2: Transition entre états.

Activation	**Prêt ⟶ Actif.**
Désactivation (ou préemption, ou réquisition)	**Actif ⟶ Prêt.**
Mise en attente (ou blocage)	**Actif ⟶ En attente.**
Réveil	**Attente ⟶ Prêt**

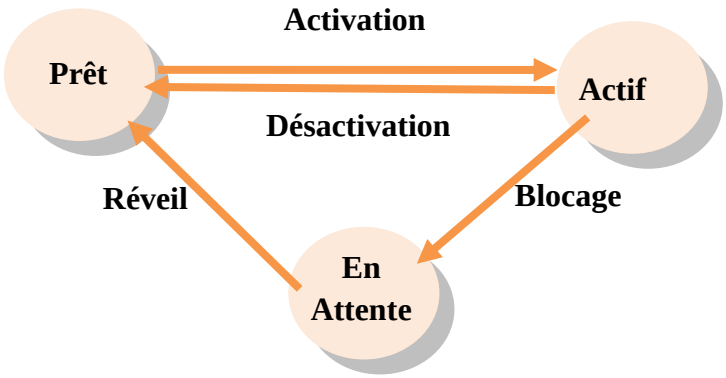


Figure V. 5: Etats du processus.

Les opération qu'un utilisateur de S.E. peut effectuer sur les processus sont :

- Création (ex : **fork**, création d'un processus sous UNIX) ;
- Destruction (ex : **kill**, destruction d'un processus sous UNIX) ;
- Activation, désactivation, blocage (ex : **wait** sous UNIX).

Le système d'exploitation gère les transitions entre les différents états des processus en maintenant deux listes : une liste des processus prêts à être exécutés et une liste des processus en attente. De plus, il doit définir une politique pour choisir quel processus prêt à être exécuté doit être activé en priorité. Cette politique peut être basée sur des critères tels que la durée d'éligibilité du processus, sa priorité ou la durée estimée de son utilisation du processeur.

Les processus sont organisés dans la file des processus prêts à être exécutés par l'ordonnanceur (Scheduler), puis c'est au dispatcher de les activer lorsque le processeur est disponible.

V.6 Commutation de contexte :

Lors des opérations de commutation de contexte ou **"context switch"** en anglais, le système d'exploitation doit sauvegarder/restaurer le contexte mémoire (partager dans l'espace) et le contexte processeur (partager dans le temps). Cela signifie que lorsque le processeur passe d'un processus à un autre pour l'exécuter, le **SE** préserver le contexte du processus en cours (mémoire et registres du processeur). Ensuite, lorsque le processeur revient à ce processus, il doit restaurer son contexte pour reprendre l'exécution là où elle s'était arrêtée.

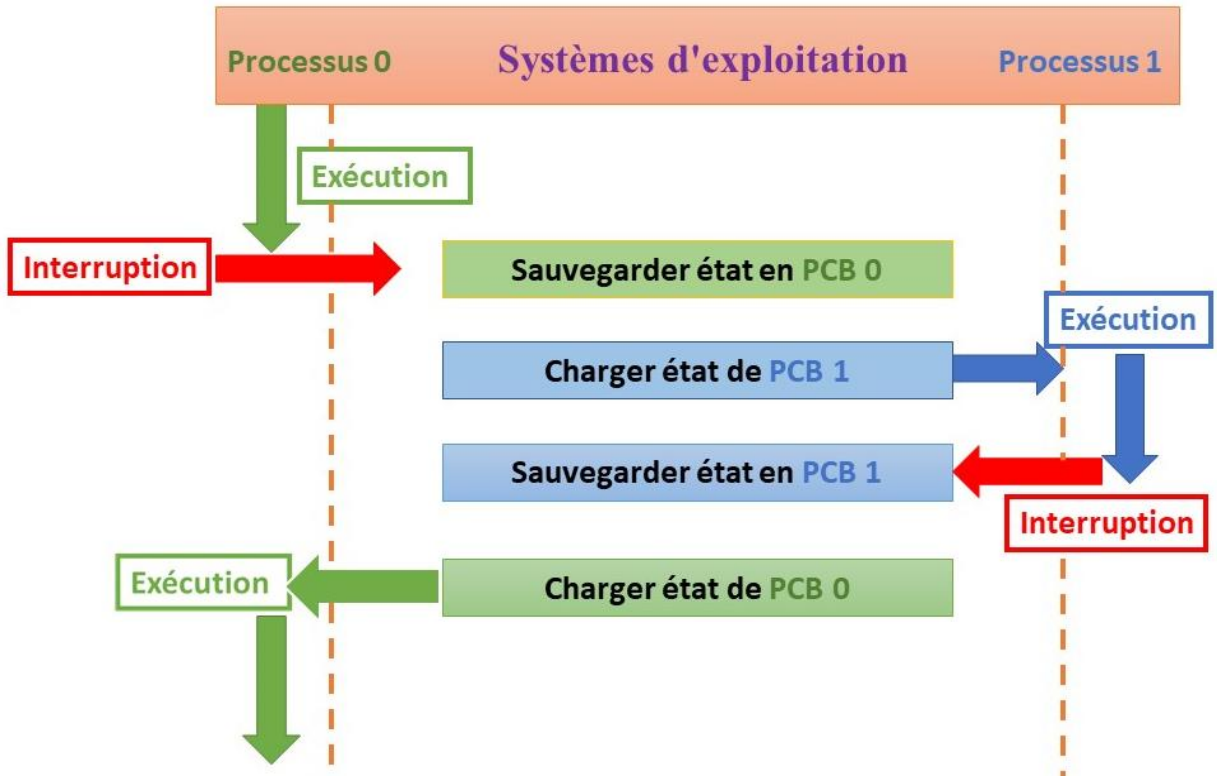


Figure V. 6: Changement ou commutation de contexte.

La commutation de contexte est le processus par lequel un système d'exploitation (**OS**) passe d'un processus ou d'un thread en cours d'exécution à un autre. Ce mécanisme permet à un système multitâche

de partager le temps du processeur entre plusieurs processus, assurant ainsi que tous les processus reçoivent une part équitable du temps **CPU** et que le système reste réactif.

V.6.1 Comment cela fonctionne-t-il ?

La commutation de contexte implique trois étapes clés :

1. **Sauvegarde de l'état du processus courant** :
   L'**OS** sauvegarde l'état du processus ou du thread actuellement en cours d'exécution. Cela inclut le contenu des registres du processeur, le compteur ordinal, le pointeur de pile, et d'autres informations pertinentes.
2. **Changement de contexte** :
   L'**OS** charge l'état du nouveau processus ou thread qui doit être exécuté. Cela inclut le chargement des registres du processeur avec les valeurs correspondant à ce processus, ainsi que la mise à jour du pointeur de pile et d'autres éléments.
3. **Reprise de l'exécution** :
   Le processeur reprend l'exécution à l'endroit où le nouveau processus avait été suspendu, en utilisant les informations rechargées.

V.6.2 Types de commutation de contexte :

1. **Commutation de processus (Process Switching)** :
   C'est le changement de contexte entre différents processus. Chaque processus a son propre espace d'adressage et ses propres ressources.
2. **Commutation de thread (Thread Switching)**:
   C'est le changement de contexte entre différents threads au sein du même processus. Les threads partagent le même espace d'adressage, mais ont leurs propres registres et piles.

V.7 Gestion de processus :

Le système d'exploitation manipule deux types de processus : ceux du système et ceux des utilisateurs. Les fonctionnalités du système d'exploitation en matière de gestion de processus sont :

- La création, suppression et interruption de processus ;
- L'ordonnancement des processus afin de décider d'un ordre d'exécution équitable entre les utilisateurs tout en privilégiant les processus du système ;
- La synchronisation entre les processus ainsi que la communication ;
- La gestion des conflits d'accès aux ressources partagées ;
- La protection des processus d'un utilisateur contre les actions d'un autre utilisateur.

V.7.1 Principes du Système Multitâche :

Un système multitâche est un type de système d'exploitation qui permet l'exécution de plusieurs tâches (processus) simultanément.

- **Simultanéité :** Le multitâche permet à plusieurs processus de paraître s'exécuter en même temps. En réalité, sur un système à un seul processeur, les processus s'exécutent tour à tour pendant de très courtes périodes.
- **Partage des Ressources :** Les processus partagent les ressources matérielles du système, comme le processeur, la mémoire, les périphériques d'entrée/sortie, etc.
- **Isolation et Protection** : Chaque processus est isolé des autres, ce qui empêche un processus de corrompre les données ou de perturber l'exécution d'un autre processus (sécurité).

V.7.1.1 Fonctionnement du Système Multitâche :

1. **Planification des Processus :**

- Le système d'exploitation utilise des algorithmes de planification pour déterminer l'ordre dans lequel les processus doivent être exécutés.
- Les algorithmes courants incluent le round-robin, la priorité, le first-come, first-served (**FCFS**), et d'autres.

2. **Commutation de Contexte :**

- Lorsque le système d'exploitation décide de passer d'un processus à un autre, il effectue une commutation de contexte pour sauvegarder l'état du processus actuel et charger l'état du prochain processus à exécuter.

3. **Gestion de la Mémoire :**

- Le système multitâche doit gérer la mémoire de manière à ce que chaque processus dispose de l'espace mémoire dont il a besoin tout en évitant les conflits entre processus.
- Utilisation de techniques telles que la pagination et la segmentation.

4. **Gestion des Périphériques :**

- Les processus doivent accéder aux périphériques d'entrée/sortie (disque dur, clavier, écran, etc.). Le système d'exploitation gère ces accès pour éviter les conflits et assurer une utilisation équitable des périphériques.

V.7.1.2 Types de Système Multitâche :

1. **Multitâche Coopératif** :

- Les processus doivent coopérer et libérer volontairement le processeur pour que d'autres processus puissent s'exécuter.

- Moins utilisé de nos jours en raison de sa faiblesse à garantir la réactivité et la stabilité du système.

2. **Multitâche Préemptif** :

- Le système d'exploitation décide du moment où interrompre un processus pour en exécuter un autre.
- Garantit une meilleure réactivité et une utilisation plus équitable du processeur.
- Utilisé par la majorité des systèmes d'exploitation modernes comme Windows, Linux, MacOs.

V.7.2 Processus et Hiérarchie :

La hiérarchie des processus fait référence à la manière dont les processus sont organisés et gérés par le système pour assurer le fonctionnement efficace et la gestion des ressources, de garantir la sécurité et la stabilité du système, et de fournir une interface utilisateur fluide et réactive.

Cette hiérarchie permet une gestion ordonnée des processus, facilitant leur création, terminaison et communication.

Le système d'exploitation fournit un ensemble d'appels système qui permettent la création, la destruction, la communication et la synchronisation des processus **[17]**.

- Les processus sont créés et détruits dynamiquement ;
- Un processus peut créer un ou plusieurs processus fils qui, à leur tour, peuvent créer des processus fils sous une forme de structure arborescente ;
- Dans ce cas, on appelle le processus créateur le **père**, et le processus créer le **fils** ;
- Le fils qui crée d'autre processus. Il devient père de ses processus ;
- Ainsi, un arbre de processus est constitué. Tous les processus répondant à ce schéma ;
- Chaque nœud de l'arbre est un processus avec un nom et un identifiant **PID** (Process Identifier).

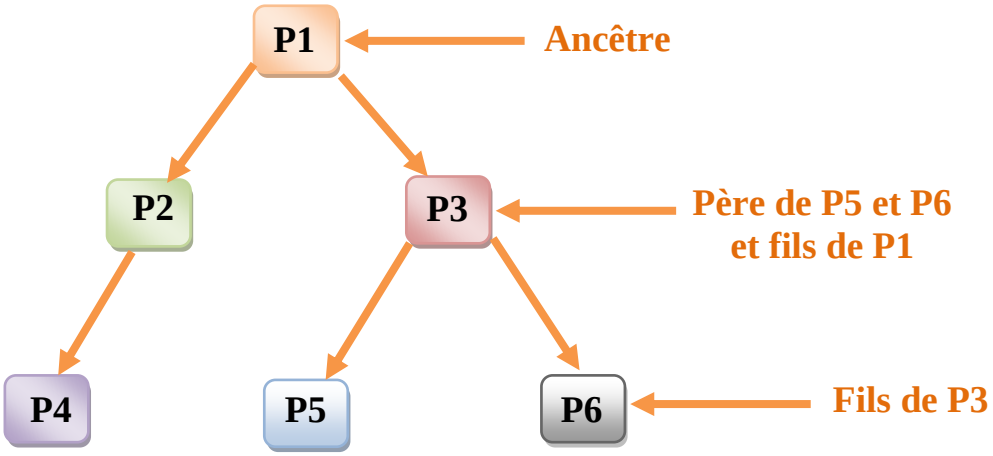


Figure V. 7: Hiérarchie des processus.

V.7.2.1 Option de partage de ressources :

- Parent et enfant partagent toutes les ressources ;
- Les enfants partagent un sous ensemble des ressources parentales ;
- Parent et enfant ne partagent aucune ressource.

V.7.2.2 Options d'exécution :

- Dans les systèmes d'exploitation **Monotâche,** l'exécution du processus père est suspendue jusqu'à terminaison du processus fils, on parle d'une **exécution séquentielle.**
- Dans les systèmes d'exploitation **Multitâche,** le père continue à s'exécuter en concurrence avec ses fils, on parle d'une **exécution asynchrone.**

V.7.3 Le concept de Fork/Join :

Le concept de **Fork/Join** est une technique de parallélisation qui permet de décomposer un problème complexe en sous-problèmes plus simples, de les résoudre en parallèle, puis de combiner les résultats des sous-problèmes pour obtenir la solution du problème initial. Ce modèle est particulièrement utile pour tirer parti des architectures multi-cœurs des processeurs modernes.

Composants Clés du Modèle Fork/Join

1. **Fork :**
    - Signifie "diviser" une tâche en plusieurs sous-tâches plus petites. Dans le contexte du framework Fork/Join, cela consiste à créer de nouvelles tâches (ou sous-tâches) qui peuvent être exécutées en parallèle.
    - Chaque sous-tâche est soumise à un pool de threads qui les exécute.
2. **Join :**
    - Signifie "combiner" les résultats des sous-tâches une fois qu'elles sont complétées. Chaque tâche attend la fin de ses sous-tâches, récupère leurs résultats et les combine pour produire le résultat final.

V.7.4 Processus Général du Modèle Fork/Join :

- **Diviser le problème :** Un problème initial est divisé en sous-problèmes plus petits jusqu'à ce que les sous-problèmes soient suffisamment simples pour être résolus directement.
- **Résoudre les sous-problèmes :** Les sous-problèmes sont résolus de manière récursive, en utilisant le même processus de division s'ils sont encore trop complexes.
- **Combiner les résultats :** Les résultats des sous-problèmes résolus sont combinés pour obtenir le résultat du problème initial.

V.7.5 Création de processus (primitives fork/join) :

Lorsqu'un processus fait un appel système avec la fonction **fork(),** il crée un processus fils identique à lui **(clone)** sauf pour le **PID** (chaque processus à son propre identifiant **PID).**

Pour ne pas exécuter le même code, la fonction **exec()** fournit le code à exécuter par le fils. Elle charge le programme dans l'espace de processus actuel et l'exécute à partir du point d'entrée.

Les processus fils et parent continuent leur exécution en parallèle après l'appel **fork()**, jusqu'à ce que le fils termine son exécution, la fonction **wait()** peut être utilisée pour suspendre le processus de l'appelant.

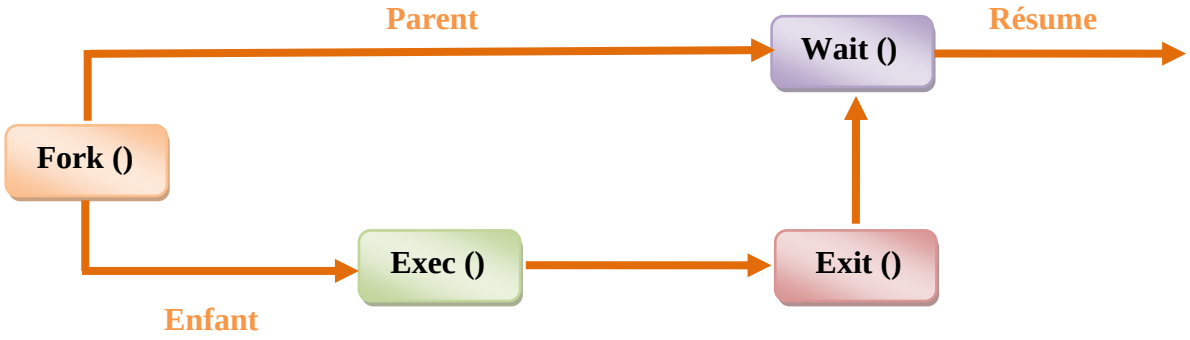


Figure V. 8: Création de processus (primitives fork/join).

- Si le parent n'est pas en attente (c'est-à-dire n'a pas atteint **wait()**), le processus fils terminé est un **zombie** (il va rester dans la mémoire).
- Si le parent a terminé, les processus fils sont des orphelins.

V.7.5.1 Les taches de la fonction fork () :

- Teste l'existence des ressources mémoires suffisante pour permettre la création du nouveau processus ;
- Calcule le **PID** du processus enfant. Le noyau explore ensuite la table proc [] pour vérifier que ce numéro n'est pas en cours d'utilisation. Si le numéro existe dans la table, le calcul est relancé ;
- Recherche une entrée libre dans la table des processus ;
- Duplique le contexte processeur et mémoire du processus parent (cloner) ;
- Met à jour les tables système pour considérer le nouveau processus crée.

V.7.5.2 Les opérations effectuées par le noyau, lors de l'exécution de l'appel système fork () :

- Il alloue un bloc de contrôle dans la table de processus ;
- Il copie les informations continues dans le bloc de contrôle du père dans celui du fils sauf les identificateurs (**PID, PPID**…) ;
- Il alloue un **PID** au processus fils ;
- Il associe au processus fils un segment de code dans son espace d'adressage.
    - Le segment de donnée et la pile sont également partagés et mis en accès lecture seul.

- Lors d'un accès en écriture par l'un des deux processus, le segment de donnée ou la pile sont effectivement dupliquer (la duplication ne sera réalisée que lors de l'écriture).
- Cette technique nommée **copie on write** ou **copie sur écriture,** permettant de réduire le temps de création de processus.

- Finalement le SE retourne le **PID** du processus crée au père et zéro au processus fils.

Chaque des processus père et fils reprend son exécution après le **fork**(), le code et les données étant strictement identiques, ce qui nécessite la disposition d'un mécanisme pour différencier le comportement des deux processus après le **fork**() (**code de retour du fork()**).

➢ Voici une explication détaillée de **fork()** et un exemple de code en **C** montrant son utilisation :

- **Fonction fork ()**
  - **Prototype** : pid_t fork(void) ;
  - **Valeur de retour** :

- **fork ()** retourne un entier de type **'pid_t.'**
- Dans le processus parent, la valeur de retour est le **PID** du processus enfant.
- Dans le processus enfant, la valeur de retour est **0**.
- Si une erreur survient, **fork()** retourne **-1** dans le processus parent, et aucune duplication de processus n'a lieu.

- **Explication du Code :**

1. **Inclusion des bibliothèques nécessaires** :

- `**stdio.h**` pour les fonctions d'entrée/sortie.
- `**stdlib.h**` pour les fonctions standard telles que '**exit**'.
- `**unistd.h**` pour les appels système POSIX comme `**fork**`, `**getpid**` et `**getppid**`.
- `**sys/types.h**` pour les types de données comme `**pid_t**`.
- `**sys/wait.h**` pour utiliser `**wait**`.

2. **Appel à fork()** :

- `**pid_t pid = fork();**` crée un nouveau processus.
- Si `**fork()**` retourne une valeur négative **(`pid < 0`)**, une erreur s'est produite lors de la création du processus.
- Si `**fork()**` retourne **0**, le code s'exécute dans le processus enfant.
- Si `**fork()**` retourne une valeur positive, le code s'exécute dans le processus parent, et cette valeur est le **PID** du processus enfant.

3. **Code spécifique au processus enfant** :

- Le processus enfant affiche son propre PID **(`getpid()`)** et le **PID** de son parent **(`getppid()`).**

4. **Code spécifique au processus parent** :

- Le processus parent affiche son propre **PID** et le **PID** de l'enfant.
- Le parent attend la fin du processus enfant avec **`wait (NULL)`**.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main( ) {
   pid_t pid;

   // Créer un nouveau processus
   pid = fork( );

   if (pid < 0)  {
      // Erreur lors de la création du processus
      fprintf ( stderr, "Erreur de fork\n" );
      return 1;
   } else if (pid  == 0)  {
      // Code exécuté par le processus enfant
      printf ( "Je suis le processus enfant. Mon PID est %d\n", getpid() );
      printf ( "Le PID de mon parent est %d\n", getppid() );
   } else {
      // Code exécuté par le processus parent
      printf ( "Je suis le processus parent. Mon PID est %d\n", getpid() );
      printf ( "J'ai créé un processus enfant avec le PID %d\n" , pid);

      // Attendre que le processus enfant se termine
      wait(NULL);
      printf ( "Le processus enfant s'est terminé\n" );
   }
   return 0;
}
```

V.7.6 Destructions des processus :

Le processus fils peut se terminer de différentes façons :

- Il se termine normalement après l'exécution de la dernière instruction du code qui lui est associée ;
- Il peut exécuter une instruction d'auto destruction. Example : primitive **Exit** ;
- Un processus peut être détruit par un autre processus. Example : primitive **kill ;**
- Une erreur peut provoquer la terminaison d'un processus.

La relation hiérarchique entre un processus et ses descendants est utilisée essentiellement pour contrôler la destruction des processus.

- L'enfant a dépassé les ressources allouées ;
- La tache assignée à l'enfant n'est plus requise.

Généralement la destruction d'un processus entraine :

- La libération des ressources qui lui avait été affecté ;
- Le **PCB** est effacé. Il disparait de la table et des files d'attente du système.

V.8 Thread (fils d'exécution) :

V.8.1 Pourquoi les threads ?

Les processus classiques **(appelées aussi processus lourd)** ont de nombreux inconvénients :

- Une gestion lourde par recopie pour créer de nouveaux processus ;
- Un gaspillage de mémoire (il y a beaucoup de données qui vont être dupliqué) ;
- Une perte de temps à cause de l'exécution séquentielle du code (on sait qu'on exécute une instruction après l'autre il y a un seul fil d'exécution pour un processus).

Dans les systèmes d'exploitation traditionnel chaque processus possède un espace d'adressage et un thread de contrôle unique. Le thread de contrôle représente le chemin d'exécution **(fils d'exécution).**

Dans le processus lourd existe un ou plusieurs threads (processus légère) :

- **Plusieurs activités ont lieu simultanément :** plusieurs taches dans l'application peuvent être implémentés par des threads qui s'exécute en parallèle (affichage, récupérer des données, répondre aux requêtes du réseau, …etc.)
- **Gestion plus simplifiée :** peut simplifier le code, augmenter l'efficacité.
- **Meilleur performance…etc.**

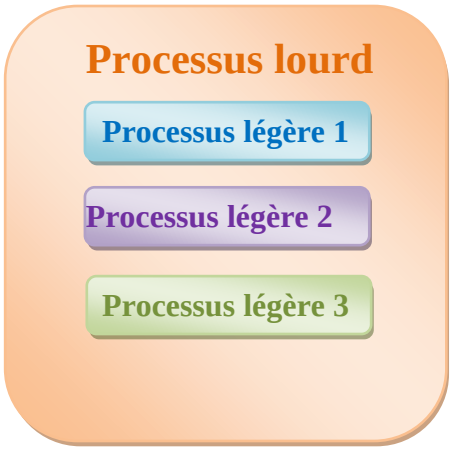
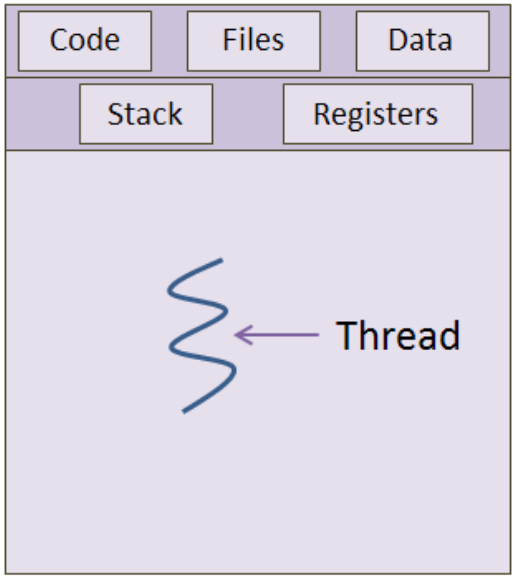


Figure V. 9: Structure d'un thread.

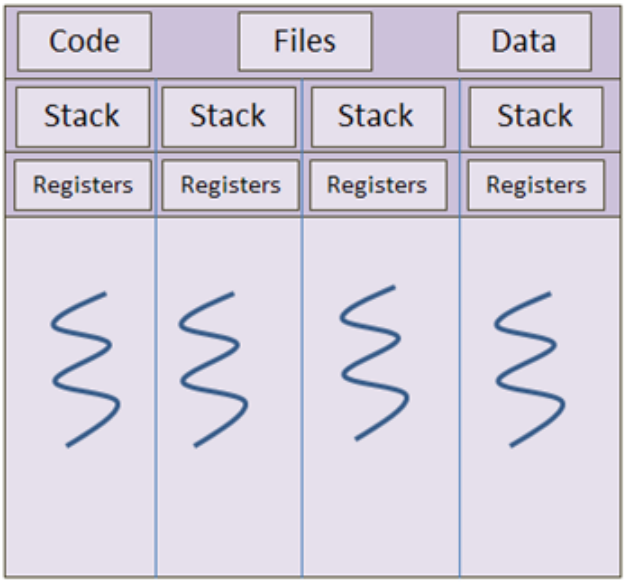
V.8.2 Qu'est-ce qu'un thread ?

- Le modèle processus décrit précédemment est un programme qui s'exécute selon un chemin unique (compteur ordinale), on dit qu'il a un fils d'exécution ou flot de contrôle unique **(single thread).**
- Nombreux systèmes d'exploitation offrent la possibilité d'associer à un même processus plusieurs chemins d'exécution **(multithreading).**

Le **multithreading** permet l'exécution simultanées en pseudo-parallèle de plusieurs parties d'un même processus.



Un seul processus avec un seul thread



Un seul processus avec multithread

- Un thread est une unité d'exécution rattaché à un processus, chargé d'exécuter une partie du programme du processus.

- Pour un processus, ses threads (fils d'exécution) partagent l'ensemble de ses ressources (l'espace d'adressage, fichiers,).
- Un processus léger (thread) est par nomination plus léger à gérer et sa gestion peut être personnalisé.

Lorsqu'un processus est créé, un seul fil d'exécution (thread) est associé au processus :

- Le thread principal qui exécute la fonction main du programme du processus ;
- Ce fil principal d'en créer d'autres ;
- Etats uniques pour chaque processus léger, chaque file possède un **TCB (Thread Control Block) :**
  - Un identificateur unique **ID**,
  - Un état.
  - Des registre (**CO** …),
  - Une priorité.
  - Pointeur vers le processus qui la crée.
  - Pointeurs vers les threads qui créent.

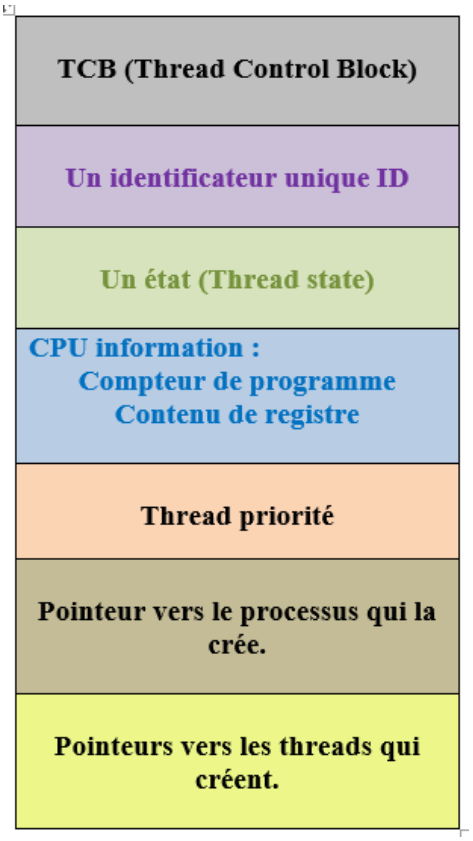


Figure V. 10: Thread Control Block **TCB**.

V.8.3 Avantages des threads:

- **Réactivité :** Le processus peut continuer à s'exécuter même si certains de ces parties sont bloquées (très importante pour les interfaces utilisateurs).
- **Partage de ressources :** Les threads partagent des ressources du processus, il est plus facile pour le programmeur que la mémoire partagée ou les mécanismes de passage de messages facilité de coopération, améliore les performances.
- **Économie d'espacé mémoire et de temps :** Il faut moins de temps pour changer de contexte entre deux threads d'un même processus.
- **Scalabilité :** Peut tirer parti des architectures multi processeurs. Ce qui permet de profiter plus de la **concurrence** et le **parallélisme.**
- **Concurrence :** Plusieurs tâches (processus ou threads) qui progressent, mais avec un processeur unique, la concurrence est seulement une illusion de parallélisme.

Un seul cœur (single core)

T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	…

- **Parallélisme :** effectués plus qu'une tâche **simultanément**.

Core 1

T_1	T_3	T_1	T_3	T_1	…

Core 2

T_2	T_4	T_2	T_4	T_1	…

Il existe deux types de parallélisme :

- **Parallélisme des données :** Distribuées des sous-ensembles des mêmes données sur plusieurs cœurs, en effectuant la même opération sur chacun d'entre eux.
- **Parallèle des tâches** : Distribution des threads à travers les cœurs, chaque thread effectué une opération unique.

V.8.4 les types des threads :

En informatique, et plus particulièrement en programmation concurrente, les threads sont des unités d'exécution indépendantes dans le contexte d'un programme plus large. Voici les principaux types de threads :

V.8.4.1 Threads Noyau :

Lorsqu'un processus est créé, il est composé d'un seul thread noyau qui exécute la fonction main. Ce thread, appelé le thread principal du processus, peu ensuite créer d'autre threads noyau (ou encore des threads utilisateur). En plus de la table des processus, la table des threads est dans le noyau qui alloue du temps **CPU** aux thread noyau et non pas au processus. Si un thread noyau d'un processus se bloque suite à un appel système, un autre thread noyau du même processus peut être élu et donc exécuté.

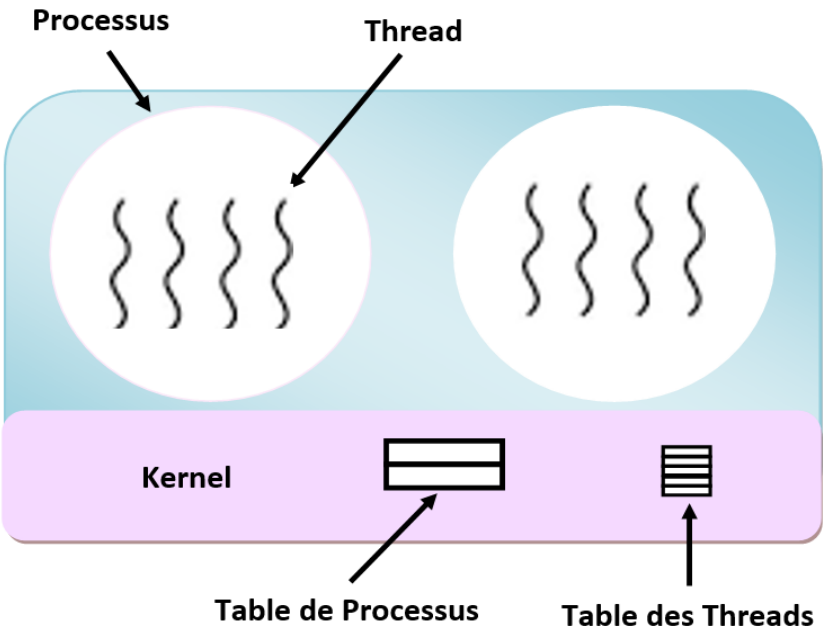


Figure V. 11: Threads Noyau.

- **Avantages et inconvénients :**

Tableau V. 3: Avantages et inconvénients (Threads noyau).

Avantages	Inconvénients
Le noyau peut attribuer plus de temps d'exécution aux processus qui possèdent un plus grand nombre de threads utilisateur.	**Lent :** gestion par appels systèmes.
Le processus n'est pas bloqué si un de ses threads est bloqué.	**Gestion complexe :** utilisateur de toute le **TCB** pour chaque thread.

V.8.4.2 Threads Utilisateur :

Les threads utilisateur sont généralement crées, et gérés plus rapidement (en mode utilisateur) que les threads noyau.

Chaque processus crée sa propre table de thread pour surveiller ses threads utilisateur, cette table est gérée par un système d'exécution (Runtime système).

Le noyau n'est pas au courant des threads utilisateurs, un ordonnanceur est créé pour répartir le temps **CPU** alloué au thread noyau entre tous les threads utilisateur.

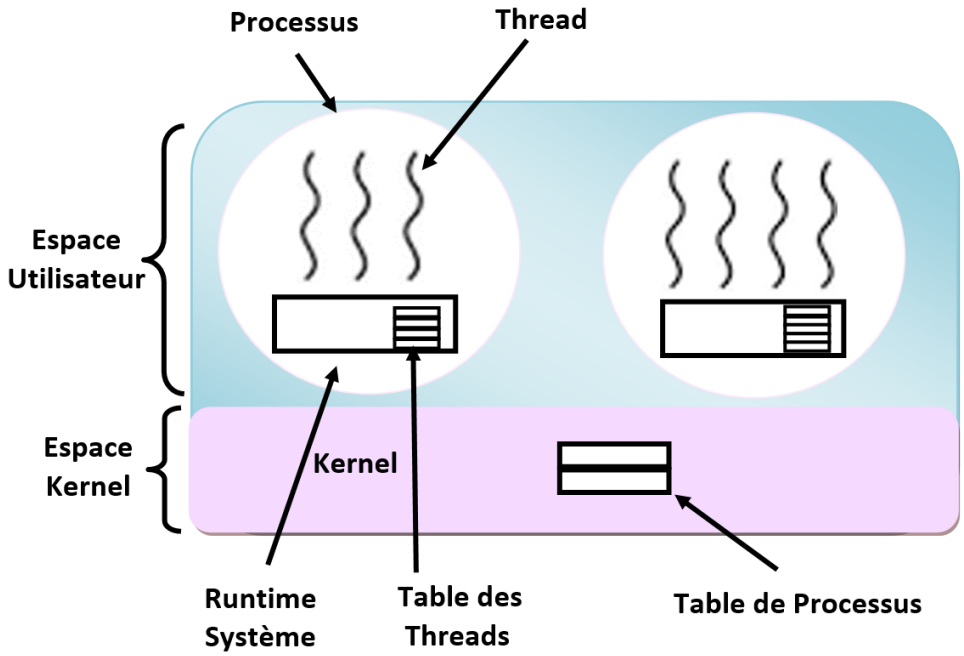


Figure V. 12: Threads Utilisateur.

Avantages et inconvénients :

Tableau V. 4: Avantages et inconvénients (Threads Utilisateur).

Avantages	Inconvénients
Peut-être implémenté dans un SE qui ne supporte pas les threads.	Le noyau attribue même temps d'exécution aux processus qui peuvent contenir un nombre très différent de threads utilisateur.
Présentation simple : CO, registre, pile et un petit control block.	Si un seul thread utilisateur est bloqué, tout le processus est bloqué.
Gestion simple : créer, bascule et synchroniser les threads se font sans l'intervention du noyau (rapide).	
Chaque processus peut avoir son propre algorithme d'ordonnancement.	

V.8.5 les modèles du multithreading :

Un thread utilisateur ne peut pas être exécuté par lui-même. Il doit mapper à un thread noyau. Il existe trois modèles de relation entre les threads utilisateur et thread noyau :

V.8.5.1 One-to-one :

- Chaque thread de niveau utilisateur correspond à un thread du noyau.
- La création d'un thread utilisateur crée un thread de noyau.
- Le nombre de thread par processus est parfois limité en raison des threads de noyau

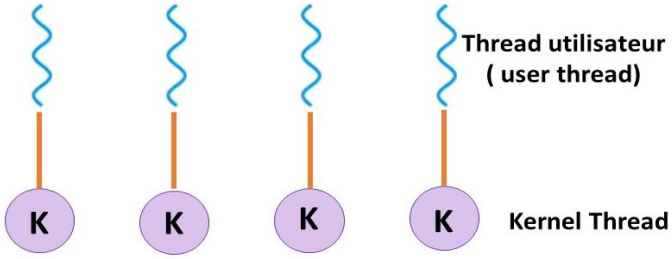


Figure V. 13: Modèles du multithreading One-to-one.

V.8.5.2 Many-to-one :

- De nombreux threads utilisateur sont associés à un seul thread noyau.
- Le blocage d'un thread (requête **E/S** par Example) provoque le blocage de tous les threads associés au même thread noyau.
- Les threads associés au même thread noyau ne peuvent pas s'exécuter en parallèle sur un système multicœur car un seul peut accéder au noyau à la fois.

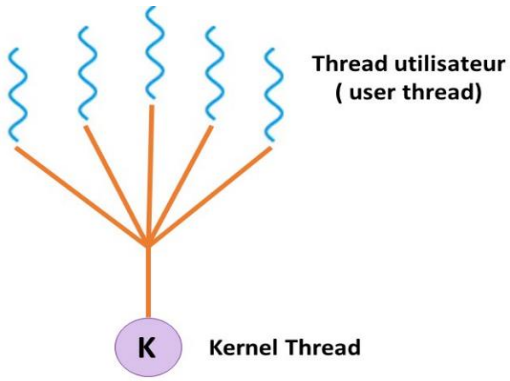


Figure V. 14: Modèles du multithreading Many-to-one.

V.8.5.3 Many-to-many.

- Plusieurs threads utilisateur peuvent être mappé de plusieurs threads noyau (plus petits ou égaux).
- Permet au SE de crée un nombre suffisant de threads noyau, en fonction de sa configuration matérielle, mais donne la possibilité à l'utilisateur de créer autant de threads utilisateur que souhaité.

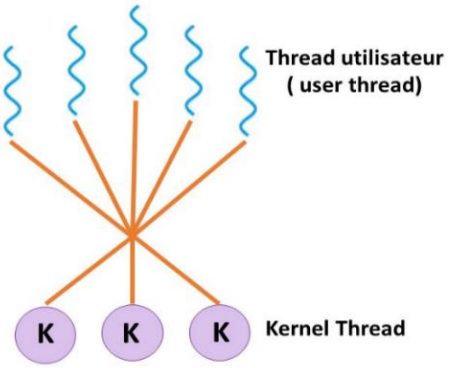
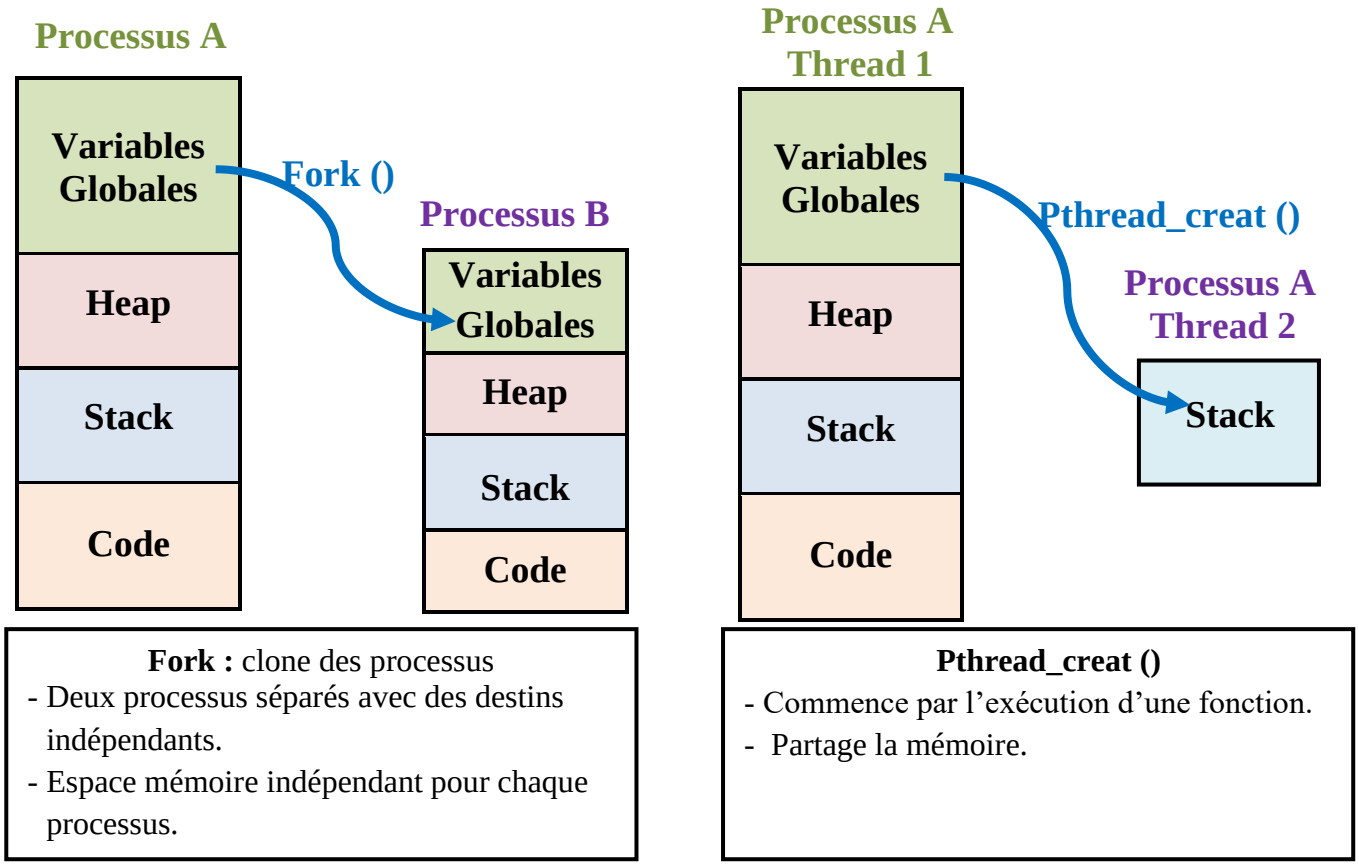


Figure V. 15: Modèles du multithreading Many-to- many.

V.8 Processus VS Threads :



Fork : clone des processus
- Deux processus séparés avec des destins indépendants.
- Espace mémoire indépendant pour chaque processus.

Pthread_creat ()
- Commence par l'exécution d'une fonction.
- Partage la mémoire.

Tableau V. 5: Processus VS Threads.

Propriétés	Processus crées avec fork	Threads d'un processus
Variables	Ils sont une copie de toutes les variables	Partage des variables globales
IDs	Nouveau processus **IDs**	Ont le même processus **ID** mais ont un unique thread **ID**
Data/control	Doivent communiquer explicitement, ex., utilisation des valeurs de retours.	Peuvent communiquer avec des valeurs de retour ou data partagée
Parallélisme (One CPU)	Concurrent	Concurrent
Parallélisme (Multiple CPU)	Peuvent s'exécuter simultanément	Le Kernel thread peuvent s'exécuter simultanément

V.10 Conclusion :

En conclusion, la gestion des processus est un élément clé des systèmes d'exploitation, permettant une utilisation optimale des ressources de l'ordinateur et la bonne exécution des programmes de manière concurrente et ordonnée. Elle englobe de nombreux aspects complexes liés à la création, l'ordonnancement, la communication et la synchronisation des processus.

CHAPITRE-VI- ORDONNANCEMENT DES PROCESSUS (SCHEDULING DE L'UC)

VI.1 Introduction :

La multiprogrammation permet à plusieurs processus d'être dans l'état prêt. Un ordinateur possède forcément plusieurs processus en concurrence pour l'obtention du temps processeur. Cette situation se produit lorsque plusieurs processus sont en état prêt simultanément. Le processeur est alloué à un seul processus à un moment donné.

Le rôle de l'ordonnanceur (ou **Scheduler**) est de choisir parmi les processus prêts à être exécutés. L'ordonnancement des processus, aussi connu sous le terme de **Scheduling**, est une fonction fondamentale des systèmes d'exploitation. Il consiste à gérer l'exécution des processus (ou tâches) par le processeur de manière efficace et équitable. Cette gestion est cruciale pour optimiser les performances globales du système et assurer une expérience utilisateur fluide.

VI.2 Objectif :

- **Comprendre les Fondamentaux de l'Ordonnancement des Processus :**
  - Apprendre les concepts de base, y compris les états des processus, les files d'attente des processus, et la distinction entre processus et threads ce que nous avons vu précédemment.
  - Identifier les objectifs principaux de l'ordonnancement, tels que la maximisation de l'utilisation du **CPU**, la minimisation du temps de réponse, et l'équité entre les processus.
- **Explorer et Comparer les Algorithmes d'Ordonnancement :**
  - Étudier différents types d'algorithmes d'ordonnancement, notamment les algorithmes **non préemptifs (FCFS, SJN)** et **préemptifs (Round Robin, SRT, Priority Scheduling).**
  - Analyser les avantages et les inconvénients de chaque algorithme et comprendre les contextes dans lesquels ils sont les plus efficaces.
- **Évaluer les Performances des Algorithmes d'Ordonnancement :**
  - Utiliser des critères d'évaluation comme le temps d'attente, le temps de réponse, le temps de traitement et le temps de retour pour mesurer l'efficacité des algorithmes d'ordonnancement.
  - Appliquer ces critères pour résoudre des problèmes pratiques et améliorer les performances du système d'exploitation.
- **Implémenter et Observer l'Ordonnancement des Processus :**
  - Découvrir comment les systèmes d'exploitation modernes implémentent l'ordonnancement des processus **(exemples : Windows, Linux, MacOs).**

- Utiliser des outils et des commandes **(par exemple, top, ps sous Linux)** pour observer et gérer l'ordonnancement des processus en temps réel.

VI.3 Définition et Importance :

L'ordonnancement des processus détermine l'ordre et la durée pendant laquelle chaque processus obtient l'accès au **CPU**. Étant donné que le nombre de processus en cours d'exécution dépasse généralement le nombre de **CPU** disponibles, le système d'exploitation doit décider quels processus exécuter, dans quel ordre et pour combien de temps.

- On appelle **« Scheduling » :** Est la fonction du système d'exploitation qui détermine l'ordre et la durée d'exécution des processus par le processeur **(CPU)**. Son but est d'optimiser l'utilisation du **CPU** tout en assurant l'équité et des temps de réponse acceptables.
- On appelle **« Scheduler » :** Est le composant du système d'exploitation responsable de la mise en œuvre du **Scheduling**, c'est-à-dire de la sélection et de l'allocation des processus au **CPU** en fonction de l'algorithme d'ordonnancement choisi.

L'ordonnanceur doit aussi éviter la monopolisation du processus, la famine et les interblocages.

- **La famine :** Un processus peut souffrir d'une famine quand l'ordonnanceur choisit tout le temps des processus plus prioritaires.
- **Interblocage :** Cette situation se produit quand deux processus s'attendent mutuellement.

Example :

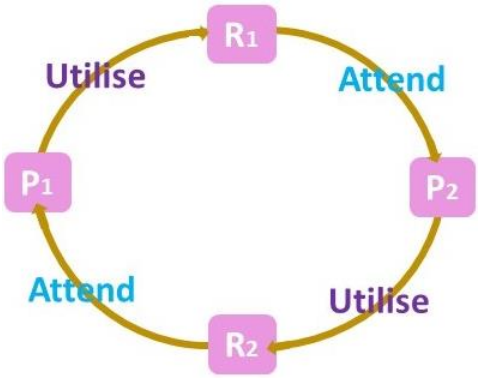


Figure VI. 1: Interblocage des processus.

Le processus **P1** utilise la ressource **R1** qui est attendue par le processus **P2** qui utilise la ressource **R2** attendue par **P1**.

L'objectif d'un algorithme d'ordonnancement consiste à identifier le processus qui conduira à la meilleure performance possible du système. Certes, il s'agit là d'une évaluation subjective dans laquelle entrent en compte différents critères à l'importance relative variable. La politique d'ordonnancement détermine l'importance de chaque critère.

VI.4 Les Critères de Scheduling :

Les critères de **Scheduling** sont des mesures utilisées pour évaluer et comparer l'efficacité des algorithmes de **Scheduling**. Voici les principaux critères :

- **Temps de réponse (Response Time)** : Le temps écoulé entre la soumission d'un processus et le début de sa première exécution.

Importance : Critique pour les systèmes interactifs où les utilisateurs s'attendent à des réponses rapides après une action.

- **Temps de Rotation (Turnaround Time) :** Est le temps total écoulé entre la soumission d'un processus et son achèvement. Il inclut tous les temps d'attente dans les files d'attente, les temps d'exécution sur le **CPU**, et les temps d'entrée/sortie.

  Importance : Indicateur global de la rapidité avec laquelle un système peut traiter des processus.

Temps de Rotation = Temps fin d'exécution - Temps d'arrivée.

- **Temps d'attente (Waiting Time)** : Le temps total qu'un processus passe dans la file d'attente prêt avant de commencer son exécution.

  Importance : Un faible temps d'attente améliore la réactivité du système, surtout pour les processus interactifs.

Temps d'attente = Temps de Rotation – Durée d'exécution.

$$\textbf{Temps moyen d'attente} = \frac{\sum \textbf{Temps d'attente}}{\textbf{nbre de processus}}$$

- **Utilisation du CPU (CPU Utilization)** : Le pourcentage de temps durant lequel le **CPU** est actif et effectue des traitements utiles.

  Importance : Maximiser l'utilisation du **CPU** permet d'optimiser les performances globales du système.

- **Débit (Throughput)** : Le nombre de processus complétés dans un intervalle de temps donné.

  Importance : Un débit élevé indique que le système peut traiter plus de processus dans un laps de temps, améliorant ainsi l'efficacité globale.

- **Équité (Fairness)** : Assurer que chaque processus reçoit une part équitable du temps **CPU**.

  Importance : Empêche la famine de processus, où certains processus pourraient ne jamais obtenir de temps **CPU**.

Ces critères permettent d'évaluer et de comparer les performances des différents algorithmes de **Scheduling**, aidant ainsi à choisir l'algorithme le plus approprié pour un système donné en fonction des besoins spécifiques en termes de performance et de réactivité.

VI.5 Priorité de Scheduling :

La priorité de Scheduling est un concept central dans les systèmes d'exploitation, où chaque processus a assigné une priorité déterminant l'ordre de leur exécution par le **CPU**. Les processus ayant une priorité plus élevée sont exécutés avant ceux avec une priorité plus basse. Cette priorité peut être affectée automatiquement par le système de Scheduling en fonction de critères prédéfinis ou être ajustée manuellement par les utilisateurs ou les administrateurs système,

On distingue deux types de priorité d'ordonnancement :

- **Priorité Statique** : La priorité est fixée au moment de la création du processus et ne change pas pendant la durée de vie du processus (une fois qu'une priorité est allouée à un programme, celle-ci ne changera pas jusqu'à la fin de son exécution).
- **Priorité Dynamique** : La priorité peut changer au cours du temps, souvent basée sur des critères comme l'âge du processus, son comportement (**CPU**-bound vs **I/O**-bound), ou des besoins spécifiques du système (la priorité affectée à un programme change en fonction de l'environnement d'exécution du programme).

VI.6 Types d'Ordonnanceur :

Il utilise différentes files d'attente (Queues) selon l'état du processus :

- **Jobe queue (file d'attente de travaux) :** Cette file d'attente contient tous les processus du système.
- **Ready queue (file d'attente des processus prêts) :** Les processus qui sont prêt dans la mémoire centrale et attendre l'exécution.
- **Device queue (file d'attente du périphérique) :** Les processus qui attendent un périphérique.

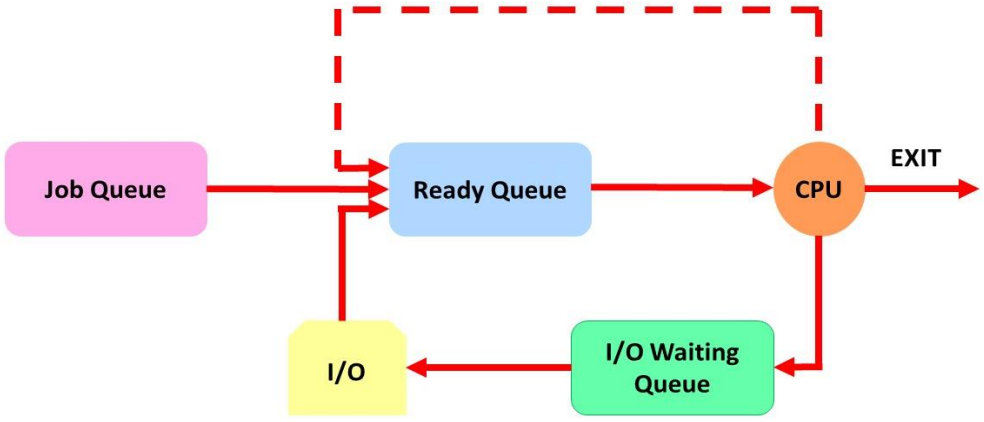


Figure VI. 2: Les files d'attente (Queues) des processus.

- L'opération d'ordonnancement s'effectue par trois types d'ordonnanceur :
  - L'ordonnanceur à long terme ;
  - L'ordonnanceur à courte terme ;
  - L'ordonnanceur à moyen terme.

VI.6.1 L'ordonnanceur à long terme (Long-Term Scheduler) :

L'ordonnanceur à long terme est responsable de décider quels processus, parmi ceux en attente, doivent être admis dans le système et placés dans la file d'attente des processus prêts **(Ready queue)**. Il sélectionne les processus résidant sur le disque et les charge en mémoire pour qu'ils soient prêts à être exécutés par le **CPU**. Cet ordonnanceur gère la multiprogrammation, déterminant ainsi le nombre de processus pouvant résider simultanément en mémoire.

VI.6.2 L'ordonnanceur à courte terme (Short-Term Scheduler) :

L'ordonnanceur à court terme est responsable de sélectionner, parmi les processus prêts qui résident déjà en mémoire, celui qui sera exécuté par le **CPU** à un instant précis. Il prend des décisions de Scheduling très fréquentes, souvent plusieurs fois par seconde, pour optimiser l'utilisation du **CPU**. Cet ordonnanceur se concentre sur des critères tels que le temps d'exécution restant **(burst time)** des processus, leurs priorités, ou d'autres métriques définies par l'algorithme de Scheduling en place.

VI.6.3 L'ordonnanceur à moyen terme (Medium-Term Scheduler) :

L'ordonnanceur à moyen terme intervient entre les ordonnanceurs à long terme et à court terme. Il est responsable de la décision de mettre en attente (swapper out) ou de rapatrier (swapper in) des processus entre la mémoire principale et le stockage de masse (comme le disque dur), en réponse à des conditions telles que la pénurie de mémoire ou la nécessité de libérer de la mémoire pour d'autres processus plus prioritaires. Cela permet de maintenir un équilibre entre l'efficacité du **CPU** et l'utilisation optimale de la mémoire.

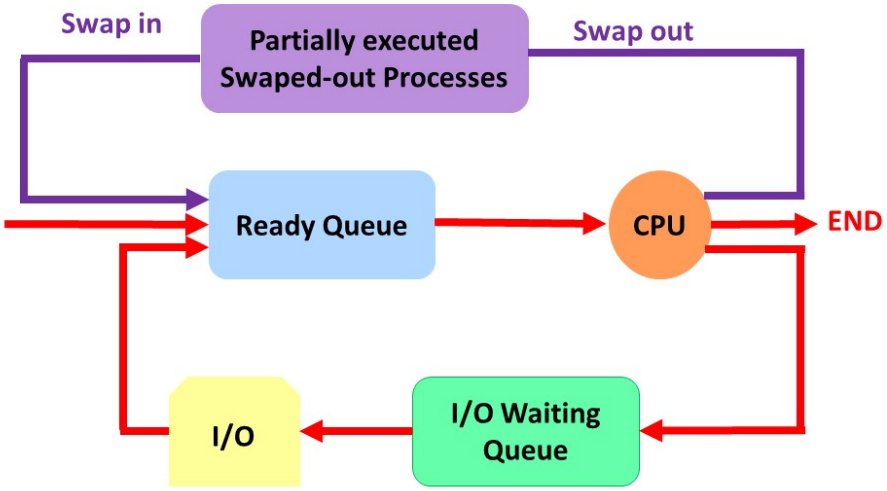


Figure VI. 3: Mécanisme de Swapping.

VI.7 Décision d'ordonnancement :

La décision d'ordonnancement se fait lorsqu'un processus **[18]** :

1. **Passe de running à waiting (bloqué) :** Un processus commute de l'état en exécution vers l'état en attente (requête **E/S**).
2. **Passe de running à ready (interrompu) :** Un processus commute de l'état en exécution vers l'état prêt (quantum de temps utilisé).
3. **Passe de waiting à ready (E/S terminé) :** Un processus commute de l'état en attente vers l'état prêt (terminaison d'une **E/S**).
4. Se termine.

L'ordonnancement peut être classifié en deux types de stratégie :

- ✓ **Non Préemptif** (sous contrôle de processus comme l'Example **#1, #4**).
- ✓ **Préemptif** (hors de contrôle du processus comme l'Example **#2, #3**).

- **Ordonnancement non préemptif (coopératif ou sans réquisition) :** Une fois qu'un processus commence à s'exécuter, il ne peut pas être interrompu jusqu'à ce qu'il soit terminé ou qu'il passe en état d'attente.

  Exemples: First-Come, First-Served **(FCFS)**, Shortest Job Next/ First **(SJN/ SJF).**
- **Ordonnancement préemptif** : Le système d'exploitation peut interrompre un processus en cours d'exécution pour donner le CPU à un autre processus. Exemples: Round Robin **(RR)**, Shortest Remaining Time **(SRT)**, Priority Scheduling.

VI.7.1 Quand Prendre des Décisions d'Ordonnancement :

- **Création d'un Processus** : Lorsqu'un nouveau processus est créé, le système doit décider s'il doit être immédiatement placé dans l'état prêt à être exécuté ou s'il doit attendre.
- **Fin d'un Processus** : Lorsqu'un processus se termine, le système doit choisir un autre processus dans la file d'attente des processus prêts pour prendre sa place sur le **CPU**.
- **Blocage d'un Processus** : Lorsqu'un processus se bloque, par exemple en attendant une opération d'entrée/sortie, le système doit sélectionner un autre processus prêt à exécuter.
- **Réveil d'un Processus** : Lorsqu'un processus bloqué devient prêt, le système doit décider si ce processus doit être exécuté immédiatement ou s'il doit attendre dans la file d'attente des processus prêts.
- **Interruptions de Minuterie** : Dans les systèmes préemptifs, des interruptions régulières du timer permettent au système de vérifier si un processus en cours d'exécution doit être interrompu pour donner la priorité à un autre processus.

Les différents états de préemption du processus entre les files d'attente sont résumés par la **Figure VI.4** suivantes.

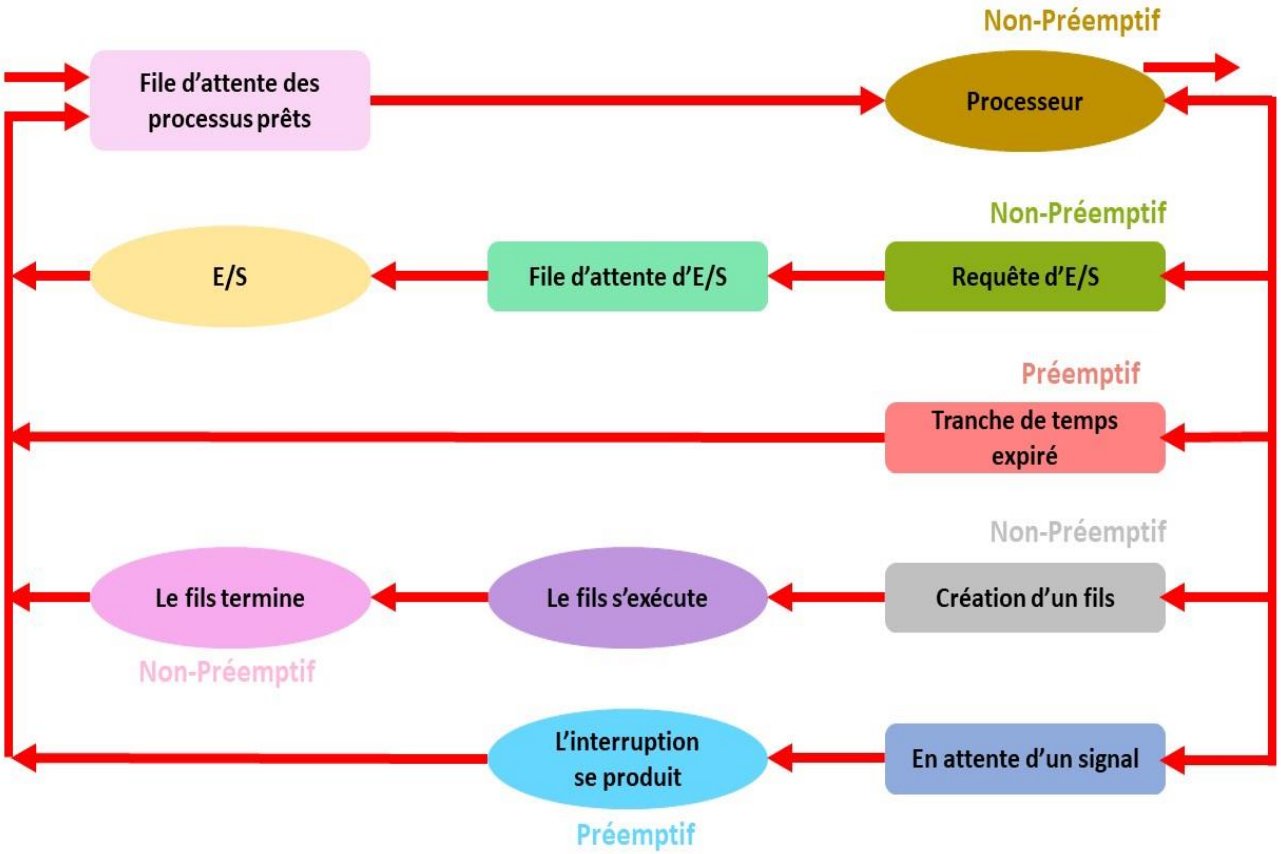


Figure VI. 4: Diagramme des files d'attente du Scheduling des processus avec préemption.

VI.7.2 Ordonnanceur du CPU :

- Les processus prêtes et les processus bloqués sont gérés dans deux files d'attentes distinctes chainant leur **PCB** ;
- Le **répartiteur** permet de choisir sur quel **CPU** exécuté le processus sélectionné par l'ordonnancement à court terme ;
- Le rôle du répartiteur est très réduit dans un contexte monoprocesseur ;
- L'ordonnanceur trie la file des processus prêts selon les critères qui spécifient la politique d'ordonnancement.

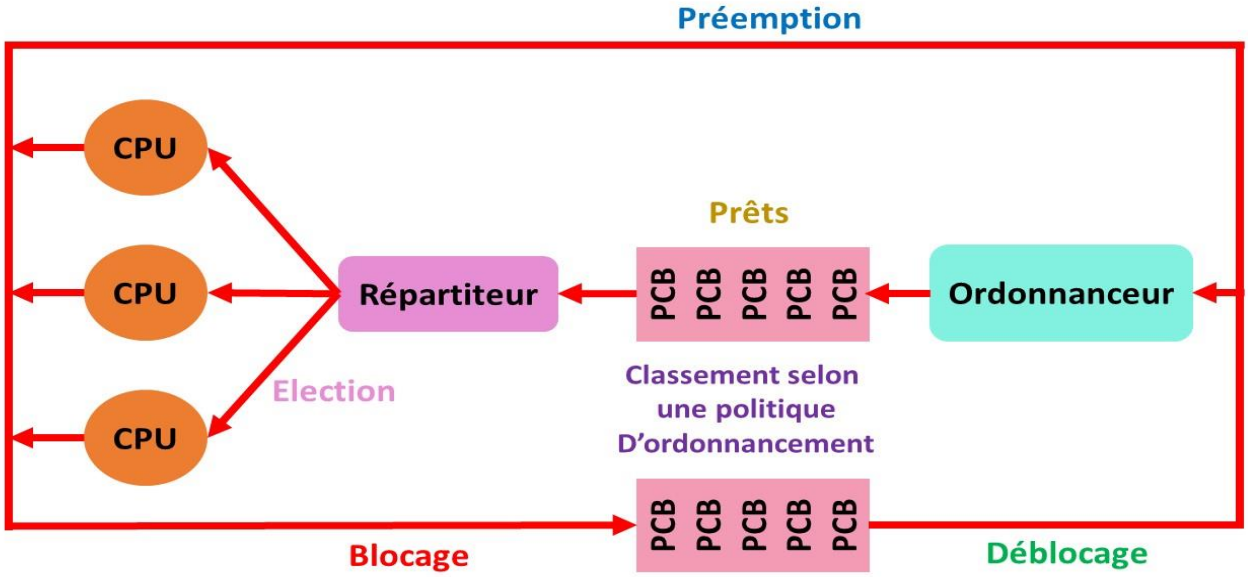


Figure VI. 5: Ordonnanceur du CPU (système multiprocesseur).

VI.8 Politique d'ordonnancement :

Une politique d'ordonnancement définit les règles et les critères utilisés par un système d'exploitation pour déterminer l'ordre et la durée d'exécution des processus. L'objectif principal de la politique d'ordonnancement est d'optimiser l'utilisation du **CPU**, de minimiser les temps d'attente, de maximiser le débit et d'assurer une répartition équitable des ressources entre les processus. Voici quelques-unes des politiques d'ordonnancement les plus courantes **[19]** :

VI.8.1 Algorithmes d'ordonnancement sans réquisition :

Avec les algorithmes d'ordonnancement sans réquisition, un processus affecté au processeur ne peut pas être interrompu. La décision d'ordonnancement aura lieu lorsque le processus courant termine son exécution ou réalise une opération d'E/S. Trois algorithmes d'ordonnancement sans réquisition sont présentés dans cette section : premier arrivé premier servi, le plus court d'abord et l'ordonnancement avec priorité simple.

VI.8.1.1 Politique « Premier Arrivé Premier Servi » (FCFS, First Come First Served- FIFO) :

L'algorithme d'ordonnancement premier arrivé premier servi **(First Come First Served : FCFS)** est le plus simple des algorithmes d'ordonnancement. Sa mise en œuvre peut être réalisée grâce à une file d'attente **FIFO**. Dès que le processeur est libre, il est alloué au processus en tête de la file d'attente. Une fois le processeur est alloué à un processus, celui-ci le garde jusqu'à la fin du processus ou suite à la demande d'une entrée/sortie.

Informations données :

Processus	Durée d'execution	Temps d'arrivée
A	3	0
B	6	1
C	4	4
D	2	6
E	1	7

1. Construire un schéma d'exécution ;
2. Calculer le temps de rotation (Turnaround Time) pour chaque processus ;
3. Calculer le temps moyen de rotation ;
4. Calculer le temps d'attente pour chaque processus ;
5. Calculer le temps moyen d'attente ;

➢ **Schéma d'exécution (Grantt) :**

A	A	A	B	B	B	B	B	B	C	C	C	C	D	D	E

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

➢ **Temps de rotation/ d'attente moyen :**

Processus	Date Début d'Exe.	Date Fin d'Exe.	Temps De rotation	Temps D'attente
A	**0**	**3**	**3**	**0**
B	**3**	**9**	**8**	**2**
C	**9**	**13**	**9**	**5**
D	**13**	**15**	**9**	**7**
E	**15**	**16**	**9**	**8**
Temps de rotation moyen		**(3+8+9+9+9) /5= 38/5 = 7,6 UT**		
Temps d'attente moyen		**(0+2+5+7+8) /5 = 23/5 = 4,6 UT**		

- **Principe** : Les processus sont exécutés dans l'ordre de leur arrivée dans la file d'attente des processus prêts.
- **Avantages** : Simple à implémenter, équitable dans l'ordre d'arrivée.
- **Inconvénients** : Peut entraîner des temps d'attente élevés pour les processus longs (problème du covoiturage).

VI.8.1.2 Politique « Job le Plus Court d'Abord » (SJF, Shortest Job First) :

L'algorithme d'ordonnancement le plus court d'abord **(SJF)** affecte le processeur au processus possédant le plus court temps d'exécution. Chaque processus est associé à la longueur de son prochain cycle. Le

processeur est alloué au processus de plus court cycle. Dans le cas où plusieurs processus possèdent la même durée, la politique **FCFS** sera alors utilisée.

Exemple : on reprend l'exemple précédent avec la politique **SJF**.

- **Schéma d'exécution (Grantt) :**

A	A	A	B	B	B	B	B	B	E	D	D	C	C	C	C

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

- **Temps de rotation/ d'attente moyen :**

Processus	Durée D'Exe.	Temps D'arrivée	Date Début d'Exe.	Date Fin d'Exe.	Temps De rotation	Temps D'attente
A	3	0	0	3	3	0
B	6	1	3	9	8	2
C	4	4	12	16	12	8
D	2	6	10	12	6	4
E	1	7	9	10	3	2
Temps de rotation moyen			(3+8+12+6+3) /5= 32/5 = 6,4 UT			
Temps d'attente moyen			(0+2+8+4+2) /5 = 16/5 = 3,2 UT			

- **Principe** : Le processus avec le temps d'exécution le plus court est exécuté en premier.
- **Avantages** : Minimise le temps d'attente moyen.
- **Inconvénients** : Difficile à mettre en œuvre car le temps d'exécution des processus doit être connu à l'avance, peut entraîner la famine pour les processus longs.

VI.8.1.3 Politique avec priorité simple :

Le principe de cet algorithme consiste à attribuer à chaque processus une valeur qui indique sa priorité. Le processeur est alloué au processus de plus haute priorité. La priorité varie selon les systèmes et peut aller de **0** à **127.** Un classement possible des processus est donné comme suit (des processus les plus prioritaires aux processus les moins prioritaires) :

1. Processus système.
2. Processus interactifs.
3. Processus batch.
4. Processus utilisateurs.

- **Principe** : Chaque processus a assigné une priorité et les processus avec des priorités plus élevées sont exécutés avant ceux avec des priorités plus basses.
- **Avantages** : Permet de gérer les processus critiques ou temps réel.
- **Inconvénients** : Peut entraîner la famine des processus de basse priorité, nécessite une gestion dynamique des priorités (par exemple, vieillissement).

VI.8.2 Algorithmes d'ordonnancement avec réquisition :

L'idée de la réquisition est d'interrompre un processus en cours d'exécution à n'importe quel moment. Cette stratégie permet de s'assurer qu'un processus ne prendra pas tout le temps **CPU**. Une horloge génère une interruption plusieurs fois par seconde. La décision d'ordonnancement peut avoir lieu lorsque le processus courant termine son exécution, réalise une opération d'**E/S** ou lors de l'interruption d'un processus courant.

Les algorithmes d'ordonnancement avec réquisition sont présentés dans cette section :

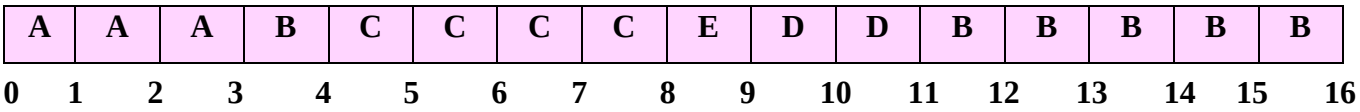
- Algorithme ayant le Plus Court Temps Restant ;
- Algorithme à base de priorité ;
- Algorithme Tourniquet ;
- Algorithme avec priorité statique multi-niveaux ;
- Algorithme avec priorité dynamique multi-niveaux.

VI.8.2.1 Politique « Job ayant le Plus Court Temps Restant » (SRTF, Shortest Remaining Time First) :

La politique **SRTF** consiste à toujours sélectionner le processus ayant le temps restant le plus court pour l'exécution. Cela signifie que si un nouveau processus arrive et que son temps de traitement restant est inférieur au temps restant du processus en cours d'exécution, le **CPU** bascule immédiatement sur ce nouveau processus. Cette méthode équivaut à la méthode SJF mais **préemptive**.

Exemple : Prenant notre premier exemple précédent avec la politique **SRTF**.

- **Schéma d'exécution (Grantt) :**



- **Temps de rotation/ d'attente moyen/ nombre de commutation :**

Processus	Durée D'Exe	Temps D'arrivée	Début d'Exe.	Préempté	Repris	Fin d'Exe.	Temps De rotation	Temps D'attente
A	3	0	0			3	3	0
B	6	1	3	4	11	16	15	9
C	4	4	4			8	4	0
D	2	6	9			11	5	3
E	1	7	8			9	2	1
Temps de rotation moyen			(3+15+4+5+2) /5= 29/5 = 5,8 UT					
Temps d'attente moyen			(0+9+0+3+1) /5 = 13/5 = 2,6 UT					
Nbr de changements de CTXT			5					

- **Avantages**

- **Minimisation du temps de réponse moyen :** En choisissant toujours le processus avec le temps de traitement le plus court, cette politique minimise le temps moyen que les processus passent dans le système**.**
- **Efficacité pour les tâches courtes :** Les tâches courtes sont exécutées rapidement, ce qui améliore la réactivité du système pour ces tâches.

- **Inconvénients**

- **Starvation :** Les processus avec de longs temps de traitement peuvent être constamment préemptés par des processus plus courts, ce qui peut entraîner une starvation (famine) de ces processus.
- **Difficile à mettre en œuvre :** La nécessité de connaître ou d'estimer le temps de traitement restant des processus peut rendre cette politique difficile à mettre en œuvre, surtout dans les systèmes où les temps de traitement ne sont pas connus à l'avance.

VI.8.2.2 Politique à base de priorité :

La priorité affectée à un processeur peut dépendre :

- De l'utilisateur qui a lancé l'exécution du processus
- De la quantité de ressources demandée par le processus, etc.

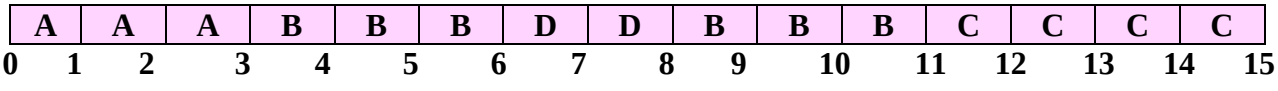
Ce type de politiques est particulièrement utile dans les systèmes temps réel où il s'agit souvent de répondre à des événements à caractères urgents. Dans ce contexte, la priorité est affectée en fonction des contraintes liées au processus lui-même.

Dans le cas d'une politique **à base de priorité préemptive**, l'arrivée d'un processus plus prioritaire entraîne l'arrêt du processus en cours d'exécution et l'attribution du processeur au nouveau processus.

Informations données :

Processus	Durée d'exécution	Temps d'arrivée	Priorité
A	**3**	**0**	**3**
B	**6**	**3**	**1**
C	**4**	**5**	**2**
D	**2**	**6**	**0**

- **Schéma d'exécution (Grantt) :**



➢ **Temps de rotation/ d'attente moyen :**

Processus	Date Début D'exécution	Préempté	Repris	Date fin D'exécution	Temps de Rotation	Temps d'attente
A	**0**			**3**	**3**	**0**
B	**3**	**6**	**8**	**11**	**8**	**2**
C	**11**			**15**	**10**	**6**
D	**6**			**8**	**2**	**0**
Temps de rotation moyen			**(3+8+10+2) /4= 23/4 = 5,75 UT**			
Temps d'attente moyen			**(0+2+6+0) /4 = 8/4 = 2 UT**			

VI.8.2.3 Politique « Tourniquet » (Round Robin, RR) :

L'algorithme d'ordonnancement tourniquet **(Round Robin : RR)** a été conçu pour les systèmes à temps partagé. Le principe de cet algorithme consiste à définir un intervalle de temps appelé **quantum**. Le processeur est alloué aux processus à tour de rôle pendant la durée du quantum. Dans la pratique la durée d'un quantum varie entre **10** et **100** ms.

- Si le Quantum est très grand, la politique **Round Robin** se confond avec la politique **FCFS**.
- Si le Quantum est très court, provoque trop de commutations de contexte et abaisse l'efficacité du processeur.

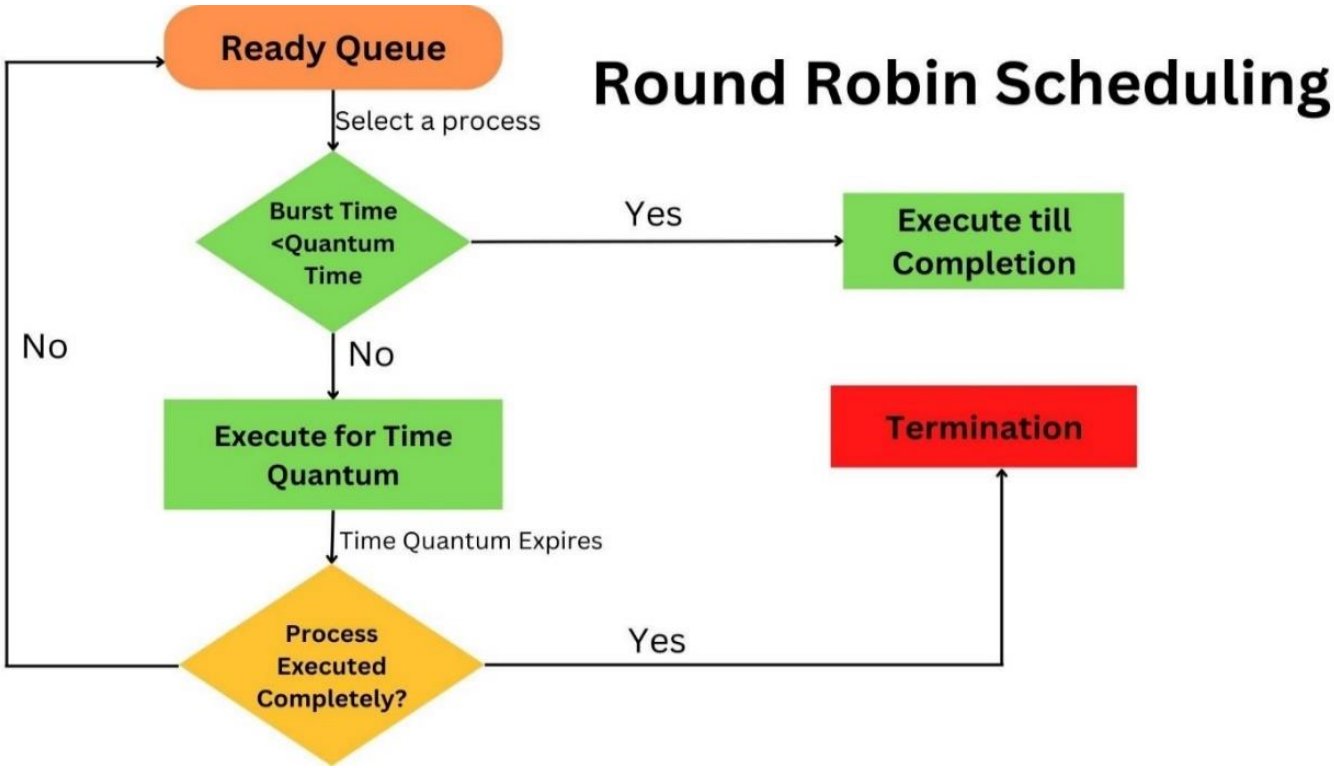


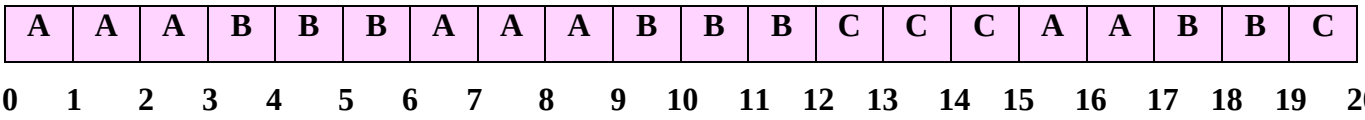
Figure VI. 6: Round Robin Scheduling.

Informations données :

Processus	Durée d'exécution	Temps d'arrivée
A	**8**	**0**
B	**8**	**2**
C	**4**	**7**

On suppose que le **Quantum = 3** et que la durée de **commutation = 0**.

- **Schéma d'exécution (Grantt) :**



- **Temps de rotation/ d'attente moyen/ nombre de commutation :**

Processus	Date Début d'Exe.	Préempté	Repris	Date Fin d'Exe.	Temps De rotation	Temps D'attente
A	**0**	**3**	**6**	**17**	**17**	**9**
		9	**15**			
B	**3**	**6**	**9**	**19**	**16**	**8**
		12	**17**			
C	**12**	**15**	**19**	**20**	**13**	**9**
Temps de rotation moyen		**(17+16+13) /3= 46/3 = 15,33 UT**				
Temps d'attente moyen		**(9+8+9) /3 = 26/3 = 8,66 UT**				
Nbr de changements de CTXT		**7**				

Example 2 : On suppose que le **Quantum = 3** et que la durée de **commutation = 1**.

Processus	Durée d'exécution	Temps d'arrivée
A	**8**	**0**
B	**$5^{(2)}3$**	**3**
C	**4**	**7**

La durée d'exécution **5$^{(2)}$3** : (signifie que le processus **B** se divise en deux exécutions, l'une est de **5** et l'autre de **3**).

- **Schéma d'exécution (Grantt) :** en symbolise la commutation avec **: #**.

#	A	A	A	#	B	B	B	#	A	A	A	#	C	C	C	#	B	B	#	A	A	#	C	#	B	B	B
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**

- **Temps de rotation/ d'attente moyen/ nombre de commutation :**

Processus	Date Début D'Exe.	Préempté	Repris	Date Fin d'Exe.	Temps de Rotation	Temps d'attente
A	1	4	9	22	22	14
		12	20			
B	5	8	17	28	25	15
		19	25			
C	13	16	23	24	17	13
Temps de rotation moyen			(22+25+17) /3= 64/3 = 21,33 UT			
Temps d'attente moyen			(14+15+13) /3 = 42/3 = 14 UT			
Nbr de changements de CTXT			7			

- **Principe** : Chaque processus reçoit un quantum de temps **CPU** fixe et est replacé en fin de file d'attente après l'expiration de son quantum.
- **Avantages** : Équitable, assure un temps de réponse raisonnable pour tous les processus.
- **Inconvénients** : Le choix d'un quantum de temps inapproprié peut affecter les performances (trop court = surcharge contextuelle, trop long = retour à **FCFS**).

VI.8.2.4 Politique « Plusieurs Niveaux de Queues » (Multilevel Queue Scheduling) :

Cette classe d'algorithmes de Scheduling a été développée pour des situations où on peut facilement classer les processus dans des groupes différents (subdivisée en plusieurs files d'attentes séparées), suivant la classe des :

- Processus système.
- Processus de premier plan (interactifs).
- Processus d'arrière-plan (batch).
- Processus utilisateurs.

➢ **L'ordonnancement avec priorité statique (ne change pas de files d'attente).**

Chaque file est gérée suivant une politique de Scheduling propre à elle, et s'adaptant mieux à la classe des processus qu'elle contient (les file **Interactifs** est plus prioritaire que les file **Batch**).

- Processus batch ⟹ **FCFS**.
- Processus interactifs ⟹ **Round Robin.**

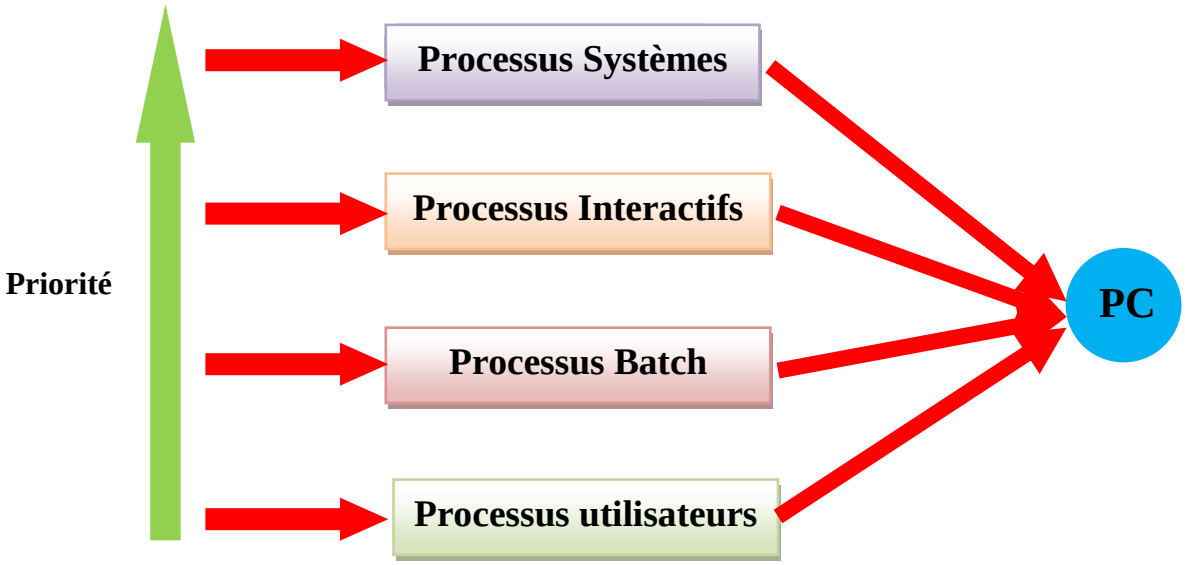


Figure VI. 7: Ordonnancement avec priorité statique.

- **Principe** : Les processus sont classés dans différentes files d'attente selon des critères comme la priorité ou le type de processus (systèmes, interactifs, batch). Chaque file d'attente peut avoir son propre algorithme d'ordonnancement.
- **Avantages** : Permet de séparer les processus selon leurs besoins spécifiques, flexible.
- **Inconvénients** : Complexe à gérer, nécessite un bon ajustement des politiques entre les files d'attente.

VI.8.2.5 Politique à Plusieurs Niveaux de Queues Dépendantes (Multilevel Feedback Queues Scheduling) :

L'ordonnancement avec priorité dynamique multi-niveaux (Multi-level-feedback round robin queues) permet aux processus de changer en permanence de file d'attente. Chaque file d'attente dispose d'un quantum de temps pour exécuter ses processus. A la fin de ce quantum, les processus qui n'ont pas été traités changent de file. Ainsi, un processus est affecté à une file pendant un quantum de temps. Si son exécution n'est pas terminée à la fin du quantum, il passe aux queues de la file immédiatement inférieure.

Example : Le système d'ordonnancement des processus sous UNIX (**BSD** 4.3 et system V4) utilise cette stratégie avec pour chaque file d'attente un algorithme d'ordonnancement Tourniquet.

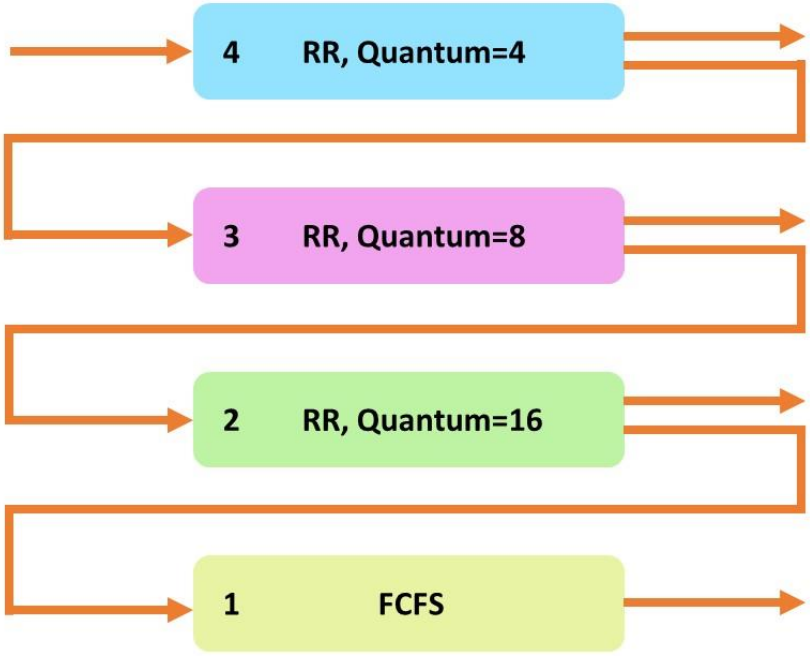


Figure VI. 8: Ordonnancement avec priorité dynamique multi-niveaux.

- **Principe** : Extension du multilevel queue Scheduling où les processus peuvent changer de file d'attente en fonction de leur comportement et de leur temps passé dans le système.
- **Avantages** : Combine les avantages de plusieurs algorithmes, adapte dynamiquement l'ordonnancement aux besoins des processus.
- **Inconvénients** : Très complexe à implémenter et à ajuster correctement.

VI.8 Conclusion :

L'ordonnanceur doit également prendre en compte des facteurs comme la préemption, la commutation de contexte et l'ordonnancement en temps réel pour fournir un système réactif et efficace. Dans l'ensemble, l'ordonnanceur de processus est un élément crucial qui permet l'illusion de l'exécution simultanée de plusieurs processus sur un nombre limité de processeurs physiques.

CHAPITRE-VII- COMMUNICATION INTERPROCESSUS ET SYNCHRONISATION

VII.1 Introduction :

Une ressource désigne toute entité dont a besoin un processus pour s'exécuter, elle peut être matérielle (processeur, périphérique, etc.) ou logicielle (variable, etc.).

Chaque processus dispose d'un espace d'adressage propre et indépendant, protégé par rapport aux autres processus, les processus doivent souvent communiquer et partager différentes ressources entre eux.

VII.2 Objectif :

- Étudier les mécanismes de communication entre processus/threads.
- Apprendre à synchroniser l'accès aux ressources partagées pour éviter les conditions de concurrence et les deadlocks.

VII.3 Communication interprocessus (IPC) :

La communication interprocessus (**IPC**) est un ensemble de mécanismes permettant aux processus de s'échanger des données et de se synchroniser dans un environnement informatique. Ces mécanismes sont essentiels pour la construction de systèmes multi-tâches, distribués ou parallèles **[20]**.

VII.3.1 Modèles de communication interprocessus :

Les processus au sein d'un système peuvent être **indépendants** ou **coopératif** :

- Un processus **indépendant** ne peut pas affecter ou être affecté par l'exécution d'un autre processus.
- Un processus **coopératif** peut affecter ou être affecté par d'autre processus (par Example : partage des données).

La communication interprocessus (**IPC**) permet à plusieurs processus de s'échanger des données et de se synchroniser. Voici les principales méthodes de communication interprocessus :

VII.3.1.1 Pipes et FIFOs (First In, First Out) :

- **Pipes** :

Les pipes permettent une communication séquentielle entre deux processus. Ils sont utilisés pour envoyer des données d'un processus à un autre de manière unidirectionnelle. Les pipes sont couramment utilisées entre un processus parent et un processus enfant.

- **Files FIFO (First In, First Out):**

Les files **FIFO**, ou tubes nommés, fonctionnent comme des pipes, mais elles peuvent être utilisées par des processus sans lien de parenté. Elles reçoivent un nom dans le système de fichiers, ce qui permet à tout processus ayant les permissions appropriées d'écrire ou de lire dans la file.

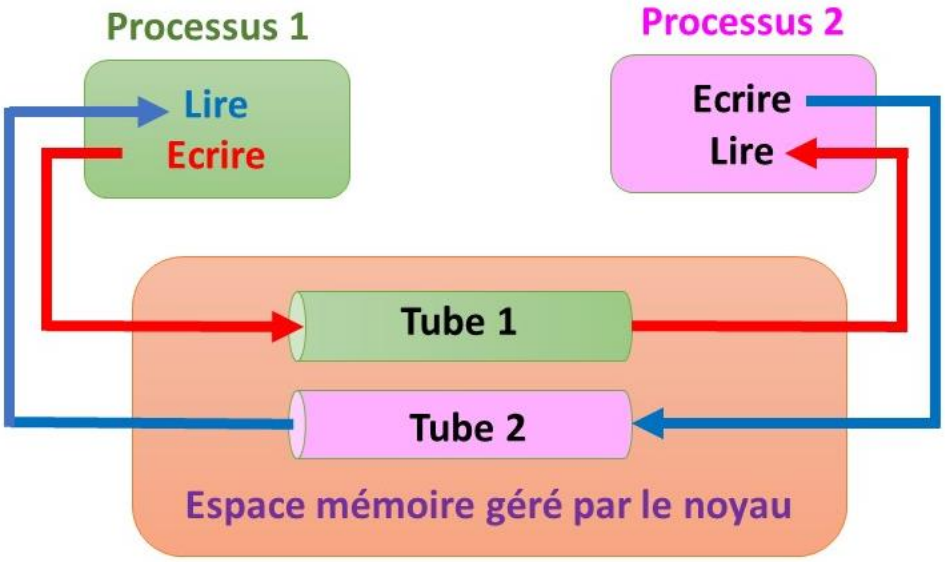


Figure VII. 1: Les Files Fifo (Tubes nommés).

VII.3.1.2 Passage de message (message passing) :

Files de messages permettent l'envoi et la réception de messages entre processus. Les messages peuvent être structurés de manière à inclure des types et des priorités.

Le passage de messages est une méthode de communication interprocessus où les processus échangent des messages explicites pour partager des informations et de synchroniser. Cette méthode est largement utilisée dans les systèmes distribués et parallèles.

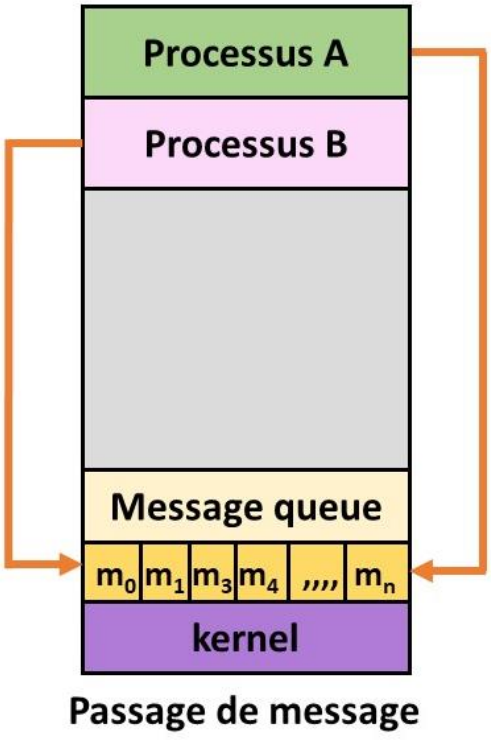


Figure VII. 2: Passage de message (message passing).

VII.3.1.3 La mémoire partagée (shared memory) :

La mémoire partagée permet aux processus de communiquer en lisant ou en écrivant dans une région de mémoire commune. Cette méthode est rapide, car elle évite la duplication des données entre les processus. Toutefois, elle nécessite des mécanismes de synchronisation pour éviter les conflits d'accès.

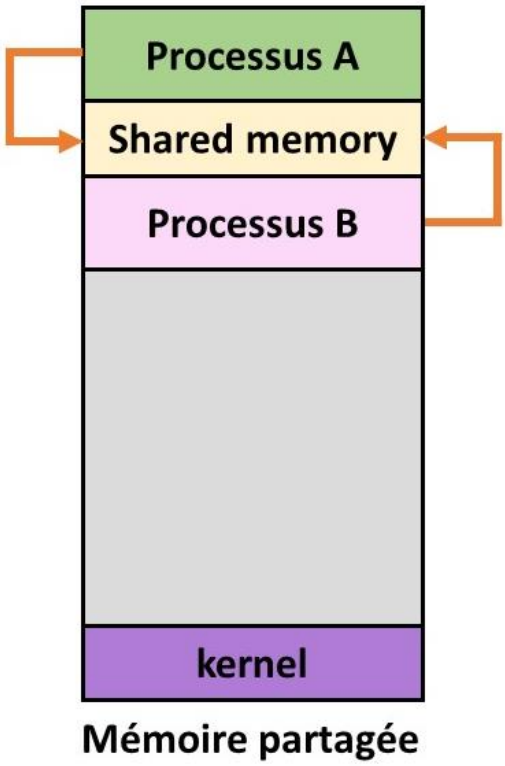


Figure VII. 3: La mémoire partagée (shared memory).

VII.3.1.4 Mémoire Mappée :

La mémoire mappée est similaire à la mémoire partagée, mais elle est associée à un fichier. Cela permet aux processus de partager des données via un fichier mappé en mémoire, facilitant la persistance des données et le partage entre processus indépendants.

VII.3.1.5 Signaux :

Les signaux sont des notifications asynchrones envoyées par un processus pour signaler un événement à un autre processus. Chaque signal a une signification spécifique et peut déclencher des actions prédéfinies.

VII.3.1.6 Sockets :

Les sockets permettent la communication entre des processus sans lien direct, pouvant même se trouver sur des machines distinctes. Ils sont utilisés pour les communications réseau, que ce soit au sein d'une même machine (sockets **UNIX**) ou entre machines différentes (sockets **TCP/IP**).

VII.3.2 Problème de communication interprocessus :

Les problèmes de communication interprocessus sont des défis courants rencontrés lors de la conception et de l'implémentation de systèmes informatiques où plusieurs processus doivent collaborer et échanger des informations de manière efficace et sécurisée.

Plusieurs problèmes doivent être résolus pour gérer la communication interprocessus :

- Comment passer des infos d'un processus à un autre ?
- Comment éviter les conflits dans des activités critique ?
- Parfois un processus **B**, qui utilise des données générées par un autre processus **A**, doit attendre la fin d'exécution de ce dernier avant de s'exécuter.

Il faut gérer aussi les processus et threads qui peuvent s'exécuter en **concurrence :**

- Peuvent être interrompus à tout moment (en complétant partiellement l'exécution).
- Les accès concurrents aux données partagées peuvent être incohérents.

VII.3.2.1 Exemples de Scénarios de Problèmes :

- **Scénario de Deadlock** : Deux processus **A** et **B** attendent chacun qu'une ressource détenue par l'autre soit libérée.
- **Scénario de Starvation** : Un processus de faible priorité est continuellement ignoré lors de l'allocation de ressources, même si ces ressources sont disponibles.
- **Scénario de Race Condition** : Deux processus tentent d'incrémenter une variable partagée sans synchronisation appropriée, conduisant à des résultats imprévisibles selon l'ordre d'exécution.

VII.3.2.2 Solutions et Bonnes Pratiques :

Pour résoudre les problèmes de communication interprocessus, il est essentiel de :

- Utiliser des mécanismes appropriés de synchronisation comme les verrous mutex, les sémaphores, ou les moniteurs pour gérer l'accès aux ressources partagées ;
- Éviter les deadlocks en adoptant des stratégies de gestion de ressources sûres, telles que l'allocation et la libération de ressources dans un ordre prédéfini ;
- Assurer une priorisation adéquate des processus pour éviter la **starvation**, en garantissant que tous les processus ont une chance équitable d'accéder aux ressources nécessaires ;
- Concevoir des protocoles de communication efficaces et bien définis pour minimiser la surcharge et optimiser les performances du système.

VII.4 Synchronisation :

Différents mécanismes sont proposés pour la synchronisation entre des processus concurrents. Nous présentons dans ce qui suit un aperçu de solutions proposées pour assurer une exclusion mutuelle entre processus concurrents : le verrouillage de fichiers, les solutions avec attente active et les solutions avec blocage.

Pour garantir la cohérence, il faut implémenter des mécanismes de **synchronisation.**

VII.4.1 Section critique :

- Une **ressource critique** est une ressource qui ne peut être accéder que par un seul processus à la fois.
- Le code d'utilisation d'une ressource critique est appelé **section critique.**
- Pour accéder à une ressource critique, le processus (Thread) doit :
  - ✓ Demander la permission avant de commencer la section critique avec **« entry section ».**
  - ✓ Terminer la section critique avec **« exit section ».**

Il peut par la suite exécuter le reste du code qui ne touche pas aux ressources critiques dans la **« remainder section » [20].**

VII.4.2 Exclusion mutuelle :

L'exclusion mutuelle est une solution permettant d'assurer que deux processus ne peuvent pas accéder simultanément à une ressource critique. Elle empêche tout processus d'entrer dans sa section critique tant qu'un autre processus est déjà en train d'utiliser sa propre section critique.

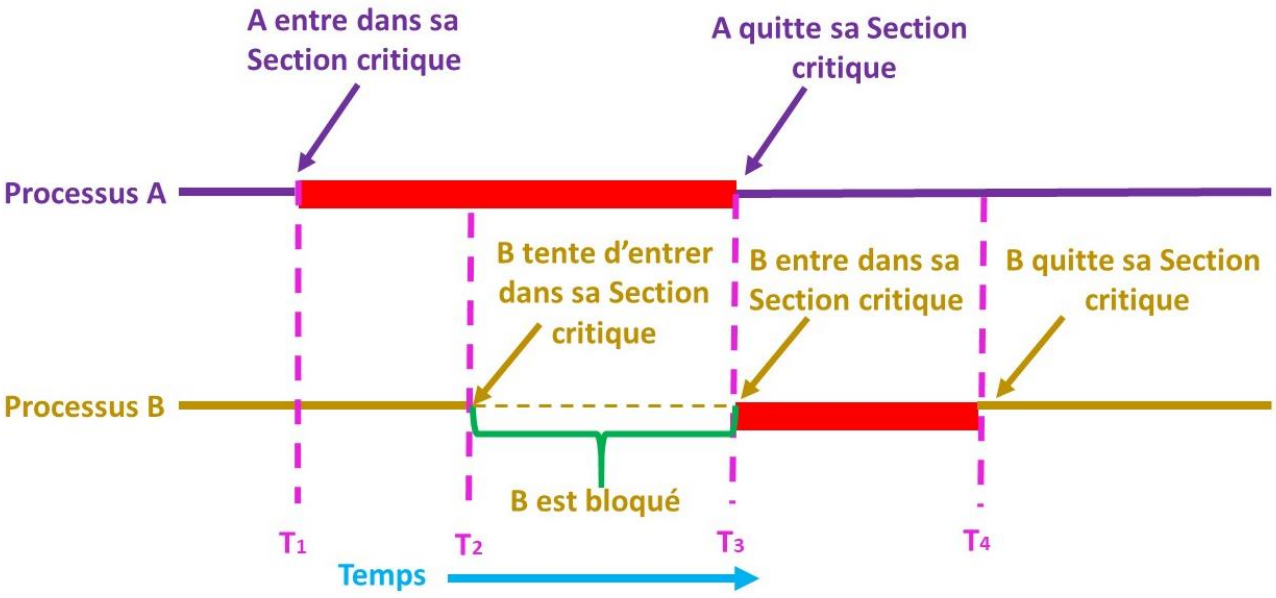


Figure VII. 4: L'exclusion mutuelle.

- **L'exclusion mutuelle :** Si un processus (thread) est dans une section critique, aucun autre processus (thread) ne peut interférer.
- **Progrès :** Si aucun processus n'est en cours d'exécution dans sa section critique et qu'il existe des processus souhaitant entrer dans leur section critique, alors le choix du processus qui entrera dans la section critique ne peut être différé indéfiniment.
- **Attente limitée :** Une limite doit exister sur le nombre de fois que les autres processus sont autorisés à entrer dans leurs sections critiques après qu'un processus ait fait une demande pour entrer dans sa section critique.

VII.4.3 Implémentation de l'exclusion mutuelle :

L'exclusion mutuelle peut être implémenter de différentes façons :

1. Désactivation des interruptions ;
2. Variable de verrou ;
3. Alternance stricte ;
4. Solution de Peterson ;
5. Instruction atomique.

VII.4.3.1 Désactivation des interruptions :

La désactivation des interruptions est une technique utilisée pour garantir l'exclusion mutuelle en empêchant les interruptions pendant l'exécution d'une section critique.

- **Fonctionnement :**

- **Désactivation des Interruptions** :
  - ✓ Un processus désactive toutes les interruptions avant d'entrer dans sa section critique.
  - ✓ Cela empêche le système d'interrompre le processus pour exécuter un autre processus ou gestionnaire d'interruptions.
- **Exécution de la Section Critique** :
  - ✓ Le processus effectue les opérations critiques sans risque d'être interrompu.
- **Réactivation des Interruptions** :
  - ✓ Une fois la section critique terminée, le processus réactive les interruptions, permettant au système de gérer les autres processus et interruptions en attente.

- **Avantages :**

- **Simplicité** : Facile à comprendre et à implémenter.
- **Efficacité** : Garantit une exclusion mutuelle parfaite pendant la désactivation des interruptions.

- **Inconvénients :**

- **Impact sur la Réactivité** : La désactivation des interruptions peut affecter la réactivité du système.
- **Utilisation Limitée** : Appropriée pour les sections critiques très courtes uniquement.
- **Non Applicable dans les Systèmes Multicœurs** : Inefficace dans les systèmes multicœurs, car désactiver les interruptions sur un cœur n'empêche pas les autres cœurs d'accéder à la section critique.

VII.4.3.2 Variable de verrou :

La variable de verrou (ou lock variable) est une méthode simple utilisée pour garantir l'exclusion mutuelle en contrôlant l'accès à une section critique.

- **Fonctionnement :**

- **Déclaration de la Variable de Verrou** : Une variable partagée, souvent appelée lock ou mutex, est utilisée pour indiquer si la section critique est occupée.
- **Acquisition du Verrou** : Avant d'entrer dans la section critique, un processus vérifie la valeur de la variable de verrou. Si la variable indique que la section critique est libre, le processus la modifie pour indiquer qu'elle est maintenant occupée.
- **Exécution de la Section Critique** : Le processus exécute les opérations dans la section critique.
- **Libération du Verrou** : Une fois la section critique terminée, le processus modifie la variable de verrou pour indiquer que la section critique est libre.

- **Avantages :**

- **Simplicité** : La méthode est facile à comprendre et à implémenter.
- **Contrôle Direct** : Elle offre un contrôle direct sur l'accès à la section critique.

- **Inconvénients :**

- **Attente Active** : Les processus peuvent consommer beaucoup de ressources en attendant activement que le verrou soit libéré.
- **Risque de Concurrence** : La vérification et la modification de la variable de verrou doivent être atomiques pour éviter les conditions de course. Sinon, deux processus pourraient simultanément vérifier et modifier la variable, ce qui pourrait conduire à une défaillance de l'exclusion mutuelle.
- **Problèmes de Performance** : L'attente active peut entraîner une baisse des performances, surtout si la section critique est fréquemment occupée.

VII.4.3.3 Alternance stricte :

L'alternance stricte est une méthode d'exclusion mutuelle utilisée pour garantir qu'aucuns deux processus ne peuvent accéder à leur section critique en même temps.

- **Fonctionnement :**

- **Initialisation d'un Tour** : Un indicateur de tour, souvent une variable partagée appelée `**turn**`, est utilisé pour indiquer quel processus a le droit d'entrer dans la section critique.
- **Attente Active** : Chaque processus vérifie la valeur de `**turn**`. Si la valeur correspond à son identifiant, il peut entrer dans la section critique. Sinon, il attend que son tour vienne.
- **Changement de Tour** : Une fois qu'un processus quitte sa section critique, il met à jour la variable `**turn**` pour permettre à l'autre processus d'entrer dans sa section critique.

- **Exemple** : Supposons deux processus, **P0** et **P1** :

- La variable partagée `**turn**` peut avoir les valeurs **0** ou **1**.
- Si `**turn**` est **0**, c'est le tour de **P0** d'entrer dans la section critique.
- Si `**turn**` est **1**, c'est le tour de **P1** d'entrer dans la section critique.

- **Avantages :**

- **Simplicité** : L'algorithme est simple à comprendre et à implémenter.
- **Garantie d'Exclusion Mutuelle** : Il assure qu'un seul processus peut être dans la section critique à la fois.

- **Inconvénients :**

- **Attente Active** : Les processus peuvent consommer beaucoup de ressources en attendant activement leur tour.
- **Manque de Flexibilité** : La stricte alternance peut entraîner une inefficacité si l'un des processus n'a pas besoin d'accéder fréquemment à la section critique.
- **Problème de Famine** : Dans certains scénarios, un processus peut être constamment bloqué si l'autre processus reste longtemps dans sa section critique.

VIII.4.3.4 Solution de Peterson :

La solution de Peterson est un algorithme classique qui garantit l'exclusion mutuelle entre deux processus. Elle est conçue pour fonctionner dans un environnement de mémoire partagée et utilise des variables partagées pour coordonner l'accès aux sections critiques **[20]**.

- **Fonctionnement :**

- **Variables Partagées :**
  - ✓ **`flag[2]`:** Un tableau de deux éléments où **`flag[i]`** est True si le processus **Pi** souhaite entrer dans sa section critique.
  - ✓ **`turn` :** Une variable partagée qui indique quel processus a la priorité en cas de conflit.
- **Entrée dans la Section Critique :**
  - ✓ Chaque processus signale son intention d'entrer dans la section critique en réglant son drapeau **`flag`** à **`True`**.
  - ✓ Il attribue ensuite la variable **`turn`** à l'autre processus, indiquant qu'il donne la priorité à l'autre processus.
  - ✓ Ensuite, il attend que l'autre processus ne souhaite pas entrer (**`flag[j] == False`**) ou que ce soit son tour (**`turn == i`**).
- **Sortie de la Section Critique :**
  - ✓ Le processus réinitialise son drapeau **`flag`** à **`False`**, indiquant qu'il a quitté la section critique.

- **Propriétés :**

- **Exclusion Mutuelle** : Un seul processus peut être dans la section critique à la fois.
- **Absence de Famine** : Chaque processus a une chance équitable d'entrer dans la section critique.
- **Attente Bornée** : Aucun processus n'attend indéfiniment pour entrer dans la section critique.

- **Avantages :**

- **Simplicité** : L'algorithme est relativement simple à comprendre et à implémenter.
- **Efficacité** : Il n'utilise que quelques variables partagées et des opérations simples.

- **Inconvénients :**

- **Limitations à Deux Processus** : La solution de Peterson est conçue pour fonctionner uniquement avec deux processus. Pour plus de deux processus, des algorithmes plus complexes comme les sémaphores ou les algorithmes de **Ricart-Agrawala** sont nécessaires.
- **Dépendance aux Opérations Atomiques** : La solution suppose que les opérations sur les variables partagées sont atomiques, ce qui peut ne pas être le cas sur certains systèmes ou architectures matérielles.

VII.4.3.5 Instruction atomique :

Les instructions atomiques sont des opérations qui se déroulent de manière indivisible, garantissant qu'aucune autre opération ne peut interférer pendant leur exécution. Elles sont essentielles pour assurer l'exclusion mutuelle et la synchronisation dans les systèmes **multithread** et **multiprocesseur**.

- **Fonctionnement :**

Les instructions atomiques sont exécutées par le matériel en une seule opération non interruptible. Voici quelques exemples courants d'instructions atomiques :

- **Test-and-Set** : Vérifie la valeur d'une variable et la modifie si elle satisfait à une certaine condition.
- **Compare-and-Swap (CAS)** : Compare la valeur d'une variable à une valeur attendue et, si elles sont égales, la remplace par une nouvelle valeur.
- **Fetch-and-Add**: Récupère la valeur actuelle d'une variable et ajoute une certaine valeur à cette variable de manière atomique.

- **Avantages**

- **Exclusion Mutuelle Garantie** : Les opérations atomiques garantissent que les sections critiques sont protégées sans besoin de verrous complexes.
- **Performance** : Elles sont souvent plus performantes que les autres mécanismes de synchronisation car elles évitent l'attente active et les changements de contexte.
- **Simplicité** : Facilite la conception de structures de données concurrentes sans verrous.

- **Inconvénients**

- **Support Matériel Nécessaire** : Les instructions atomiques dépendent du support matériel. Toutes les architectures ne les supportent pas de manière égale.
- **Complexité** : Bien que les instructions soient simples, leur utilisation correcte peut être complexe, surtout dans les systèmes avec de nombreux threads ou processus.
- **Portée Limitée** : Les instructions atomiques sont généralement limitées à des opérations simples sur des types de données de base (par exemple, entiers, pointeurs).

VII.4.4 Le Sommeil et l'Activation :

Au lieu que le processus boucle en attendant la sortie d'un autre processus de sa section critique, utilisation du **CPU**, il est préférable de bloquer **(sleep/block)** et de ne le réveiller qu'à la libération de sa section critique **(Wake up).**

VII.4.4.1 Sommeil (Sleep) :

- Lorsqu'un processus ou un thread ne peut pas entrer dans sa section critique parce qu'une autre unité d'exécution l'occupe, il est mis en sommeil.
- Au lieu de boucler activement en attendant que la section critique soit libre (ce qui consomme des ressources **CPU**), le processus est mis en attente jusqu'à ce qu'une condition soit remplie.

VII.4.4.2 Activation (Wake) :

- Une fois que le processus ou le thread en cours dans la section critique la quitte, il réveille les processus ou threads en sommeil.
- Cela permet à un ou plusieurs processus ou threads en attente de tenter à nouveau d'entrer dans la section critique.

- **Avantages :**

- **Efficacité** : Évite l'attente active, économisant ainsi les ressources **CPU**.
- **Réactivité** : Réduit le temps pendant lequel les processus ou threads sont inactifs mais consomment des ressources.

- **Inconvénients :**

- **Complexité de l'Implémentation :** La mise en œuvre peut être plus complexe, nécessitant une gestion des files d'attente et des opérations de sommeil et de réveil.
- **Latence** : Il peut y avoir une légère latence associée à la mise en sommeil et au réveil des threads.

VII.4.5 Verrous Mutex (Mutex Locks) :

Les verrous mutex (de **"mutual exclusion"**) sont des mécanismes utilisés pour assurer l'exclusion mutuelle dans des environnements multi-thread ou multi-processus. Ils garantissent qu'un seul thread ou processus peut accéder à une ressource critique à la fois, empêchant ainsi les conditions de course et assurant la cohérence des données.

- **Fonctionnement :**

- **Acquisition du Verrou** :
  - ✓ Avant d'entrer dans une section critique, un thread doit acquérir le verrou.
  - ✓ Si le verrou est déjà détenu par un autre thread, le thread en attente est bloqué jusqu'à ce que le verrou soit libéré.
- **Exécution de la Section Critique** :
  - ✓ Une fois le verrou acquis, le thread peut exécuter la section critique en toute sécurité.
- **Libération du Verrou** :
  - ✓ Après avoir terminé les opérations dans la section critique, le thread libère le verrou.
  - ✓ Un des threads en attente peut alors acquérir le verrou et entrer dans la section critique.

- **Avantages :**

- **Exclusion Mutuelle Garantie** : Assure qu'un seul thread peut accéder à la section critique à la fois.
- **Simplicité d'Utilisation :** Les **API** des verrous mutex sont simples et faciles à utiliser.

- **Réduction des Risques de Concurrence** : Protège efficacement contre les conditions de course et les incohérences de données.

- **Inconvénients :**

- **Blocage** : Si un thread ne libère pas le verrou correctement (par exemple, en cas de panne), cela peut entraîner un blocage.
- **Inversion de Priorité** : Peut entraîner des problèmes d'inversion de priorité, où un thread de haute priorité attend un thread de basse priorité.
- **Surcoût en Performance** : L'acquisition et la libération des verrous ajoutent un certain surcoût en termes de performance, surtout dans les environnements hautement concurrentiels.

Note :

Verrous Ré-entrants :

Les verrous ré-entrants **(reentrant locks)** sont une variante des mutex qui permettent à un thread qui détient déjà le verrou de l'acquérir à nouveau sans être bloqué. Cela est utile pour éviter les blocages dans les scénarios où une méthode synchronisée appelle une autre méthode synchronisée.

VII.4.6 Les sémaphores :

Les sémaphores sont des outils de synchronisation avancés utilisés pour la gestion de l'accès concurrentiel aux ressources partagées dans les systèmes multi-thread et multi-processus. Ils ont été introduits par **Edsger Dijkstra** en **1965** et sont largement utilisés pour coordonner l'accès à des ressources critiques.

- **Fonctionnement :**

- **Type de Sémaphore :**
  - ✓ **Sémaphore Binaire :** Peut avoir deux états (**0** ou **1**), utilisé pour la synchronisation entre deux processus ou threads, similaire à un verrou mutex.
  - ✓ **Sémaphore Compteur (Général) :** Peut avoir une valeur entière positive quelconque, utilisé pour gérer un ensemble de ressources.
- **Opérations Fondamentales :**
  - ✓ **P (Wait) :** Diminue la valeur du sémaphore. Si la valeur devient négative, le processus ou thread est mis en attente jusqu'à ce que la ressource devienne disponible.
  - ✓ **V (Signal) :** Augmente la valeur du sémaphore. Si des processus ou threads sont en attente (en raison d'une opération P précédente), l'un d'eux est réveillé pour utiliser la ressource.

- **Avantages :**

- **Flexibilité :** Les sémaphores peuvent être utilisés pour des situations plus complexes que les verrous simples, comme la gestion de plusieurs ressources.

- **Évitement d'Attente Active :** Contrairement aux méthodes basées sur l'attente active, les sémaphores mettent les threads en attente de manière efficace, économisant ainsi les ressources du processeur.
- **Synchronisation Avancée :** Permet une synchronisation plus fine et plus complexe entre plusieurs processus ou threads.

- **Inconvénients :**

- **Complexité :** Les sémaphores nécessitent une gestion prudente pour éviter les problèmes de synchronisation tels que les blocages ou les incohérences.
- **Difficulté de Débogage :** En raison de leur utilisation dans des scénarios complexes, les erreurs de synchronisation peuvent être difficiles à identifier et à corriger.

- **Sémaphores et Gestion de Ressources :**

Les sémaphores sont souvent utilisés pour gérer des ressources telles que des buffers, des fichiers, des espaces mémoire partagés, etc. Ils permettent de contrôler l'accès à ces ressources de manière à éviter les conflits et à garantir leur intégrité.

VII.5 Conclusion :

La communication interprocessus et la synchronisation sont des éléments essentiels pour le bon fonctionnement des systèmes multi-processus et multi-thread. Ils permettent aux processus de coopérer efficacement tout en évitant les problèmes de concurrence, partagent les ressources de manière sécurisée et évitent les conflits et les conditions de course. Une gestion efficace de l'IPC et de la synchronisation est cruciale pour la performance, la stabilité et la sécurité des systèmes d'exploitation.

CHAPITRE-VIII- GESTION DE MEMOIRE

VIII.1 Introduction :

La gestion de la mémoire dans les systèmes d'exploitation est le processus d'allocation, de suivi et d'organisation de la mémoire d'un ordinateur pour optimiser les performances et l'efficacité. Elle permet de gérer la mémoire vive (**RAM**) et la mémoire virtuelle, garantissant que les programmes disposent des ressources nécessaires tout en évitant les conflits et la fragmentation. Les techniques incluent la pagination, la segmentation et l'utilisation de la mémoire cache pour accélérer l'accès aux données fréquemment utilisées. Une gestion efficace de la mémoire est cruciale pour le fonctionnement stable et rapide des applications et du système global.

VIII.2 Objectifs Spécifiques :

- **Connaître les Types de Mémoire et leur Rôle**
  - Identifier et décrire les différents types de mémoire (mémoire primaire, mémoire secondaire, cache).
  - Comprendre les caractéristiques et les utilisations spécifiques de chaque type de mémoire.
- **Maîtriser les Techniques de Gestion de la Mémoire**
  - Expliquer le partitionnement fixe et dynamique et leurs avantages/inconvénients.
  - Comprendre et mettre en œuvre la pagination et la segmentation.
  - Décrire le concept de mémoire virtuelle et son importance dans les systèmes modernes.
- **Appliquer les Algorithmes d'Allocation de Mémoire**
  - Comparer et appliquer les différentes stratégies d'allocation contiguë **(First-Fit, Best-Fit, Worst-Fit).**
  - Mettre en œuvre des techniques d'allocation non-contiguë **(pagination, segmentation).**

VIII.3 Définition :

La mémoire est un composant essentiel des systèmes informatiques, responsable du stockage et de la récupération des données nécessaires à l'exécution des programmes. Elle permet de conserver temporairement ou de manière permanente des informations que le processeur peut accéder pour effectuer des opérations.

VIII.3.1 Utilisations des mémoires :

- Lors de démarrage d'un ordinateur, un programme qui contenant des instructions de démarrage préliminaire s'exécute ce programme est préenregistré dans une mémoire appelé **mémoire morte ROM.**

- Les données et les programmes sont enregistrées d'une manière permanente dans des mémoires appelée **mémoire de masse** (auxiliaires) comme les disques durs, flash disques, CD ...etc.
- Pour être exécutés, les programmes et les données sont chargés dans une mémoire appelée **RAM** (charge d'une manière temporelle pour exécuter un programme qui est stocké d'une manière durable).
- Lors de l'exécution d'une instruction le microprocesseur peut utiliser des **registres** pour un stockage temporel.
- Pour diminuer le temps d'accès à **RAM**. L'ordinateur utilise une mémoire plus rapide et plus proche du processeur, cette mémoire contient des copies des données enregistrées dans la **RAM** les plus fréquemment utilisés, elle est appelée **mémoire cache.**
- Pour diminuer le temps d'accès aux mémoires de masse, le processeur utilise une mémoire d'appui (équivalente à la mémoire Cache).

VIII.3.2 Types de Mémoire :

Il existe plusieurs types de mémoire utilisés dans les systèmes informatiques, chacun ayant des caractéristiques et des fonctions spécifiques. Voici un aperçu des principaux types de mémoire **[21]** :

a. Mémoire Primaire (RAM - Random Access Memory) :

Mémoire à accès direct et volatile utilisée pour stocker les données et les instructions en cours d'exécution. Utilisée pour le stockage temporaire des programmes en cours d'exécution et des données actives.

- **Exemples** :
  - **DRAM** (Dynamic **RAM**)
  - **SRAM** (Static **RAM**)

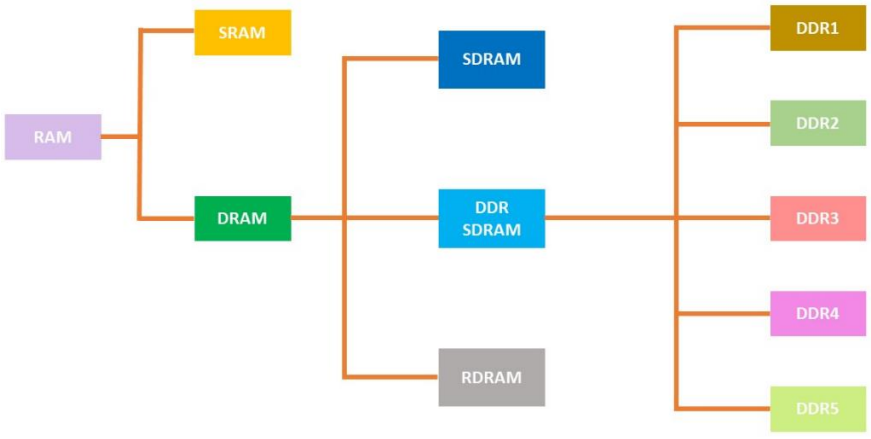


Figure VIII. 1: Type de RAM.

b. Mémoire Secondaire :

Mémoire non volatile (les données sont conservées même lorsque l'alimentation est coupée). Accès plus lent que la **RAM**, utilisée pour stocker les données et les programmes de manière permanente.

- **Exemples** :
  - Disques durs (**HDD**)
  - Disques **SSD** (Solid State Drive)
  - **CD/DVD**
  - Clés **USB**

c. Mémoire Cache :

Mémoire très rapide volatile située entre le processeur et la **RAM** pour améliorer les performances en stockant des copies des données fréquemment utilisées.

- **Types** :
  - **L1 Cache :** Intégré directement dans le processeur.
  - **L2 Cache :** Peut être intégré dans le processeur ou séparé.
  - **L3 Cache :** Souvent partagé entre plusieurs cœurs de processeurs.

d. Mémoire Virtuelle

Technique permettant de simuler une plus grande quantité de mémoire **RAM** en utilisant de la mémoire secondaire (disque dur ou **SSD**) pour étendre l'espace d'adresse accessible aux programmes.

- **Fonctionnement** :
  - Utilise la pagination pour diviser la mémoire en pages et gérer leur emplacement entre la **RAM** et le disque.
  - Permet d'exécuter des programmes plus grands que la mémoire physique disponible.
  - Peut entraîner des temps d'accès plus longs si les données doivent être échangées entre la **RAM** et le disque.

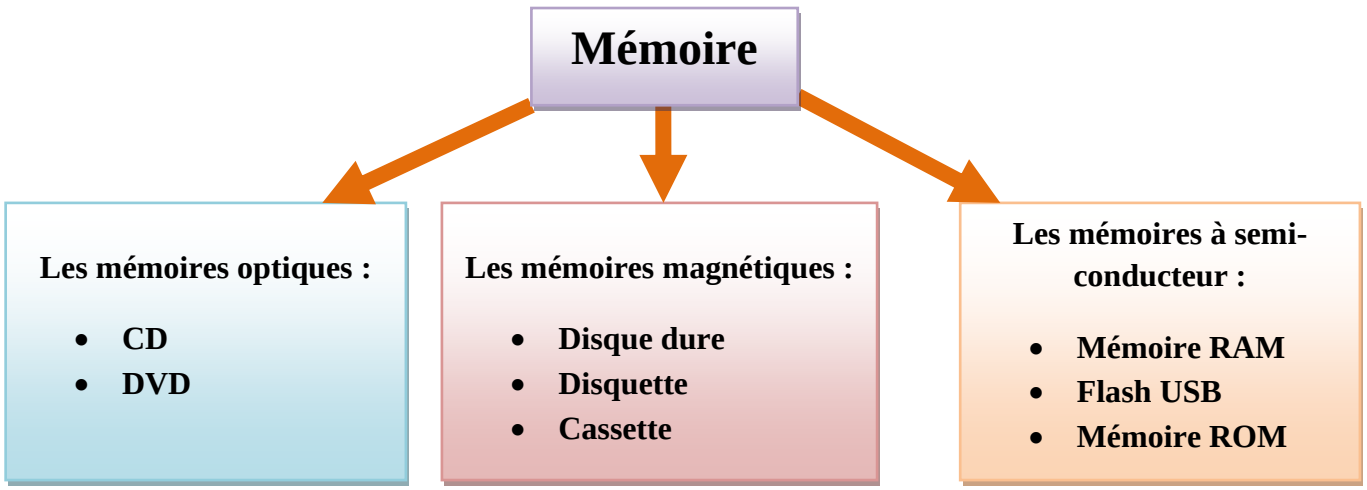


Figure VIII. 2: Type de mémoire.

VIII.3.3 Les mémoires à semi-conducteur :

Sont des dispositifs de stockage d'informations numériques. Ils sont utilisés dans tous les appareils, y compris les microprocesseurs et également dans la mise en œuvre de circuit logique programmables.

Alors une mémoire est un circuit à **semi-conducteurs** qui permet d'enregistrer de conserver et de restituer des informations, il y a donc :

- **Écriture :** lorsqu'on enregistre des informations en mémoire.
- **Lecture :** lorsqu'on récupère ces informations précédemment enregistrées.

VIII.3.3.1 RAM - Random Access Memory:

Dispositif capable de **stocker** puis de **restituer** une information. L'unité d'information (bit, octet, etc.) s'appelle « **point de mémoire** » ou « **cellule** ».

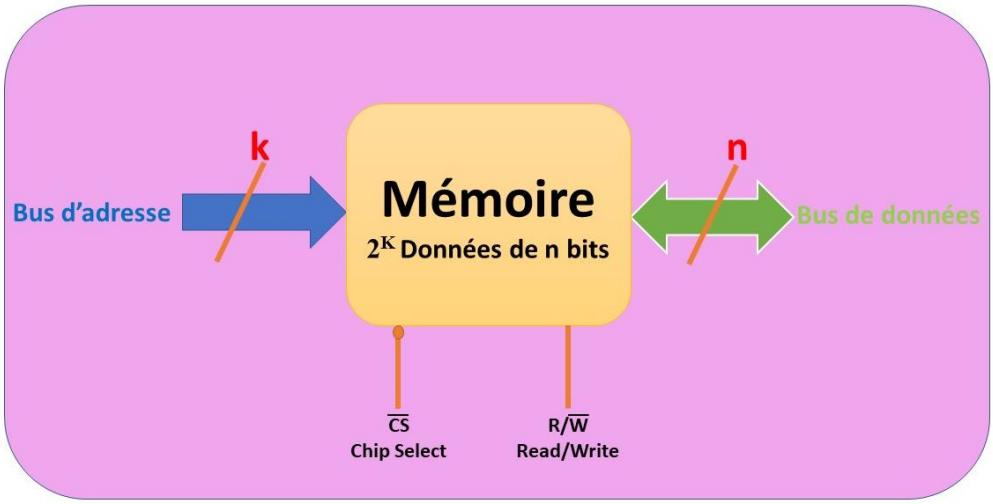


Figure VIII. 3: Schéma fonctionnel d'une mémoire.

- La commande **R/W (Read/Write) :** C'est la commande qui détermine la direction de transfert des données (lecture ou écriture).
- La commande **CS (Chip Select) :** C'est l'entrée qui permet d'activer ou désactiver la mémoire.
- **n** lignes d'entrée pour envoyer l'information à écrire dans le mot mémoire ;
- **n** lignes de sortie pour recevoir l'information à lire du mot mémoire ;
- **k** lignes d'adresses pour accéder aux mot mémoire ;
- Un décodeur **K→2^K** qui permet de sélectionner le mot mémoire référencé par l'adresse introduite.

VIII.3.3.2 ROM (Read-Only Memory):

Est un type de mémoire non volatile utilisée principalement pour stocker des données permanentes et des instructions nécessaires au démarrage et au fonctionnement de base des ordinateurs.

- **Utilisations de la ROM :**
  - ✓ **BIOS/UEFI** : Stocke les instructions nécessaires au démarrage initial de l'ordinateur et à la configuration matérielle.
  - ✓ **Micrologiciels** : Contient le logiciel de bas niveau pour contrôler le matériel, comme dans les périphériques et les contrôleurs intégrés.
  - ✓ **Dispositifs embarqués** : Utilisée dans des appareils comme les téléphones mobiles, les microcontrôleurs et les équipements industriels pour stocker le logiciel intégré.
- **Types de ROM :**
  - ✓ **ROM Masquée** : Programmée lors de la fabrication, ne peut pas être modifiée.
  - ✓ **PROM (Programmable ROM)** : Peut être programmée une seule fois après fabrication à l'aide d'un équipement spécial.
  - ✓ **EPROM (Erasable Programmable ROM)** : Peut être effacée par exposition aux **UV** et reprogrammée.
  - ✓ **EEPROM (Electrically Erasable Programmable ROM)** : Peut être effacée et reprogrammée électriquement, couramment utilisée pour stocker les micrologiciels et les paramètres de configuration.

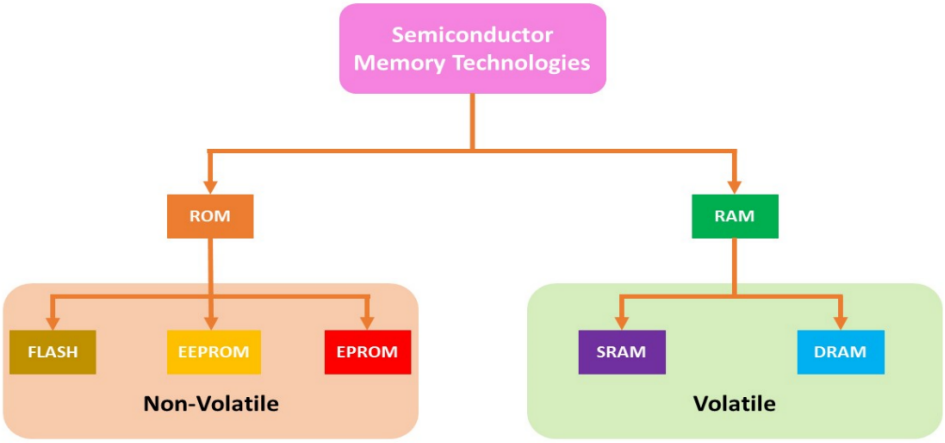


Figure VIII. 4: Type de mémoires à semi-conducteur.

VIII.3.4 Fonction de la Mémoire dans un Système Informatique :

- **Stockage Temporaire** : Conserve les données et les instructions nécessaires aux programmes en cours d'exécution.
- **Stockage Permanent** : Sauvegarde les données et les applications pour une utilisation à long terme.
- **Amélioration des Performances** : La mémoire cache et la mémoire virtuelle permettent d'accélérer l'accès aux données et de gérer efficacement les ressources mémoire.
- **Isolation et Protection** : Assure que chaque processus dispose de son propre espace mémoire isolé, protégeant les données et les instructions contre les accès non autorisés par d'autres processus.

VIII.4 Caractéristiques :

Une mémoire est caractérisée par plusieurs facteurs qui répondent à des besoins spécifiques en matière de stockage et de traitement des données **[22]** :

a. **Capacité :** Elle représente la quantité d'informations stockables, elle est exprimée en bit, octet, kilo-octet, …etc.

Tableau VIII. 1: Capacité de mémoire (quantité d'informations stockables).

Terminologie	Abréviation	Valeur		
Octet	~~Ɵ~~	8 bits	2^3 bits	
Kilo- Octet	Ko	1024 ~~Ɵ~~	2^{10} ~~Ɵ~~	10^3 ~~Ɵ~~
Méga- Octet	Mo	1024 Ko	2^{20} ~~Ɵ~~	10^6 ~~Ɵ~~
Giga- Octet	Go	1024 Mo	2^{30} ~~Ɵ~~	10^9 ~~Ɵ~~
Téra- Octet	To	1024 Go	2^{40} ~~Ɵ~~	10^{12} ~~Ɵ~~
Péta- Octet	Po	1024 To	2^{50} ~~Ɵ~~	10^{15} ~~Ɵ~~
Exa- Octet	Eo	1024 Po	2^{60} ~~Ɵ~~	10^{18} ~~Ɵ~~
Zetta- Octet	Zo	1024 Eo	2^{70} ~~Ɵ~~	10^{21} ~~Ɵ~~
Yotta- Octet	Yo	1024 Zo	2^{80} ~~Ɵ~~	10^{24} ~~Ɵ~~

b. **Temps d'accès :** C'est le temps nécessaire à une opération de Lecture/Ecriture.

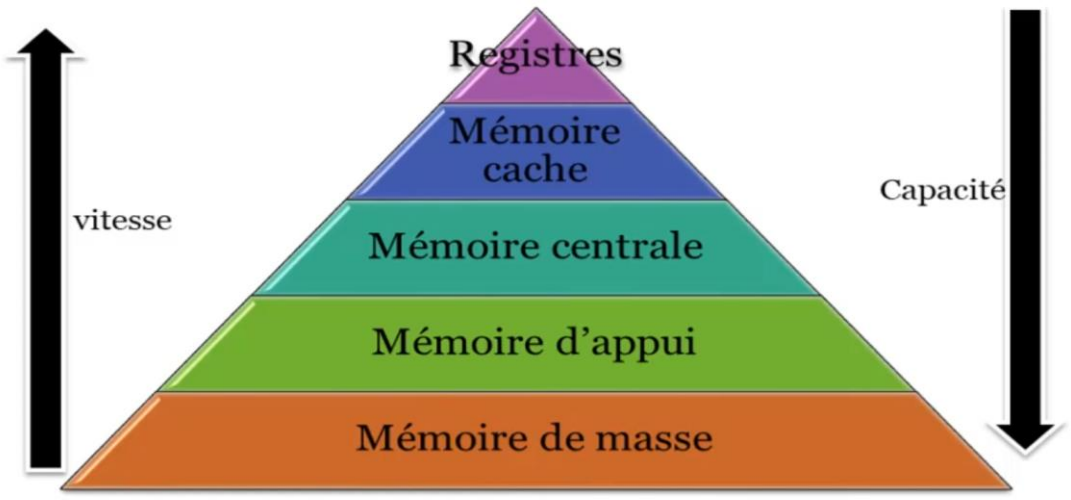


Figure VIII. 5: La hiérarchie mémoire.

- En termes de **temps d'accès** les registre sont les mémoires les plus rapide.
- En termes de **capacité de stockage** en remarque les mémoires de masse sont les plus puisant.

c. **Débit :** C'est la quantité d'information lues/écrites par unité de temps. **Example** : Mo/s.

d. **Durée de mémorisation :** C'est la durée pendant laquelle une information est conservée en mémoire, elle peut être :

- Quasi-permanente (**ROM**, Disque Dure) ;
- Temporaire (**DRAM**) ;
- Volatile (en fonction de présent du courant électrique).

e. Mode d'accès : Les modes d'accès de la mémoire désignent les différentes méthodes par lesquelles les données peuvent être lues ou écrites dans divers types de mémoire. Voici une description des principaux modes d'accès de la mémoire.

Tableau VIII. 2: Modes d'accès de la mémoire.

Mode d'Accès	Description	Exemples de Mémoire
Accès Séquentiel	Les données sont lues ou écrites dans un ordre préétabli, une après l'autre.	Bandes magnétiques, certaines mémoires flash.
Accès Direct (ou Aléatoire)	Les données peuvent être lues ou écrites dans n'importe quel ordre, sans suivre une séquence spécifique.	**RAM**, disques durs, **SSD.**
Accès par Page	Les données sont organisées en pages et l'accès se fait par page entière.	Mémoire virtuelle, certains types de **DRAM.**
Accès par Bloc	Les données sont organisées en blocs et l'accès se fait par bloc entier.	Disques durs, **SSD.**
Accès Associatif	Les données sont accédées par une clé ou un identifiant plutôt que par une adresse.	Cache, **TLB** (Translation Lookaside Buffer).

f. Localisation : La localisation de la mémoire dans un système informatique se réfère à l'endroit où chaque type de mémoire est physiquement ou logiquement situé par rapport au processeur (**CPU**).

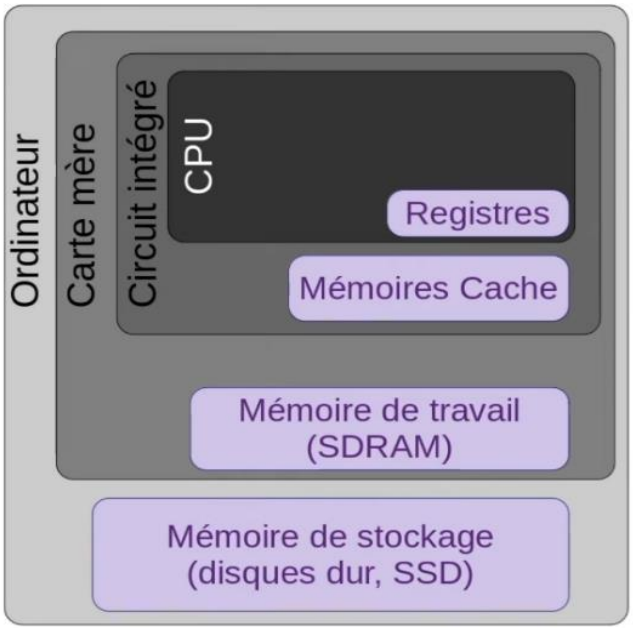


Figure VIII. 6: Localisation de mémoire.

Voici un tableau détaillant les types de mémoire et leur localisation :

Tableau VIII. 3: Localisation de la mémoire dans un système informatique.

Niveau de Mémoire	Type de Mémoire	Localisation	Description
1	Registres du **CPU.**	Directement sur le **CPU.**	Petites unités de mémoire intégrées dans le processeur, utilisées pour les opérations immédiates.
2	Cache **L1.**	Sur le **CPU.**	Mémoire cache très rapide, intégrée dans le processeur, pour stocker les données et instructions fréquemment utilisées.
3	Cache **L2.**	Sur le **CPU** (parfois à l'extérieur).	Mémoire cache rapide, souvent plus grande que L1, utilisée pour stocker des données et instructions à accès légèrement moins fréquent.
4	Cache **L3.**	Entre les cœurs du **CPU** (partagée).	Mémoire cache plus grande et légèrement moins rapide, partagée entre plusieurs cœurs du processeur.
5	**RAM** (Random Access Memory).	Modules de mémoire sur la carte mère.	Mémoire principale volatile, utilisée pour stocker les programmes en cours d'exécution et les données actives.
6	Mémoire Flash.	Connectée via des ports **USB** ou des bus.	Utilisée pour le stockage portable (clés **USB**, cartes **SD**) ou comme stockage principal (**SSD**).
7	Disque Dure (**HDD**).	Connectés via des interfaces (**SATA, NVMe**).	Stockage secondaire non volatile, utilisé pour stocker des systèmes d'exploitation, des applications et des fichiers de données.
8	Mémoire virtuelle.	Implémentée via le disque dur/**SSD.**	Technique utilisant le stockage secondaire pour simuler une extension de la mémoire physique disponible.

VIII.5 Gestion de la mémoire centrale :

Dans le cadre d'un système multiprogrammé, plusieurs programmes sont chargés en mémoire centrale **RAM** (ce sont des processus), ce qui engendre trois grands problèmes à résoudre :

- Définir un espace d'adressage séparés pour chaque processus.
- Alloués de la place en mémoire **RAM** pour le code et les données d'un processus.

- Protéger cet espace d'adressent vis-à-vis des accès des autres processus.

Plusieurs autres concepts sont utilisés pour améliorer la gestion de la mémoire comme **la mémoire virtuelle**, ne charger que les parties utiles à un instant donné de chaque processus ou qu'on appelle le **Swapping**.

La chaine de production de programme produit un programme exécutable **relogeable**, dont toutes les adresses sont calculées à partir d'une origine fixée à **0**.

L'ensemble des adresses du programme exécutable, générées par le processeur au moment de l'exécution des instructions, est appelé **espace d'adressage logique** ou **espace d'adresse virtuelles.**

L'ensemble des adresses physique réellement occupées par le programme exécutable suite au chargement en **RAM** est appelé **espace adressage physique.**

Le **Memory Management Unit MMU** est le dispositif matériel qui réalise la **conversion** des **adresses logique** en **adresse physiques**.

Pour faire la transition entre **l'adresse réelle** et **l'adresse logique**, **le Memory Management Unit MMU** peut être conçu de façon très simple : il suffit de d'additionner l'adresse de base du programme (sa première adresse dans la mémoire physique) à son adresse dans le programme (adresse virtuelle).

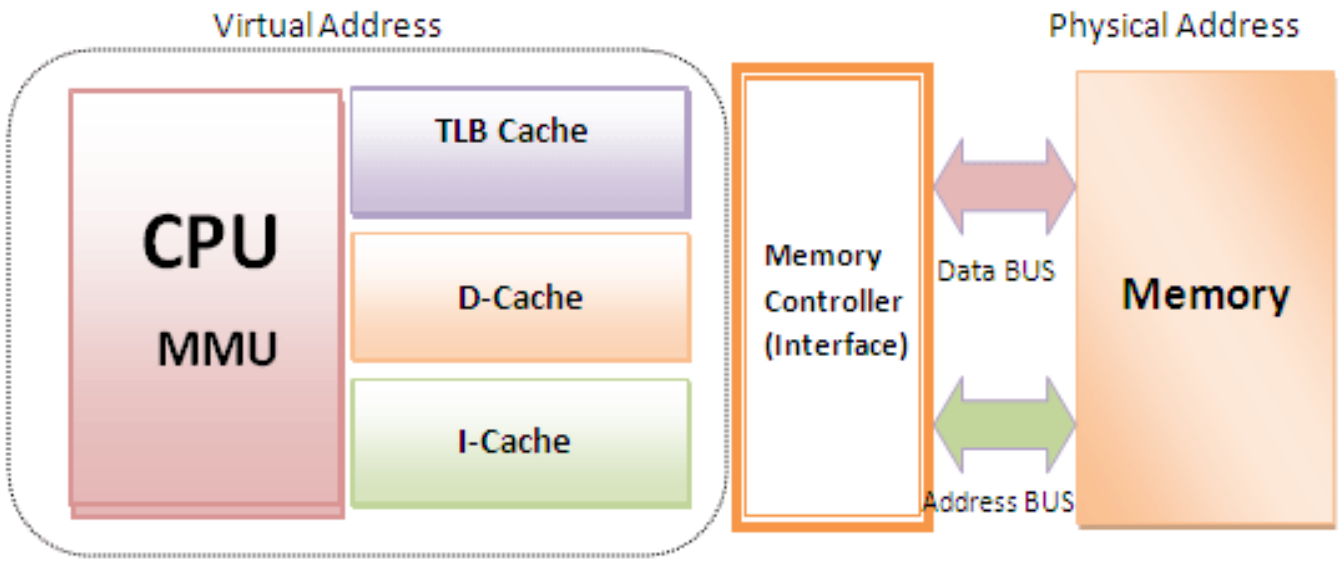


Figure VIII. 7: Memory management unit **MMU**.

La **RAM** est divisée en deux grandes zones, une pour le **système d'exploitation** et l'autre pour **les programmes des utilisateurs**.

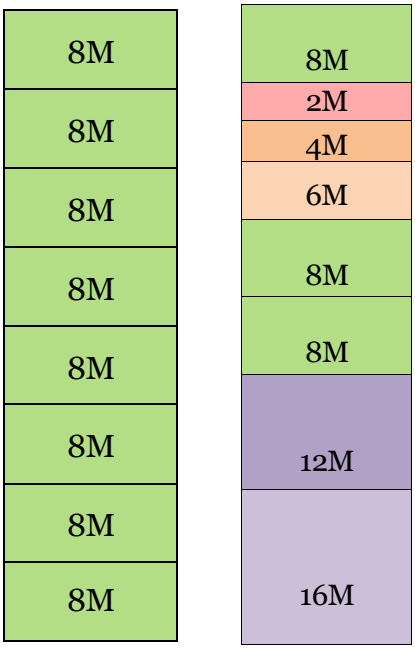
Lorsque l'utilisateur demande l'exécution d'un programme, le chargeur doit trouver une place libre suffisamment grande dans la zone des programmes utilisateurs, une fois l'exécution du programme terminé, la rame doit être libérée.

Il existe plusieurs stratégies **d'allocation/libération** de la mémoire centrale **RAM** :

- **Allocation contiguë :**
  - Partitions fixes.
  - Partitions dynamiques.
  - Allocation par bourgeonnement.
- **Allocation non contiguë :**
  - Pagination.
  - Segmentation.

VIII.6 Allocation contiguë :

VIII.6.1 Partitions fixes :



- La mémoire principale est divisée en zones de taille fixe qui ne se chevauchent pas.
- Chaque zone et appelée partition et peut être de taille égale ou non.
- La taille des partitions et déterminée au départ et ne change pas durant l'exécution des programmes.
- Un algorithme de placement est en charge de trouver une partition libre pour charger les processus en mémoire.
- Tout processus dont la taille est inférieure ou égale à la taille des partitions peut être chargé en mémoire.
- Si toutes les partitions sont occupées, le **SE** peut déplacer un processus d'une partition vers la mémoire secondaire.
- Si un programme est assez grand pour tenir dans une seule partition, il doit être découpées en partie nommée des **« overley »**.
- Dans ce modèle, c'est le programmeur qui gère le découpage du programme.
  - **Avantage :** Simplicité de mise en œuvre, gestion facile des processus.
  - **Inconvénient** : Fragmentation interne (de l'espace mémoire peut être gaspillé si un processus ne remplit pas entièrement sa partition).

L'allocation contiguë de partitions à taille fixe crée de la **fragmentation interne**. Entre chaque partition de taille fixe, un peu de mémoire est perdue parce que le programme contenu dans la partition n'a pas nécessairement la même taille que la partition.

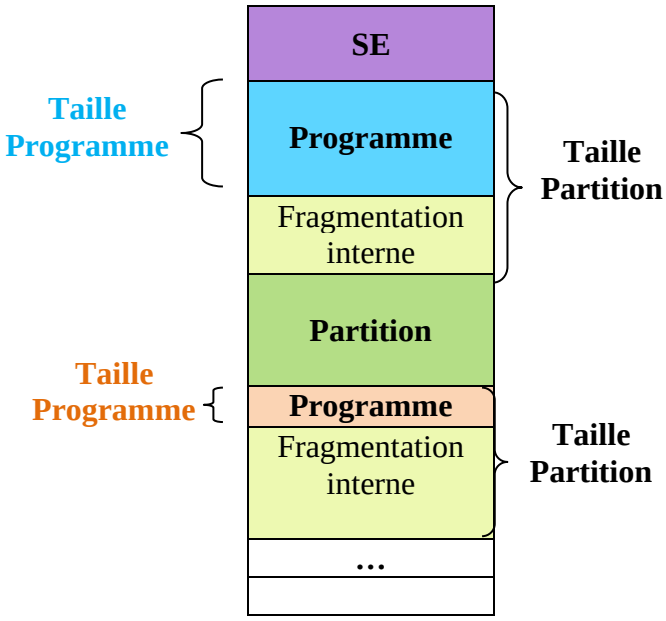


Figure VIII. 8: Fragmentation interne.

VIII.6.2 Partitions dynamiques :

- L'idée est de créer des partitions dynamiquement selon la taille des processus.
- Les partitions sont créées à l'exécution des processus. Elles sont allouées de la taille correspondant aux besoins du processus.

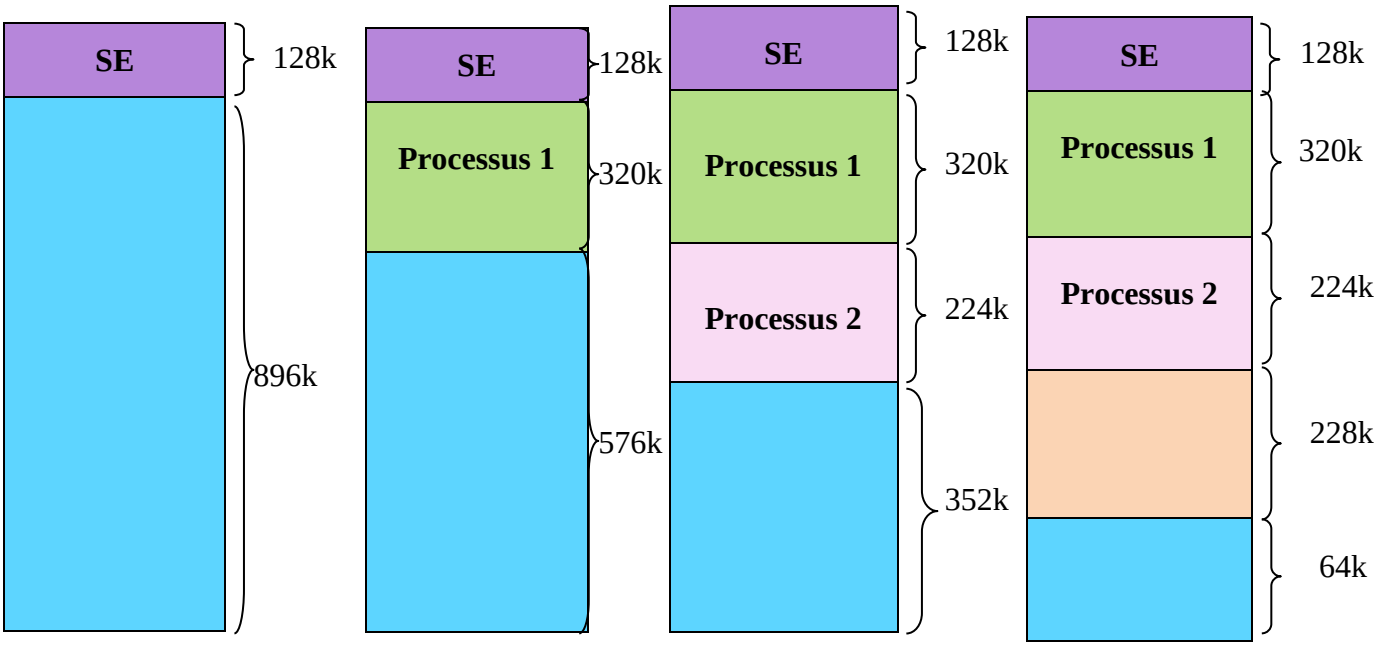


Figure VIII. 9: Partitions dynamiques.

- **Avantage :**

- Réduction de la fragmentation interne.

- **Inconvénient :**

- Fragmentation externe (des petits blocs de mémoire inutilisés peuvent se former entre les partitions actives), gestion plus complexe des partitions libres et occupées.

L'allocation contiguë de partitions à taille variable crée de la **fragmentation externe**. Lorsqu'un programme est retiré de la mémoire, il laisse un bloc de mémoire libre. Il est possible, par la suite, que ce bloc soit rempli partiellement par un processus de taille moindre (dans une nouvelle partition). Il reste alors de la mémoire libre à l'extérieur des partitions.

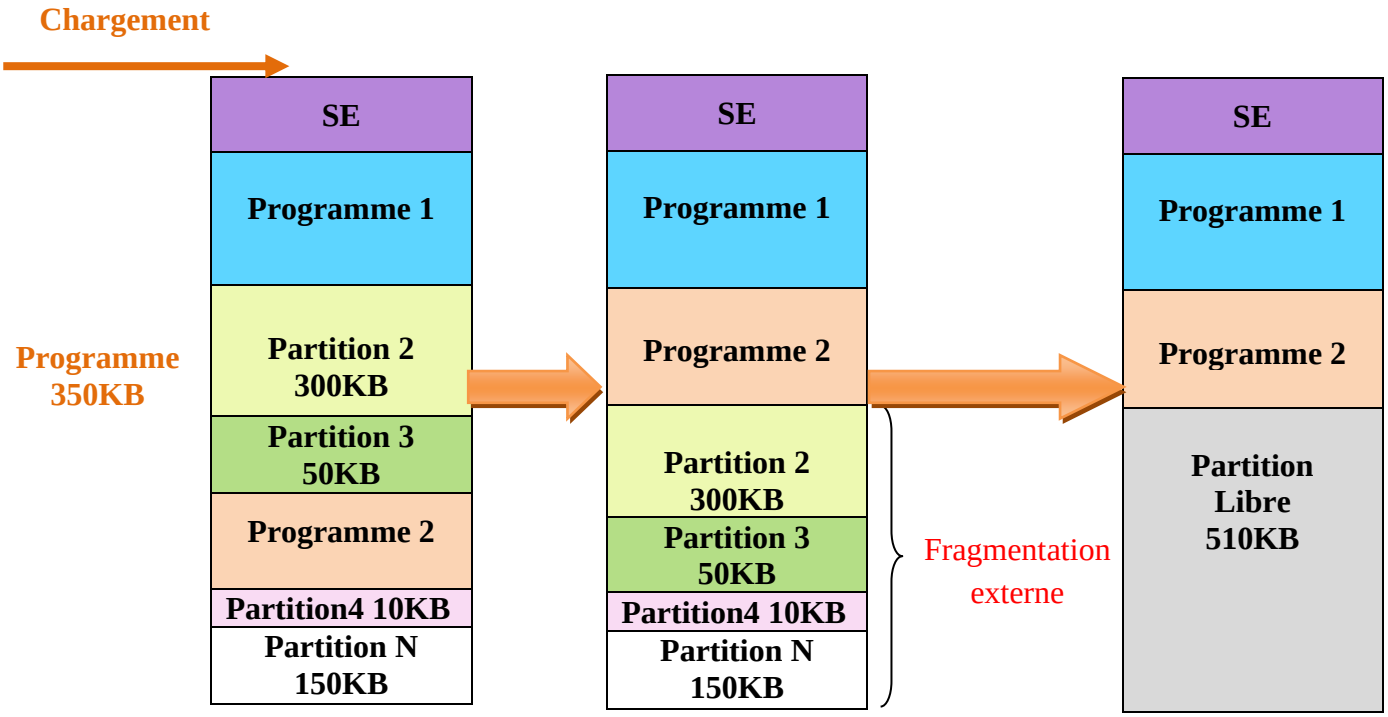


Figure VIII. 10: Fragmentation externe.

- **Fragmentation externe :** C'est des partitions de la mémoire qui ne peuvent accueillir un programme de taille supérieure, alors que la somme de toutes ou certaines partitions dépasse la taille de programme (Le problème que aucune partition ne peut loger le programme **350 KB** alors que l'espace physique de la **RAM** dépasse largement cette taille).
  Pour réduire l'impact de la fragmentation externe, le compactage consiste à déplacer les programmes en une seule partie et rassembler les partitions libres résidus en une seule partition plus grande. Cette technique est utilisée avec les programmes **relogeables**.

Il existe plusieurs algorithmes afin de déterminer l'emplacement d'un programme en mémoire (allocation contiguë). Le but de tous ces algorithmes est de maximiser l'espace mémoire occupé et minimiser la fragmentation :

- **First Fit :** Le programme est mis dans le premier bloc de mémoire qui peut contenir ce dernier à partir du début de la mémoire.
- **Best Fit :** Le programme est mis dans le bloc de mémoire le plus petit pouvant contenir.
- **Next Fit :** Le programme est mis dans le premier bloc de mémoire suivant le dernier alloué pouvant contenir ce processus.
- **Worse Fit :** Le programme est mis dans le bloc de mémoire le plus grand.

Example : Selon chacun des trois algorithmes, **First Fit, Best Fit** et **Next Fit**, ou sera placé un bloc de **16K**.

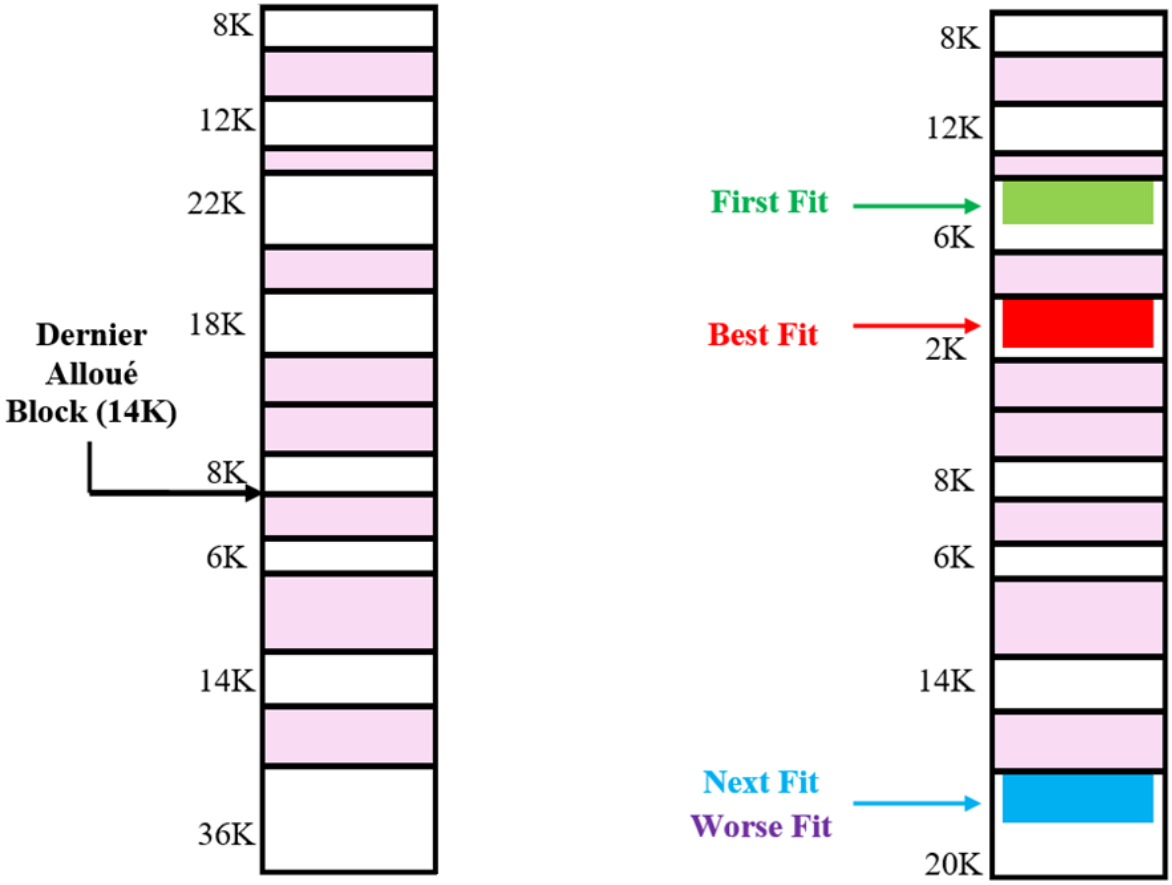


Figure VIII. 11: Emplacement d'un programme en mémoire (allocation contiguë).

VIII.6.3 Allocation par bourgeonnement :

- Partition fixe limite le nombre de processus.
- Partition dynamique fragmentation externe est difficile à maintenir.
- La méthode connue par système de bourgeons constitue un bon compromis pour contrecarrer les problèmes liés aux méthodes précédentes.

L'algorithme d'allocations par bourgeonnement fonctions avec des partitions variables qui peuvent au besoin être subdivisé ou visionner les unes avec les autres.

La technique d'allocation par bourgeonnement **(buddy system)** divise la mémoire en blocs de taille variable mais toujours en puissance de deux.

- **Fonctionnement** :

- La mémoire est initialement divisée en un seul grand bloc.
- Lorsqu'une demande de mémoire arrive, ce bloc est divisé en deux blocs plus petits **(bourgeonnement)** de taille égale, jusqu'à ce que la taille appropriée soit trouvée.
- Les blocs peuvent être fusionnés **(réassemblés)** lorsqu'ils sont libérés, s'ils sont des **"buddies"** (paires adjacentes de blocs de même taille).

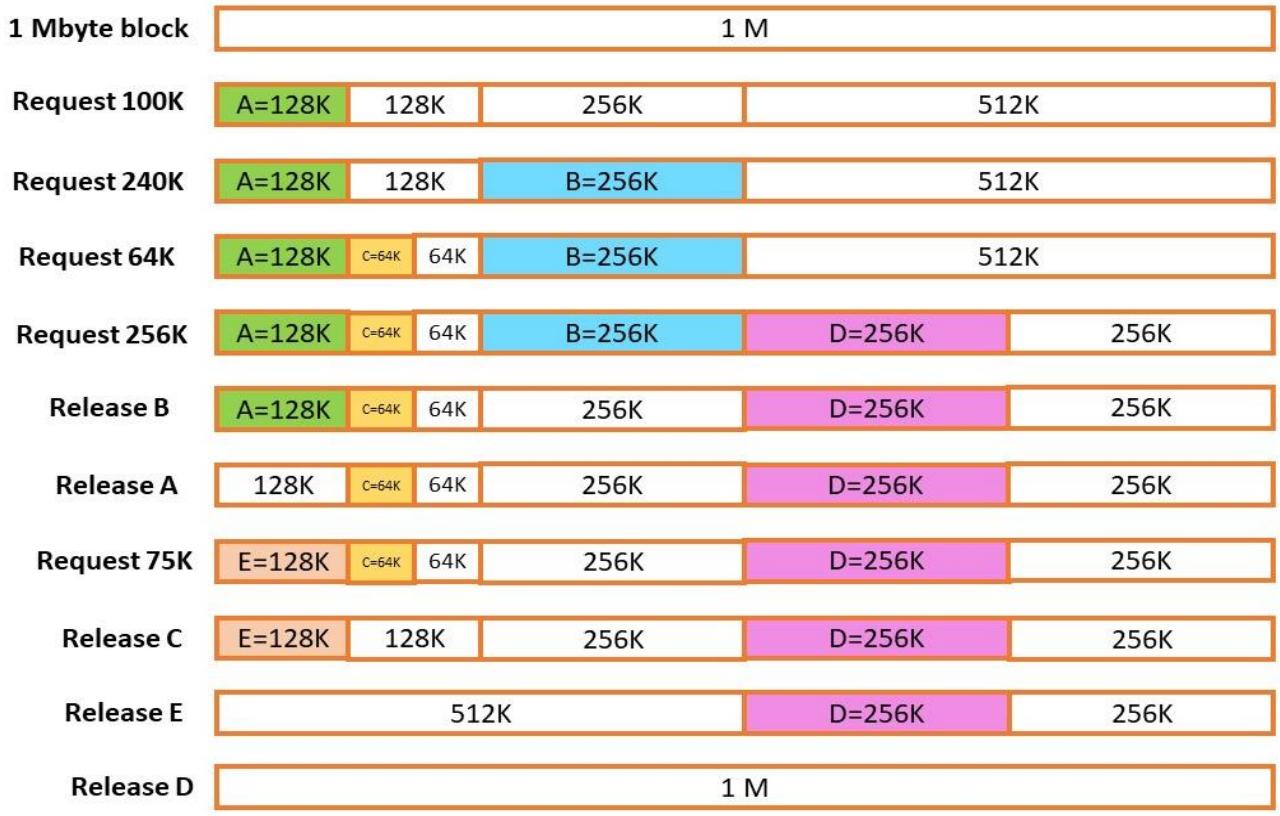


Figure VIII. 12: Allocation par bourgeonnement.

- La fragmentation moyenne est d'environ **25%**.
- Chaque bloc et au moins **50%** occupé.
- Les programmes ne sont pas déplacés en mémoire c'est une gestion simple et dynamique.

VIII.6.4 Protection des partitions dont l'allocation :

Chaque partition allouée à un programme constitue un espace d'adressage protégé pour le processus correspondant qui doit être gardée des accès mémoire des autres processus.

Dans l'allocation contiguë, cette opération est réalisée en vérifiant que chaque adresse logique est strictement inférieure à la taille de la partition alloués au processus concernés.

La protection des partitions fait référence à la prévention de l'accès non autorisé ou incorrect à la mémoire allouée à un processus par d'autres processus. Cela assure que chaque processus fonctionne dans son propre espace mémoire, sans interférer avec les autres **[25]**.

- **Méthodes de Protection** :

a. Registres de Base et de Limite :

- **Registre de Base** : Contient l'adresse de départ de la partition allouée à un processus.
- **Registre de Limite** : Contient la taille de la partition allouée.

Ces registres assurent que chaque accès mémoire par le processus est vérifié pour se situer dans les limites de la partition.

b. Unités de Gestion de Mémoire (MMU) :

Utilisation de la MMU pour traduire les adresses virtuelles en adresses physiques, ajoutant une couche de protection et d'isolation.

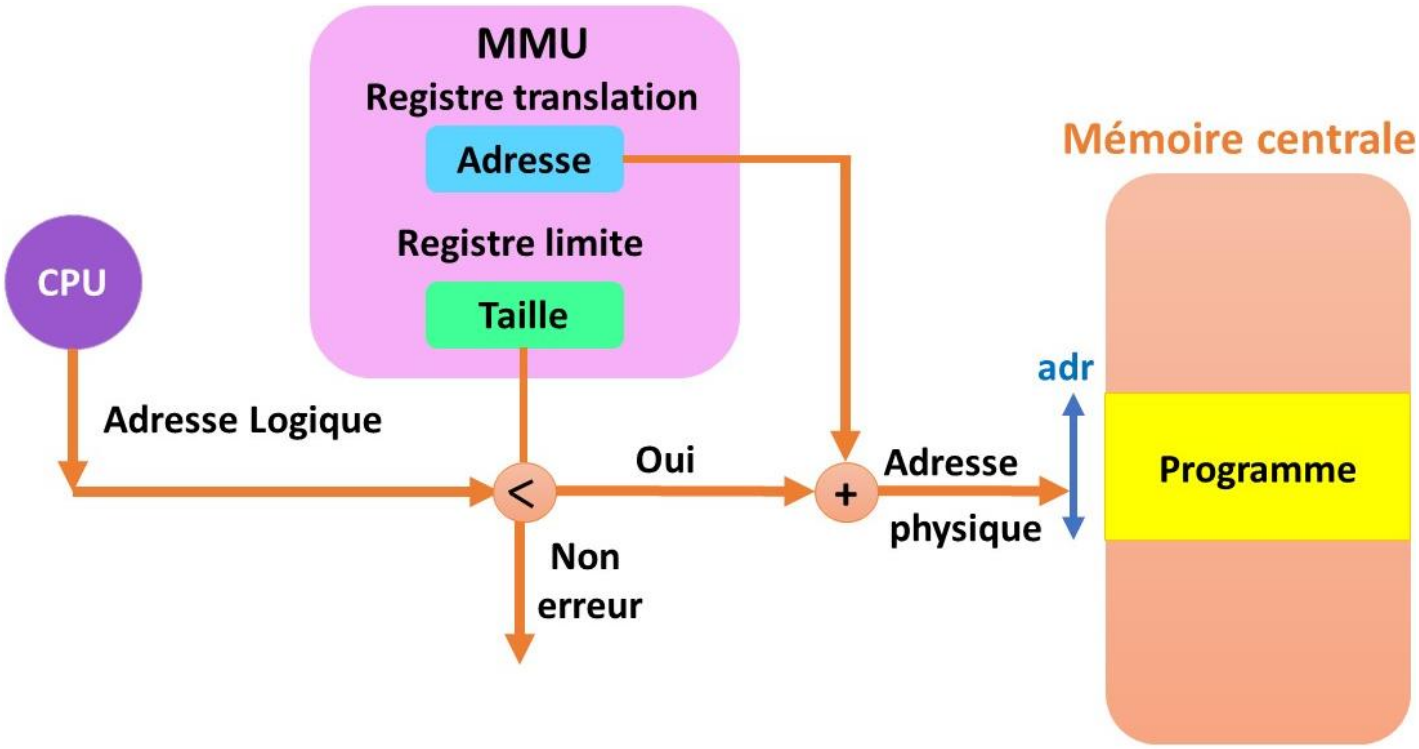


Figure VIII. 13: Protection des partitions d'allocation contiguë avec **MMU**.

VIII.7 Allocation non contiguë :

Les méthodes avec partition fixe créer de la fragmentation car les partitions sont relativement grandes.

L'allocation non contiguës utilise alors des petites partitions pour éviter la défragmentation :

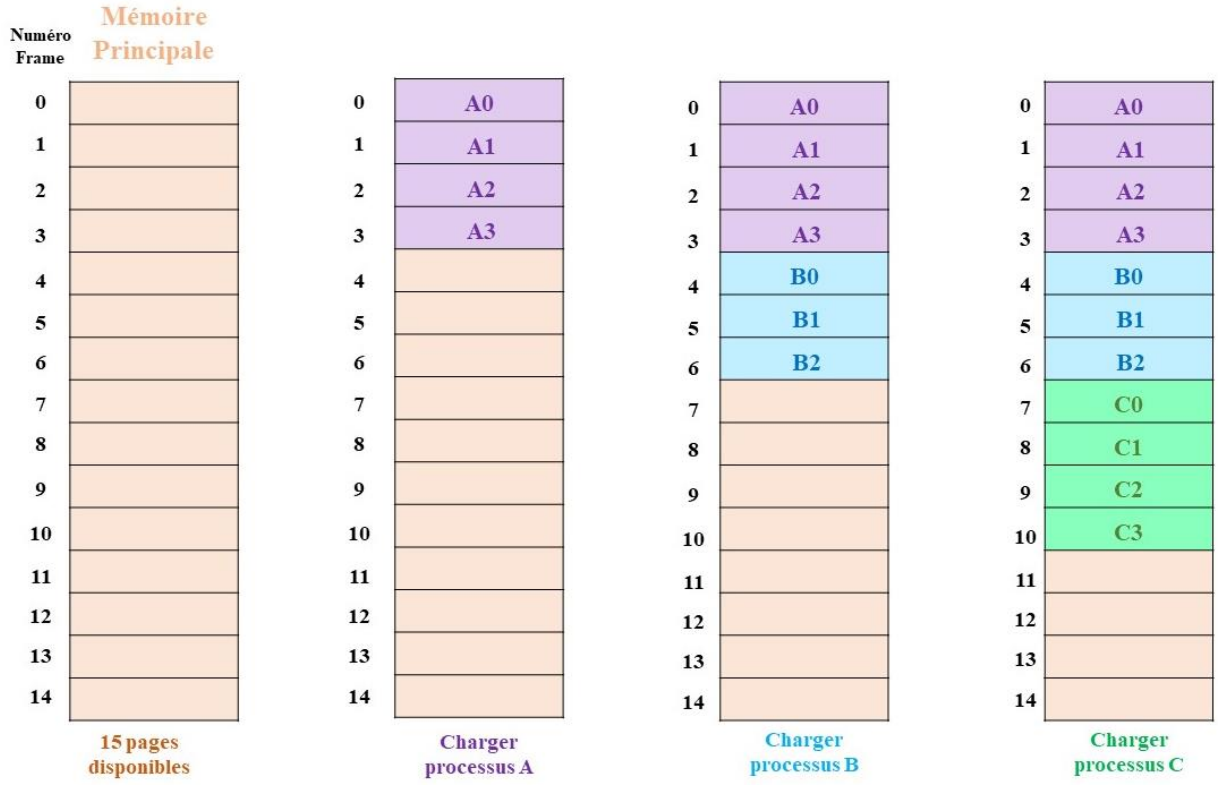
- Diviser le **programme** en plusieurs petites **pages**.
- Divisé la **mémoire** en plusieurs petites **frame**.
- Mettre une **page** par **frame**.
- Les pages d'un même processus ne sont pas forcément dans des frames adjacents de la mémoire.
  - Quand le programme est divisé en page de taille **fixe** on parle de **pagination.**
  - Quand les pages sont de taille **variable** on parle de **segmentation.**

VIII.7.1 Pagination :

La pagination est une technique de gestion de la mémoire qui permet de diviser l'espace d'adressage logique d'un processus en blocs de taille fixe appelés pages, et la mémoire physique en blocs de taille égale appelés cadres (ou frames) **[24]**.

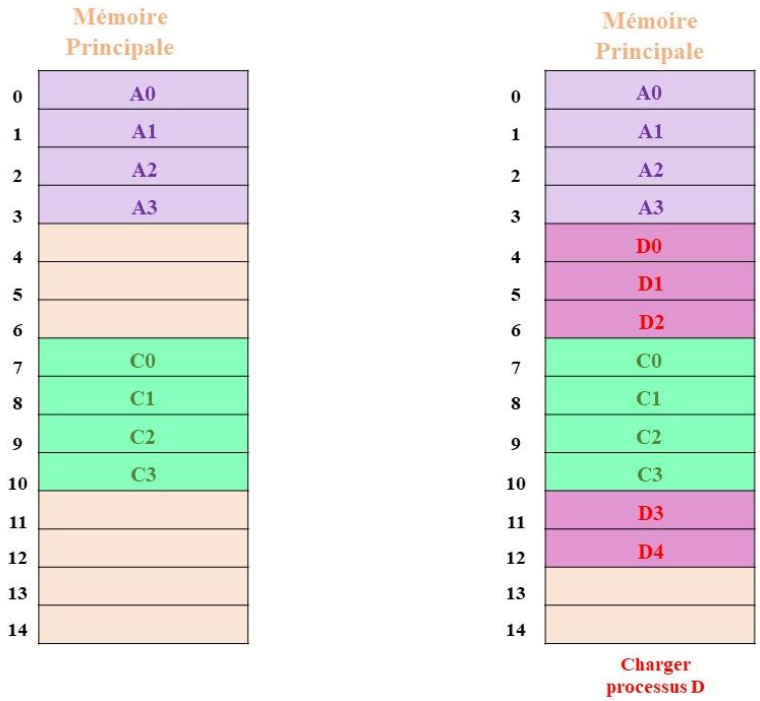
- **Fonctionnement** :
  - Un processus est divisé en pages de taille fixe.
  - Les pages sont mappées aux cadres de page dans la mémoire physique.
  - Une table des pages maintient les informations de mappage pour chaque processus.

Soit trois processus **A**, **B** et **C** qui occupent respectivement **4**, **3** et **4** pages :



Pour charger un nouveau processus **D** de **5** pages :

- On ne peut pas mettre le processus **D** de **5** pages dans la mémoire, parce que nous avons que **4** pages libre, donc on swappe le processus **B** (de **3** pages).
- On mettre les **3** premiers pages de processus **D** sur la place libérée par le processus **B** swappé, et les autres pages à la fin après le processus **C** (les pages de processus **D** ne sont pas adjacentes).



- **Avantages** :
  - Élimination de la fragmentation externe.
  - Permet une utilisation plus efficace de la mémoire.
- **Inconvénients** :
  - Peut entraîner une fragmentation interne si la taille des pages ne correspond pas exactement à la taille des segments de mémoire utilisés par le processus.
  - Complexité accrue de la gestion de la table des pages.

VIII.7.1.1 Table de page :

Pour connaître les frames libres de celles utilisées, pour ne pas écraser l'information d'un processus d'un autre, une table de page est conservée en mémoire.

Tableau VIII. 4: Table de frame.

Frames	Page	Processus
frame 1	-1 (libre)	
frame 2	1	P_1
frame 3	3	P_1
frame 4	1	P_2
frame 5	-1	
frame 6	2	P_1
frame 7	4	P_1
frame 8	2	P_2

La table des pages est une structure de données essentielle utilisée dans les systèmes d'exploitation pour gérer la mémoire virtuelle. Elle sert à maintenir la correspondance entre les adresses logiques (virtuelles) utilisées par un programme et les adresses physiques réelles dans la mémoire.

La table des pages est une structure de données utilisée dans les systèmes de gestion de mémoire paginée pour traduire les adresses virtuelles générées par les processus en adresses physiques réelles. Chaque processus possède sa propre table des pages, qui contient des entrées correspondant aux pages de son espace d'adressage logique.

Pour pouvoir convertir l'adresse logique en adresse physique, chaque processus doit conserver sa propre table de page ainsi qu'une adresse paginée pour chaque octet.

Une adresse paginée est constituée de :

« **Numéro de page *p*** à quelle l'octet appartient, **déplacement *d*** relativement au début de la page *p* ».

- A partir de **numéro de page *p*** on peut trouver le frame dans la mémoire.
- A partir de **déplacement *d*** au début de la page, on va faire la translation pour trouver l'octet.

a. Structure de la Table des Pages :

Entrées de la Table des Pages : Chaque entrée dans la table des pages correspond à une page de l'espace d'adressage virtuel du processus. Une entrée typique contient :

- **Adresse de Cadre (Frame Address)** : L'adresse du cadre de mémoire physique où la page est stockée.
- **Bit de Présence (Present Bit)** : Indique si la page est actuellement en mémoire physique.

- **Bits de Protection** : Indiquent les permissions d'accès à la page (**lecture « r »**, **écriture « w »**, **exécution « x »**).
- **Bit de Modification (Dirty Bit)** : Indique si la page a été modifiée.
- **Bit de Référence (Accessed Bit)** : Indique si la page a été récemment accédée.

b. Fonctionnement de la Table des Pages :

- **Traduction d'Adresse :**

- Lorsque le processeur génère une adresse virtuelle, celle-ci est divisée en deux parties : le numéro de page et le décalage (offset).
- Le numéro de page est utilisé pour indexer la table des pages et trouver l'entrée correspondante.
- L'entrée de la table des pages fournit l'adresse de cadre de mémoire physique.
- Le décalage est ajouté à l'adresse de cadre pour obtenir l'adresse physique finale.

- **Gestion des Défauts de Page :**

- Si l'entrée de la table des pages indique que la page n'est pas présente en mémoire physique (bit de présence désactivé), un défaut de page est déclenché.
- Le système d'exploitation doit alors charger la page manquante depuis le stockage secondaire (disque) et mettre à jour la table des pages.

- **Optimisation via le TLB (Translation Lookaside Buffer) :**

- Le **TLB** est une petite mémoire cache qui stocke les traductions d'adresses les plus fréquemment utilisées pour accélérer le processus de traduction.
- Si une traduction d'adresse se trouve dans le **TLB**, elle peut être utilisée directement sans consulter la table des pages.

c. Types de Tables des Pages :

- **Table des Pages Uniniveau** :

- Une seule table contenant toutes les entrées de pages pour un processus.
- Simple mais peut devenir très grande pour des espaces d'adressage importants.

- **Table des Pages Multiniveau** :

- Utilise plusieurs niveaux de tables des pages pour réduire la taille de la mémoire nécessaire pour stocker la table.
- Chaque niveau de la table des pages pointe vers une autre table de pages.

- **Table des Pages Inversée** :

- Une seule table globale pour tout le système, avec une entrée pour chaque cadre de mémoire physique.
- Chaque entrée contient des informations sur la page logique stockée dans ce cadre.

Deux approches existent pour la réalisation de la table des pages :

- La table de pages est une **structure matérielle** réalisé grâce à des registres de la **MMU**.
  Un seul accès à la mémoire pour accéder à un octet dans la **RAM**.
  Par contre elle ne convient que pour des petites tables de pages.
- La table des pages est une **structure logicielle** placés en **RAM** elle-même elle est placée en mémoire
  L'adresse de cette table en mémoire du processus actif et placé dans le registre **PTBR** (page-table base registre) de la **MMU**.

Pour accélérer les accès à la **RAM**, le cache associatif **TLB** peut être utiliser. Il contient les couples ***<p, c>*** les plus récemment accédés :

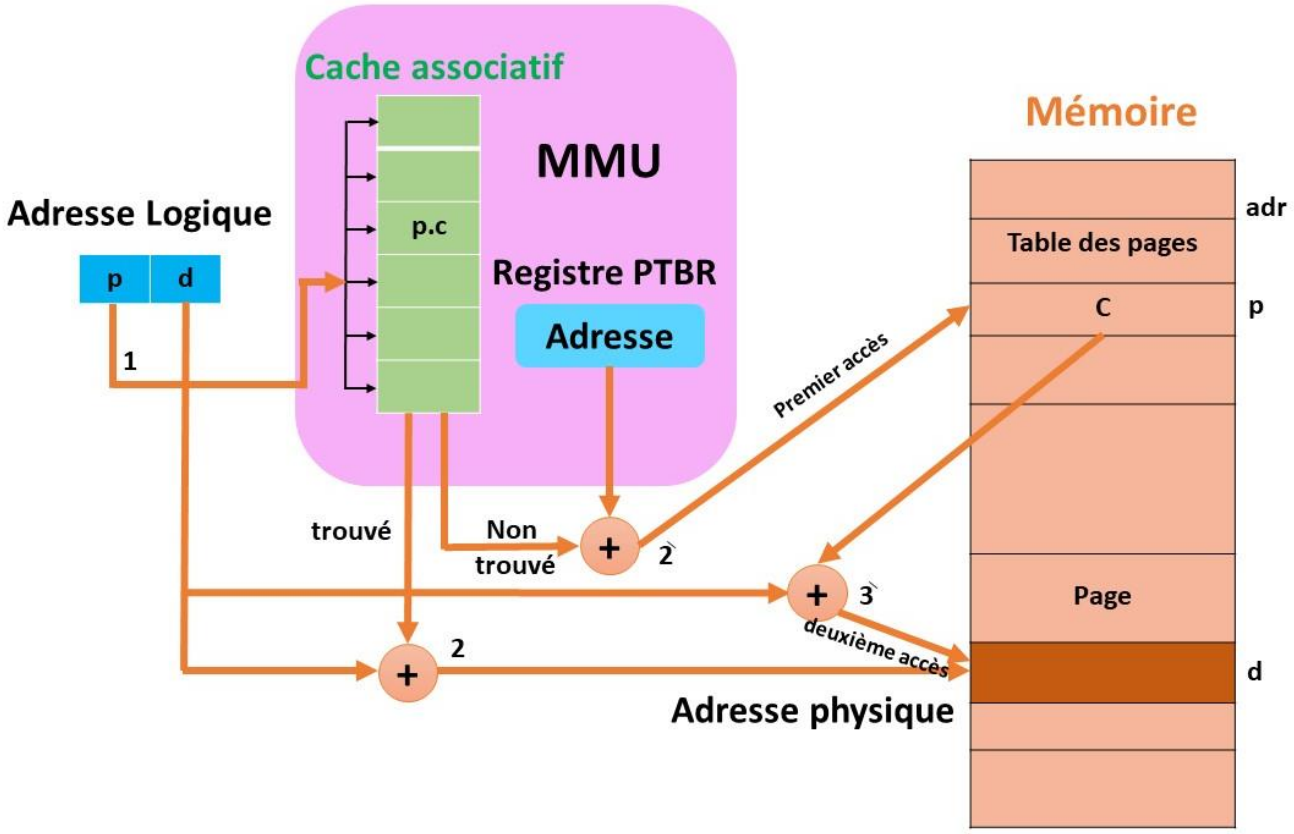


Figure VIII. 14: Cache associatif **TLB** (Translation Lookaside Buffer).

VIII.7.1.2 Défaut de page :

Pour que le **CPU** puisse savoir si une page est chargée en **RAM**, un bit de **validation V** est ajouté aux tables de pages (il est à 1 quand la page est chargée en **RAM**).

Le principe de la mémoire virtuelle consiste à ne chargée, à un instant donné, que les pages utiles, elle utilise une combinaison de mémoire vive (**RAM**) et de stockage secondaire (disque dur ou **SSD**) pour simuler un espace de mémoire continue et étendue pour les applications. Cela permet aux programmes de fonctionner comme s'ils disposaient de plus de mémoire que ce qui est physiquement installé.

La réalisation de la mémoire virtuelle engendre deux grandes questions :

- Que se passe-t-il si on veut accéder à une page non chargée à la mémoire ?
- Si la mémoire est pleine, quelle page décharger ?

Un défaut de page se produit lorsqu'un processus veut accéder à une page non présente en mémoire centrale.

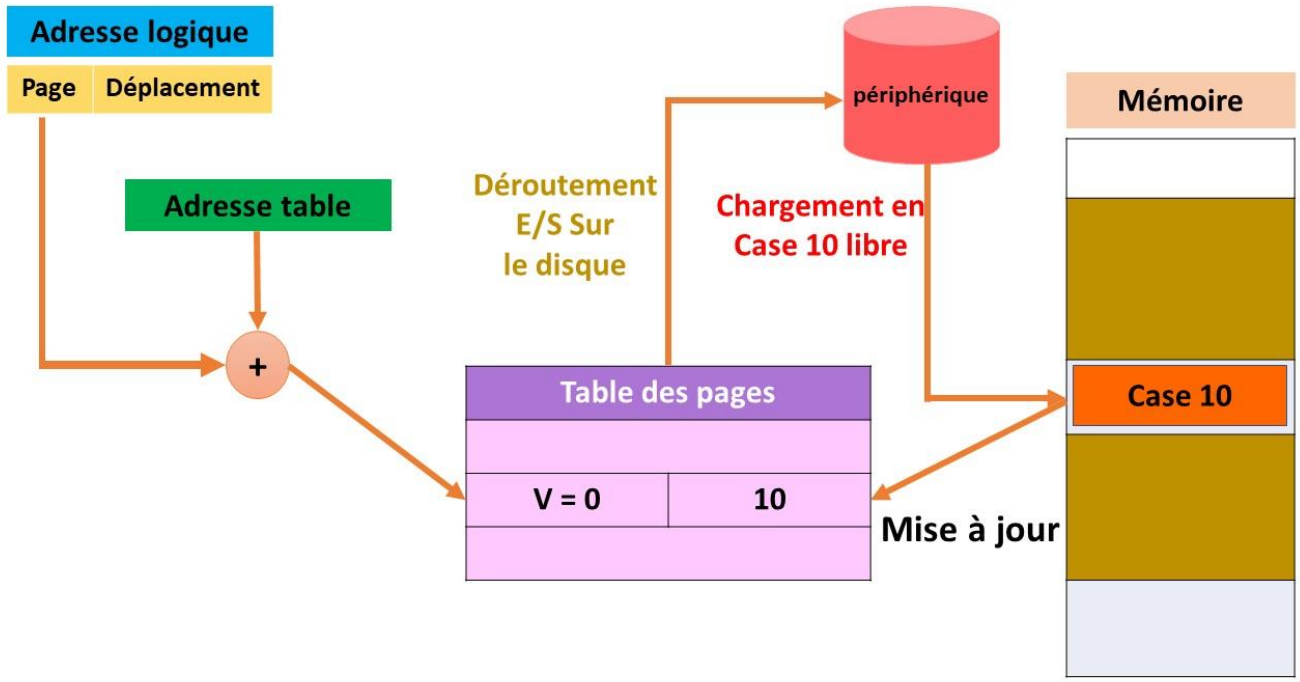


Figure VIII. 15: Défaut de page.

A l'occurrence d'un défaut de page, un déroutement est réalisé, le SE charge la page manquante dans une case vide. Si toutes les cases sont pleines, il doit d'abord vider une case (la mettre en mémoire auxiliaire) puis charge une nouvelle page.

VIII.7.2.3 MMU avec TLB :

L'utilisation de la **MMU** avec un **TLB** améliore considérablement l'efficacité de la traduction des adresses dans les systèmes de mémoire virtuelle. En stockant les traductions d'adresses les plus fréquemment utilisées dans le **TLB**, le système réduit la nécessité de consulter la table des pages, ce qui accélère l'accès à la mémoire et améliore les performances globales du système.

L'accès à un octet de mémoire dans un système utilisant la pagination et une Unité de Gestion de Mémoire (**MMU**) avec un Translation Lookaside Buffer (**TLB**) implique plusieurs étapes. Le **TLB** est une mémoire cache spécialisée utilisée pour réduire le temps nécessaire pour effectuer des traductions d'adresses de la mémoire virtuelle à la mémoire physique. Voici une description détaillée de ce processus.

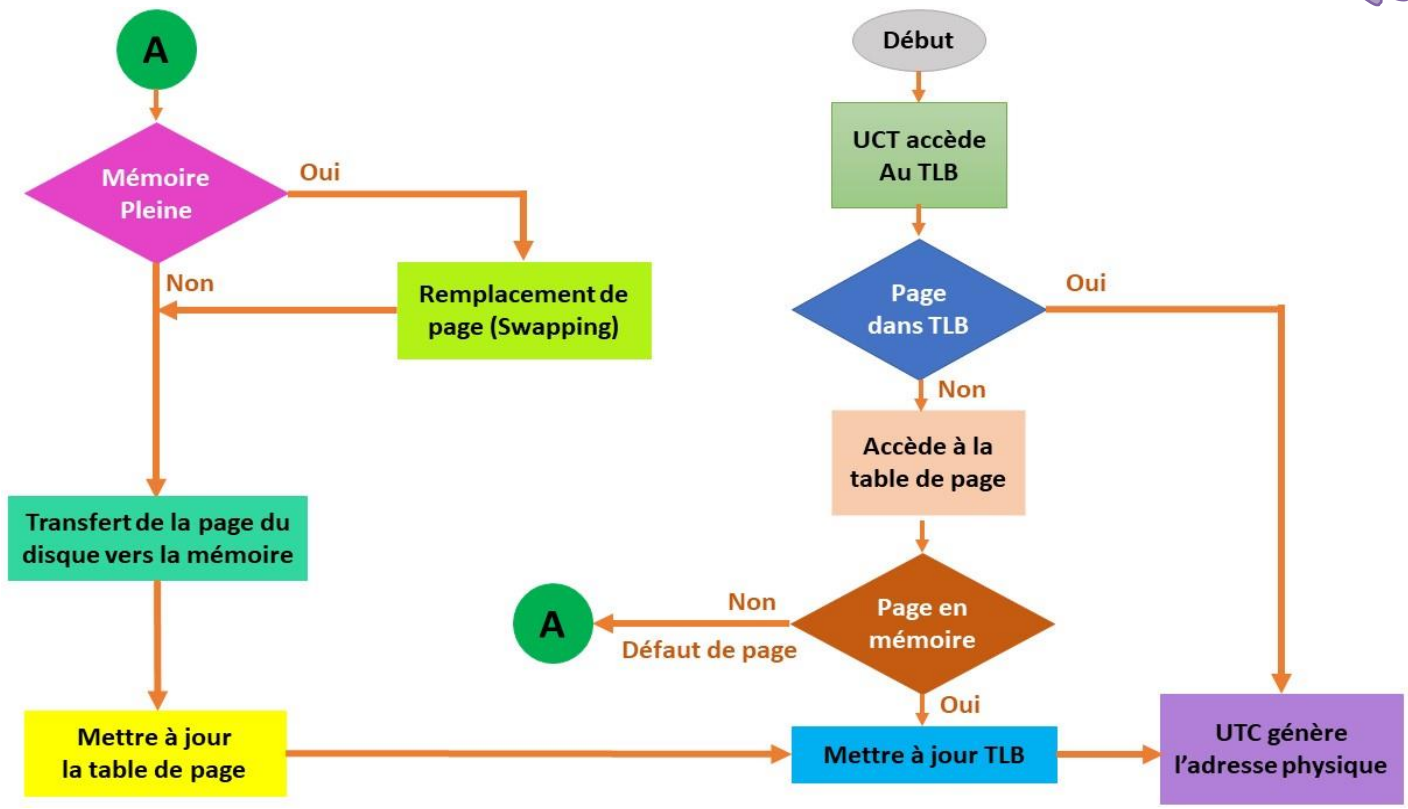


Figure VIII. 16: Le cas général d'accédé à 1 octet avec une adresse paginée.

VIII.7.2.4 Algorithmes de remplacement de pages en pagination :

Lorsqu'une page doit être chargée en mémoire, mais que toutes les cadres de page sont déjà occupés, un système de pagination doit décider quelle page existante remplacer. Plusieurs algorithmes de remplacement de pages peuvent être utilisés pour prendre cette décision. Voici les principaux algorithmes

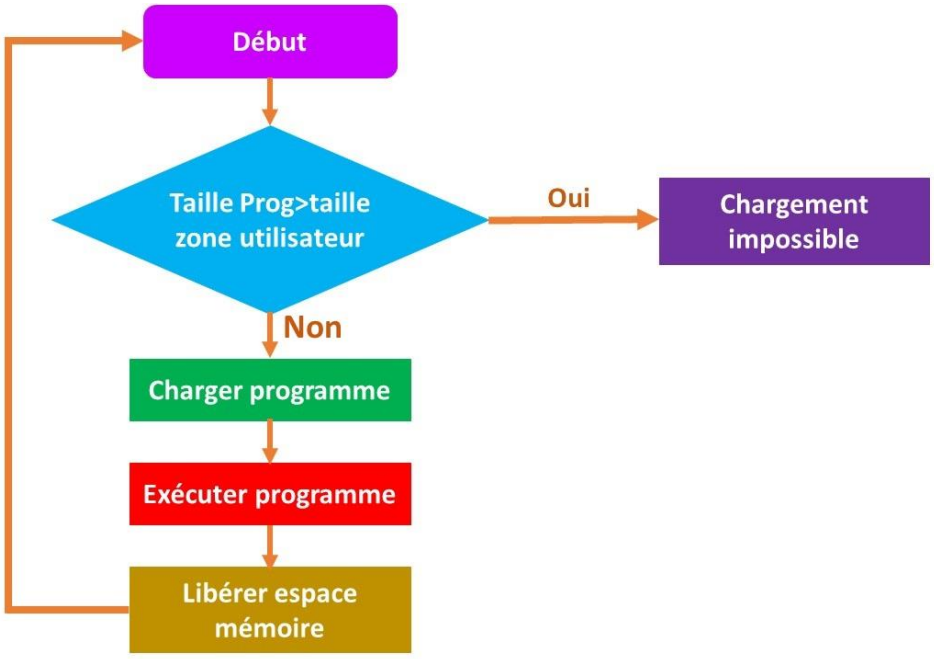


Figure VIII. 17: Algorithme d'Allocation mémoire d'un programme utilisateur.

1. Optimal Page Replacement (OPT) :

Cet algorithme remplace la page qui ne sera pas utilisée pendant la plus longue durée à l'avenir. C'est l'algorithme optimal théorique, mais il n'est pas réalisable en pratique car il faudrait prédire les futurs accès mémoire.

Exemple :

Chaîne de page	**7**	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
Frame 1	**7**	**7**	**7**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**7**	**7**	**7**
Frame 2		**0**	**0**	**0**	**0**	**0**	**0**	**4**	**4**	**4**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Frame 3			**1**	**1**	**1**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
Nbr défaut de page	**D**	**D**	**D**	**D**		**D**		**D**			**D**			**D**				**D**		
Taux de défaut de page	**9/20 (45%)**																			

2. First-In-First-Out (FIFO) :

C'est l'algorithme le plus simple, le système d'exploitation garde une trace de toutes les pages en mémoire dans une file d'attente, et la page la plus ancienne (en tête de file) est sélectionnée pour être remplacée.

Exemple :

Chaîne de page	**7**	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
Frame 1	**7**	**0**	**1**	**2**	**2**	**3**	**0**	**4**	**2**	**3**	**0**	**0**	**0**	**1**	**2**	**2**	**2**	**7**	**0**	**1**
Frame 2		**7**	**0**	**1**	**1**	**2**	**3**	**0**	**4**	**2**	**3**	**3**	**3**	**0**	**1**	**1**	**1**	**2**	**7**	**0**
Frame 3			**7**	**0**	**0**	**1**	**2**	**3**	**0**	**4**	**2**	**2**	**2**	**3**	**0**	**0**	**0**	**1**	**2**	**7**
Nbr défaut de page	**D**	**D**	**D**	**D**		**D**	**D**	**D**	**D**	**D**	**D**			**D**	**D**			**D**	**D**	**D**
Taux de défaut de page	**15/20 (75%)**																			

Intuitivement, on peut penser que plus il y a de frames disponibles, moins il y a de défauts de pages. Ceci n'est pas toujours le cas. **Belady** a montré par un contre- exemple que l'algorithme **FIFO** donne plus de défauts de pages avec l'accroissement du nombre de frames. Ceci est connu sous le nom de l'**anomalie de Belady.**

Exemple avec 3 cadres :

Chaîne de page	7	0	1	2	7	0	3	7	0	1	2	3	7	0	1	2	3	2	4	3
Frame 1	7	0	1	2	7	0	3	3	3	1	2	2	7	0	1	2	3	3	0	1
Frame 2		7	0	1	2	7	0	0	0	3	1	1	2	7	0	1	2	2	7	0
Frame 3			7	0	1	2	7	7	7	0	3	3	1	2	7	0	1	1	2	7
Nbr défaut de page	D	D	D	D	D	D	D			D	D		D	D	D	D	D		D	
Taux de défaut de page	15/20 (75%)																			

Le même exemple avec 4 cadres :

Chaîne de page	7	0	1	2	7	0	3	7	0	1	2	3	7	0	1	2	3	2	4	3
Frame 1	7	0	1	2	2	2	3	7	0	1	2	3	7	0	1	2	3	3	4	4
Frame 2		7	0	1	1	1	2	3	7	0	1	2	3	7	0	1	2	2	3	3
Frame 3			7	0	0	0	1	2	3	7	0	1	2	3	7	0	1	1	2	2
Frame 4				7	7	7	0	1	2	3	7	0	1	2	3	7	0	0	1	1
Nbr défaut de page	D	D	D	D			D	D	D	D	D	D	D	D	D	D	D		D	
Taux de défaut de page	16/20 (80%)																			

Anomalie de Belady : augmenter le nombre de cadres peut augmenter le nombre de défauts de page au lieu de le diminuer (dans certaine situation atypique).

3. Least Recently Used (LRU)

Cet algorithme remplace la page qui n'a pas été utilisée depuis le plus longtemps. L'idée est que les pages utilisées récemment seront probablement réutilisées bientôt. La mise en œuvre pratique de **LRU** est complexe car il faut maintenir une liste ordonnée de toutes les pages.

Chaîne de page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Nbr défaut de page	D	D	D	D		D		D	D	D	D			D		D		D		
Taux de défaut de page	12/20 (60%)																			

4. Algorithme de seconde chance :

Cet algorithme est une variante de **FIFO**. Lorsqu'une page doit être remplacée, il regarde d'abord si le bit de référence est à 0, auquel cas il remplace cette page. Sinon, si le bit de référence est à 1, il met le bit de référence à 0 et passe à la page suivante.

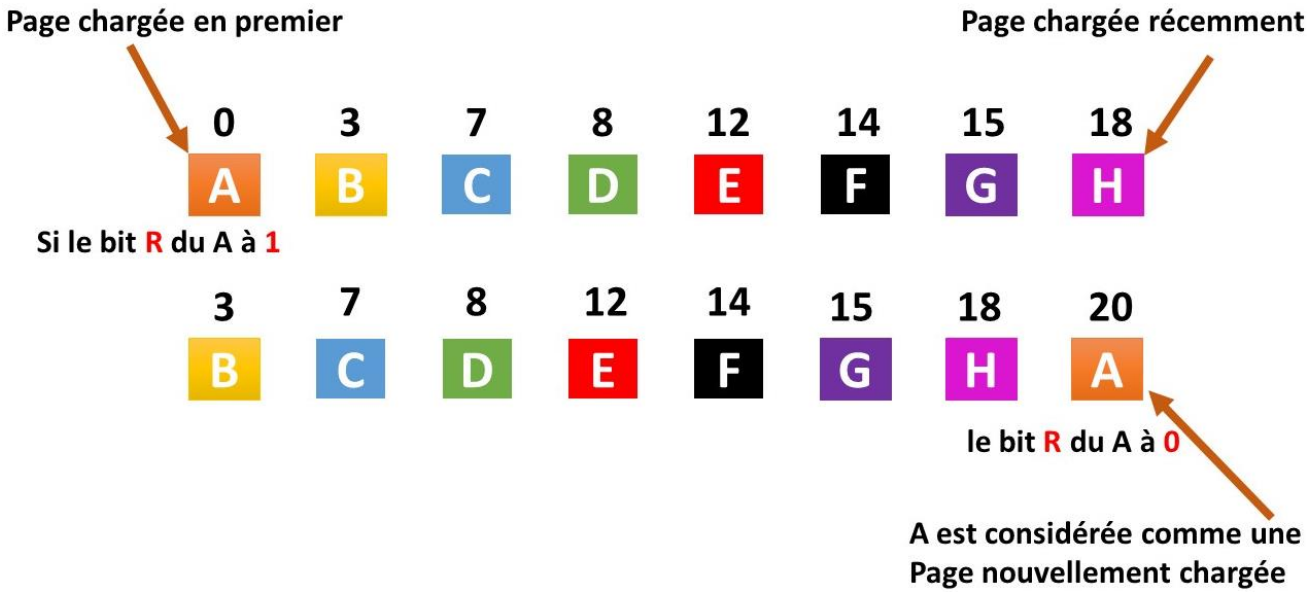


Figure VIII. 18: Algorithme de seconde chance.

Chaîne de page	7	0	1	7	3	0	3	7	2	1	4	3	4	0	1	2	3	2	4	3
Frame 1	7_0	0_0	1_0	1_0	3_0	0_0	0_0	0_0	2_0	1_0	4_0	3_0	3_0	0_0	1_0	2_0	3_0	3_0	4_0	4_0
Frame 2		7_0	0_0	0_0	7_0	3_0	3_1	3_1	3_0	2_0	1_0	4_0	4_1	3_0	4_0	1_0	2_0	2_1	3_0	3_1
Frame 3			7_0	7_1	1_0	7_0	7_0	7_1	7_0	3_0	2_0	1_0	1_0	4_1	0_0	4_0	1_0	1_0	2_1	2_0
Nbr défaut de page	D	D	D		D	D			D	D	D	D		D	D	D	D		D	
Taux de défaut de page	14/20 (70%)																			

5. Algorithme de l'horloge :

L'algorithme de l'horloge est une variante de l'algorithme **FIFO (First-In-First-Out)** qui prend en compte l'utilisation récente des pages. Il est plus performant que **FIFO** tout en étant plus simple à mettre en œuvre que **LRU (Least Recently Used)**.

- **Voici comment fonctionne l'algorithme de l'horloge :**

1. Chaque cadre de page en mémoire a un bit "utilisé" (référence bit) qui est initialisé à **0**.
2. Lorsqu'une page est accédée, son bit "utilisé" est mis à **1**.
3. Quand un défaut de page se produit et que la mémoire est pleine, l'algorithme parcourt circulairement les cadres de page comme les heures d'une horloge.

4. Pour chaque cadre rencontré :
   - Si le bit "utilisé" est à **0**, la page de ce cadre est sélectionnée pour être remplacée.
   - Si le bit "utilisé" est à **1**, il est remis à **0** et l'algorithme passe au cadre suivant.
5. Le remplacement a lieu une fois qu'un cadre avec le bit "utilisé" à **0** a été trouvé.

Cet algorithme protège les pages fréquemment utilisées du remplacement en remettant à **0** leur bit "utilisé" à chaque accès. Seules les pages peu ou pas utilisées récemment seront remplacées.

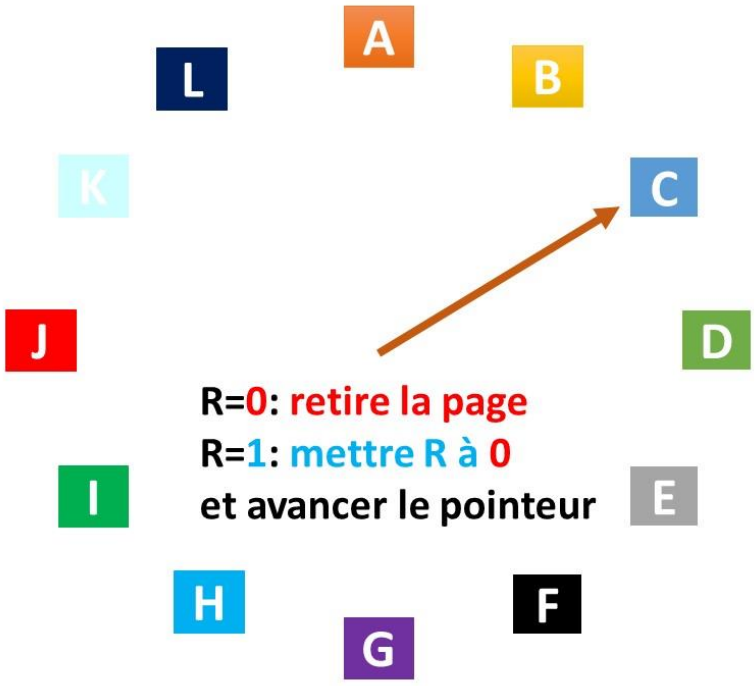


Figure VIII. 19: Algorithme de l'horloge.

- **Algorithmes basés sur le comptage :**

6. Least Frequently Used (LFU) :

L'algorithme **LFU** remplace la page qui a été accédée le moins de fois. Il maintient un compteur pour chaque page et remplace la page avec le compteur le plus faible. En cas d'égalité, il remplace la page la plus ancienne.

Chaîne de page	**7**	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**
Frame 1	7_0	7_0	7_0	2_0	2_0	2_0	2_0	4_0	4_0	3_0	3_0	3_1	3_1	3_1	3_1
Frame 2		0_0	0_0	0_0	0_1	0_0	0_2	0_2	0_2	0_2	0_3	0_3	0_3	0_3	0_3
Frame 3			1_0	1_0	1_0	3_0	3_0	3_0	2_0	2_0	2_0	2_0	2_0	1_0	2_0
Nbr défaut de page	D	D	D	D		D		D	D	D				D	D
Taux de défaut de page	**10/15= 66,67%**														

7. Most Frequently Used (MFU) :

L'algorithme **MFU** remplace la page qui a été accédée le plus de fois. Il maintient également un compteur pour chaque page et remplace la page avec le compteur le plus élevé. En cas d'égalité, il remplace la page la plus ancienne.

Chaîne de page	**7**	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**
Frame 1	7_0	7_0	7_0	2_0	2_0	2_0	2_0	4_0	4_0	4_0	0_0	0_0	0_0	0_0	0_0
Frame 2		0_0	0_0	0_0	0_1	3_0	3_0	3_0	2_0	2_0	2_0	2_0	2_1	1_0	1_0
Frame 3			1_0	1_0	1_0	1_0	0_0	0_0	0_0	3_0	3_0	3_1	3_1	3_1	2_0
Nbr défaut de page	**D**	**D**	**D**	**D**		**D**	**D**	**D**	**D**	**D**	**D**			**D**	**D**
Taux de défaut de page	**12/15= 80%**														

8. Not Frequently Used (NFU) :

L'algorithme **NFU** est une variante de **LFU**. Il classe les pages en **4** catégories selon leur fréquence d'accès et remplace la page de la catégorie le moins fréquemment utilisé.

- **Soit l'exercice suivant** :

On considère un système de pagination dont la **taille de page** est égale à **4 ko,** la **mémoire physique** est de **12ko** et le **mot de mémoire** de **1 octet**.

1/ Combien de cadre de pages contient cette mémoire ?

2/ Soit un **programme** de taille **24ko**, qui fit référence aux adresses logique suivantes :

2, 5012, 6200, 8215, 2000, 17800, 50, 13248, 18456, 1203, 5741, 9442, 16524, 23580, 16895, 22630, 123.

- Donner pour chaque adresse le numéro de page et le déplacement dans la page ***< p, d >***.
- Déduire la chaine de pages référencées durant l'exécution de ce programme.

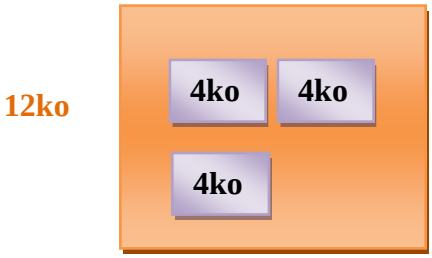
3/ Calculer le taux de défaut de page résulté par un remplacement **FIFO**, et **LRU**. Quel algorithme minimise ce taux ?

Solution :

1/ Le nombre de cadre de pages contient cette mémoire :

$$\textbf{nbr de cadre} = \frac{\textbf{la mémoire physique}}{\textbf{la taille de page}}$$

Soit 12/4 = 3 est le nombre de cadre de pages contient cette mémoire.



2/ Déduire la chaine de pages référencées durant l'exécution de ce programme.

$$\text{Chaine de page} = \frac{adresses\ logique}{\text{la taille de page}}$$

Chaine	2	5012	6200	8215	2000	17800	50	13248	18456	1203	5741	9442	16524	23580	16895	22630	123
÷4096	0	1	1	2	0	4	0	3	4	0	1	2	4	5	4	5	0
Reste	2	916	2104	23	2000	1416	50	960	2072	1203	1645	1250	140	3100	511	2150	123

3/ Calculer le taux de défaut de page résulté par un remplacement **FIFO** :

Chaîne de page	0	1	1	2	0	4	0	3	4	0	1	2	4	5	4	5	0
Frame 1	0	1	1	2	2	4	0	3	3	3	1	2	4	5	5	5	0
Frame 2		0	0	1	1	2	4	0	0	0	3	1	2	4	4	4	5
Frame 3				0	0	1	2	4	4	4	0	3	1	2	2	2	4
Nbr défaut de page	D	D		D		D	D	D			D	D	D	D			D
Taux de défaut de page	11/17=64,70%																

- Calculer le taux de défaut de page résulté par un remplacement **LRU :**

Chaîne de page	0	1	1	2	0	4	0	3	4	0	1	2	4	5	4	5	0
Frame 1	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	4	4
Frame 2		1	1	1	1	1	0	0	0	0	1	1	1	5	5	5	5
Frame 3				2	2	2	2	3	3	3	3	2	2	2	2	2	0
Nbr défaut de page	D	D		D		D	D	D			D	D		D			D
Taux de défaut de page	10/17=58,82%																

- **Comparaison des Algorithmes**
- **FIFO** : 11 défauts de page.
- **LRU** : 10 défauts de page.
- **LRU** plus performant que **FIFO.**

VIII.7.1.5 Comparaison des Algorithmes :

Tableau VIII. 5: Comparaison des algorithmes de remplacement de pages.

Algorithme	Avantages	Inconvénients
First-In-First-Out (FIFO)	Simple à implémenter.	Peut conduire à des performances sous-optimales (anomalie de Belady).
Optimal Page Replacement (OPT)	Théoriquement optimal, offre le nombre minimum de fautes de page.	Impossible à mettre en œuvre en pratique, nécessite une connaissance parfaite du futur.
Least Recently Used (LRU)	Prend en compte l'utilisation récente, souvent plus performant que **FIFO.**	Complexité de mise en œuvre et de gestion.
Algorithme de Seconde Chance	Évite de remplacer des pages fréquemment utilisées, améliore **FIFO.**	Peut nécessiter plusieurs passes pour trouver une page à remplacer.
Algorithme de L'horloge	Similaire à l'algorithme de seconde chance mais plus efficace plus facile à implémenter.	Peut encore nécessiter plusieurs passes pour trouver une page à remplacer.
Not Recently Used (NRU)	Simple à mettre en œuvre et à comprendre.	Moins précis que **LRU**, peut ne pas toujours choisir la meilleure page à remplacer.
Least Frequently Used (LFU)	Prend en compte l'utilisation à long terme des pages.	Peut conserver des pages rarement utilisées dans le passé mais souvent utilisées récemment.
Most Frequently Used (MFU)	Évite de remplacer les pages qui sont utilisées souvent, utile pour certaines charges de travail.	Peut conserver des pages qui ne sont plus pertinentes mais ont été souvent utilisées dans le passé.
Not Frequently Used (NFU)	Simple à implémenter, favorise les pages récemment introduites.	Peut conserver des pages rarement utilisées dans le passé, similaire à **LFU.**

VIII.7.2 Segmentation :

La segmentation est un découpage de l'espace d'adressage, comme la pagination, mais elle conserve la vue de programmeur de son programme (reproduit le découpage mémoire tel qu'il est décrit logiquement par l'usager) : - Le programme principal.

- Fonctions (sous-programme).
- Les données (initialisées et non initialisées).
- La pile d'exécution.

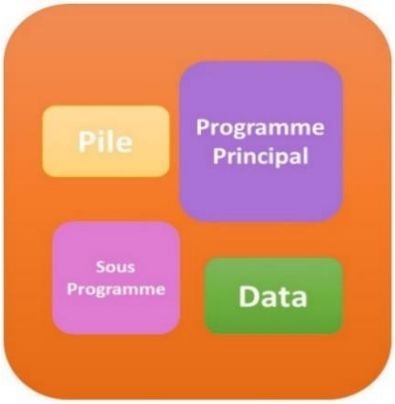


Figure VIII. 20: Vue de l'utilisateur d'un programme.

Lors de la compilation, le compilateur associe un segment à chaque morceau du programme compilé.

Un segment est un ensemble d'emplacement mémoire consécutif non sécable, à la différence des pages, les segments d'un même espace d'adressage peuvent être de taille différente.

La segmentation facilite le partage entre processus de segment de données ou de code.

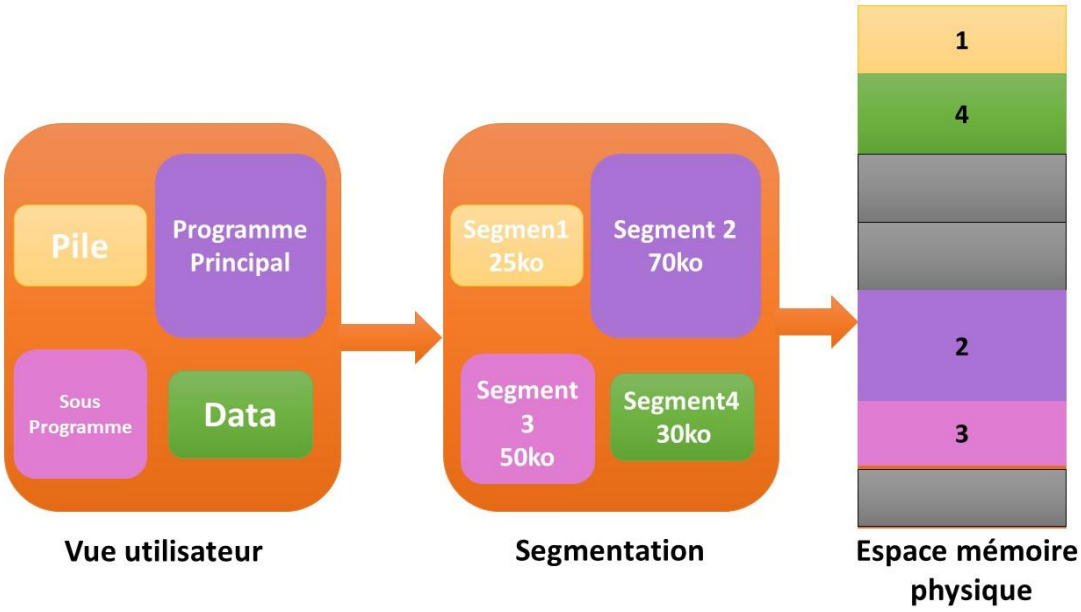


Figure VIII. 21: Allocation d'espace mémoire par segmentation.

Contrairement au schéma de pagination, la segmentation permet d'éliminer la fragmentation interne car l'espace mémoire alloué à un segment est de taille égale exactement à la taille du segment. Cependant, une fragmentation externe peut se produire due au fait qu'on fait une allocation dynamique de l'espace mémoire. Pour remédier à la fragmentation externe, la pagination et la segmentation peuvent être combinée : **paginer les segments :**

- Chaque segment est composé d'un ensemble de page.

- Les adresses virtuelles sont dans ce cas des triples : < Numéro de segment, numéro de page, déplacement dans la page >.

Un segment est une entité contenant des informations précises pour le système et pour sa gestion.

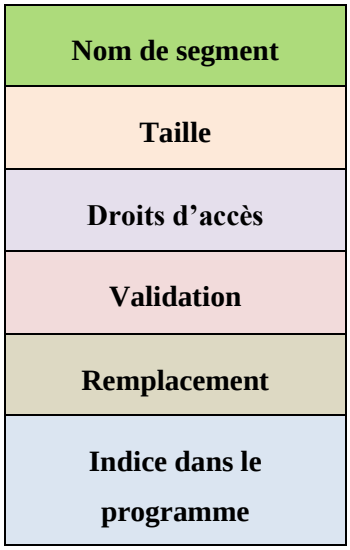


Figure VIII. 22: Schéma illustratif d'un segment.

Validation : Indique si le segment doit rester en mémoire.

Remplacement : Propose des informations à l'algorithme de remplacement.

VIII.7.2.1 Schéma de translation d'adresses :

Chaque segment est repéré par son numéro **S** et sa longueur variable **L**. Un segment est un ensemble d'adresses logiques contiguës. Une adresse logique est donnée par un couple **(S, d)**, où **S** est le numéro du segment et **d** le déplacement dans le segment. L'association d'une adresse logique à une adresse physique est décrite dans une table appelée **table de segments (Segment Table).** Chaque entrée de la table de segments possède un **segment de base** et un **segment limite**. Le segment de base contient l'adresse physique de début où le segment réside en mémoire, tandis que le registre limite spécifie la longueur du segment. La translation des adresses logiques en adresses physiques est à la charge de la **MMU**. Le support matériel pour la segmentation est montré dans la figure ci-après :

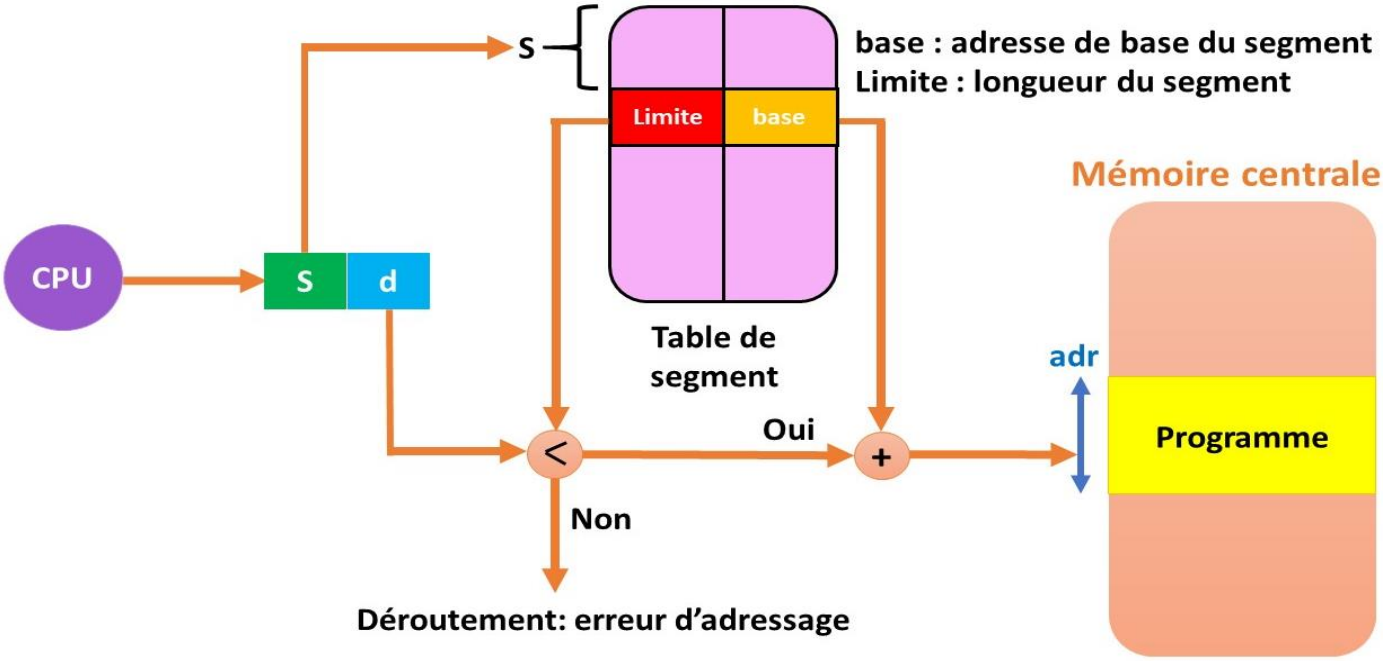


Figure VIII. 23: Matériel pour la segmentation.

Le déplacement **d** dans le segment doit être compris entre **0** et la limite du segment et dans ce cas **l'adresse physique est calculée** en additionnant la valeur de **d** à la base du segment. Dans le cas contraire, une **erreur d'adressage** est générée.

VIII.7.2.2 Comparaison entre Pagination et Segmentation :

Tableau VIII. 6: Comparaison entre Pagination et Segmentation.

Critère	Pagination	Segmentation
Unité de mémoire	Pages de taille fixe.	Segments de tailles variables.
Fragmentation	Fragmentation interne (à l'intérieur des pages).	Fragmentation externe (entre les segments).
Correspondance logique	Pas de correspondance logique directe avec les unités du programme.	Correspondance directe avec les unités logiques du programme.
Taille des unités	Taille fixe (généralement entre **4KB** et **64KB**).	Taille variable selon les besoins du programme.
Gestion de la mémoire	Simple et efficace, mais nécessite une table des pages pour chaque processus.	Plus complexe en raison des tailles variables des segments.
Protection de la mémoire	Protection de la mémoire par page (moins flexible).	Protection par segment, chaque segment ayant ses propres droits d'accès.
Partage de la mémoire	Partage facile des pages de code entre processus.	Partage facile des segments de code entre processus.

VIII.7.3 Mémoire virtuelle :

La mémoire virtuelle est une technique de gestion de la mémoire dans laquelle la mémoire secondaire **(comme le disque dur)** peut être utilisée comme si elle faisait partie intégrante de la mémoire principale **(RAM)**. Le système d'exploitation utilise une combinaison de matériel et de logiciel pour mapper les adresses mémoire utilisées par un programme **(appelées adresses virtuelles)** en adresses physiques dans la mémoire de l'ordinateur. Avec la mémoire virtuelle, la mémoire vue par un processus apparaît comme un espace d'adressage contiguë ou une collection de segments contiguës. Le système d'exploitation gère les espaces d'adressage virtuels et l'assignation de la mémoire réelle à la mémoire virtuelle.

- **Avantages de la mémoire virtuelle :**
  - Les principaux avantages de la mémoire virtuelle sont :
    - ✓ Permettre l'exécution de programmes plus grands que la mémoire physique
    - ✓ Utiliser la mémoire de manière plus efficace
    - ✓ Augmenter le niveau de multiprogrammation (exécution de plusieurs programmes en parallèle)
    - ✓ Gérer automatiquement le transfert des données entre mémoire physique et disque
    - ✓ Offrir une isolation mémoire et donc plus de sécurité
    - ✓ Permettre l'allocation mémoire à moindre coût
    - ✓ Éviter la fragmentation externe de la mémoire
- **Fonctionnement :**

La mémoire virtuelle fonctionne généralement selon le principe de la pagination. La mémoire est divisée en pages de taille fixe. Lorsqu'un programme accède à une page qui n'est pas présente en mémoire **RAM**, le processeur déclenche une exception **(page fault)**. Le système d'exploitation charge alors la page depuis le disque vers la mémoire et redémarre le programme. Les performances dépendent du nombre total de défauts de page, qui dépendent eux-mêmes des algorithmes de remplacement de pages **(choix des pages à remplacer en mémoire)** et d'allocation de cadres **(nombre de pages allouées à chaque processus)**.

- **Limites**

Bien que très utile, la mémoire virtuelle a aussi quelques inconvénients :

- Les applications s'exécutent plus lentement quand elles tournent depuis la mémoire virtuelle
- Le système peut ralentir si la **RAM** est insuffisante et que trop de pages doivent être échangées avec le disque **(thrashing).**
- La taille de la mémoire virtuelle est limitée par l'espace disque disponible.
- Cela réduit l'espace disque disponible pour les données.

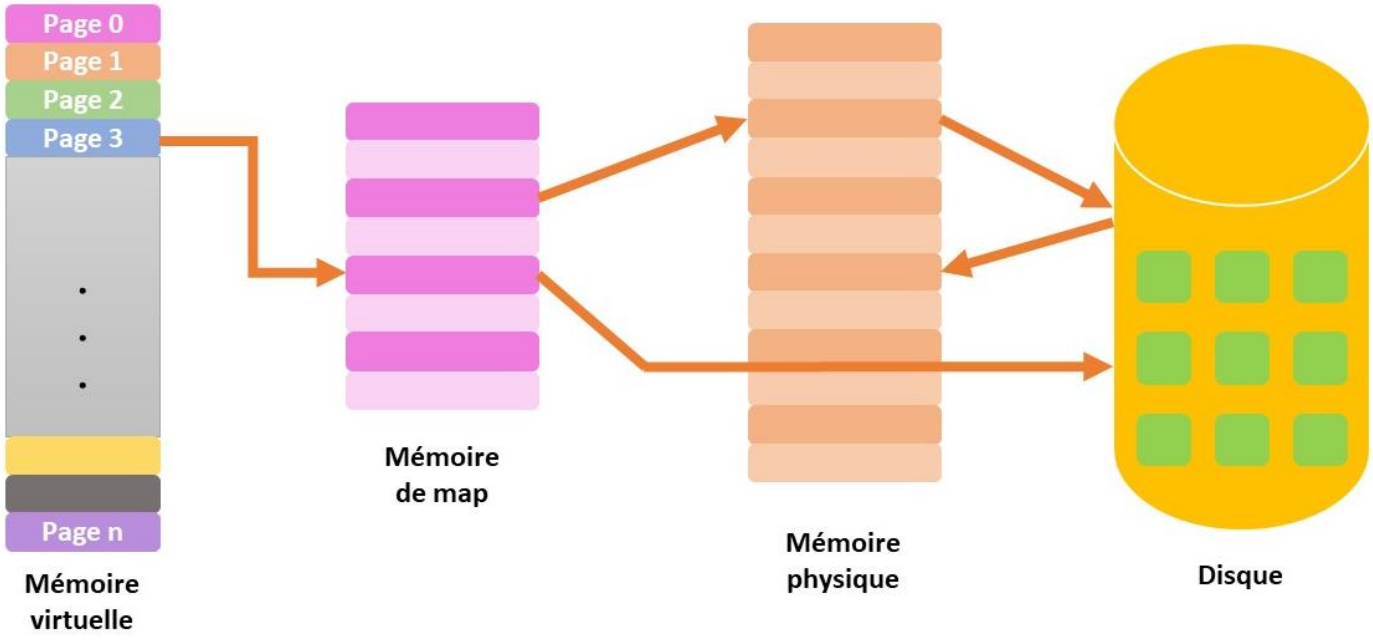


Figure VIII. 24: Mémoire virtuelle vs mémoire physique.

Voici un tableau comparatif des principales différences entre la mémoire virtuelle et la mémoire physique :

Tableau VIII. 7: Comparaison entre mémoire virtuelle et mémoire physique.

Critère	Mémoire Physique	Mémoire Virtuelle
Définition	Mémoire primaire de l'ordinateur (RAM)	Technique de gestion de la mémoire utilisant le disque dur
Taille	Limitée par la capacité des barrettes RAM	Limitée par la taille du disque dur
Type de mémoire	Volatile (perd son contenu sans alimentation)	Non volatile (stockée sur le disque)
Accès par le processeur	Accès direct	Nécessite une translation d'adresses (MMU)
Performances	Très rapide	Plus lent (latence des accès disque)
Utilisation de la mémoire	Chargement complet des programmes	Chargement par pages à la demande
Sécurité	Accès direct, moins de protection	Isolation des espaces mémoire, plus de sécurité
Avantages	Accès rapide, simplicité	Exécution de programmes plus gros, meilleure utilisation de la mémoire
Inconvénients	Taille limitée	Surcoût en performances, complexité de gestion

VIII.8. Conclusion :

La gestion de la mémoire est essentielle pour assurer l'efficacité, la sécurité et la fiabilité des systèmes d'exploitation. Elle permet de maximiser l'utilisation des ressources mémoire, de protéger les processus les uns des autres, et d'assurer que les applications peuvent fonctionner sans interruption. Une gestion de la mémoire bien conçue et mise en œuvre est cruciale pour la performance globale du système, permettant aux utilisateurs et aux applications de tirer le meilleur parti de l'infrastructure matérielle disponible.

Conclusion générale :

Le module de systèmes d'exploitation jouant un rôle déterminant dans la gestion et l'optimisation des ressources matérielles et logicielles d'un ordinateur. Au travers des différentes sections abordées dans ce polycopié, nous avons exploré les multiples facettes et fonctions essentielles des systèmes d'exploitation, ainsi que leurs implications pratiques et théoriques.

Le système d'exploitation est au cœur du fonctionnement de tout système informatique. Il sert de médiateur entre le matériel et les applications, assurant une gestion efficace et sécurisée des ressources. La compréhension approfondie des mécanismes et des principes qui régissent les systèmes d'exploitation est cruciale pour les professionnels de l'informatique, les développeurs et les administrateurs système. Ce module fournit les bases théoriques et pratiques nécessaires pour concevoir, déployer et maintenir des systèmes informatiques robustes, performants et sécurisés.

En somme, le système d'exploitation, par sa capacité à gérer de manière optimale les processus, la mémoire, le stockage et les périphériques, ainsi qu'à garantir la sécurité, constitue le pilier sur lequel repose l'ensemble de l'écosystème informatique. Sa maîtrise est indispensable pour relever les défis technologiques actuels et futurs.

Ce polycopié de cours sur les systèmes d'exploitation offre une synthèse des concepts fondamentaux et des fonctions clés de ces systèmes logiciels essentiels. Ils présentent différentes interfaces aux utilisateurs et applications, allant des langages de commande aux appels système permettant d'accéder aux services du système.

On observe une transition des systèmes propriétaires vers des systèmes ouverts et portables, notamment avec l'essor du logiciel libre comme Linux. Le passage à l'informatique distribuée et l'émergence des systèmes embarqués et de l'internet des objets sont d'autres tendances majeures. Le développement des techniques de virtualisation et du cloud computing constituent également des évolutions importantes.

Bien que les principes de base soient stables, les systèmes d'exploitation évoluent en fonction des progrès matériels et des besoins applicatifs. Maîtriser leurs commandes et la programmation de scripts est essentiel pour les informaticiens.

En résumé, ce cahier offre une vue d'ensemble des systèmes d'exploitation, de leur rôle central dans l'informatique moderne, de leur fonctionnement et de leur évolution. Ils constituent un support pédagogique important pour les étudiants en informatique.

Bibliographie :

- **[1]**. **D. P. BOVET** et **M. Cesati,** « Understanding the Linux Kernel (3rd Edition) », O'Reilly Media, (2005).
- **[2]**. **T. MASSART,** « MOOC Numérique et Sciences Informatiques », Université Libre de Bruxelles (ULB) - l'école Polytechnique de l'ULB, (2023).
- **[3]. J. REKIK** et **Z. DHAOUI,** « Système d'Exploitation ». Université de Manouba Ecole Supérieure d'Economie Numérique. (2013).
- **[4]. M. LOUKAM**, « Système d'exploitation des ordinateurs 1 », Université Hassiba Benbouali Chlef - Département informatique, (2017).
- **[5]. A. ABBAS**, « Systèmes d'exploitation », Université de Bouira Faculté des Sciences et des Sciences appliquées - Département d'Informatique, (2016).
- **[6]. J-F LALONDE,** « GIF-1001 Ordinateurs : Structure et Applications », Faculté des sciences et de génie à l'Université Laval, (2015).
- **[7]. D. BOUKHLOUF,** « Systèmes d'exploitation I : Gestion des entrées / sorties », Université Mohamed Kheider Biskra - Faculté des sciences exactes et sciences de la nature et de la vie Département d'informatique, (2018).
- **[8]. C. BENZAID**, « Cours systèmes d'exploitation (partie II) ». LSI-Département Informatique, Faculté d'Electronique & Informatique, USTHB, (2021).
- **[9]. A. JUTON,** « Principes et fonctionnement d'un système d'exploitation », ENS Paris Saclay, (2021).
- **[10]. N. SALMI,** « Principes des Systèmes d'Exploitation », Pages Bleues, les Manuels de l'Etudiant, (2007).
- **[11]. A. OUARED,** « Synthèse des cours et recueil d'exercices corrigés pour le module du système d'exploitation 1 », Université Ibn khaldoun - Tiaret Faculté Mathématiques et Informatique, (2020).
- **[12]**. **M.A. PERALDI-FRATI,** « Cours Architecture des Ordinateurs », IUT de Nice- Côte d'Azur Département Informatique, (2007).
- **[13]**. **I. SGHAIER,** « Systèmes d'exploitation – Gestion des périphériques », Université Yahia Fares de Médéa-département informatique, (2007).
- **[14] N. Farah**, « Cours processus », Département d'Informatique Faculté des Sciences Université Badji Mokhtar Annaba, (2020).
- **[15]. M. DAOUDI**, « Système d'Exploitation I ». Université Mohammed Premier Faculté des Sciences Département d'Informatique, Oujda, (2019).

- **[16]. B. LAMIROY**, **L. NAJMAN** et **H. TALBOT**, « Systèmes d'exploitation », Collection Synthex, École supérieure d'ingénieurs en électronique et électrotechnique, à Noisy-le-Grand-Paris, (2006).
- **[17]. M. BEN ABDELJELIL,** « Synchronisation de processus », Université IHEC Sousse Tunisie, (2020).
- **[18]. S. MONNIER**, « Ordonnancement », Université de Montréal Faculté des arts et des sciences - Département d'informatique et de recherche opérationnelle, (2017).
- **[19]. M. CHENAIT, B. ZEBBANE** et **C. BENZAID**, « Introduction aux systèmes d'exploitation 1 », University of Sciences and Technology Houari Boumediene, (2022).
- **[20]. Z. KOUAHLA**, « Cours système d'exploitation 2 », Université Guelma-Département d'Informatique, (2022).
- **[21]. L. GHALOUCI,** « Architecture de l'Ordinateur », Université d'Oran Des Sciences et de la Technologie - Mohamed Boudiaf, (2015).
- **[22]. M. MAHSEUR**, « Cours Mémoires », Université des Sciences et de la Technologie Houari Boumediène, (2020).
- **[23]. T. PETAZZONI**, « Gestion des interruptions », Université de Technologie de Belfort-Montbéliard, (2008).
- **[24]. P. REUTER**, « Systèmes d'exploitation », LIPSI-ESTIA INRIA Futurs Bordeaux, (2005).
- **[25]. W. STALLINGS**, « Operating Systems: Internals and Design Principles », 7th Edition, Prentice Hall, Pearson Education, (2011).