

Séance 1

Gestion avancée de la mémoire



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Objectifs

- Méthodes de **partitionnement** de la mémoire
Fixe, dynamique ou avec un système de buddies
- Gestion avancée de la **mémoire virtuelle**
 - Protection, page partagée et fichier mappé en mémoire
 - Allocation de la mémoire du kernel
 - Considérations avancées : prepaging, taille pages, TLB reach...
- Techniques de buffering pour gérer les **entrées/sorties**
- Gestion de l'**espace libre** sur le disque

Gestion de la mémoire



Gestion de la mémoire

- Exigences à satisfaire par **système de gestion mémoire**
 - Relocalisation dans la mémoire physique
 - Protection contre l'accès par d'autres processus
 - Partage de zones mémoire avec d'autres processus
 - Organisation logique et physique de la mémoire
- Plusieurs **tâches** gérées par l'OS concernant la mémoire
 - Notamment déplacement entre mémoire secondaire et principale*

Partitionnement de la mémoire

- Partie fixe mémoire principale occupée par l'OS

Le reste est utilisé pour placer plusieurs processus

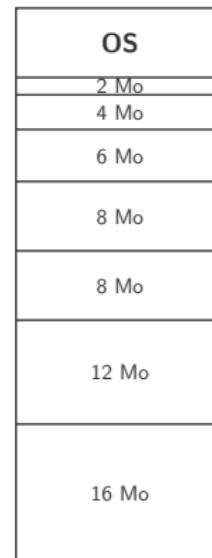
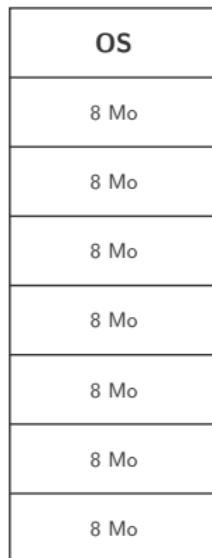
- Plusieurs techniques de partitionnement possibles

- Fixe avec partitions de même taille ou de tailles différentes
- Dynamique avec nombre et tailles différentes des partitions
- Systèmes avec des buddy's

Partitionnement fixe (1)

■ Partitions de tailles égales ou inégales

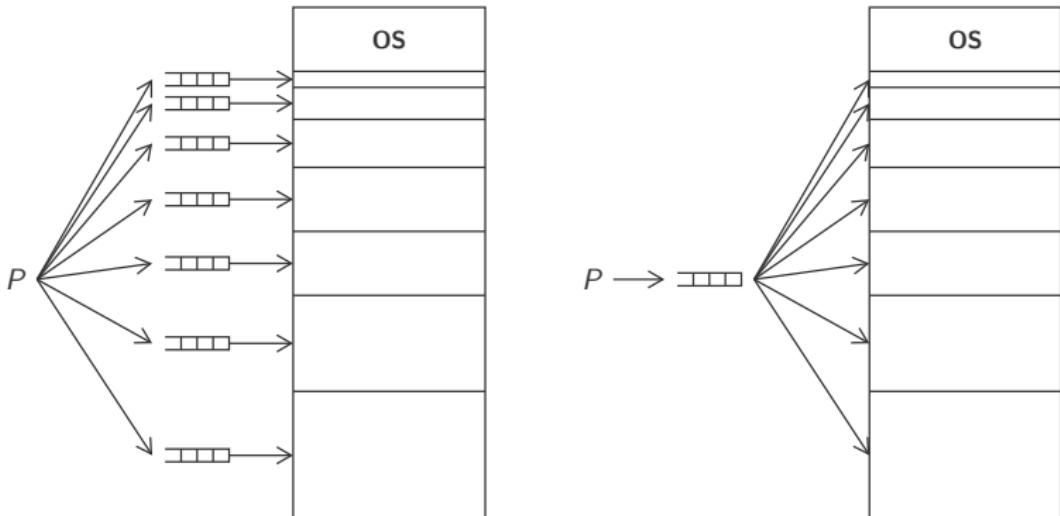
Choix d'une partition libre avec suffisamment de place



Partitionnement fixe (2)

- Une file **globale** ou **locale** par partition

Gaspillage de place si pas de processus de bonne taille



Partitionnement dynamique

- Un processus occupe précisément la **place dont il a besoin**
 - Fragmentation externe qui provoque création de trous
 - Élimination des trous par compaction de la mémoire
- Plusieurs **stratégies de placement** d'un processus
 - First-fit le plus simple et généralement meilleur et plus rapide
 - Best-fit généralement le pire en créant pleins de petits trous
 - Worst-fit et next-fit moins bons que first-fit

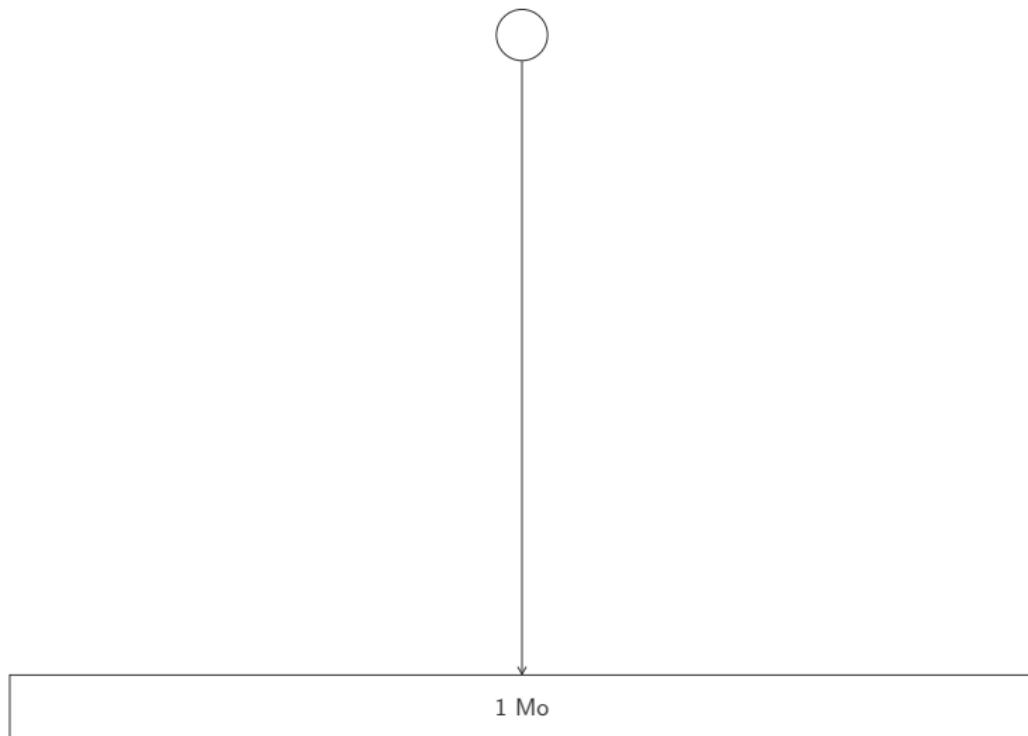
Buddy system (1)

- **Compromis** entre partitionnement fixe et dynamique
 - Utilisation inefficace de la mémoire avec fixe
 - Dynamique complexe à gérer et besoin de compactation
- Bloc mémoire de 2^K avec le **buddy system** ($L \leq K \leq U$)
 - 2^L est la plus petite taille de bloc autorisée
 - 2^U correspond souvent à la mémoire totale disponible
- Algorithme de recherche/création d'un **bloc de taille adéquate**
Structure arborescente de buddies de tailles identiques

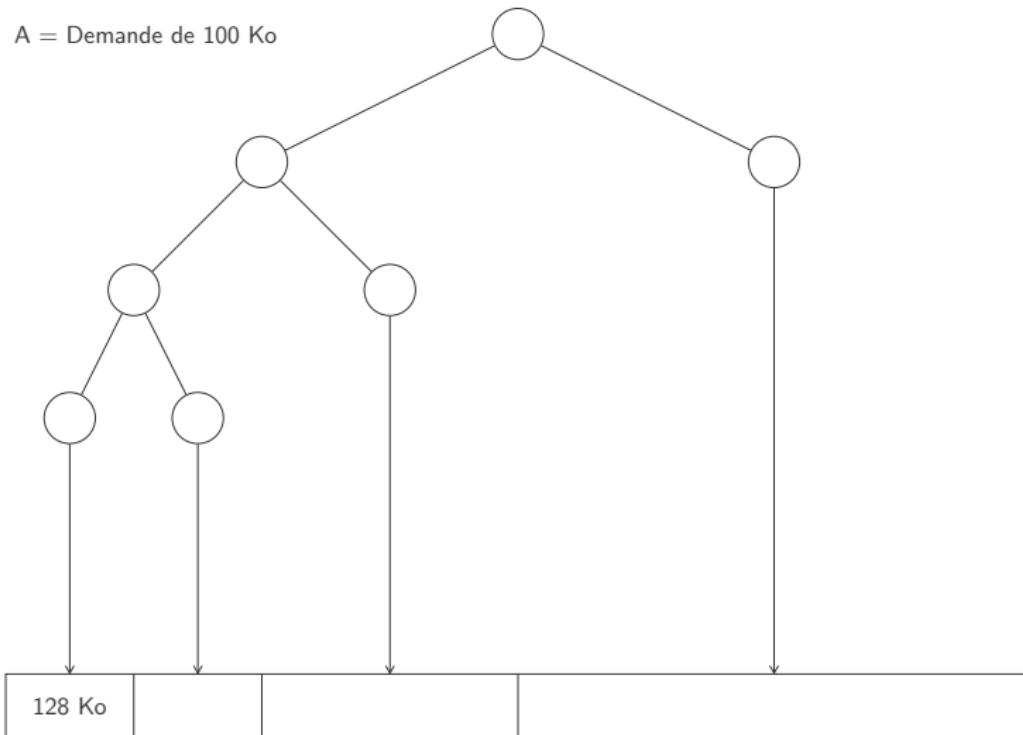
Buddy system (2)

- Traitement d'une **requête de taille s** de demande mémoire
 - 1 Taille totale disponible comme un seul bloc de taille 2^U
 - 2 Si $2^{U-1} \leq s \leq 2^U$, allocation de tout le bloc
 - 3 Sinon, découpe en deux buddies de même taille 2^{U-1}
 - 4 Si $2^{U-2} \leq s \leq 2^{U-1}$, allocation du bloc...
- Deux principales **opérations** sur les blocs
 - Découpe d'un bloc en deux blocs de tailles identiques
 - Regroupement de deux blocs en un seul bloc

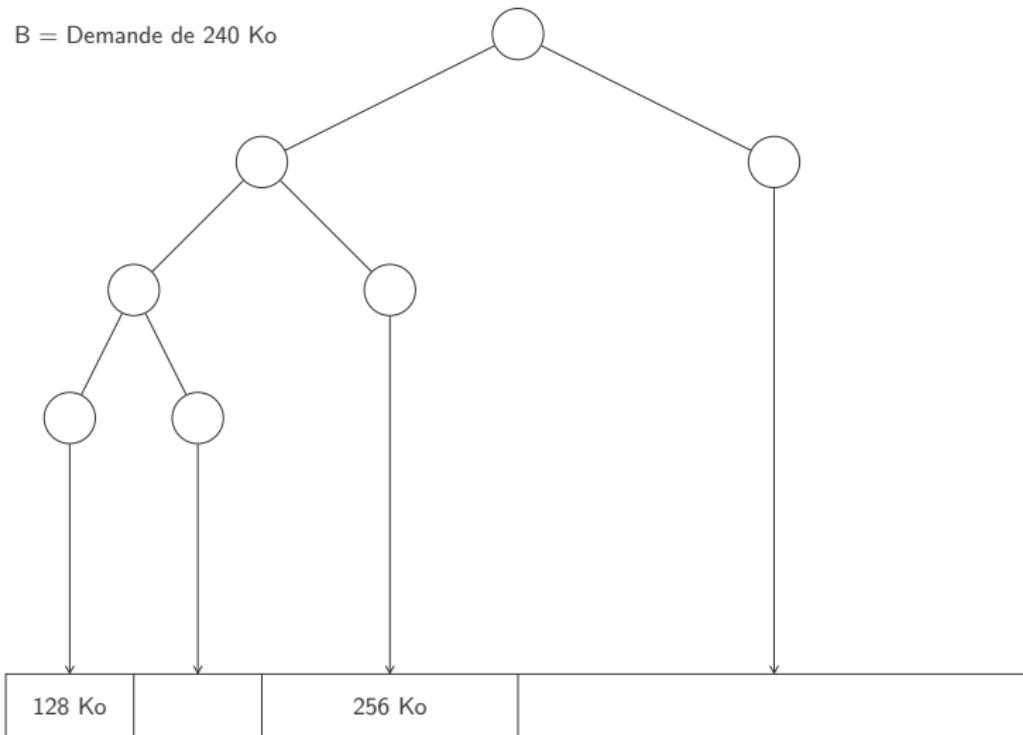
Exemple du buddy system



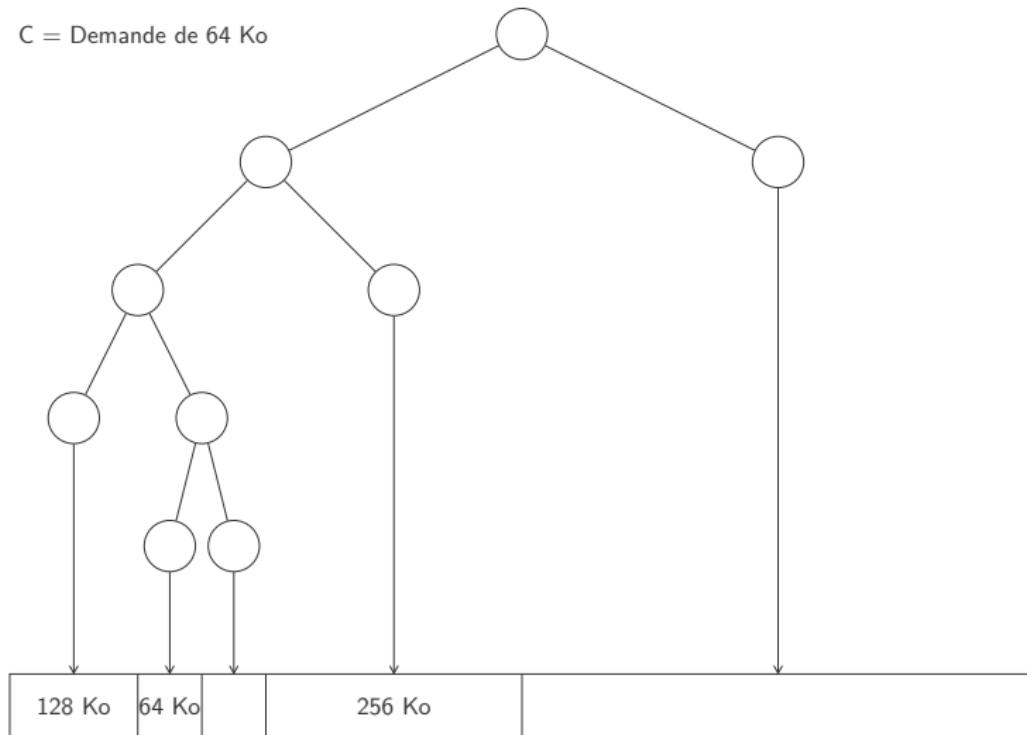
Exemple du buddy system



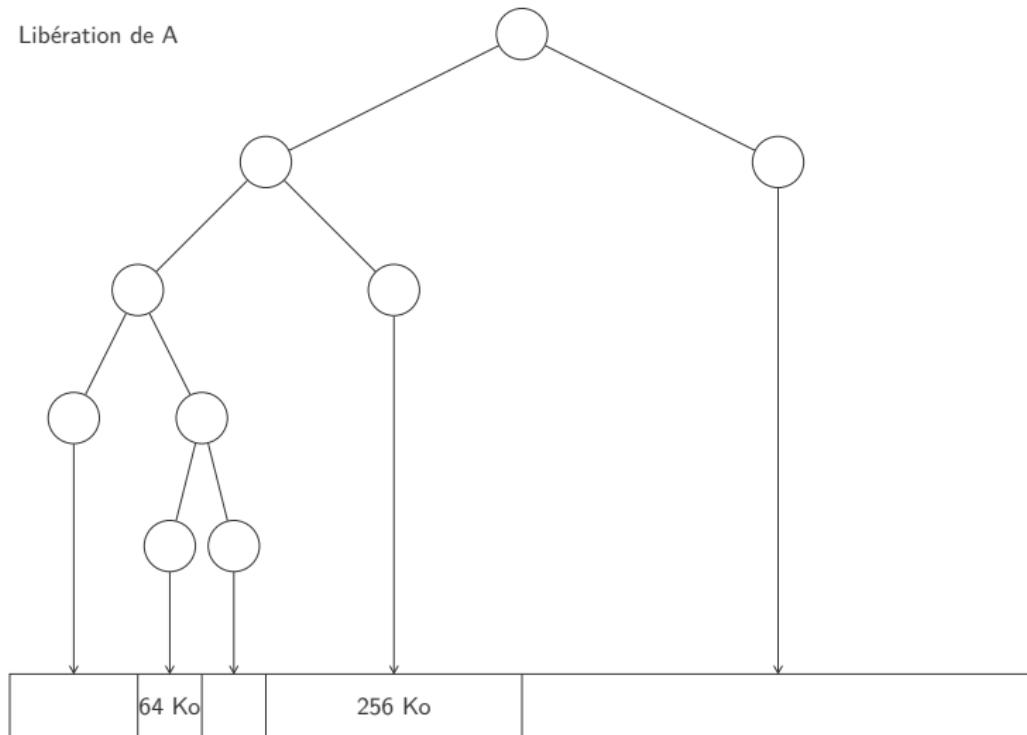
Exemple du buddy system



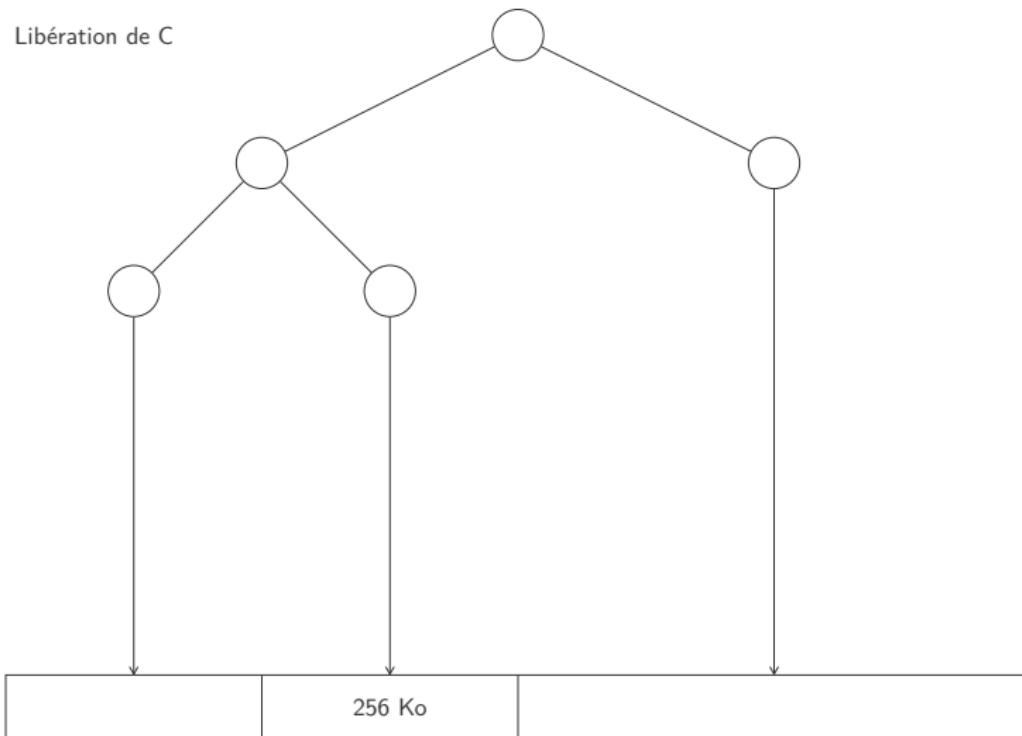
Exemple du buddy system



Exemple du buddy system



Exemple du buddy system



Mémoire virtuelle



Protection de la mémoire

- Protection mémoire dans un **système paginé** géré par des bits
 - Plusieurs bits associés aux cadres, stockés dans table des pages
 - Vérification de règles à chaque accès à la table des pages
- Plusieurs **types de protection** possibles
 - Bit read/write ou read-only, avec trap vers l'OS si violation
 - Bit valid/invalid indique si page dans espace logique processus
 - Page-table length register vérifie si bonne range d'adresses

Page partagée

- Possibilité de partager du **code commun** avec le paging

Par exemple pour un éditeur texte sur un time-sharing

- Page partageable si code contenu est **réentrant**

Code qui ne se modifie pas lui-même

- **Partage** éditeur texte entre 40 users (150 Ko code, 50 Ko données)

- Sans partage de page : $40 \times (150 + 50) = 8000 \text{ Ko}$
- Avec partage de page : $150 + 40 \times 50 = 2150 \text{ Ko}$

Fichier mappé en mémoire

- Lecture séquentielle d'un fichier depuis le disque dur
Nécessité de faire des appels systèmes et accès disque
- Gérer les accès E/S fichier comme des accès mémoire
Associer logiquement partie espace d'adresses virtuel à fichier
- Accès au fichier se base sur le mécanisme de défauts de page
 - Associer un bloc disque à une ou des pages en mémoire
 - Premier accès provoque défaut de page et chargement disque
 - Écriture pas directement répercutée sur le disque

Mémoire et E/S mappée

- Mémoire partagée implémentée par fichier mappé en mémoire
Communication inter-processus par fichier partagé
- Mapping d'E/S dans la mémoire
 - Permet d'échanger des données avec les registres E/S
 - Pratique pour périphériques avec temps de réponse rapide

Allocation mémoire kernel (1)

- Séparation de la **mémoire kernel** de celle des utilisateurs

Pages allouées lorsqu'un processus demande plus de mémoire
- **Pool de mémoire** libre différent pour le kernel
 - Requête mémoire du kernel de tailles variées et petites
 - Minimisation de l'espace perdu par fragmentation
 - Certains périphériques nécessitent pages contigües physique
- **Deux principales stratégies** de gestion de la mémoire libre

Buddy system ou slab allocation

Allocation mémoire kernel (2)

- Utilisation du **buddy system** pour gérer la mémoire kernel
 - Fragmentation, e.g. demande de 33 Ko nécessite bloc 64 Ko
 - Pas de garantie que moins de 50% de l'espace gaspillé
- Aucun espace perdu par fragmentation avec **slab allocation**
 - Un slab consiste en une ou plusieurs pages contigües physique
 - Plusieurs slabs par cache, un par structure de données kernel
 - Cache rempli d'objets, instance des données kernel
 - Allocation mémoire gérée par un slab allocator

Prepaging

- Pallier le grand nombre de **défaux de page initiaux**
Également lorsqu'un processus revient de swap
- Précharger des pages qui vont être nécessaires par **prepaging**
 - Prendre tout le contenu pour les petits fichiers
 - Précharger les pages du working set

Taille des pages

- Choix de la **taille des pages** peut être critique

Pas trop le choix pour machine existante

- Pas une **meilleure taille de page** dans l'absolu

- Invariablement des puissances de deux entre 2^{12} et 2^{22}
- Va influencer directement la taille de la table des pages
- La mémoire est mieux utilisée avec des pages de petites tailles
- La taille de page influence le temps de lecture/écriture
- Relation avec la taille des secteurs disque...

TLB reach

- Traductions d'adresses virtuelles résolues par TLB
 - Taux de succès des traductions mesuré par le hit ratio
 - Augmenté en même temps que la taille de la mémoire
- **TLB reach** mesure quantité de mémoire accessible
 - Nombre d'entrées de la table multiplié par taille des pages
 - Working set d'un processus devrait être dans le TLB

Verrouillage de page

- Nécessité de **verrouiller des pages** en mémoire

Notamment lorsque des opérations E/S sont faites en mémoire

- **Deux solutions** possibles pour éviter des problèmes

- Jamais exécuter opérations E/S en mémoire user, mais système
- Avoir un lock bit pour chaque cadre
- Peut perturber les algorithmes de remplacement de pages

- Plusieurs situations de **pinning** de pages en mémoire

- Tout ou une grande partie du kernel est lockée
- Une base de données peut vouloir pinner certaines pages

Gestion des entrées/sorties



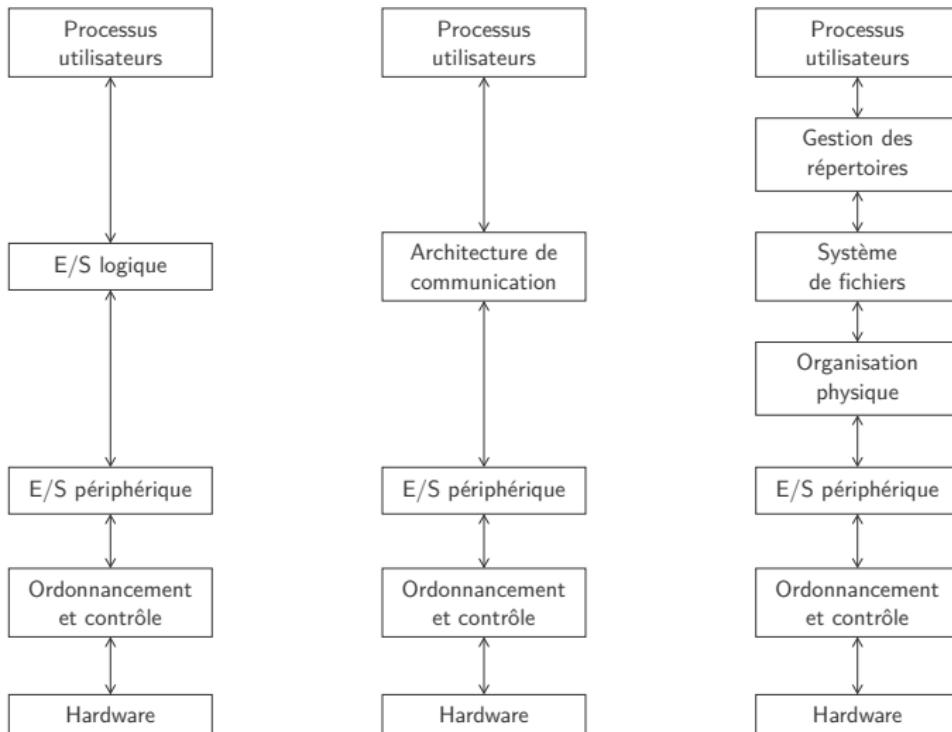
Système d'entrée/sortie

- Deux éléments clés pour le design du **système d'entrée/sortie**
 - **Efficacité** pour combattre le bottleneck mémoire
 - **Généralité** pour gérer les périphériques de manière similaire
- Maximiser le nombre de **processus sur le CPU**
 - File d'attente de processus en attente d'E/S
 - Swapping pour avoir plus de processus actifs

Structure logique du système E/S (1)

- **Trois principales couches** pour gérer périphérique local
 - E/S logique offre des fonctions haut niveau à l'utilisateur
 - E/S du périphérique avec contrôle de bas niveau
 - Ordonnancement et contrôle (interruptions, état...)
- Deux autres types de **périphériques plus spécifiques**
 - Périphérique de communication via un port
 - E/S de stockage secondaire avec un système de fichiers

Structure logique du système E/S (2)



Demande E/S

- Demande d'un **bloc** depuis le disque par un processus
 - Bloc de 512 octets à lire et placer en mémoire de 1000 à 1511
 - Le processus est mis en attente et ne peut pas être swappé
 - Blocage en RAM de toute la page contenant ces adresses

- Écriture d'un **bloc** par un processus vers le disque

Processus ne peut pas être immédiatement swappé

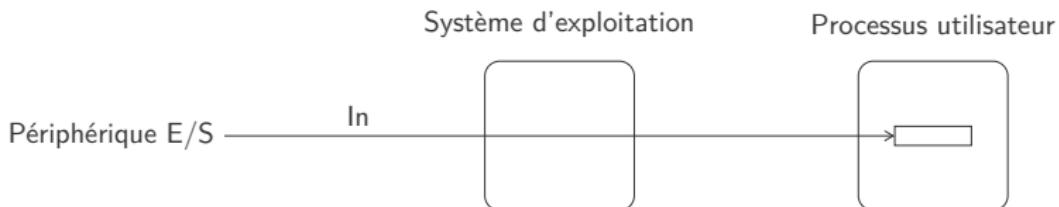
- Utilisation de **buffers** pour pallier ces problèmes

Lecture à l'avance et écriture retardée de données

Single buffering

- Utilisation d'**un buffer intermédiaire** au sein de l'OS
 - Simple zone mémoire de buffer dans la mémoire système
 - Lecture vers buffer, puis déplacement dans mémoire processus
- **Lecture à l'avance** de données depuis le disque

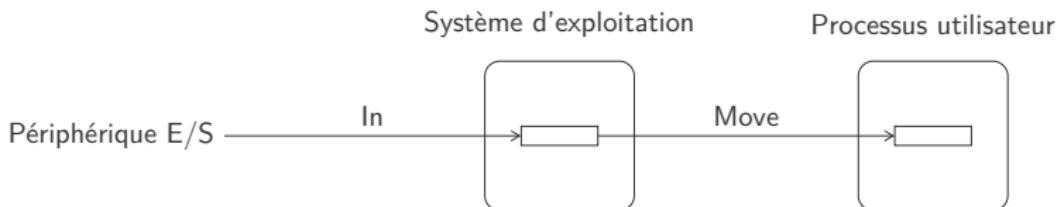
Traitement d'un bloc de données pendant que le suivant est lu



Single buffering

- Utilisation d'**un buffer intermédiaire** au sein de l'OS
 - Simple zone mémoire de buffer dans la mémoire système
 - Lecture vers buffer, puis déplacement dans mémoire processus
- **Lecture à l'avance** de données depuis le disque

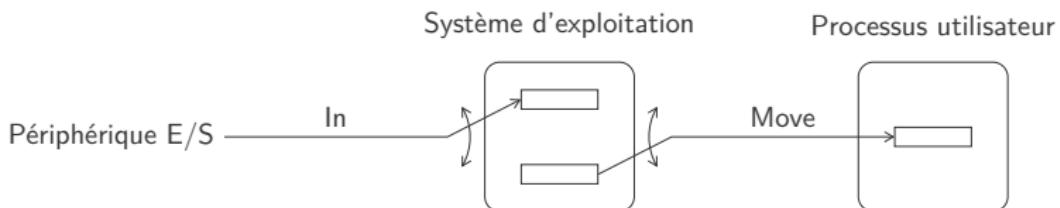
Traitement d'un bloc de données pendant que le suivant est lu



Double buffering

- Deux buffers systèmes par opération
 - Lecture par le processus d'un buffer, écriture de l'autre par l'OS
 - Meilleure performance que single buffering contre complexité
- Intéressant pour les périphériques **orientés blocs**

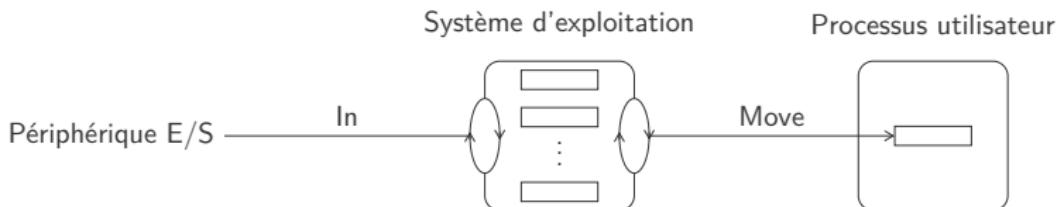
Pas de gain pour les périphériques orientés flux



Buffering circulaire

- Plusieurs buffers pour processus avec **salves E/S rapides**
 - Liste circulaire de buffers entre lesquels les opérations alternent
 - Similaire au buffer circulaire du producteur/consommateur
- Buffering permet d'adoucir les **pics de demandes E/S**

Toujours problème si un processus demande plus que possible



Comparaison de performances

- Comparaison **crue, mais informative** des performances
 - T temps requis pour récupérer un bloc depuis le périphérique
 - C temps de calcul entre des requêtes d'entrées
 - M temps nécessaire pour déplacer les données du buffer
- Comparaison des **performances** selon le type de buffering
 - Temps par bloc sans buffering $T + C$
 - Avec un simple buffer, temps réduit à $\max(C, T) + M$
 - Avec double buffer, plus que $\max(C, T)$



Gestion de l'espace libre

Table d'allocation du disque

- Deux éléments à gérer pour l'**allocation des fichiers** sur disque
 - Espace qui est actuellement alloué à des fichiers
 - Blocs libres sur le disque qui ne sont pas alloués à des fichiers
- **Table d'allocation du disque** retient l'espace libre

Plusieurs techniques pour stocker efficacement cette table

Bit vector

- Chaque bloc **représenté par un bit** (1 si libre et 0 si alloué)
 - Grande simplicité à stocker et manipuler
 - Pas efficace sauf si maintenu en permanence en mémoire

Disque de 1 To avec blocs de 4 Ko nécessite 256 Mo de mémoire
- **Recherche efficace** du premier bloc libre
 - 1 Trouver le premier mot non nul
 - 2 Scanner ce premier mot pour trouver la position du 1

(# bits par mot) × (# mots nuls) + (offset du premier bit à 1)

Liste chainée

- Garder la **liste complète** de tous les blocs libres

Faire une liste chainée avec des pointeurs de bloc en bloc

- Plusieurs **désavantages** possibles

- Le pointeur vers le premier bloc libre est crucial
- Traverser toute la liste coute cher en opérations E/S

- Quelques **variantes** possibles

- Garder les adresses des n premiers blocs libres dans le premier
- Retenir les suites de blocs libres contigus

Crédits

- <https://www.flickr.com/photos/seeminglee/3935911390>
- <https://www.flickr.com/photos/apolloscribe/25995779683>
- <https://www.flickr.com/photos/paulbence/279744368>
- <https://www.flickr.com/photos/gsfc/15284914831>