Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Operating Systems

Résumé TE01

Auteurs :
Marc Roten

Professeur :
Nicolas Schroeter

20 novembre 2018

Table des matières

1 THCHAP1 Introduction

Dans ce cours, nous allons nous intéresser à ce qui se passe dans les «entrailles» de la machine. Un système d'exploitation sert à gérer les ressources d'un

2 THCHAP2 Concept de base

Avec un niveau d'abstraction, on peut considérer le graphique suivant

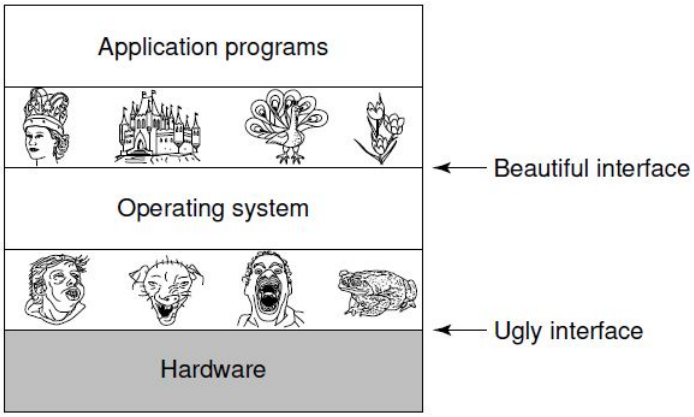
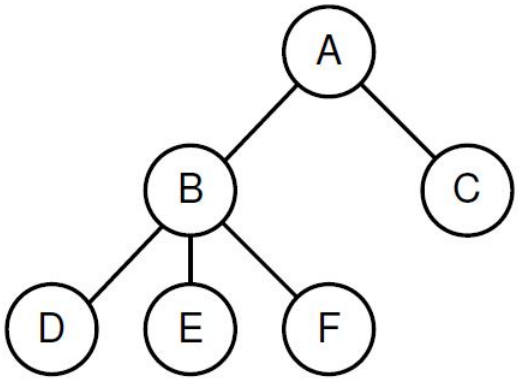


FIGURE 1 – Niveau d'abstraction

Quand un processus crée un autre processus, on peut le représenter sous forme d'arbre comme suit :



L'arbre des processus. Le processus A a créé les deux processus B et C. Le processus B a créé les processus D, E et F.

FIGURE 2 – Arbre de processus

2.1 Les appels systemes

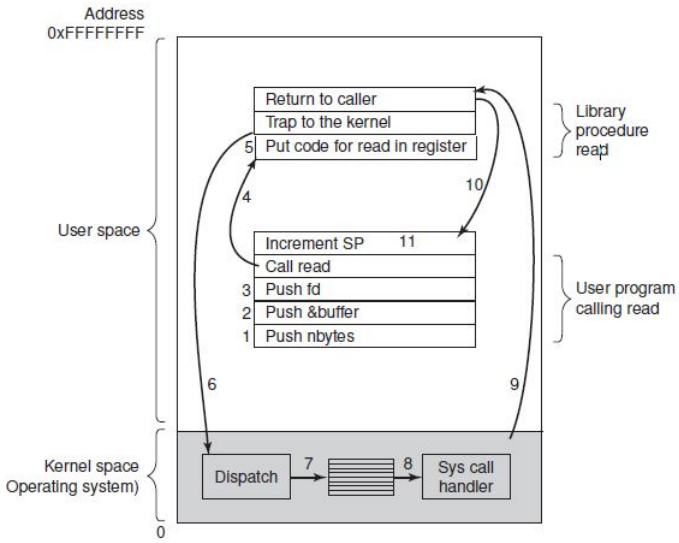


FIGURE 3 – appels systèmes simplifiés

2.1.1 Gestion des processus

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

FIGURE 4 – les différenta appels systemes

2.1.2 Gestion de fichier

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

FIGURE 5 – les différenta appels systemes

2.1.3 Gestion répertoires

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

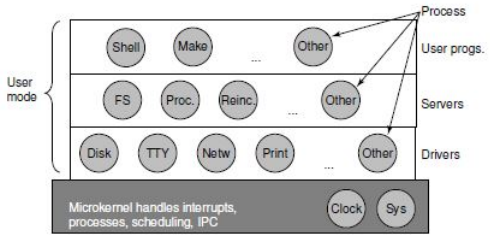
FIGURE 6 – les différenta appels systemes

2.2 Structure d'un OS

2.2.1 Système monolithiques

Systèmes tels que linux

2.2.2 MicroNoyaux



En moyenne, on considère qu'on code contient entre 2 et 10 bugs par 1'000 lignes de code. Le noyau Linux contient plus de 20'000'000 de lignes de code et Windows 10 plus de 50'000'000. En comparaison, MINIX contient moins de 15'000 lignes de code.

FIGURE 7 – MicroNoyaux

2.2.3 Modèle client serveur

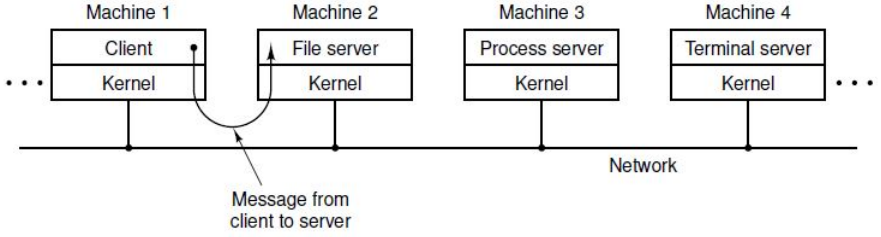


FIGURE 8 – client-serveur

2.2.4 Machines virtuelles

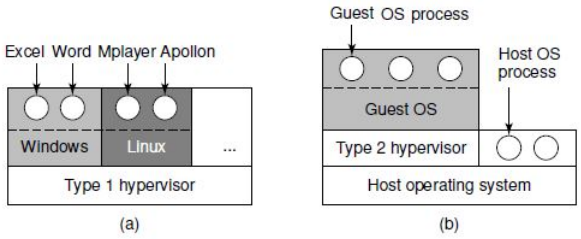


FIGURE 9 – VM

3 THCHAP3 Processus et threads

Les processus sont :

- le concept central d'un OS
- Abstraction d'un programme en cours d'execution
- Concurrence même avec un seul processeur

3.1 Définitions simplifiées

- **Le programme** est la recette de cuisine (algorithme)
- **Le processeur** est le cuisinier qui suit la recette
- **le processus** est l'action de cuisiner (activité)
- **Si le cuisiner se brûle, il remplace le livre de cuisine par celui des premiers secours et se soigne**

3.2 Naissance d'un processus

Les evénements qui induisent la création de processus sont :

- initialisation du systeme
- Execution d'un appel système de création de processus en cours d'execution (Fork)
- Requête utilisateur sollicitant éa création de nouveaux processus.
- Lancement d'un travail en traitement par lots (batch)

3.3 Mort d'un processus

- Arrêt normal, volontaire
- arrêt pour erreur, volontaire
- Fatal error, involontaire
- autre process qui arrête notre processus

3.4 Machine d'état processus

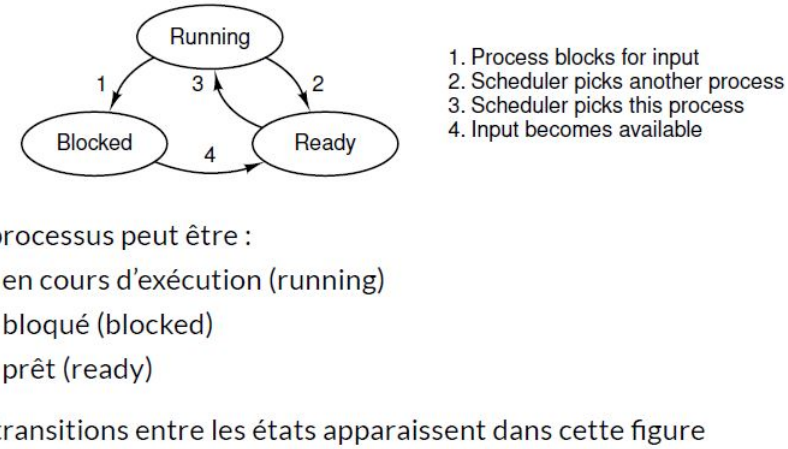


FIGURE 10 – FSM Processus

Le Scheduler s'occuppe de dire quel process travaille et quand le process travaille. C'est en quelque sorte le chef d'orchestre dans tout ce bordel.

3.5 Utilisation Processus / process en mémoire

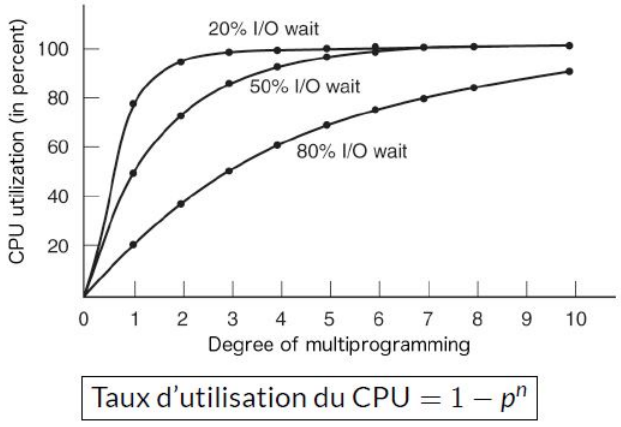


FIGURE 11 – FSM Processus

3.6 Les threads

3.6.1 Schéma conceptuel Threads

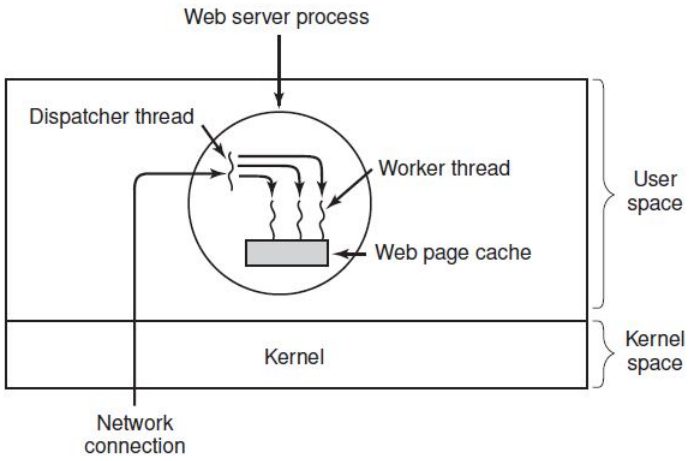


FIGURE 12 – Concept threads

3.6.2 Code Threads

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

Thread dispatcher

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

Thread worker

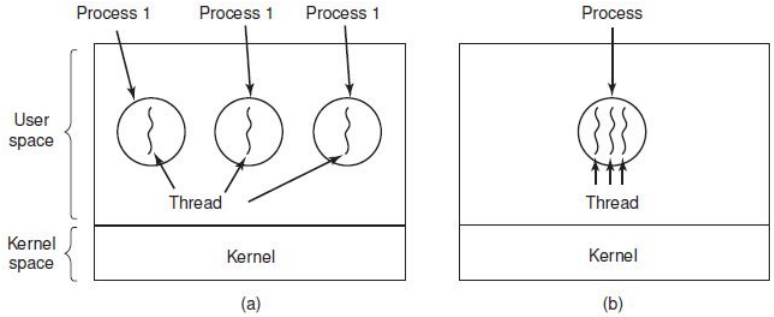
FIGURE 13 – Exemple de threads

3.6.3 Manière de créer un serveur

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

FIGURE 14 – How to build a server and get a life

3.6.4 MultiThread



3 processus avec 1 thread chacun
1 processus avec 3 threads

FIGURE 15 – le fameux multi srède

3.7 Conditions pour eviter les conditions de concurrence

- deux process ne doivent pas se trouver dans leurs sections critiques.
- Il ne daut pas faire de supposition quant à la vitesse ou au nombre de processeur mis en oeuvre
- Aucun processus s'executant à l'intérieur de la section critique ne doit bloquer d'autres processus.
- Aucun processeur ne doit attendre indéfiniment pour pouvoir entrer dans sa section critique.

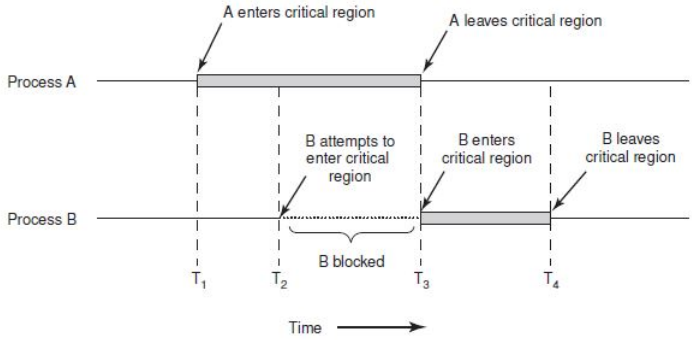


FIGURE 16 – La Zone critique, et si on connait pas, on trikite pas

4 THCHAP4 L'ordonnancement des processus

C'est quand qu'on ordonnance Jamie ?

Et bien c'est très simple, on le fait :

- Lors de la création d'un process, parent ou enfant qui prend le contrôle ?
- Lorsqu'un process se termine
- Lorsqu'un process se met en attente d'un event sur une I/O
- Lors d'une interruption faite par un periphérique
- Lors d'une interruption par l'horloge pour des processus préemptif

4.1 Système d'exploitation préemptif

C'est la capacité qu'a un systême d'exploitation multiTache à executer ou stopper des tâches planifiées en cours.

4.2 Catégories d'ordonnancement d'algorithmes

- **Non préemptifs :** traitement par lot (batch) non-préemptif, ou alors avec une très grande période.
- **Interactifs :** interactifs. préemptif avec courte période
- **Collaboratifs :** temps réel. Préemption non nécessaires. les processus sont connus et collaborent.

4.3 Objectifs : ordonnancement des process

- **Equité :** attribuer à chaque processus un temps processeur équitable
- **Application de la politique :** faire en sorte que la politique soit bien appliquée, **MAKE SCHEDULER GREAT AGAIN**
- **Equilibre :** faire en sorte que toutes les parties du systême soient occupées.

4.3.1 traitement par lots (batch)

- **Capacité de traitement :** Optimiser le nombre de jobs à l'heure
- **Délai de rotation :** réduire le délai entre soumission et achèvement
- **Utilisation du CPU :** Faire en sorte que le processeur soit occupé en permanence.

4.3.2 systêmes interactifs

- **Temps de réponse :** répondre rapidement aux requêtes
- **Proportionnalité :** satisfaire aux attentes des utilisateurs

4.3.3 Systême en temps réel

- **Respecter les délais :** Eviter de perdre des données
- **Prévisibilité :** Eviter la dégradation de la qualités dans les systémes multimédias

4.3.4 Mashup ALL

All systems
Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems
Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems
Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems
Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

FIGURE 17 – Mashup all

4.4 Traitement par lots

- FIFO, premier arrivé, premier servi **NO PREEMPTION**
- Shortest job first
- Shortest remaining time next

4.4.1 Pourquoi les plus courts en premiers

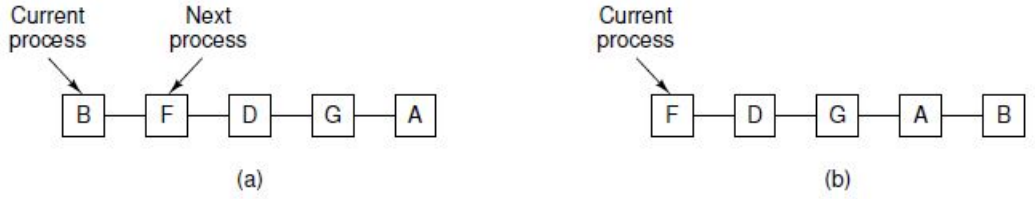
Dans ce cas ci

- on admet que le temps de traitement des processus est connu à l'avance par l'ordonnanceur.
- Le but est d'optimiser le délai de rotation so tous les jobs sont soumis en même temps

4.5 Les systêmes interactifs

- L'ordonnancement est de type **Round Robin Scheduling**
- Ordonnancement par priorités **Priority Scheduling**
- Files d'attentes multiples
- Processus le plus court apres**Shortest process next**
- ordonnancement garanti
- ordonnancement par tirage au sorte
- ordonnancement équitable

4.6 Round Robin



Ordonnancement de type «tourniquet» (round robin scheduling).

(a) La liste des processus exécutables

(b) La liste des processus exécutables après que B a utilisé son quantum

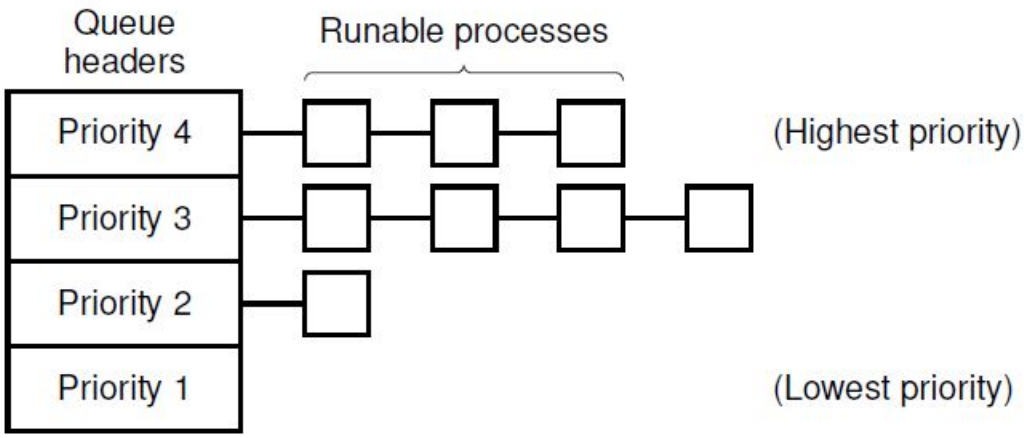
FIGURE 18 – Round robin

Le «quantum» est l'intervalle de temps pendant lequel un processus est autorisé à s'exécuter. Après avoir épuisé son quantum, le processus doit faire place à un autre processus.

- Un quantum court améliore le temps de réponse
- Un quantum long améliore la performance
- Le quantum est généralement choisi entre 10 mS et 1000 mS
- Les processeurs plus rapides permettent de réduire le quantum

FIGURE 19 – Round robin,explication

4.7 Ordonnancement par priorité



Ordonnancement par priorités avec 4 catégories de priorités

FIGURE 20 – Par priorité

4.8 File d'attentes multiples

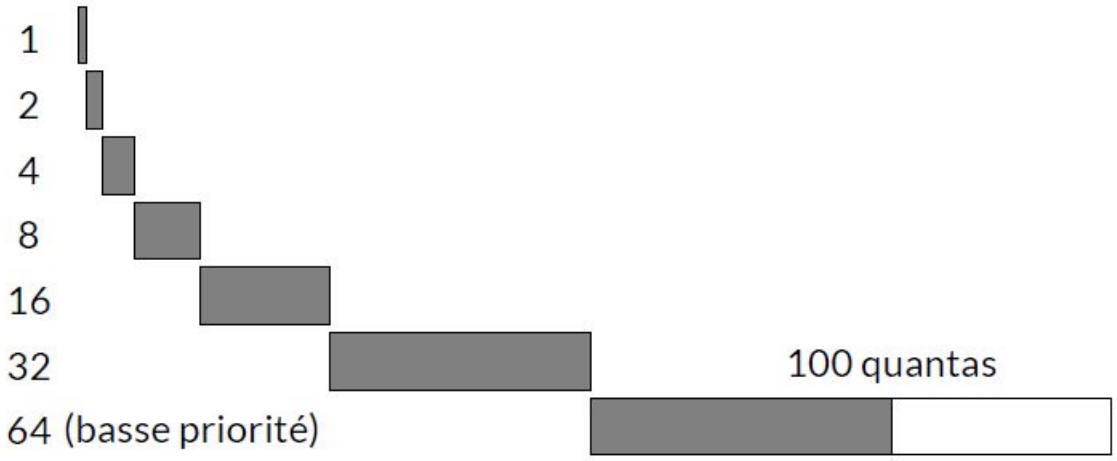


FIGURE 21 – files d'attentes multiples

4.9 Shortest process next

- Estimations fondées sur les comportements antérieurs
- Calcul de vieillissement
  - T_0
  - $\frac{T_0}{2} + \frac{T_1}{2}$
  - $\frac{T_0}{4} + \frac{T_1}{4} + \frac{T_2}{2}$
  - $\frac{T_0}{8} + \frac{T_1}{8} + \frac{T_2}{4} + \frac{T_3}{2}$

FIGURE 22 – Shortest process next

4.10 Ordonnement Garanti

- Si n processus s'exécutent, chaque processus récupère $\frac{1}{n}$ cycle du CPU
- Le système effectue le suivi du temps processeur dont chaque processus a bénéficié depuis sa création

FIGURE 23 – Shortest process next

4.11 Par tirage au sort

- Chaque processus reçoit un ou plusieurs «billets de loterie»
- L'ordonnanceur tire au sort un billet au hasard
- Un processus nouvellement créé pourrait obtenir plus de billets que les anciens processus
- Des processus coopératifs peuvent échanger des billets s' ils le souhaitent

FIGURE 24 – Shortest process next

4.12 Ordonnancement équitable

- Cet algorithme tient compte des propriétaires (utilisateurs) des processus
- Chaque **utilisateur** reçoit la même part du CPU
- Si 2 utilisateurs ont droit à 50% du temps de CPU, ils les obtiendront, quel que soit le nombre de processus qu'ils ont démarrés

FIGURE 25 – Shortest process next

5 THCHAP6 Gestion de la mémoire

Solution pour les programmes plus gros que la taille de la mémoire

5.1 Le Swapping

L'allocation mémoire change au gré des processus qui viennent en mémoire et qui la quittent. Les zones grisées indiquent que la mémoire est inutilisée. Comme illustré ci dessous.

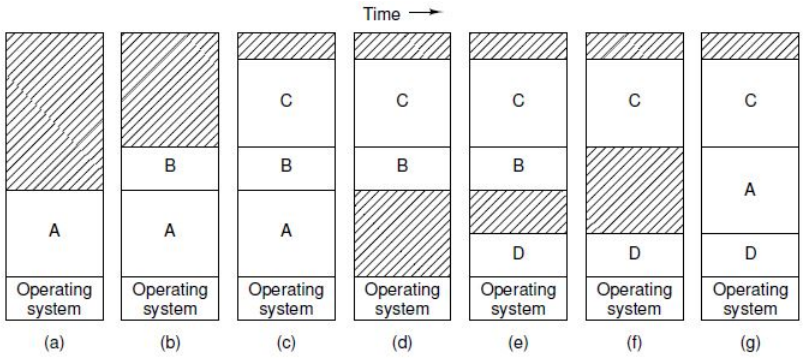


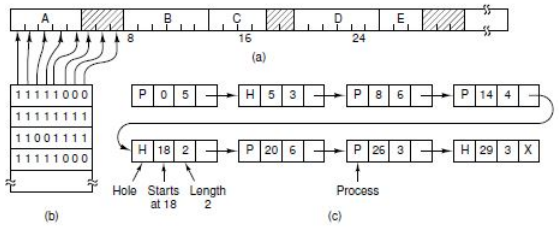
FIGURE 26 – Swapping

5.1.1 Problème induits

- Premier problème : La mémoire devient fragmentée
- Solution : le compactage de la mémoire
- Mais cette solution est gourmande en ressources (pour compacter 1 GiB, il faut compter environ 5 secondes)
- Deuxième problème, la taille de la mémoire utilisée par les processus n'est pas fixe (dynamic data segment)
- Solution : allouer plus de mémoire que nécessaire

FIGURE 27 – problèmes

5.2 Bitmap



(a) Une partie de la mémoire avec 5 processus et 3 trous. Les petites marques verticales indiquent les unités d'allocation. Les zones grisées (valeur 0 dans le tableau des bits) sont des zones libres.

(b) Le tableau de bits correspondant.

(c) Les mêmes informations sous forme d'une liste chaînée.

FIGURE 28 – Bitmap

- La taille de la table des bits est inversement proportionnelle à la taille des segments (qui va de quelques Bytes à quelques kiBytes)
- Avec des petits segments, la table est grande, mais on a peu de gaspillage dans les blocs
- Avec des grands segments, la table est petite, mais on a beaucoup de gaspillage dans les blocs
- Avec des segments de 4 Byte, 1 bit représente 32 bit. La table des bits prendra $\frac{1}{33}$ de la mémoire totale, soit environ 3%
- Problème : la recherche d'un bloc libre n'est pas efficace (il faut chercher une séquence de «0» dans la table)

FIGURE 29 – Bitmap specs

5.3 Linked List

- Chaque nœud contient un bit qui indique si le bloc correspond à un processus ou à un «trou», l'adresse de début du bloc, ainsi que la taille du bloc.
- Habituellement, l'entrée dans la table des processus contient un pointeur vers le début de la liste
- On peut simplifier la gestion de la liste en utilisant un double chaînage (mais on utilise plus de mémoire)

FIGURE 30 – Linked List specs

Avec les listes chaînées on peut implémenter différents algorithmes :

- **First Fit :** on itère depuis le début de la liste et on prend le premier qui passe
- **Next Fit :** On itère depuis l'endroit pointé actuellement et on prend le premier qui vient
- **Best Fit :** On itère dans toute la liste et on prend le meilleur
- **Worst Fit :** On recherche le bloc le plus grand parmis toute la liste

5.4 Best and WorstFit

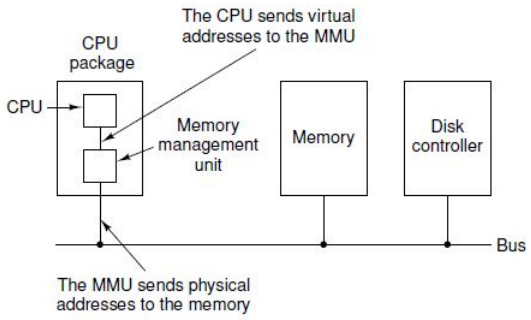
Si on doit utiliser ces types d'algorithmes, il est judicieux de trier la liste par taille de blocs

5.5 Concept de mémoire virtuelle

- La mémoire virtuelle permet la multiprogrammation.
- La memoire virtuelle permet d'avoir **Plus de mémoire physique que virtuelle**
- L'espace d'adressage est découpé en pages (suites d'addresses contigues)
- chaque programme a son espace d'adressage (de 0 à n)

5.6 La pagination

C'est la MMU (memory management unit) qui s'en charge.



Localisation et fonction d'une MMU. Ici, la MMU est montée comme une partie intégrante du CPU. Cette configuration est usuelle de nos jours. Toutefois, elle pourrait se trouver dans un circuit séparé, comme c'était le cas il y a quelques années.

FIGURE 31 – MMU

Ci-dessous Exemple trivial de mapping entre physique et virtuel.

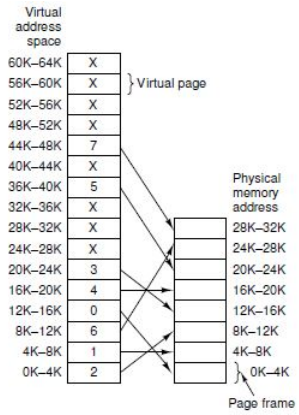


FIGURE 32 – Mapping virtuel <—> physique

- Les pages de la mémoire virtuelle sont mappées sur la mémoire physique.
- Il n'est pas nécessaire d'avoir toutes les pages mappées pour executer un processus.
- La multiprogrammation profite des I/O générées par le Swapping pour attribuer les ressources CPU à d'autres processus.

5.6.1 CPU essaie acceder à page virtuelle occupée

Si le CPU essaye d'accéder à une adresse d'un page virtuelle absente :

- Une exception de type **«page fault»** est produite
- Le système d'exploitation sélectionne un cadre de page peu utilisée
- il sauve le contenu ce cette page peu utilisée sur le disque
- Charge la nouvelle page depuis le disque
- Modifie le «mapping»
- Et reprend l'instruction

FIGURE 33

5.7 Speedup the pagination

Tout système de pagination doit considérer ces deux éléments :

- La correspondance de l'adresse virtuelle vers l'adresse physique doit être rapide, car elle est utilisée à chaque accès mémoire (donc très souvent).
- Si l'espace d'adressage est grand, la table des pages sera grande.Pour un espace d'adressage sur 32 bit et des pages de 4 kiByte, il y a 1M de pages.

- De plus, chaque processus à besoin de sa propre table de pages !

5.8 La TLB, translation lookaside buffer

Les références dans la table sont groupées et seule une petite partie de la table est réelement utilisée.

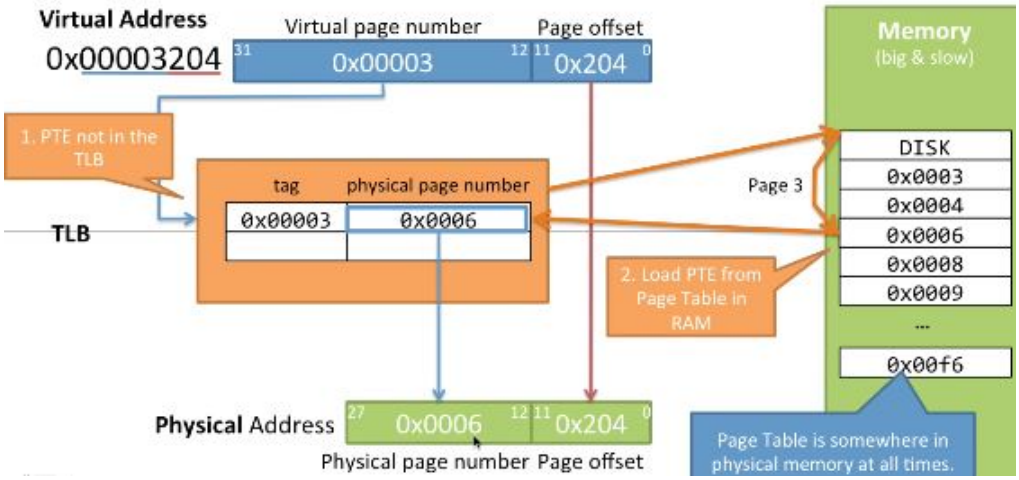


FIGURE 34 – With an empty TLB

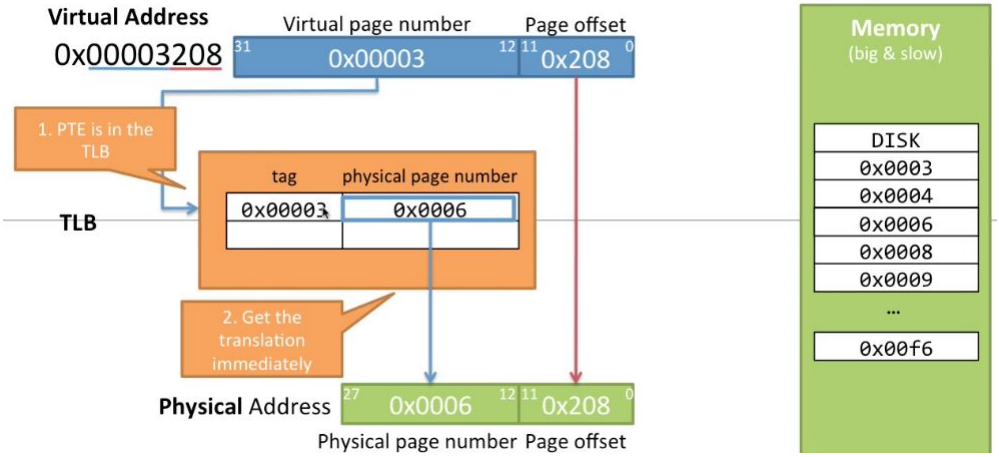


FIGURE 35 – TLB Hit

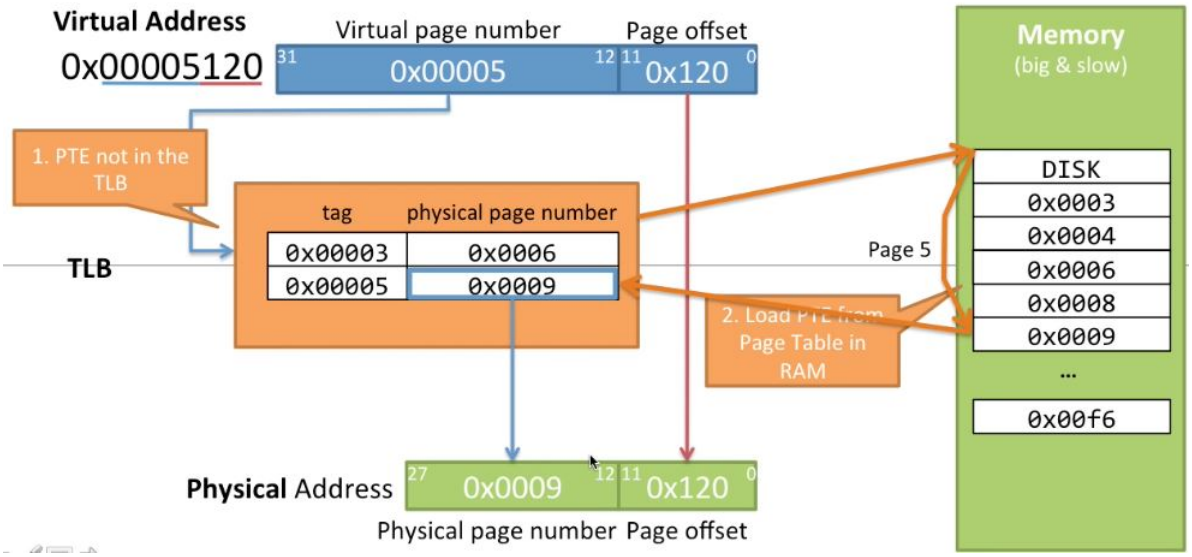


FIGURE 36 – TLB Miss

6 Conclusion

Si vous avez aimé mon résumé, faites un git clone de mon Git. Suivez moi sur gitlab.forge.heia-fr.ch github and iLoveFreeSoftware.com.