Système d'exploitation

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard
Jean-louis.lanet@unilim.fr

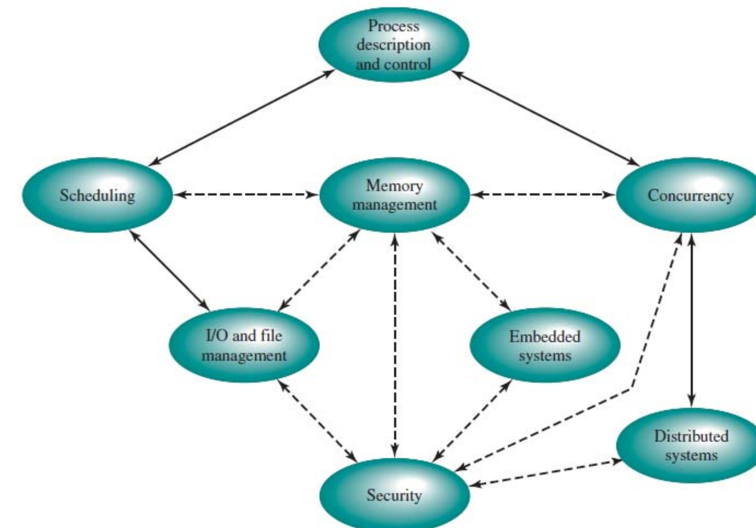
Objectif du cours

- Découvrir les mécanismes internes des systèmes d'exploitation, gestion mémoire, processus etc.
- Maitriser les algorithmes utilisés, se préparer aux cours du M2 Cryptis et ISICG,
- De la théorie à la pratique, des expérimentations en TP,
- Les cours ne sont pas obligatoires (sauf pour les boursiers) ni les TD mais les TP si.
  - Une absence => ABI au CC

Plan du cours

- 6 séances de 2x1,5 heures, 13,5 h de TD, 7,5h de TP et votre projet.
- Trois intervenants J.-L. Lanet, G. Bouffard et D. Pequegnot
- Contenu :
  1. Architecture des systèmes,
  2. Gestion des processus
  3. Mémoire,
  4. Gestionnaire d'IO,
  5. Sécurité,
  6. Ouverture…
- Supports disponibles sur FOAD

Relation des éléments du cours



Objectif des TD/TP

- TD (9 séances) : reprendre les mécanismes vus en cours et les appliquer à des problèmes de même nature, ou bien explorer des concepts laissés de côté en cours.
- TP (5 séances) : éditer, compiler puis exécuter des programmes, et aussi utiliser la documentation.

Évaluation

- Première session
  - Un partiel (TP) 1h30 : 25%
  - Un écrit 1h30 : 75%
- Seconde session
  - Écrit 2h 100%

Document personnel autorisé:

- *Votre voisin n'est pas document personnel*
- *Votre téléphone, votre pda, votre pc portable non plus*
- *Le support de cours, des notes manuscrites sur papier BLANC. OUI.*
- *Tout le reste est **interdit***

Ressources

- De très nombreuses ressources sont disponibles sur le net,
- Les livres indispensables :
  - LE **Tanenbaum**, Modern operating systems
  - Operating System: Concepts and Techniques, Naghibzadeh
  - Les systèmes d'exploitation , Samia Bouzefrane

Any question ?

Système d'exploitation
Architecture

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard
Jean-louis.lanet@unilim.fr

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d'exécution
- Exceptions & Interruptions
- Les différentes mémoires

Le rôle du système

Système d'exploitation :

- Programme dont le rôle est de gérer le matériel
- Permet de lancer les programmes des utilisateurs
- Plus qu'un simple moniteur et que le BIOS
- Offre une couche d'abstraction vis-à-vis du matériel
- Gestion des fichiers, de la mémoire, du réseau, du multi-tâches, de l'IHM...

Le rôle du système

- Système d'exploitation :
  - Années 60, premiers OS sur des mainframes : traitement par lot,
  - temps partagé
  - Années 70, Unix : multi-utilisateur, réseau
  - Années 80 et 90, PC : interactivité, plug & play
  - Années 2000, OS embarqués : PDA, téléphones portables...
- Mais aussi voitures (environs 60 CPU / voiture) , appareil photo, routeur ADSL, chaîne HIFI
- Mais encore : Internet, systèmes distribués, systèmes temps réels, multiprocesseurs

Par exemple…

Quelques OS

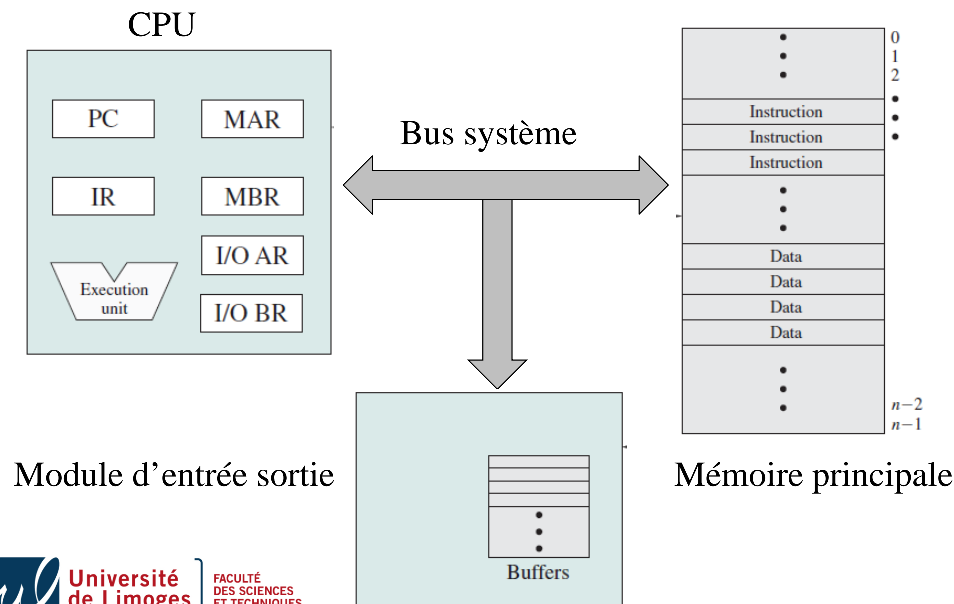
- Unix , Linux
- Windows NT, 2000, XP, Vista, 07
- MAC OS X
- Windows CE, pocket PC
- Palm OS,
- RT-Linux, µCLinux
- VxWorks, pSos
- JavaOS

Les composants

- Un système est composé de trois sous systèmes: le CPU; la mémoire et les composants d'entrée sortie.
  - Le processeurs exécute les calculs généralement sur des registres internes,
  - La mémoire principale stocke les données et les programmes avant de les transférer en mémoire secondaire,
  - Les entrées sorties transfèrent les données de la mémoire principale vers les mémoires secondaires,
  - Les bus assurent les communication entre ces entités.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Composants



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Les cœurs de microprocesseurs

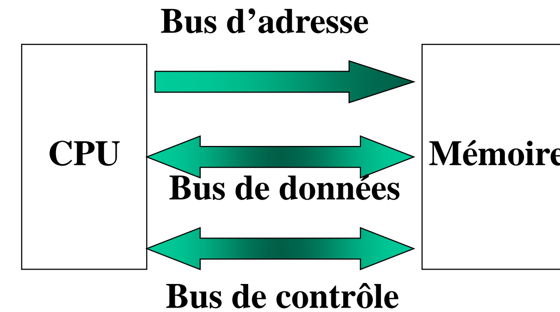
- Fin des processeurs en tranche, ils sont conçu en monochip ou agrégation de chip,
- Le GPU (Graphical Processor Unit) exécutent de manière efficace des calculs sur de multiples données,
- DSP (digital Signal Processor) sont capables d'exécuter des calculs sur des flux de signaux,
- SoC (System on Chip) incluent sur la même puce des composants différents (mémoire, ligne d'IO,…)

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

CPU

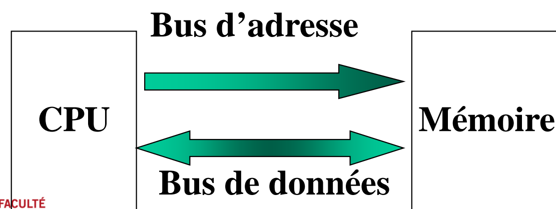
- PC: adresse prochaine instruction à exécuter
- SP: adresse du sommet de pile
- Registres généraux de calcul
- Registre (ou file) de registre de décodage d'instruction,
- Registre d'état
  - Bits de conditions (Z, LT, GT, etc.)
  - Modes d'exécution
    - Niveau de protection (user/supervisor)
    - Adressage physique/virtuel
  - Masque des interruptions

Interaction CPU / mémoire



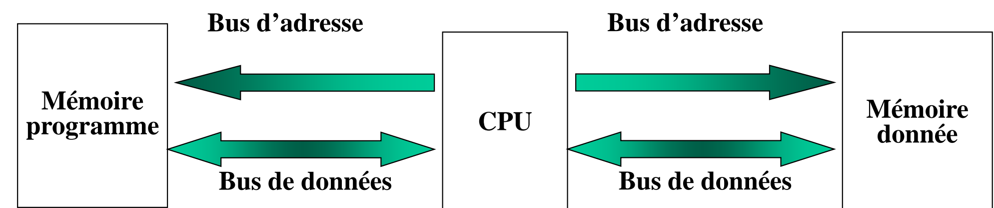
Architecture Von Neuman

- Mémoire contient instructions et données
- CPU lit les instructions depuis la mémoire
- Chargement d'un programme en mémoire
  - Instructions manipulées comme des données
- Programmes peuvent s'auto modifier.
  - Sauf certaines parties du Bios (secure bootloader)



Architecture Harvard

- Deux types de mémoire
  - Instructions et données
  - Accès en parallèle aux instruction et aux données
  - Données sont accédée plus fréquemment que les instructions,
- CPU Idem à Von Neuman
  - Ne peut pas utiliser du code qui se modifie lui-même,
  - Plus grand débit accès mémoire



Architecture

- PC: adresse prochaine instruction à exécuter
- SP: adresse du sommet de pile
- Registres généraux de calcul
- Registre d'état
  - Bits de conditions (Z, LT, GT, etc.)
  - Modes d'exécution
    - Niveau de protection (user/supervisor)
    - Adressage physique/virtuel
  - Masque des interruptions

RISC vs. CISC

- RISC (Reduced Instruction Set Computer)
  - Choisit les instructions et les modes d'adressage les plus utiles
  - Peuvent être réalisée en hardware directement
- CISC (Complex Instruction Set Computer)
  - Choisit les instructions et les modes d'adressage qui rendent l'écriture de compilateurs plus simple
  - Nécessite l'utilisation de micro-code

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

RISC vs. CISC

- Avantages CISC
  - Réduit le nombre d'instructions d'un programme
- Inconvénients CISC
  - Temps d'exécution (des instructions) plus long
  - Processeur plus complexe et donc plus lent

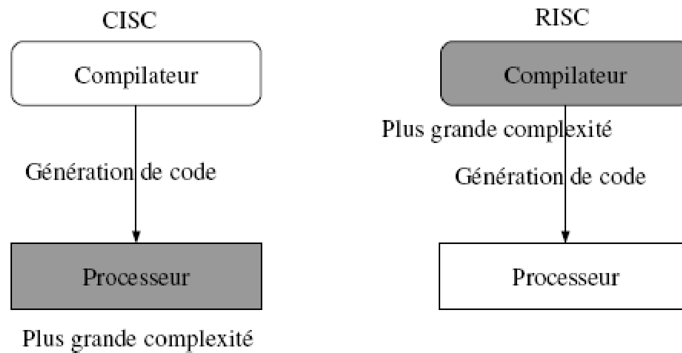
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

RISC vs. CISC

- RISC:
  - Place disponible sur la puce pour de nombreux registres généraux et des caches mémoire
  - écriture de compilateur et programmation assembleur difficile
- CISC:
  - Peu de place sur la puce: peu de registres et caches de petite taille
  - écriture de compilateur et programmation assembleur facile

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

RISC vs. CISC



Architecture RISC

- Nombre d'instructions réduit
  - Une instruction par cycle
  - Instructions de taille fixe => facile à pré-charger
- Pipelines
  - Traitement d'une instruction est décomposé en unités élémentaires
  - Exécution de ces unités en parallèle

Architecture RISC

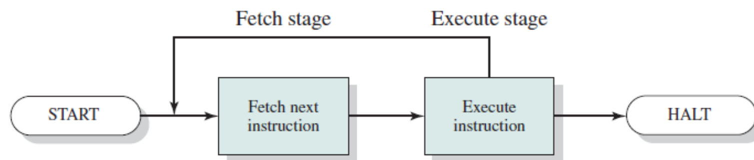
- Registres
  - Grand nombre peu spécialisés (adresses ou données)
- "Load-Store"
  - processeur opère seulement sur les données des registres
  - Instructions spécifiques pour lire un mot mémoire dans un registre, et pour écrire un registre en mémoire.

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d'exécution
- Exceptions & Interruptions
- Les différentes mémoires

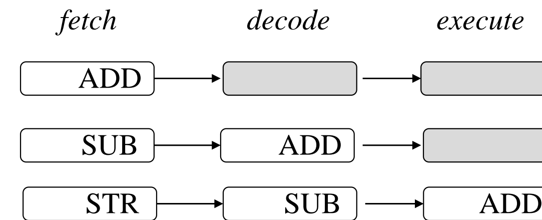
Exécution d'une instruction

- Cycle d'exécution:
  - Lecture (*fetch-stage*) des instructions de la mémoire,
  - Exécution (*execution-stage*) de l'instruction.
  - Le PC est incrémenté après le fetch,
  - L'instruction est chargée dans le registre d'instruction (IR) et il est décodé,
  - Quatre type d'instruction : Transfert mémoire-registre, Transfert processeur entrée sortie, Calcul de donnée ou Control



RISC et pipe line

- Exemple du ARM7 : 3 niveaux
  - Fetch: charge une instruction depuis la mémoire
  - Décode: identifie l'instruction à exécuter
  - Exécute: exécute l'instruction et écrit le résultat dans le registre



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Pipeline

- Instructions de branchement
  - vidage (***flush***) du pipeline
  - **Pourquoi** ?

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

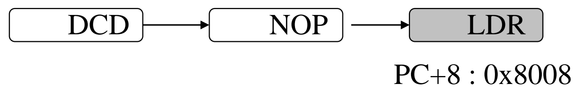
Pipeline et PC

- Le PC (au niveau *execute*) pointe sur l'adresse de l'instruction courante + n octets
  - Important quand on calcul un offset relatif,
  - Exemple
    - 0x8000 LDR pc,[pc,#0]
    - 0x8004 NOP
    - 0x800D DCD JumpAddress
- Quelle adresse pointe le PC lorsque s'exécute le LDR ?

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Pipeline et PC

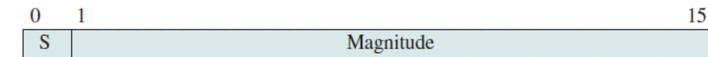
- Le PC (au niveau *execute*) pointe sur l'adresse de l'instruction courante + octets
  - Important quand on calcul un offset relatif,
  - Exemple
    - 0x8000 LDR pc,[pc,#0]
    - 0x8004 NOP
    - 0x800D DCD JumpAddress



Exécution d'une instruction

- Soit un processeur 16 bits imaginaire,
  - Des registres : PC, MAR (Memory Adress Register, IR (Instruction Register) et AC un accumulateur
  - Au moins trois opcodes
    - 0001 charge AC par un contenu mémoire
    - 0010 stocke AC dans la mémoire
    - 0101 additionne dans AC depuis la mémoire
- Un format d'instruction

- Le format d'un entier



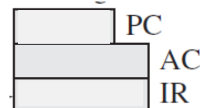
Exécution d'une instruction

- Soit le programme suivant situé en 0x300:
  - 1940, 5941 et 2941

Mémoire programme

300	1 9 4 0
301	5 9 4 1
302	2 9 4 1

Registres du processeur



Mémoire donnée

940	0 0 0 3
941	0 0 0 2

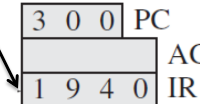
Exécution d'une instruction (1)

Étape de *Fetch*

Mémoire programme

300	1 9 4 0
301	5 9 4 1
302	2 9 4 1

Registres du processeur



Mémoire donnée

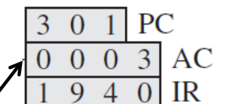
940	0 0 0 3
941	0 0 0 2

Étape de *Execute*

Mémoire programme

300	1 9 4 0
301	5 9 4 1
302	2 9 4 1

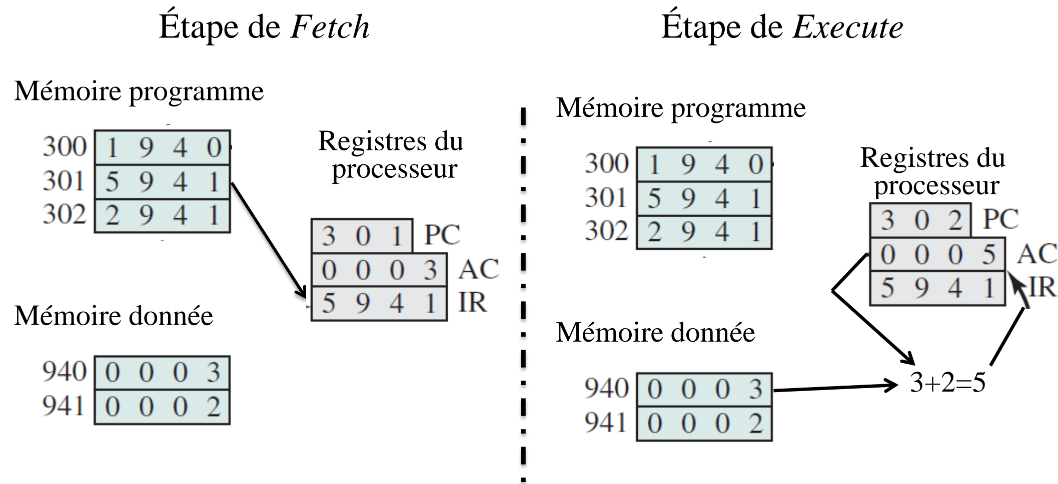
Registres du processeur



Mémoire donnée

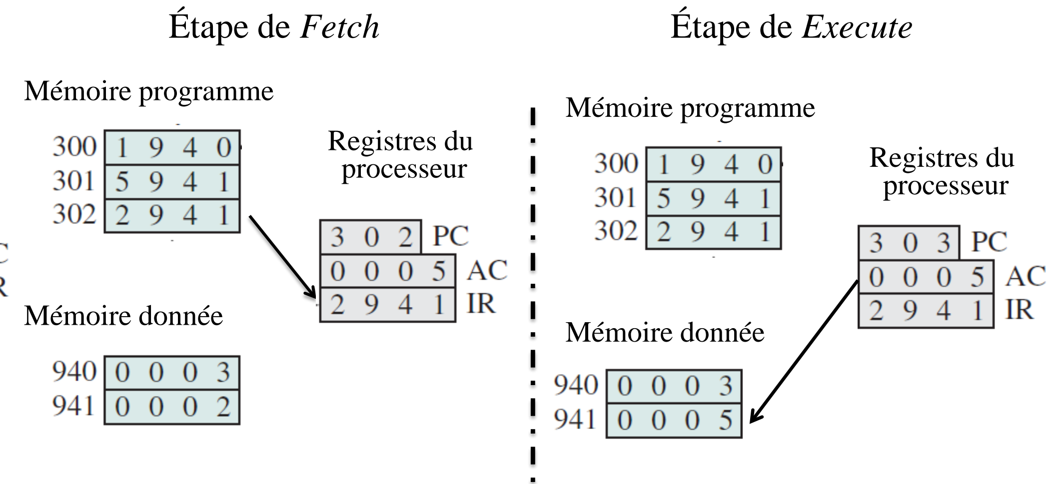
940	0 0 0 3
941	0 0 0 2

Exécution d'une instruction (2)



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Exécution d'une instruction (3)



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Exercice

- Supposons que le processeur dispose de deux instructions d'accès I/O
  - 0011 Charge AC depuis I/O
  - 0111 Stocke AC dans I/O
- Dans ce cas, l'adresse sous 12 bits identifie un périphérique particulier. Montrez l'exécution du programme suivant en utilisant le format précédent:
  - Charge AC depuis le périphérique 5.
  - Additionne son contenu avec la mémoire 940.
  - Stocke le résultat dans le périphérique 6.
- Hypothèse la valeur dans le périphérique 5 est 3 et en 940 est stockée la valeur 2.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d'exécution
- Exceptions & Interruptions
- Les différentes mémoires

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Mode Superviseur

- Accès aux ressources protégées
  - Registres d'I/O, de gestion de la mémoire
  - Registre contenant le mode d'exécution
  - Toute la mémoire physique, tous les périphériques
- Droit d'exécuter les instructions *privilégiées*
  - Masquage des interruptions
  - Changement de mode d'exécution
  - Instructions d'I/O
- Processeur démarre en mode superviseur

Mode Utilisateur

- Droits restreints
  - Accès aux registres généraux
  - Exécution des instructions standard
- Passage superviseur -> utilisateur
  - Contrôlé par OS
  - Lancement d'un programme
- Passage user -> superviseur
  - Instruction *svc* (supervisor call)
  - Opération invalide ou non autorisée =>exceptions

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Supervisor call

- Utilisé pour invoquer un service du système
  - service id. (open, write, ...) dans un registre
  - service args (pathname, fd, ...) dans registre(s) ou dans la pile utilisateur
  - résultat / code erreur dans registre(s)
- Synchrone avec (processus / thread) appelant
  - relatif au contexte système appelant
- Appels système fournis par fonctions de bibliothèque du langage (libc)

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Modèle du programmeur ARM

- 1 mode utilisateur non privilégié
- 5 modes système, privilégiés :
  - Interrupt (standard) _irq
  - Fast interrupt _fiq
  - Abort _abt
  - Indéfini _und
  - Software interrupt _svc
  - + tâches Système

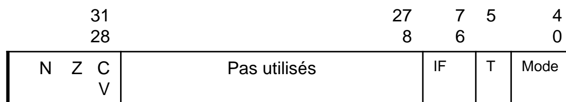
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Des registres (mode User)

R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15

- 16 Registres de 32 bits
- R0-R10, R12 : généraux
- R11 (fp) Frame pointer
- R13 (sp) Stack pointer
- R14 (lr) Link register
- R15 (pc) Program counter

- Current Program Status Register

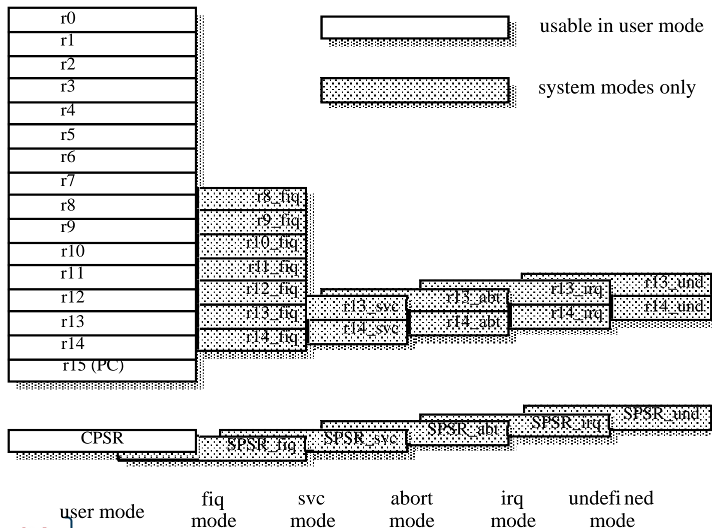


Modèle du programmeur ARM

- Associés aux divers modes de travail, des registres différents sont disponibles :
- **r13** (stack pointer) et **r14** (link register) sont distincts dans tous les modes
- **r8 à r12** sont distincts en mode **fiq,** ils ont été prévus pour réaliser une fonction DMA par logiciel sans variables en mémoires
- Les autres registres sont visibles dans tous les modes

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Registres



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Modèle du programmeur ARM

- Les registres **SPSR**_xxx :
  - **S**aved **P**rogram **S**tatus **R**egister
  - Utilisés lors des appels des exceptions :
  - Copie local du **CPSR** (**C**urrent **PSR**) à restaurer en fin de procédure
  - Le nouveau mode dépend de la source de changement de mode

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d’exécution
- Exceptions & Interruptions
- Les différentes mémoires

Les interruptions

- Mécanisme pour optimiser le traitement par le processeur,
  - Les I/O sont toujours plus lentes que le processeur,
  - Un PC à 1 Ghz exécute environ 109 instruction à la seconde,
  - Un disque dur tourne à 7200 tr/mn soit une demi rotation toutes les 4 ms.

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Interruptions

- Événement déclenché par composant matériel
  - Fin d'entrée/sortie d'un périphérique
  - Échéance de temps terminée
- Signal envoyé au processeur de l'extérieur
  - Par l'intermédiaire d'un PIC
  - Partageable ou non
- Événement asynchrone
  - Indépendant du programme courant
  - Masquable par le système

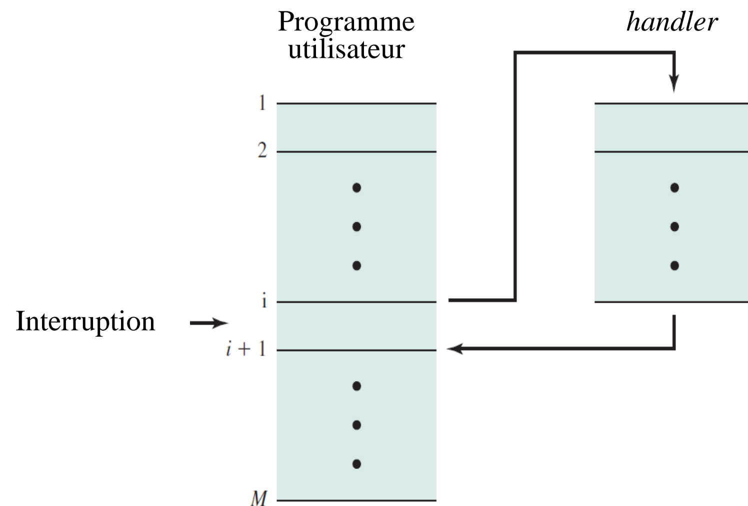
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Exceptions

- Provoquée par une condition exceptionnelle lors de l'exécution de l'instruction courante
  - Adresse mémoire invalide
  - Instruction non autorisée ou invalide
  - Division par zéro
- Synchrone avec le programme exécuté
  - Détectée par le CPU
  - Traitée dans le contexte du programme courant

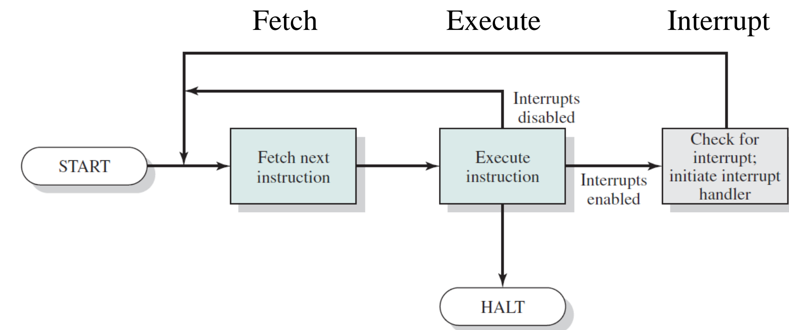
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Transfert de contrôle vers un *handler*



Conséquence sur le cycle

- Rajout d'une évaluation des condition dans le cycle d'exécution,
- Si présence met à jour son registre d'état,



Traitement Exceptions/Interruptions (1)

- Processeur suspend l'exécution en cours
- Sauvegarde état courant
  - PC, SP, registre d'état
  - Dans des registres dédiés ou dans la pile superviseur
- Ou utilise des *shadow registers*
  - 2 jeux de registres, un par mode d'exécution

Traitement Exceptions/Interruptions (2)

- Charge nouveau contexte d'exécution :
  - Registre d'état avec mode « superviseur »
  - PC avec adresse mémoire spécifique
- Opérations de sauvegarde et de chargement
- Non-interruptible
  - Sinon état non cohérent
  - Sauf par NMI, sur PowerPC par exemple
- Si exception durant opérations (« double faute»)
  - => arrêt CPU ou traitement spécial (Intel)

Exceptions/Interruptions "Vector Table"

- Adresse de branchement
  - Dépend de l'exception/interruption
- Vers une table de vecteurs localisée
  - à adresse prédéfinie (0x00000000 par exemple)
  - dans un registre (IDTR sur Intel)
- Contient une instruction de branchement
- Table de vecteurs initialisée par le système

Table des vecteurs

- La table de vecteur contient une instruction et non pas une adresse,
- En général une instruction de branchement sur une fonction appropriée

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Exceptions ARM, entrée

- Lors d'une exception **le processeur** :
  - Termine l'instruction courante
  - Change le mode selon l'exception
  - Sauve l'adresse de l'instruction qui suit dans r14 (**pc → lr**, ***link register***)
  - Copie **CPSR → SPSR** du nouveau mode
  - Interdit interruption IRQ: 1→ CPSR:bit I(7)
  - Si exception FIQ, interdit FIQ: 1 → CPSR:bit F(6)
  - Met adresse vecteur → pc (r15)
  - **La recherche de la source d'interruption matérielle est à réaliser par logiciel** !

Exceptions ARM, sortie

- Lors de la fin d'une exception **la routine d'exception** :
  - Restaure les éventuels registres modifiés depuis la pile
  - **SPSR → CPSR**
  - Récupère l'adresse de retour dans r14 (**lr → pc**)

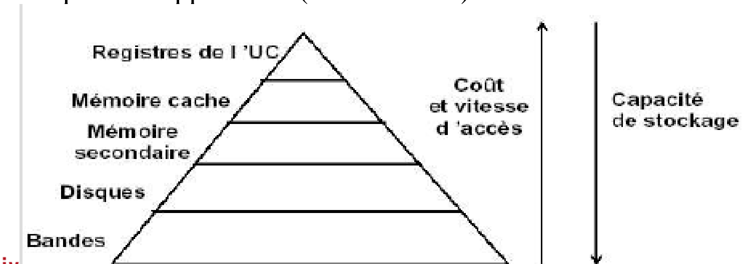
Plan du cours

- Introduction
- Architecture et évolution des systèmes
- Exécution des instructions
- Modes d'exécution
- Exceptions & Interruptions
- Les différentes mémoires

Gestion de la mémoire

Le matériel (données AMD Athlon 64):

- Les registres du processeurs – (tps d'accès moyen à un mot 64 bits : 0,3 ns)
- Cache de niveau 1 : à l'intérieur du processeur - 64 ko (1 ns)
- Cache de niveau 2 : à l'intérieur – SRAM de de 512 ko
- Cache de niveau 3 : à l'extérieur – SRAM de de 2Mo
- Mémoire principale : SDRAM – jusqu'à 4Go (de 5 ns à 100 ns))
- Disque dur : qq 100 Go (de 1us à 1ms)



Rôle de la mémoire

- Faible nombre de registres : on ne peut presque rien faire sans la mémoire.
- A part des calculs numériques, les algorithmes parcourent des structures de données complexes représentées en mémoire.
- Il y a alors des accès mémoires toutes les 2-5 instructions...
- Goulot d'étranglement
  - Les opérations mémoires sont plus lentes que les instructions des processeurs (croissance plus rapides).
  - Plusieurs niveaux de caches.

Des temps d'accès

- Niveau 1 : 1 ko tps d'accès 0.1 ms
- Niveau 2 : 100 ko tps d'accès 1ms
- Si 95% des accès mémoire sont dans le cache quel est le temps d'accès moyen d'un octet ?

Les types de mémoires

- Deux catégories :
  - volatile : elles doivent être alimentées en électricité pour conserver les informations (RAM et dérivés)
  - non-volatile : elles conservent les informations même non alimentées (ROM, EEPROM, Flash, etc.)
- La ROM
  - Non modifiable, pas cher, très miniaturisable,
  - Utilisé dans les systèmes embarqués et divers composants d'un ordinateur ne nécessitant pas de mise à jour,
- L'EEPROM
  - Electrically Erasable Programmable Read-Only Memory
  - Effaçable, utilisée dans ?

Les types de mémoires

- Deux catégories :
  - volatile : elles doivent être alimentées en électricité pour conserver les informations (RAM et dérivés)
  - non-volatile : elles conservent les informations même non alimentées (ROM, EEPROM, Flash, etc.)
- La Feram
  - Quel type ?

Les types de mémoires

- Deux catégories :
  - volatile : elles doivent être alimentées en électricité pour conserver les informations (RAM et dérivés)
  - non-volatile : elles conservent les informations même non alimentées (ROM, EEPROM, Flash, etc.)
- La Feram
  - Quel type ?
- La FLASH Nand
  - Quel type ?
  - Accès ?

Une mémoire série…

- Accès par un bus série, et dé-sérialisation en interne,
- Mémoire RAM tampon pour la dé-sérialisation,
  - Accès par shadowing uniquement,
  - Le contrôleur gère la copie de la RAM vers la Nand,
- Pages :
  - Unité minimale de lecture (perte d'efficacité pour les petits fichiers)
  - Taille des pages 4Ko + overhead,
- Bloc
  - Unité minimale d'écriture,
  - Un bloc contient 32, 64 ou 128 pages,
  - Ecriture très lente !!!

Rôle de la mémoire

- La sécurité passe par la mémoire
  - Au minimum : zone système et zone utilisateur.
  - Si possibles, protéger des zones plus fines parmi les zones utilisateurs.
- La modularité passe par la mémoire
  - Avoir plusieurs programmes chargés en parallèle.
  - Partageant du code.
- Besoin de support matériel
  - Pour pouvoir faire les vérifications de sécurité.
  - Pour les faire efficacement.
  - Pour permettre et faciliter le partage de code.

Concept de base en gestion mémoire

Manipulation symbolique implicite des adresse : variables et références

- Localisation (adresse) masquée par le système qui gère la mémoire,

L'abstraction mémoire

- Chaque processus voit la mémoire comme si elle était tout entière à lui seul avec un espace d'adressage énorme.
- C'est un espace linéaire organisé en régions contiguës protégées (séparées par des trous).
- Le matériel et le logiciel permettent :
  - De garantir la sécurité (protection mémoire).
  - Un partage de la mémoire entre processus.
  - Sans surcoût au runtime ; au contraire. . .
- Les adresses virtuelles sont traduites en adresses physiques par le **Memory Management Unit** (unité spécialisée du processeur)

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Le cache

- Le processeur accède lors de chaque instruction à la mémoire même si il ne manipule pas de donnée pourquoi ?

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Cache et localité

- Soit le code suivant :

```
for (i=0; i<20; i++)
  for (j=0; j<10; j++)
    a[i]= a[i] * j
```

- Donnez un exemple de localité spatiale dans ce code
- Donnez un exemple de localité temporelle dans ce code

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Transferts de Données

- Par interruption,
- Indirects à travers des registres
  - Simple mais pas efficace
  - Composants lents (clavier, souris, etc.)
- Par accès direct en mémoire (**DMA**)
  - Données copiées par composant dans les buffers du driver
  - Efficace (copie en parallèle avec CPU)
  - Composants rapides (disques, ethernet, USB, etc.)
  - Complexe
    - Buffers pas dans cache / « bus snooping » (Intel)
    - Continue après arrêt brutal puis re-démarrage à chaud

DMA

- Lorsque le processeur veut lire / écrire un bloc de donnée il envoi un signal au DMA comprenant,
  - Le type d'opération (R,W)
  - L'adresse du composant d'IO impliqué (usb, graphique,..)
  - Le début de l'adresse à lire ou écrire,
  - Le nombre de mots concernés.
- Le processeur continue son exécution, le DMA transfert les données.
  - Transfert mot par mot, en gérant le bus de données,
  - En cas de compétition : ajout de *t_wait* au processeur,
  - Signale la fin par une interruption au processeur.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

DMA

- Un module de DMA transfert des caractères vers la mémoire principale à partir d'un périphérique transmettant à 9600 bits par seconde.
- Le processeur peut *fetcher* des instruction à 1 million d'instructions par seconde,
- Quel est le ralentissement du processeur du au DMA ?

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Données globales / locales

```
extern char c_array[];
    /* global public to all files */
static int i_array[2048];
    /* global private to file */
int /* return value */
func (int i, char* name) {
    /* call arguments */
int* ptr;
    /* local to func() */
struct example_t ex;
    /* local to func() */
static char c_pfgd[100];
    /* global private to func() */
```

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

La pile

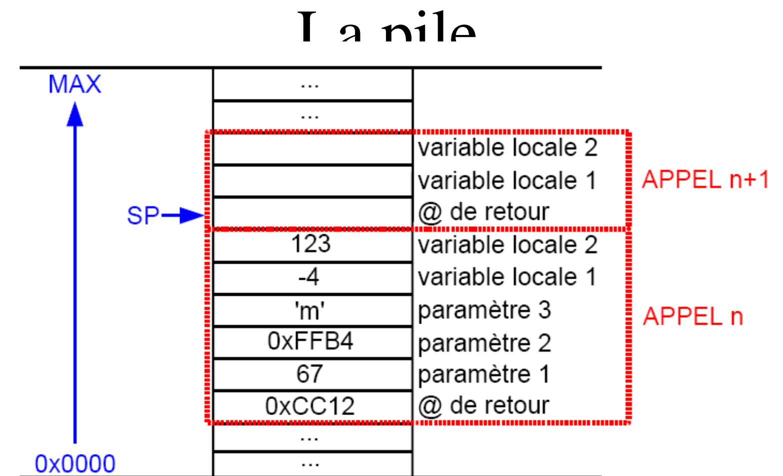
Zone « dans laquelle s'exécute le programme »

- paramètres, variables locale, adresse et valeur de retour de fonction / routine / méthode
- croît à chaque appel, décroît à chaque retour

Automatique: géré par l'environnement d'exécution (*runtime*)

+/- invisible du programme(ur)

La pile



- NB: Ici, pile croissante avec adresses croissantes.
- En pratique, la pile croît souvent vers 0x0000.

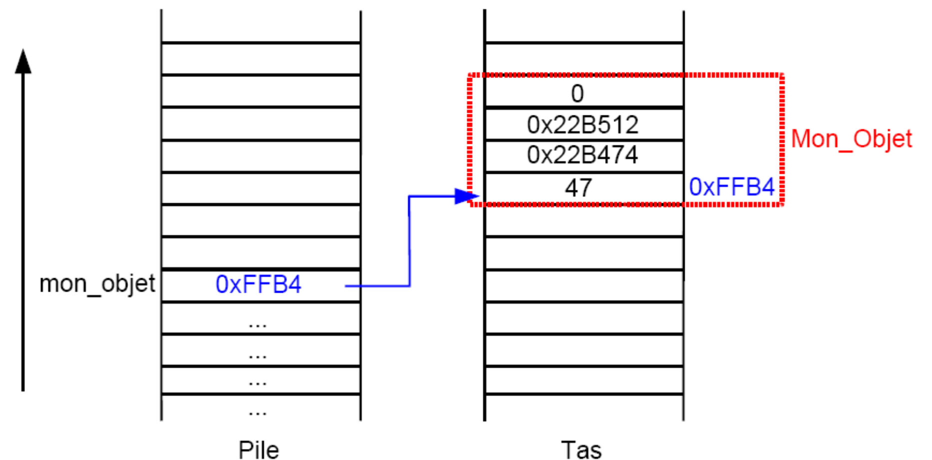
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Le tas

- Zone où le programme(ur) alloue toutes ses données qui ne sont pas en pile
- Allocation explicite: malloc C
  - `MonObjet*mon_objet=(MonObjet*)malloc(sizeof(MonObjet));`
- Allocation « implicite »: new en Java, C++...
  - `MonObjet mon_objet = new MonObjet();`

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Le tas

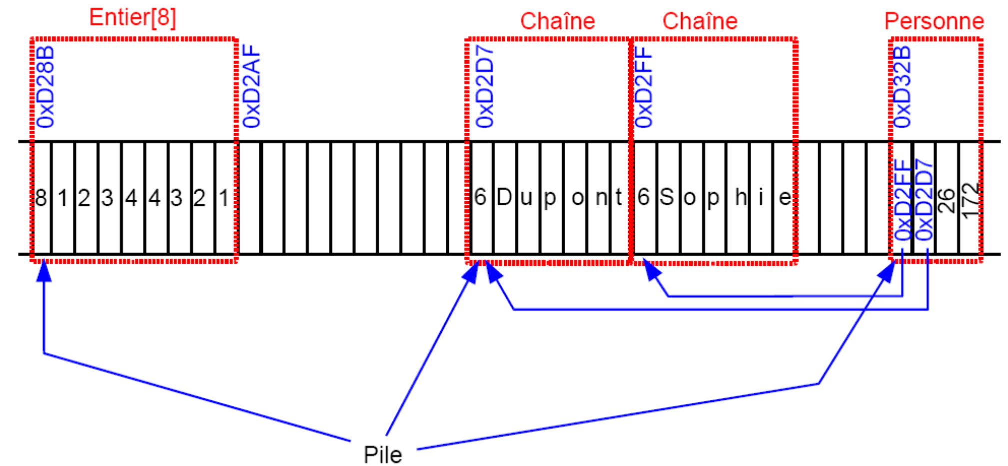


Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Le tas

Zone « désordonnée », contrairement à la pile (d'où les noms...)
La gestion mémoire concerne principalement le tas.

Le tas



Appels système

- Quelles sont les trois méthodes pour passer des paramètres au SE ?

Le processus de démarrage

- Les mémoires du processeur et la mémoire centrale est volatile
- Il faut donc un programme stocké dans une mémoire non volatile
- BIOS : Basic Input Output System
- Lors du démarrage le processeur met automatiquement son PC à l'adresse du BIOS

Démarrage

- Le BIOS effectue les opérations suivantes:
  - phase de test
  - chargement de la configuration depuis la mémoire du BIOS
  - le BIOS cherche un périphérique de stockage bootable
  - chargement des 512 premiers octets de ce périphérique contenant le bootstrap
  - Comment continuer le processus de boot ?

Processus de démarrage

- Le noyau est à son tour chargé par le *bootloader*
- Il configure le processeur:
  - la table des interruptions
  - le contrôleur de mémoire (MMU)

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Any question ?

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Système d'exploitation Gestion Mémoire

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard
Jean-louis.lanet@unilim.fr

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Plan

- Besoin de gestion mémoire
- Pagination
- Segmentation
- Problèmes de sécurité

Objectifs

- Après cette présentation vous devriez être capable de
  - Comprendre les fondements de la gestion mémoire,
  - Raisonner sur les besoins et techniques de partitionnement de la mémoire,
  - Expliquer les concepts de pagination et de segmentation ainsi que leurs avantages comparés,
  - Connaitre les problèmes de sécurité induits par la gestion mémoire.

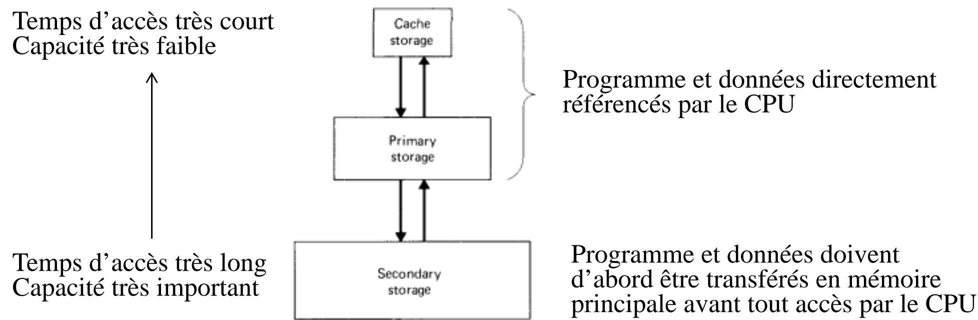
Problème de l'allocation

- L'allocation mémoire doit permettre à un processus d'accéder à un objet défini en mémoire virtuelle (prochain cours),
- Une politique d'allocation mémoire doit résoudre:
  - La correspondance entre adresses virtuelles et adresses physique,
  - Gérer la mémoire physique,
  - Réaliser le partage d'information entre les usagers,
  - Assurer la protection mutuelle d'informations appartenant à des usages distincts.

Et la gestion mémoire…

- Lorsque cette mémoire a été allouée à un processus il doit la gérer,
  - Créer des objets / détruire des objets,
  - Gérer les appels à des sous programmes
  - Gérer la récursivité
  - …

Pourquoi gérer la mémoire ?

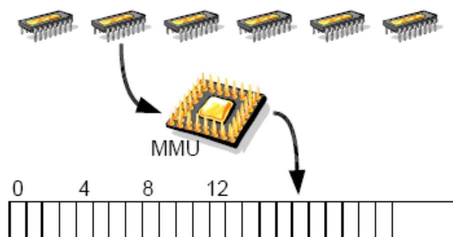


Mémoire/Adresses physiques et logiques

- Mémoire physique:
  - la mémoire principale RAM de la machine
- Adresses physiques: les adresses de cette mémoire,
- Mémoire logique: l'espace d'adressage d'un programme,
- Adresses logiques: les adresses dans cet espace,
- Il faut séparer ces concepts car normalement, les programmes sont chargés parfois dans des positions différentes de mémoire,
  - Donc adresse physique ≠ adresse logique

Représentation de la mémoire

- Mémoire: des puces (matériel)
- Vue par le système (OS/application) via des adresses (logiciel)



Représentation de la mémoire

- Manipulation des adresses à la main (ASM):

```
MOV    47,#0xFBBFC
MOV    74,#0xFBFBC
ADD    3,#0xFBFBC
INC#0xFBBFC
SUB    #0XFBBFC,#0xFBFBC
```

  - Peu clair...

Représentation de la mémoire

- Manipulation des adresses à la main (ASM)
  - Permet de mettre des données en un lieu précis de la mémoire
  - Compliqué si on veut faire cohabiter plusieurs applications
  - Lisibilité et maintenance pitoyables

Représentation de la mémoire

- Manipulation symbolique **explicite** des adresses : variables et pointeurs

  int *a = 0xFBBFC
  int *b = 0xFBFBC
  *a = 47
  *b = 74
  *b = *b+3
  *a = *a+1
  *b = *b-*a
  - C'est mieux: plus haut niveau, plus clair

Représentation de la mémoire

- Manipulation symbolique **implicite** des adresses: variables et références

  int a = 47
  int b = 74
  b += 3
  a++
  b -= a

Représentation de la mémoire

- Manipulation symbolique implicite des adresses: variables et références
  - Encore plus haut niveau, plus clair
  - Localisation (adresse) masquée (par le système qui gère la mémoire)
    - Facile d'avoir plusieurs programmes (multi-tâche, multi processus)

Programme

- Objets d'un programme
  - fonctions, variables, constantes désignés par un nom symbolique unique
  - taille des objets calculée par compilateur
  - éditeur de liens attribue une position absolue (adresse) ou relative des objets dans espace d'adressage
- Taille et adresses des instructions et des données stockées dans programme binaire

Rôle de la mémoire

- Stocker instructions et données des programmes exécutés par le processeur,
- Repérer les objets par leur adresse dans la mémoire
- Espace d'adressage
  - Ensemble des unités mémoire (octets) individuellement adressables depuis le CPU
  - Mécanisme(s) de construction de l'adresse de chaque unité de l'espace

Rôle de la mémoire

- Les opérations mémoires sont plus lentes que les instructions des processeurs.
- La sécurité passe par la mémoire
  - Au minimum : zone système et zone utilisateur.
  - Si possible, protéger des zones plus fines parmi les zones utilisateurs.
- La modularité passe par la mémoire
  - Avoir plusieurs programmes chargés en parallèle.
  - Partageant du code.
- Besoin de support matériel
  - Pour pouvoir faire les vérifications de sécurité efficacement.
  - Pour permettre et faciliter le partage de code.

Contrôle d'accès

- Pour la sûreté : avoir des zones en lecture seule pour l'utilisateur.
  - La zone système.
  - Les zones des autres utilisateurs.
- Pour la sécurité (secret) : contrôler également la lecture.
- Pour le confort : l'utilisateur peut lui-même protéger certaines zones en écritures.
- Pour l'efficacité : copie à l'écriture.
  - On interdit une zone en écriture.
  - On rattrape l'interruption de violation de droit pour sauvegarder la zone, puis autoriser l'écriture et reprendre l'exécution.

Autres opérations

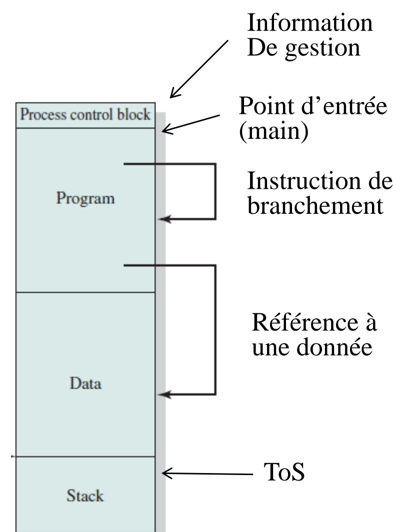
- Programme partiellement chargé en mémoire
  - Si le programme ne loge pas tout entier en mémoire centrale.
  - (La mémoire disque est plus grande que la mémoire centrale).
- Plusieurs programmes chargés en mémoire
  - Peuvent-ils partager un même sous-programme ?
  - Nécessite du code réentrant : Comment le permettre/favoriser ?
- Code relogeable
  - On veut pouvoir sauver un programme en train de s'exécuter sur le disque (par manque de place), puis plus tard le recharger en mémoire, éventuellement à un autre emplacement.

Liaison d'adresses logiques et physiques (instructions et données)

- La liaison des adresses logiques aux adresses physiques peut être effectuée en moments différents:
  - Compilation : quand l'adresse physique est connue au moment de la compilation (rare)
    - p.ex. parties du SE
  - Chargement : quand l'adresse physique où le programme est chargé est connue, les adresses logiques peuvent être traduites (rare aujourd'hui)
  - Exécution : normalement, les adresses physiques ne sont connues qu'au moment de l'exécution
    - p.ex. allocation dynamique, swap : le programme peut être relogé à des adresses différentes,

Problème d'adressage d'un processus

- Le système a besoin de connaitre:
  - Le point d'entrée,
  - La pile correspondant à ce processus,
- Il doit inférer,
  - L'adresse réel du programme,
  - Les différents sauts,
  - Les adresses des données,



Problème d'adressage d'un processus : hardware

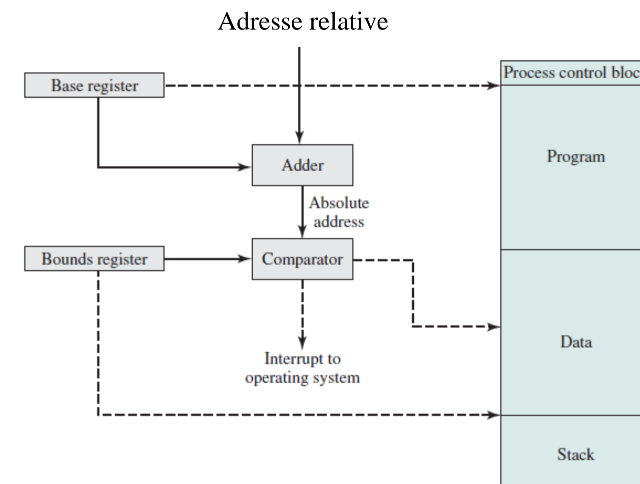


Image du processus en mémoire

Code relogeable

- On veut pouvoir déplacer le code une fois chargé en mémoire
- Il suffit que le compilateur référence les adresses du code par décalage par rapport à un registre de base (par exemple le début de la zone utilisateur).
- Permet un calcul dynamique des adresses réelles du code.
- Peut aussi se faire au niveau logiciel (génération de code) sans support matériel.
- L'adresse $p + k$ est un saut relatif de d par rapport à pc ou de k par rapport à la *base* qui vaut ici p pendant l'exécution.

Plan

- Besoin de gestion mémoire
- Pagination
- Segmentation
- Problèmes de sécurité

Et pour quelques termes de plus…

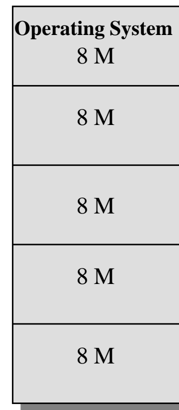
- Frame : la mémoire vive est composée de zones de même taille, appelées *cadres* (*frames* en anglais).
- Pages : un bloc de donnée de longueur fixe résidant en mémoire secondaire. Une page peut être copiée temporairement dans une frame,
- Segment : un bloc de donnée de longueur variable résidant en mémoire secondaire. Un segment peut être copié temporairement en mémoire principale.

Techniques de gestion mémoire

- Partionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple soufrant de la fragmentation interne
- Partionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne

Partitionnement à taille fixe

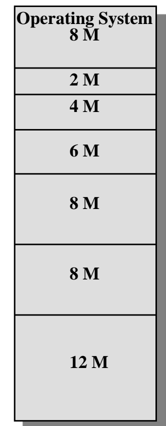
- Main memory divided into static partitions
- Simple to implement
- Inefficient use of memory
  - Small programs use entire partition
  - Maximum active processes fixed
  - Internal Fragmentation



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Partitionnement à taille fixe

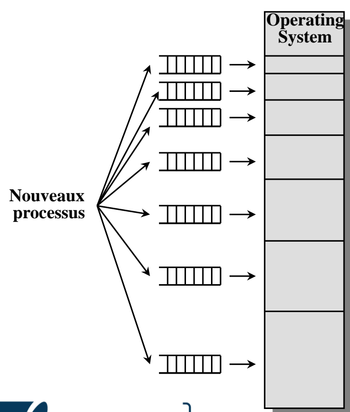
- Différentes tailles de partitions
  - Les petits programmes sont alloués aux petites partitions
  - Le problème de la fragmentation reste.
- Algorithme de placement
  - Utiliser la plus petite partition possible,
  - Une file d'attente par partition,
  - Inefficacité si les grosses partition ne sont pas requises.

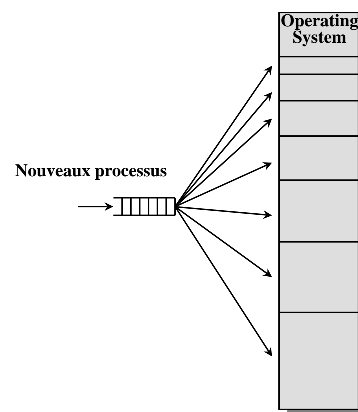


Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

File d'attente

- Soit une file d'attente par partition soit une file d'attente et allocation First Fit.





Techniques de gestion mémoire

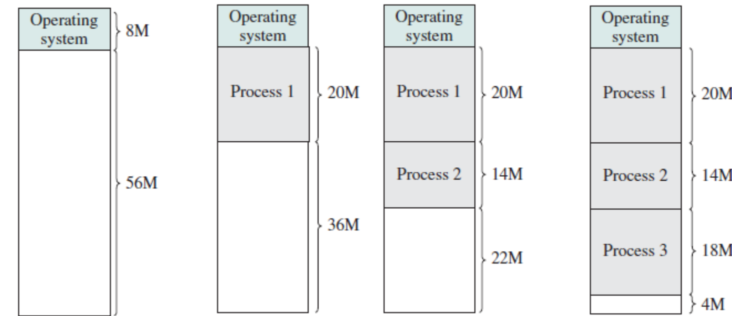
- Partionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple soufrant de la fragmentation interne
- Partionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Partitionnement dynamique

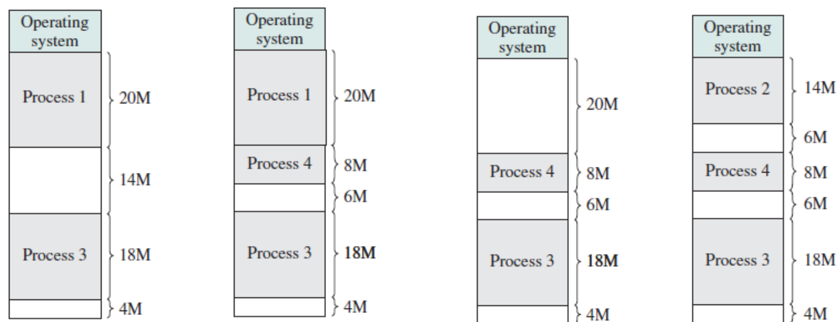
- Les partitions sont de longueurs variables et leur nombre n'est pas fixe,
- Un processus reçoit exactement la mémoire dont il a besoin,
- Éventuellement ceci créé des trous dans la mémoire (espace alloué puis dés-alloué) : fragmentation externe.

Fragmentation



Quatre processus doivent être chargés : 20, 14, 18 et 8 Mo

Fragmentation



Fragmentation mémoire

- Les statistiques montrent que pour N blocs alloués, 0.5*N blocs sont perdus par fragmentation;
- Solution le compactage.
  - Le CPU déplace les processus de manière à les rendre contigus.
  - La mémoire est relâchée par bloc.
  - L'algorithme de compactage doit être exécuté de temps en temps
    - Fréquence ?
    - A quel moment ?

Fragmentation

- Réunir 2 blocs libres adjacents
  - Minimiser fragmentation
- Immédiatement, lors de la libération d'un bloc
  - Retrouver éventuel bloc libre adjacent du bloc libéré
  - Liste des blocs libres par adresses croissantes
  - Pénalise opération de libération
- Plus tard
  - À partir d'un seuil minimum de blocs libres
  - Lorsque allocation non satisfaite immédiatement

Algorithme d'allocation

- Allocation de régions de tailles variables
  - Espace libre = ensemble de blocs de ≠ tailles
- Taille bloc libre > taille demandée
  - Reste bloc libre de plus petite taille
- Fragmentation externe
  - Somme taille des zones libres > taille demandée
  - Taille de chaque zone libre < taille demandée

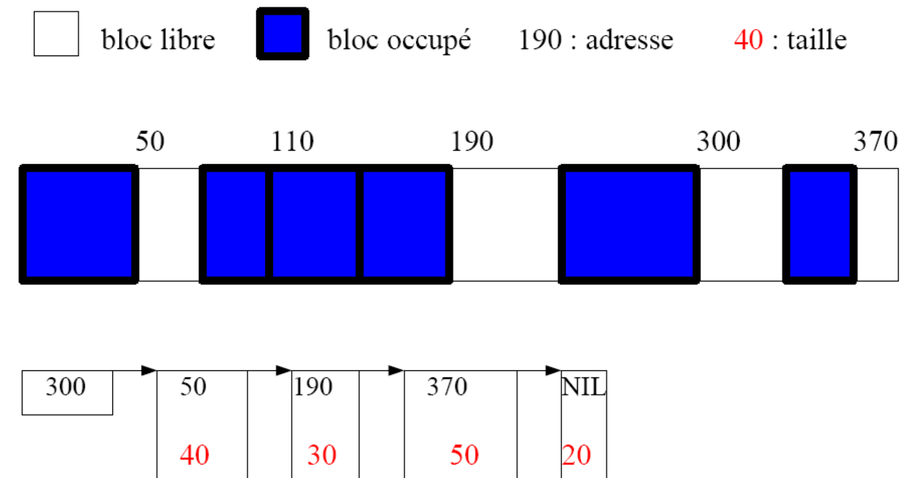
Université de Limoges | Faculté des Sciences et Techniques

Allocateur First fit

- Alloue le premier bloc libre qui convient
- Liste de blocs libres gérée en FIFO ou par adresses croissantes
- Chaque bloc libre contient
  - Sa taille
  - Adresse bloc suivant dans la liste
- Allocation doit au pire parcourir toute la liste

Université de Limoges | Faculté des Sciences et Techniques

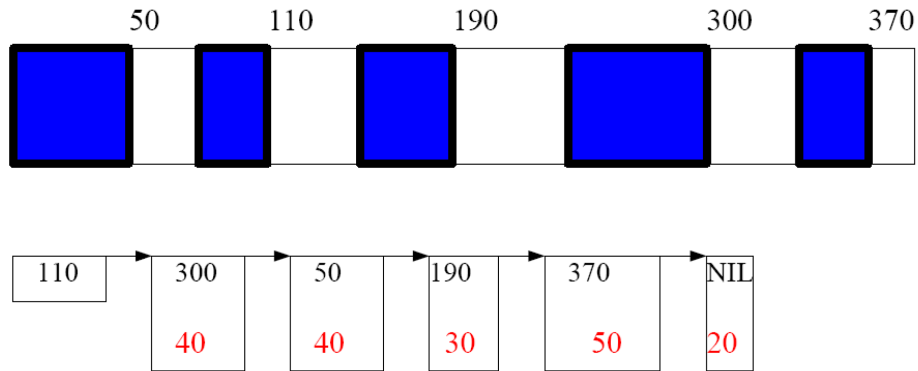
First fit



Université de Limoges | Faculté des Sciences et Techniques

First fit

Libération bloc de taille 40 à l'adresse 110



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

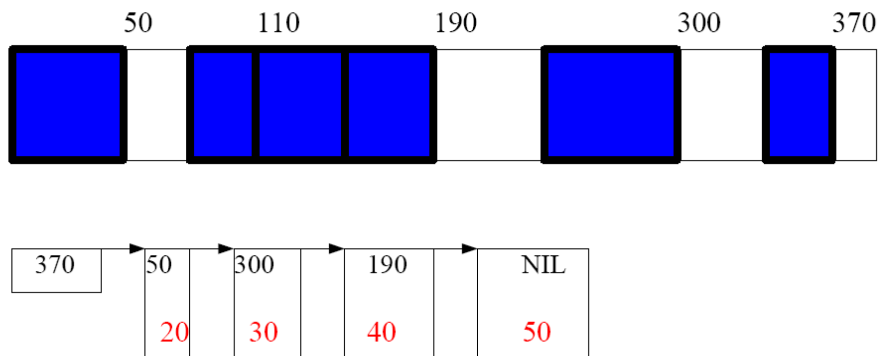
Allocateur Best fit

- Alloue depuis le bloc libre qui laisse le plus petit résidu
- Liste de blocs libres gérée par tailles croissantes
- Chaque bloc libre contient
  - Sa taille
  - Adresse bloc suivant dans la liste
- Libération rapide
- Allocation doit au pire parcourir toute la liste

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Best fit

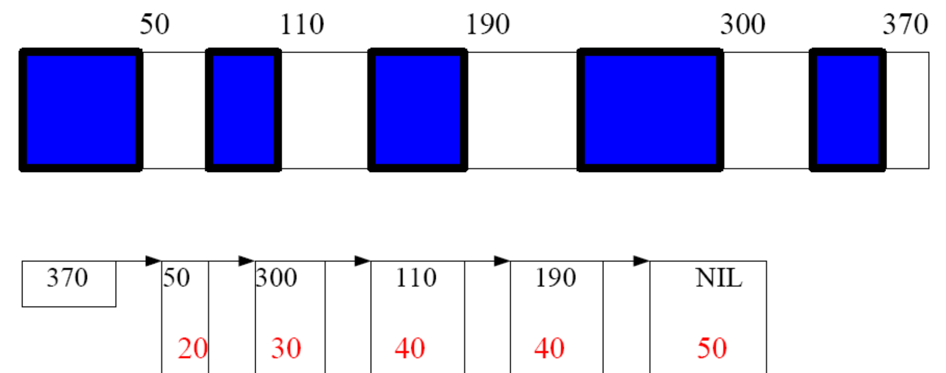
bloc libre   bloc occupé   190 : adresse   40 : taille



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Best fit

Libération bloc de taille 40 à l'adresse 110

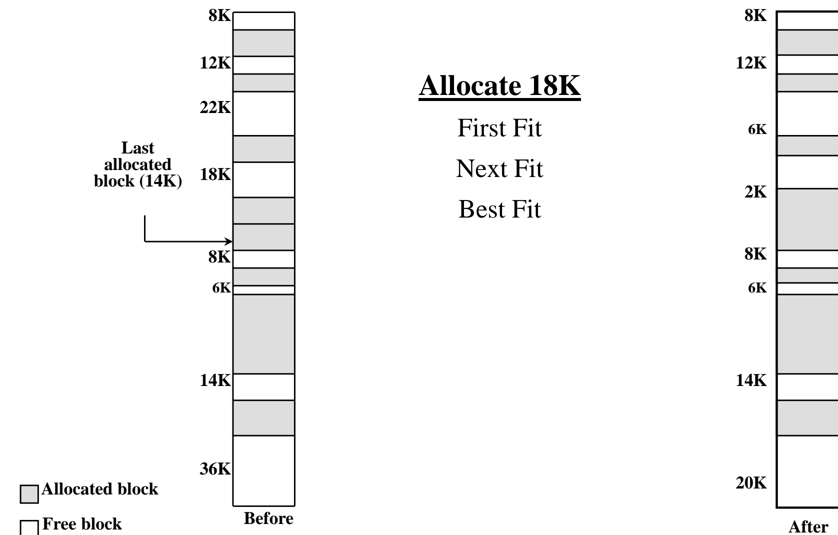


Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Worst fit

- Plus grand résidu,
- On choisit la zone telle que la taille restante soit la plus grande possible,
- On combat efficacement l'émiettement.

Les différents algorithmes…



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Techniques de gestion mémoire

- Partitionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple soufrant de la fragmentation interne
- Partitionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Pagination simple

- La mémoire est partitionnée en petits morceaux de même taille: les pages physiques ou 'cadres' ou 'frames'
- Chaque processus est aussi partitionné en petits morceaux de même taille appelés pages (logiques)
- Les pages logiques d'un processus peuvent donc être assignés aux cadres disponibles n'importe où en mémoire principale
- Conséquences:
  - un processus peut être éparpillé n'importe où dans la mémoire physique.
  - la fragmentation **externe** est éliminée

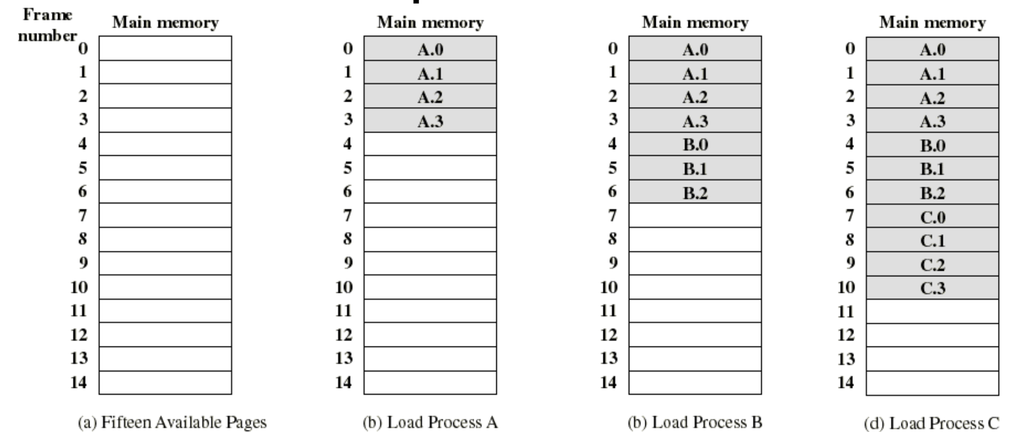
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Pagination

- Le principe de la pagination est simple
  - l'espace d'adressage de chaque programme est partagé en blocs de taille fixe (des tailles de 4-16KB sont courantes),
  - les pages utilisées sont chargées en mémoire.
  - le restant de l'espace d'adressage est stocké sur le disque dur.
- Les adresses physiques des pages en mémoire sont stockées dans une *page table*.
- Chaque processus dispose de sa *page table*, qui fait office de multiples *base register*.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Exemple de chargement de processus

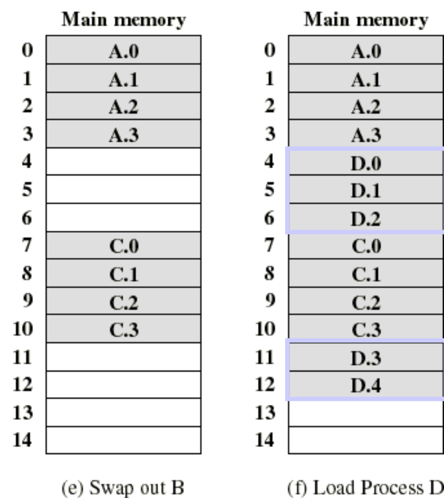


- Supposons que le processus B se termine ou soit suspendu

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

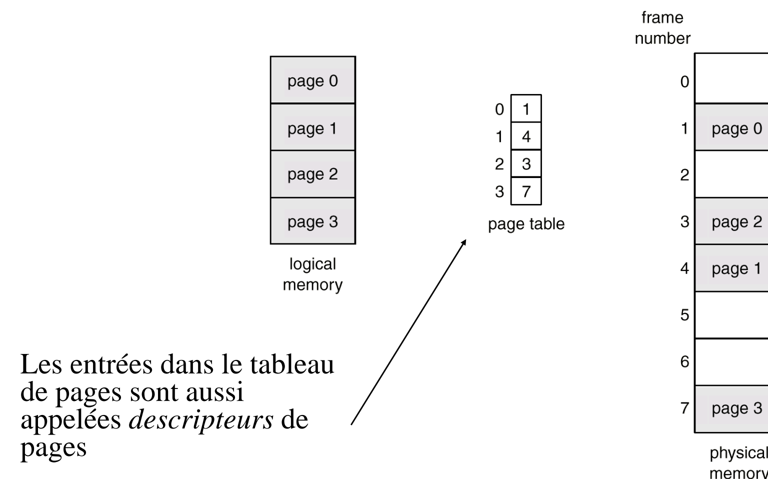
Exemple de chargement de processus

- Nous pouvons maintenant transférer en mémoire un programme D, qui demande 5 pages
  - bien qu'il n'y ait pas 5 pages contigües disponibles
- La fragmentation externe est limitée au cas où le nombre de pages disponibles n'est pas suffisant pour exécuter un programme en attente
- Seule la dernière page d'un programme peut souffrir de fragmentation interne (moy. 1/2 cadre par proc)



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

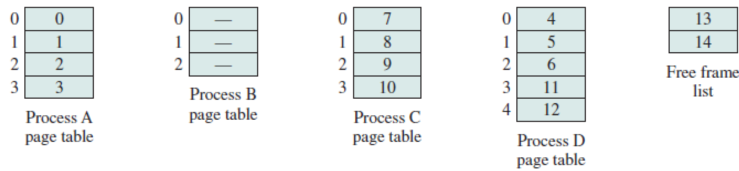
Tableaux de pages



Les entrées dans le tableau de pages sont aussi appelées *descripteurs* de pages

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Tables de pages

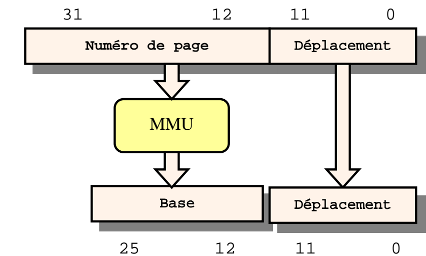


- Le SE doit maintenir une table de pages pour chaque processus
- Chaque descripteur de pages contient le numéro de frame où la page correspondante est physiquement localisée
- Une table de pages est indexée par le numéro de la page afin d'obtenir le numéro de la frame
- Une liste de frames disponibles est également maintenue (*free frame list*)

Pagination - Adressage

Une adresse virtuelle est donc composée de deux parties : un numéro de page et un déplacement dans la page.

Exemple:
Largeur des adresses = 32bits
Taille des pages = 4KB
Taille de la mémoire = 64MB



Cette transformation, réalisée par la MMU, permet de travailler avec des espaces d'adressage plus grands que la mémoire physique, et d'autre part de référencer des données sur une même page avec peu de bits.

Pagination - *Page faults*

- Une adresse virtuelle peut donc référencer
  - soit une page en mémoire (*page hit*)
  - soit une page sur le disque dur (*page miss*).
- Si la page cherchée se trouve sur le disque dur, le MMU engendre une interruption (*page fault*)
- Une procédure se charge de transférer les données en mémoire (une opération qui demande plusieurs millions de cycles d'horloge).

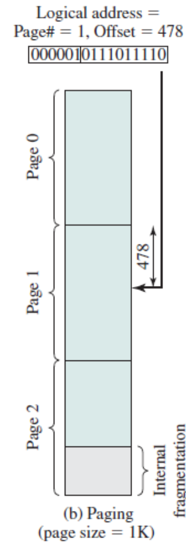
Pagination - Remplacement

- Lors qu'une nouvelle page est transférée du disque dur à la mémoire, il est parfois nécessaire d'en transférer une autre de la mémoire au disque dur (*swap*).

- Étant donné le "prix" très élevé d'un *page fault*, l'algorithme utilisé pour décider quelle page va être "swappée" est fondamental pour la performance d'un système. L'algorithme de choix pour cette opération est le LRU (*least recently used*).

Traduction d'adresses

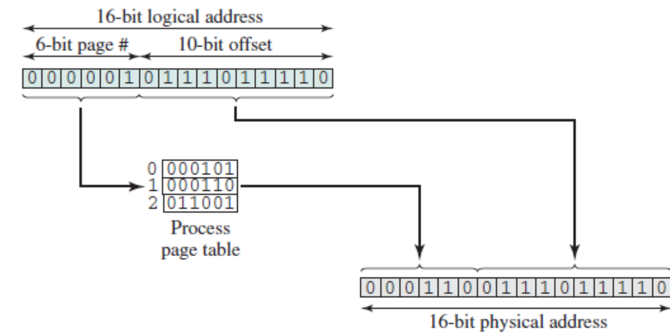
- L'adresse logique est facilement traduite en adresse physique
  - car la taille des pages est une puissance de 2
    - les pages débutent toujours à des adresses qui sont puissance de 2
    - qui ont autant de 0s à droite que la longueur de l'offset
    - donc ces 0s sont remplacés par l'offset
  - Ex: si 16 bits sont utilisés pour les adresses et que la taille d'une page = 1K (1024 bits) on a besoin de 10 bits pour le décalage, laissant ainsi 6 bits pour le numéro de page
  - L'adresse logique (n,m) est traduite à l 'adresse physique (k,m) en utilisant n comme index sur la table des pages et en le remplaçant par l'adresse k trouvée.



(b) Paging
(page size = 1K)

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Traduction d'adresse (logique-physique) pour la pagination



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Techniques de gestion mémoire (next)

- Segmentation simple *(prochain cours)*
  - Divise les programmes en segments
  - Pas de fragmentation interne, faible fragmentation externe
- Mémoire virtuelle paginée *(prochain cours)*
  - Basée sur un mécanisme de pages, mais pas toutes en mémoire centrale,
  - Autorise un vaste espace de mémoire virtuelle
  - Surcout d'exécution
- Mémoire virtuelle segmentée *(prochain cours)*
  - Basée sur un mécanisme de segments, mais pas toutes en mémoire centrale,
  - Facilité pour partager des modules.

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Any question ?

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Système d'exploitation
Gestion Mémoire

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard / David Pequegnot
Jean-louis.lanet@unilim.fr

Plan

- Besoin de gestion mémoire
- Pagination
- Segmentation
- Problèmes de sécurité

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Techniques de gestion mémoire

- Partitionnement à taille fixe
  - Divise la mémoire en partitions lors du boot, les tailles peuvent être identiques ou pas mais fixes,
  - Mécanisme simple soufrant de la fragmentation interne
- Partitionnement à taille variable
  - Les partitions sont créées lors du chargement des programmes
  - Ne crée pas de fragmentation interne mais externe,
- Pagination simple
  - Divise la mémoire en pages de taille fixes et charge les programmes dans les pages disponibles
  - Pas de fragmentation externe, mais légère fragmentation interne

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Techniques de gestion mémoire

- Segmentation simple
  - Divise les programmes en segments
  - Pas de fragmentation interne, faible fragmentation externe
- Mémoire virtuelle paginée
  - Basée sur un mécanisme de pages, mais pas toutes en mémoire centrale,
  - Autorise un vaste espace de mémoire virtuelle
  - Surcout d'exécution
- Mémoire virtuelle segmentée
  - Basée sur un mécanisme de segments, mais pas toutes en mémoire centrale,
  - Facilité pour partager des modules.

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Segmentation

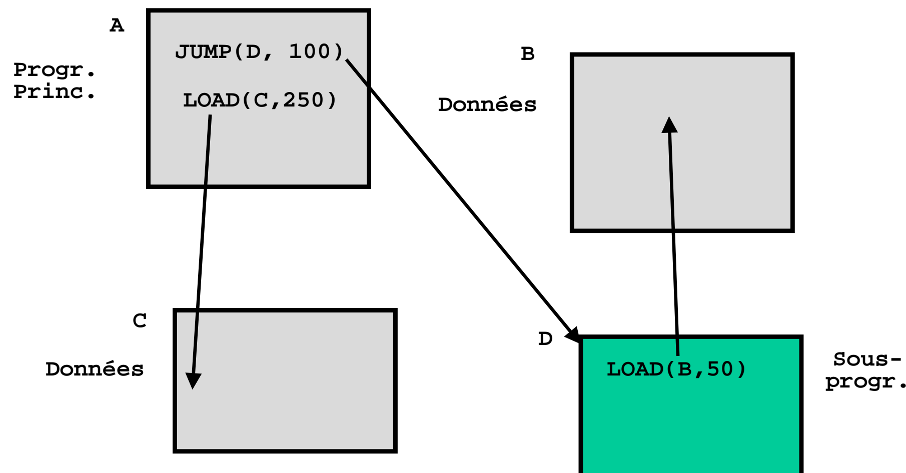
- Un segment est un ensemble d'information considéré comme une unité logique
- La mémoire est coupée en régions appelées segments,
- Tous les segments n'ont pas la même taille,
- Comme pour la pagination, la segmentation utilise un numéro de segment,
- A l'intérieur d'un segment, les informations sont désignées par un déplacement qui est une adresse relative.

Segmentation

- Les segments sont utilisés :
  - Comme unité de découpage logique d'un programme
  - Comme unité de partage entre plusieurs utilisateurs,
  - Comme unité de protection, le segment est l'entité à laquelle sont attachés des droits d'accès,
- Un descripteur est attaché à chaque segment,
  - Adresse d'implantation du segment,
  - Les droits d'accès,
  - Sa taille

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Les segments sont des parties logiques du programme

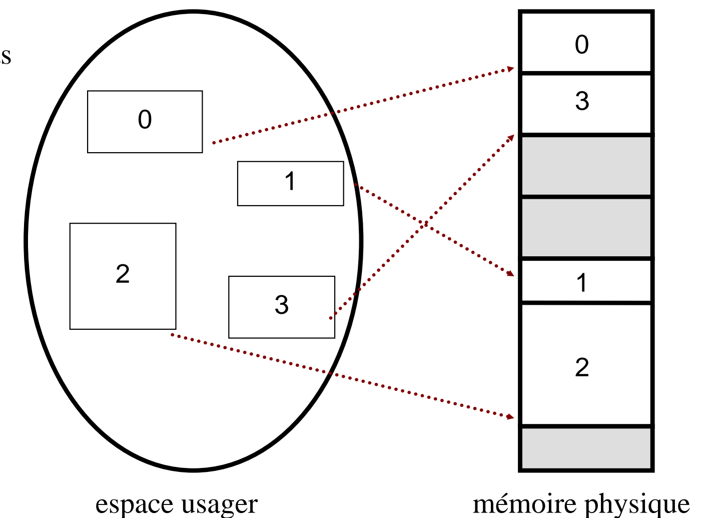


4 segments: A, B, C, D

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Les segments comme unités

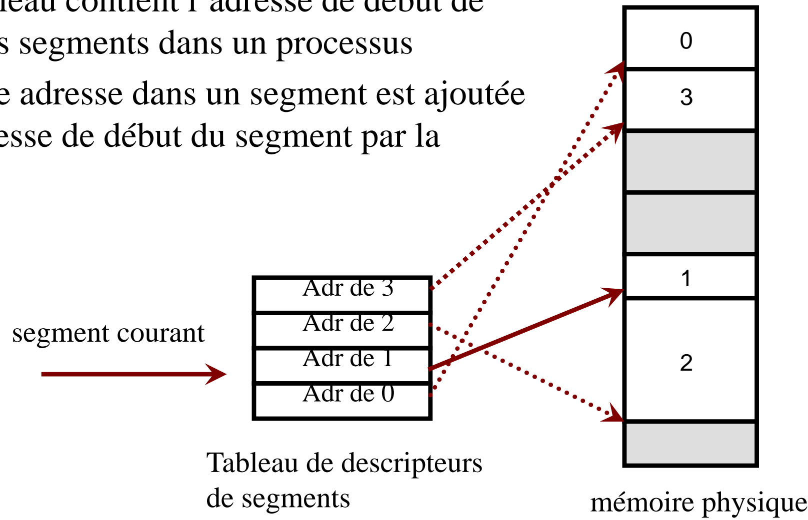
Étant donné que les segments sont plus petits que les programmes entiers, cette technique implique moins de fragmentation (qui est externe dans ce cas)



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Mécanisme pour la segmentation

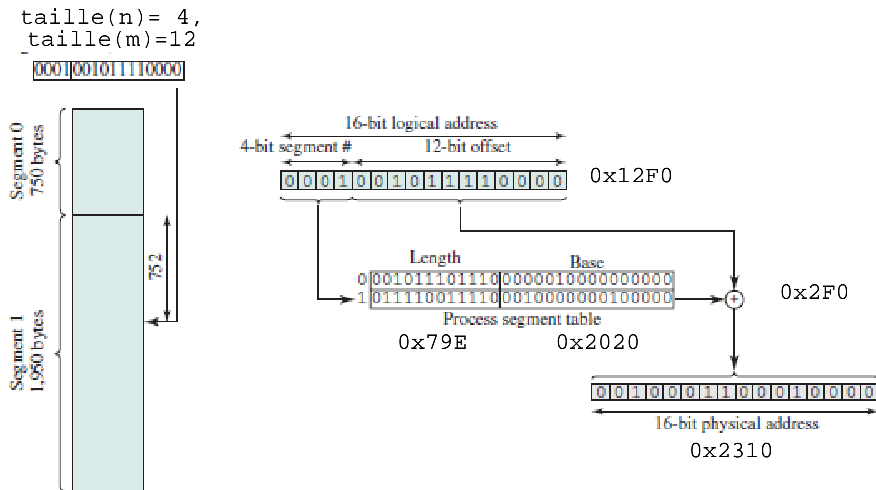
- Un tableau contient l'adresse de début de tous les segments dans un processus
- Chaque adresse dans un segment est ajoutée à l'adresse de début du segment par la MMU

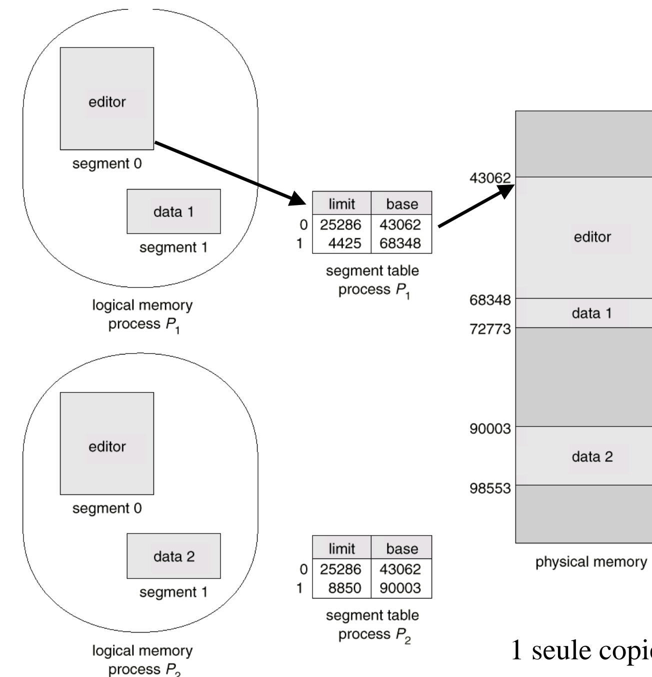


Détails

- L'adresse logique consiste d'une paire: <No de segm, décalage>
- Le tableau des segments contient des descripteurs de segments
  - adresse de base
  - longueur du segment
  - Infos de protection…
- Pour le processus il y aura un pointeur à l'adresse en mémoire du tableau des segments
- Il y aura aussi le nombre de segments dans le processus
- Au moment de la commutation de contexte, ces infos seront chargées dans les registres appropriés de la MMU

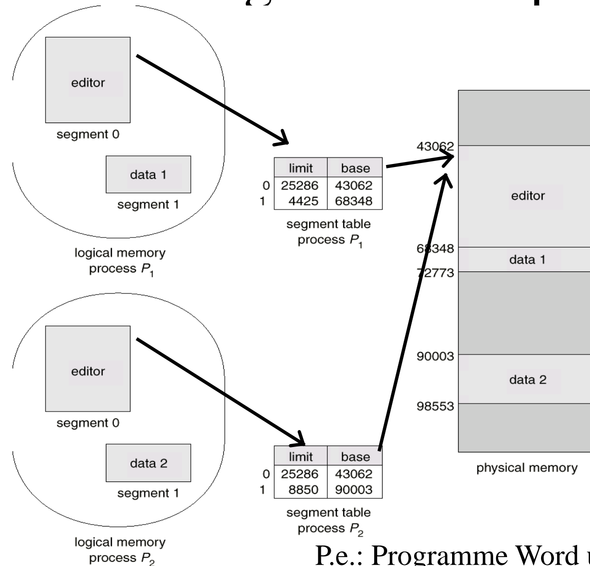
Détails





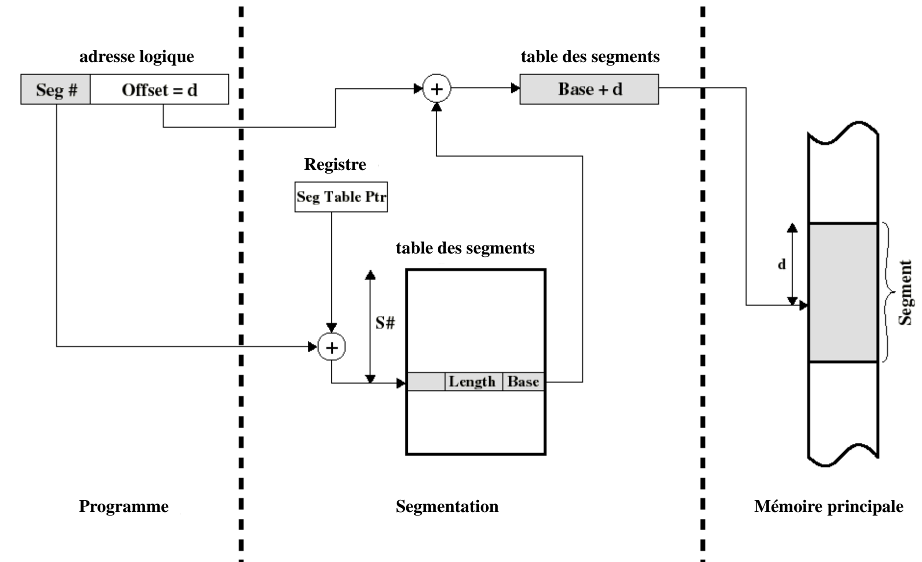
1 seule copie en mémoire du segment partagé

Partage de segments: le segment 0 est partagé



P.e.: Programme Word utilisé pour éditer différents documents , DLL utilisé par plus usagers

Traduction d'adresses par segmentation



Aussi, si d > longueur: segmentation fault

Segmentation et protection

- Chaque *descripteur de segment* peut contenir des infos de protection:
  - longueur du segment
  - privilèges de l'usager sur le segment: lecture, écriture, exécution
    - Si au moment du calcul de l'adresse on trouve que l'usager n'a pas droit d'accès→interruption
    - ces infos peuvent donc varier d'usager à usager, par rapport au même segment!

limite	base	read, write, execute?

Évaluation de la segmentation simple

- Avantages: l'unité d'allocation de mémoire est
  - Plus petite que le programme entier
  - Une entité logique connue par le programmeur
  - Les segments peuvent changer de place en mémoire
  - La protection et le partage de segments sont aisés (en principe)
- Désavantage: le problème des partitions dynamiques:
  - La fragmentation externe n'est pas éliminée:
  - Trous en mémoire, compression?
- Une autre solution est d'essayer de simplifier le mécanisme en utilisant unités d'allocation mémoire de tailles égales

Désavantage

- La zone mémoire d'un processus est contiguë (au moins à l'intérieur d'un segment s'il y a des segments).
- Un processus doit être entièrement en mémoire, ou bien, le compilateur doit le segmenter manuellement. *i.e.* prévoir manuellement le chargement/déchargement des segments.
- Il y a possibilité de fragmentation de la mémoire : impossibilité d'allouer une grosse zone alors qu'il y a plein d'espaces libres mais de petites tailles.
- Remarques
  - Les segments partitionnent la mémoire en grosses zones.
  - Les adresses peuvent être référencés par rapport au début du segment (le logiciel ajoute un décalage)

Traduction d'adresses: segmentation et pagination

- Tant dans le cas de la segmentation, que dans le cas de la pagination, nous *ajoutons* le décalage à l'adresse du segment ou page.
- Cependant, dans la pagination, l'addition peut être faite par simple concaténation:

```
11010000+1010
      =
  1101 1010
```


Segmentation simple vs Pagination simple

- La segmentation est visible au programmeur mais la pagination ne l'est pas.
- Le segment est une unité logique de protection et partage, tandis que la page ne l'est pas.
- La segmentation requiert un matériel plus complexe pour la traduction d'adresses (addition au lieu d'enchaînement)
- La segmentation souffre de fragmentation *externe* (partitions dynamiques)
- La pagination produit de fragmentation *interne*, mais pas beaucoup (1/2 cadre par programme)

Adresse logique (pagination)

- Les pages sont invisibles au programmeur, compilateur ou assembleur (seules les adresses relatives sont employées)
- Un programme peut être exécuté sur différents matériels employant des dimensions de pages différentes
  - Ce qui change est la manière dont l'adresse est découpée

Techniques de gestion mémoire

- Segmentation simple
  - Divise les programmes en segments
  - Pas de fragmentation interne, faible fragmentation externe
- Mémoire virtuelle paginée
  - Basée sur un mécanisme de pages, mais pas toutes en mémoire centrale,
  - Autorise un vaste espace de mémoire virtuelle
  - Surcout d'exécution
- Mémoire virtuelle segmentée
  - Basée sur un mécanisme de segments, mais pas toutes en mémoire centrale,
  - Facilité pour partager des modules.

Virtualisation

- Pour qu'un processus s'exécute il faut:
  - Son code est dans la mémoire principale,
  - Il a au moins toute la mémoire dont il a besoin,
  - Ses périphériques sont prêts.
- Toutes les instructions du programme sont chargées,
  - Même si certaines parties ne seront jamais exécutées,
  - L'espace d'adressage physique n'est pas forcément contigüe il peut être étalé sur plusieurs pages.

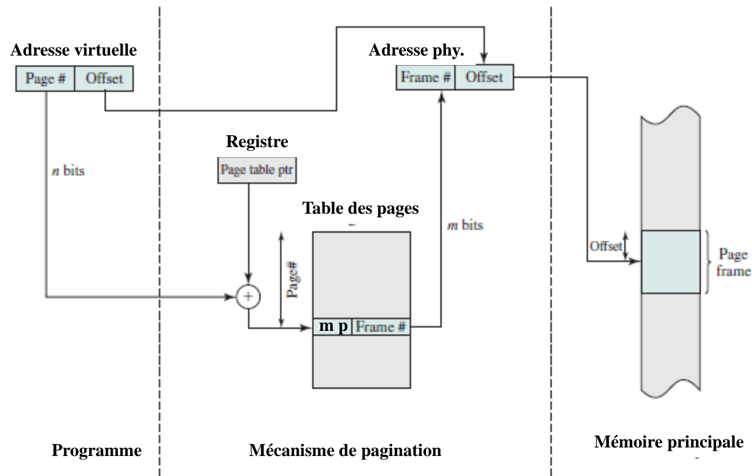
Virtualisation

- Caractéristique de la pagination et segmentation
  - Toutes les références au programme (saut, méthodes) et au données sont relatives,
  - Un processus peut être séparé en plusieurs éléments (pages, segments), pas forcément contigus.
- S'il est possible d'exécuter un programme n'ayant pas toutes ses instructions en mémoire :
  - Potentiellement un espace d'adressage plus vaste
  - Plus de programmes en mémoire centrale
  - Moins d'attente des entrées sorties
  - Programme non utilisé = programme non chargé

Virtualisation

- La mémoire secondaire peut donc être considéré comme une mémoire exécutable virtuelle dont la taille est celle de la mémoire secondaire
- Différence entre mémoire physique et mémoire logique
  - Le matériel est en charge de la translation d'adresse
  - Seule une partie du programme est chargé en mémoire
- Le système doit gérer les fragment absents (page miss) par mise en attente du processus et transfert de la mémoire.

Virtualisation



Problèmes d'efficacité

- La traduction d'adresses, y compris la recherche des adresses des pages et de segments, est exécutée par des mécanismes matériels
- Cependant, si la table des pages est en mémoire principale, chaque adresse logique occasionne au moins 2 références à la mémoire
  - Une pour lire l'entrée de la table de pages
  - L'autre pour lire le mot référencé
- Le temps d'accès mémoire est doublé...

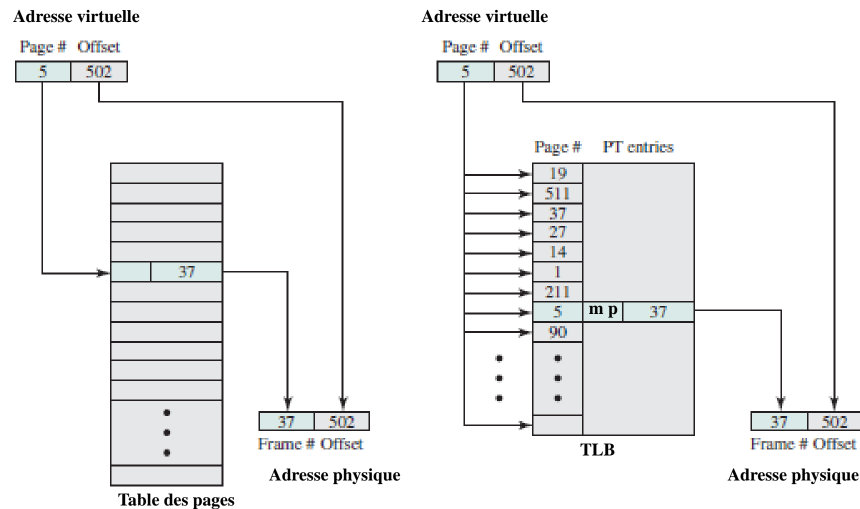
Pour améliorer l'efficacité

- Où mettre les tables des pages (les mêmes idées s 'appliquent aussi aux tables de segment)
- Solution 1: dans des registres du processeur.
  - avantage: vitesse
  - désavantage: nombre limité de pages par proc., la taille de la mémoire logique est limitée
- Solution 2: en mémoire principale
  - avantage: taille de la mémoire logique illimitée
  - désavantage: mentionné
- Solution 3 (mixte): les tableaux de pages sont en mémoire principale, mais les adresses les plus utilisées sont aussi dans des registres du processeur.

Registres associatifs TLB
Translation Lookaside Buffers, ou *caches* d'adressage

- Recherche parallèle d'une adresse:
  - L'adresse recherchée est cherchée dans la partie gauche de la table en parallèle (matériel spécial)
- Traduction page → cadre
  - Si la page recherchée a été utilisée **récemment** elle se trouvera dans les registres associatifs

TLB versus Accès direct



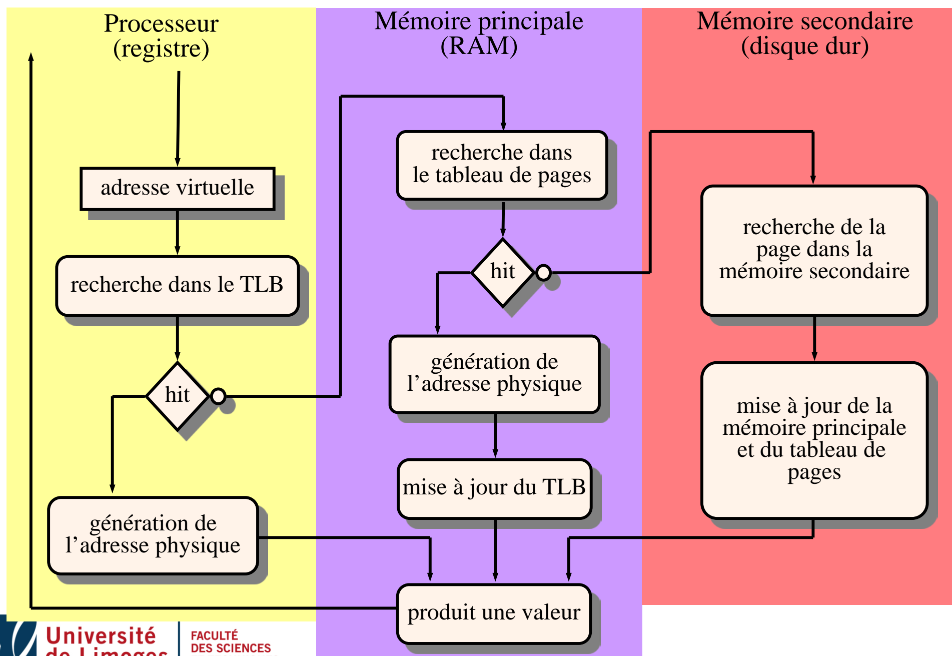
Translation Lookaside Buffer

- Sur réception d'une adresse logique, le processeur examine le cache TLB
- Si cette entrée de page y est, le numéro de cadre en est extrait
- Sinon, le numéro de page indexe la table de page du processus (en mémoire)
  - Cette nouvelle entrée de page est mise dans le TLB
  - Elle remplace une autre pas récemment utilisée
- Le TLB est vidé quand l'UCT change de proc

Les trois premières opérations sont faites par matériel

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Pagination - Algorithme



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Algorithmes de remplacement de pages

- Une faute de page force le système à choisir:
  - Quelle page doit être enlevée pour faire de la place pour la nouvelle page
- Les pages modifiées doivent d'abord être sauvegardées
  - Sinon on peut simplement les écraser
- Il est préférable de ne pas choisir une page souvent utilisée
  - Sinon il faudra la recharger bientôt

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

L'algorithme optimal

- Remplacer la page qui sera utilisée après toutes les autres
  - Optimal mais irréalisable
- On exécute deux fois un programme
  - La seconde fois on a l'information nécessaire pour implémenter l'algorithme de remplacement de page optimal
  - Permet de comparer les performances d'algorithmes réalisables avec l'algorithme optimal.

Algorithme de remplacement de la page non récemment utilisée (NRU)

- Chaque page possède deux bits: R et M
  - Ces bits sont mis à 1 lorsque la page est référencée ou modifiée (le bit R est remis à 0 périodiquement).
- 4 classes de pages
  1. non référencée, non modifiée
  2. non référencée, modifiée
  3. référencée, non modifiée
  4. référencée, modifiée
- NRU enlève une page au hasard
  - En partant du plus petit numéro de classe

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

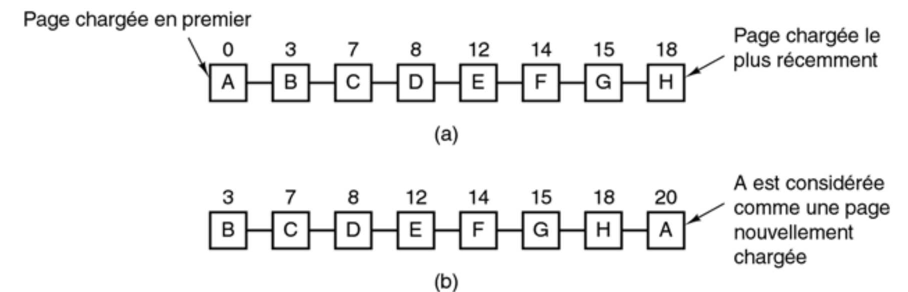
Algorithme premier arrivé, premier sorti

- On maintient un liste des pages
  - dans l'ordre de leur chargement en mémoire
- La page au début de la liste (la plus ancienne) est remplacée
- Désavantage
  - les vielles pages peuvent aussi être celles qui sont le plus utilisées.

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Algorithme de la seconde chance

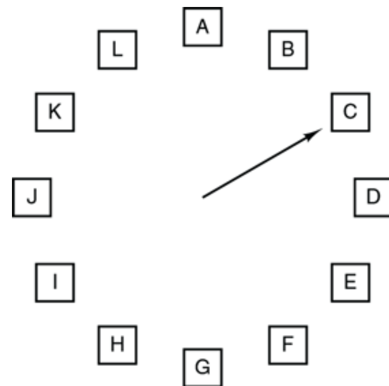
- Opération *seconde chance:*
  - Pages triées selon l'ordre d'arrivé
  - Si le bit R de la page la plus ancienne est 1 alors on la met à la fin de la liste comme une nouvelle page (son bit R à 0)



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

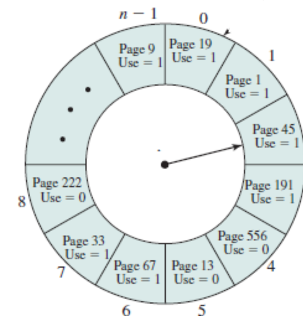
Algorithme de l'horloge

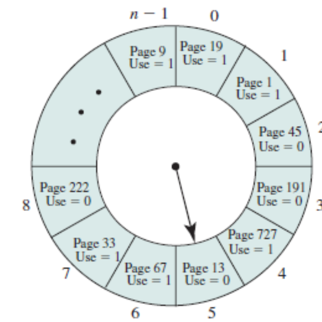
Comme l'algorithme précédent mais l'utilisation d'une liste circulaire évite de déplacer les pages.



Quand un défaut de page se produit, la page pointée est testée. L'action entreprise dépend de la valeur du bit R :
R = 0 : retirer la page
R = 1 : mettre R à 0 et avancer le pointeur

Algorithme de l'horloge





Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Algorithme de la page la moins récemment utilisée (LRU)

- On suppose que les pages qui ont été utilisées récemment le seront à nouveau prochainement
  - On enlève les pages qui n'ont pas été utilisées depuis longtemps
- Tenir à jour une liste chaînée de pages
  - Page la plus récente en premier
  - Mettre à jour cette liste à chaque référence mémoire !!
- On utilise un compteur C incrémenté après chaque instruction
  - On copie C dans la page des tables pour l'entrée de la page référencées
  - Choisir la page avec la plus petite valeur de compteur
  - On remet le compteur à 0 périodiquement

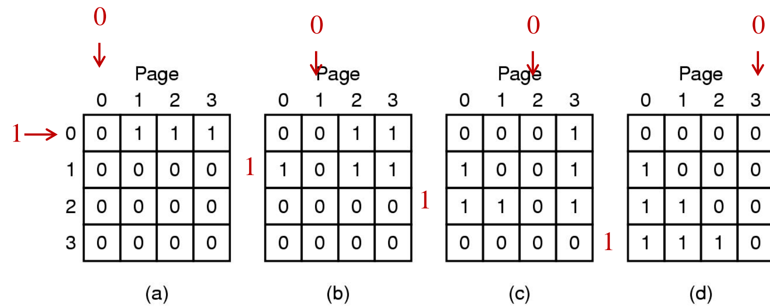
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Algorithme de la page la moins récemment utilisée (LRU)

- Autre solution matérielle
  - Pour une machine à n cadre
  - Matrice n×n
  - Quand une page k est référencée
    - mettre à 1 tous les bits de la rangée k
    - mettre à 0 tous les bits de la colonne k

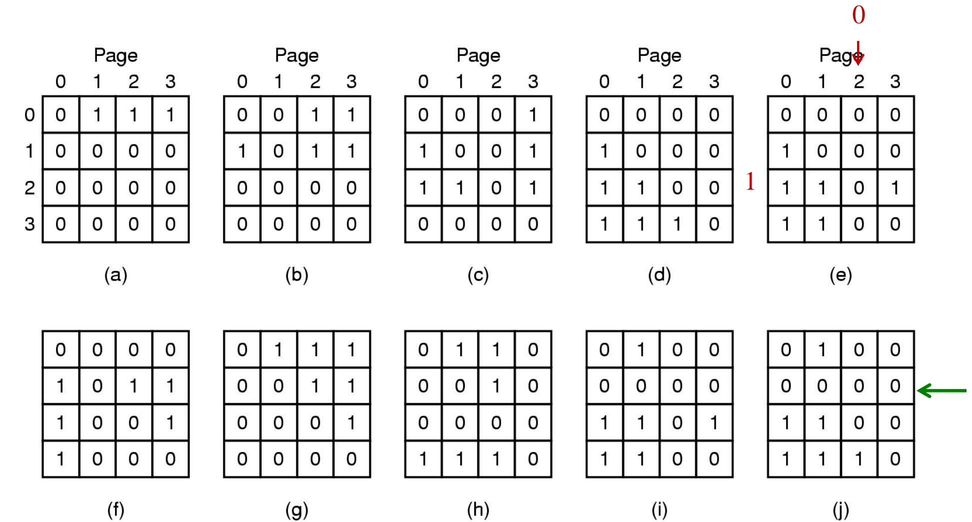
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Exemple – référence des pages: 0,1,2,3…2



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Exemple – référence des pages: 0,1,2,3,2,1,0,3,2,3



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Résumé des algorithmes de remplacement de pages

Algorithme	Commentaires
Optimal	*Utilisé comme banc d'essai*
NRU	*Très grossier*
FIFO	*Peut écarter des pages importantes*
Seconde chance	*Grande amélioration de FIFO*
Horloge	*Réaliste*
LRU	*Excellent mais difficile à implanter*
NFU	*Approximation grossière de LRU*
Vieillissement	*Approximation efficace de LRU*
Ensemble de travail	*Coûteux*
WSClock	*Bonne efficacité*

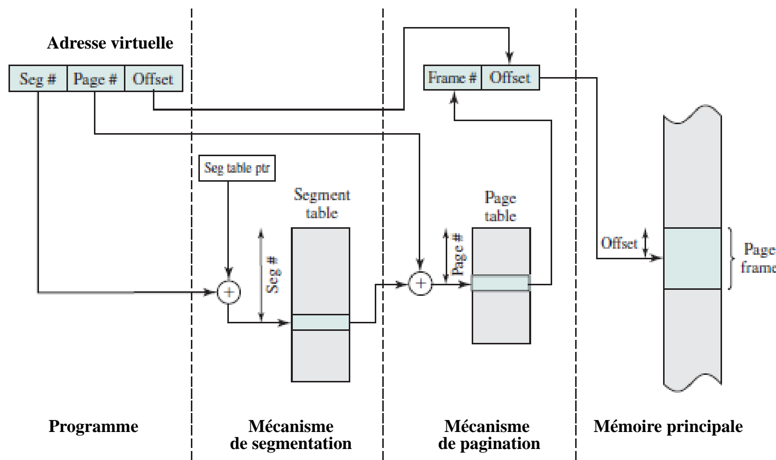
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Pagination et segmentation combinées

- Les programmes sont divisés en segments et les segments sont paginés
- Donc chaque adresse de segment n’est pas une adresse de mémoire, mais une adresse au tableau de pages du segment
- Les tableaux de segments et de pages peuvent être eux-mêmes paginés

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Pagination et segmentation



Utilisation de Translation Lookaside Buffer

- Dans le cas de systèmes de pagination à plusieurs niveaux, l'utilisation de TLB devient encore plus importante pour éviter les multiples accès en mémoire afin de calculer une adresse physique
- Les adresses les plus récemment utilisées sont trouvées directement dans la TLB.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Conclusions sur la Gestion Mémoire

- Problèmes de:
  - fragmentation (interne et externe)
  - complexité et efficacité des algorithmes
- Méthodes
  - Allocation contiguë
    - Partitions fixes
    - Partitions variables
    - Groupes de paires
  - Pagination / Segmentation
- Problèmes en pagination et segmentation:
  - taille des tableaux de segments et pages
  - pagination de ces tableaux
  - efficacité fournie par Translation Lookaside Buffer

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Récapitulation sur la fragmentation

- Partition fixes: fragmentation interne car les partitions ne peuvent pas être complètement utilisées + fragmentation externe s'il y a des partitions non utilisées.
- Partitions dynamiques: fragmentation externe qui conduit au besoin de compression.
- Segmentation sans pagination: pas de fragmentation interne, mais fragmentation externe à cause de segments de longueur différentes, stockés de façon contiguë (comme dans les partitions dynamiques)

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Récapitulation sur la fragmentation

- Pagination:
  - en moyenne, 1/2 cadre de fragmentation interne par processus
  - dans le cas de mémoire virtuelle, aucune fragmentation externe
- Donc la pagination avec mémoire virtuelle offre la meilleure solution au problème de la fragmentation

Comparaison entre la segmentation et la pagination

Considérations	Pagination	Segmentation
Le programmeur doit-il connaître la technique utilisée?	Non	Oui
Combien y a-t-il d'espaces d'adressage linéaires?	1	Plusieurs
L'espace total d'adressage peut-il dépasser la taille de la mémoire physique?	Oui	Oui
Les procédures et les données peuvent-elles être séparées et protégées séparément?	Non	Oui
Peut-on traiter facilement des tables dont les tailles varient?	Non	Oui
Le partage de procédures entre utilisateurs est-il simplifié?	Non	Oui

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Any question ?

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Système d'exploitation Ordonnancement

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard
Jean-louis.lanet@unilim.fr

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Plan

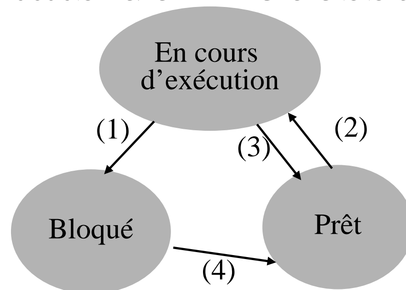
- Introduction : ordonnancement de processus
- Algorithmes classiques
- Systèmes temps réels

Processus

- Un processus est une activité: programme, entrées, sorties…
- Systèmes monoprocesseurs : *pseudo-parallélisme*
- Multiprogrammation: basculement entre processus
- Implémentation de Processus
  - Processus possède son propre espace d'adressage: programme, données, pile.
  - Le changement de contexte (changement de processus) → Table de processus, avec une entrée/ processus contenant registres, identificateur, ptr vers le segment texte, ptr vers segment de données, ptr vers le segment de pile, état …

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

États de Processus



- Le processus est bloqué, en attente d'une donnée, événement,
- L'ordonnanceur choisit un autre processus,
- L'ordonnanceur choisit ce processus,
- La donnée, l'évènement devient disponible.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Le problème

- Dans un système à processus :
  - de nombreux processus attendent qu'un événement se produise…
    - ils n'ont pas immédiatement besoin du processeur…
    - … mais doivent pouvoir l'obtenir dès que l'événement attendu se produit
  - certains processus font des calculs de façon intensive, sans attente d'événements
    - ils souhaitent garder le processeur le plus longtemps possible
  - Conflit d'intérêt : ordonnanceur (*scheduler*) = arbitre + chef d'orchestre

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Objectifs d'un ordonnanceur

- Rôle d'un algorithme d'ordonnancement :
  - décider de l'allocation d'une ressource aux processus qui l'attendent, pour atteindre certains objectifs
  - dans la suite, « processus » (au sens large) signifie un:
    - processus (au sens Unix, processus « lourd »),
    - thread : fil d'exécution à l'intérieur de la mémoire d'un processus
- Exemple de ressource : le processeur
  - Objectif : aboutir à un partage efficace du temps d'utilisation du processeur
  - Problème : que veut dire efficace ? Et pour qui?

Critères d'efficacité pour le CPU

- Respect de la priorité
  - La plupart des systèmes permettent d'accorder des priorités différentes aux processus…
  - Priorité peut être statique ou dynamique (se modifie au cours du temps) …
- Respect de l'équité
  - Deux processus qui ont le même niveau de priorité doivent pouvoir utiliser le CPU aussi souvent l'un que l'autre

Critère d'optimisation pour l'ordonnancement du CPU

- Utilisation maximale du processeur
  - Maximiser: Taux Utilisation(CPU) = Durée Activité(CPU) / Durée Totale
- Débit processus
  - Maximiser Débit = Nombre Processus Terminés / Unité Temps
- Temps de traitement moyen :
  - doit être minimal pour un traitement batch
- Temps de réponse maximum :
  - doit être minimal pour un traitement interactif ou temps réel

Classification des algorithmes d'ordonnancement

- Dans un monde idéal (statistiquement) :
  - le hasard devrait bien faire les choses : les processus endormis ne devraient pas se réveiller tous en même temps
- Dans la réalité :
  - les activités des processus sont « corrélées » : les processus ne se réveillent pas au hasard …
- Deux familles d'algorithmes :
  - Sans réquisition : c'est aux processus de relâcher volontairement la ressource (non préemptif)
  - Avec réquisition : l'ordonnanceur peut récupérer la ressource détenue par un processus au profit d'un autre (préemptif)

Mécanismes de base nécessaires

- L'activation de l'ordonnanceur est possible
  - À chaque entrée dans le noyau, à chaque appel système,
  - À chaque interruption du matériel : disque, horloge, …
- Chaque appel système peut donc potentiellement activer un autre processus
- Ressources de type CPU :
  - commutation de contexte
    - Pour permettre à un autre processus d'utiliser la ressource,
    - Contexte peut être en partie matériel (registres, état),
- Algorithmes avec réquisition : besoin d'horloge
  - pour contrôler la durée d'utilisation
  - pour percevoir l'écoulement du temps: interruptions périodiques pour mesurer le temps passé et lancer des actions à des dates fixées

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels

Sans préemption

- Ressource allouée à une entité jusqu'à ce qu'elle n'en ait plus besoin
  - Par nécessité (ex: imprimante)
- Inconvénients :
  - ne peut convenir aux activités « temps réel »
  - convient difficilement aux activités interactives :
    - Obligation de programmer des applications « sociables »
    - Tolérable dans un système faiblement mono utilisateur (Windows 3.x, 95, …)
  - ne correspond pas à de vrais processus indépendants (il s'agit en fait de co-routines)
- Avantages :
  - facile à mettre en œuvre
  - pas besoin de mécanismes matériels spécifiques…(horloges, interruptions)

Mise en oeuvre

- Au moment de la libération de la ressource :
  - L'ex-détenteur de la ressource invoque l'algorithme d'ordonnancement
    - Cette action peut être réalisée à l'insu du programmeur (exemple : win3x, win9x)
  - L'algorithme **choisit** le processus suivant
  - L'algorithme déclenche la commutation de contexte

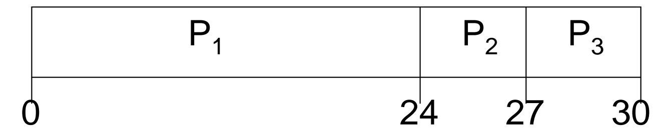
Politique de choix : FIFO

- Politique « FIFO » (First In First Out)
- Allocation dans l'ordre d'arrivée (premier arrivé = premier servi)
- Inconvénient : défavorise les entités ayant besoin d'utiliser la ressource un court laps de temps
  - Le temps d'attente n'est pas proportionnel au temps d'utilisation
    - pas équitable,
    - temps moyen de traitement élevé

Ordonnancement FIFO

Processus	Tps CPU
P_1	24
P_2	3
P_3	3

- Supposons que les processus arrivent dans l'ordre suivant: P_1 , P_2 , P_3 Le diagramme correspondant est:



- Temps d'attente de $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Temps d'attente moyen: (0 + 24 + 27)/3 = 17

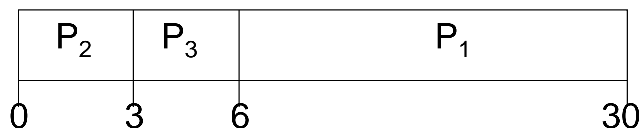
Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Ordonnancement FIFO

Supposons que les processus arrivent dans l'ordre suivant

P_2 , P_3 , P_1

- Le diagramme de Gantt serait alors:



- Temps d'attente de P_1; P_2, P_3 ?
- Temps d'attente moyen: ?
- Conclusion ?

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

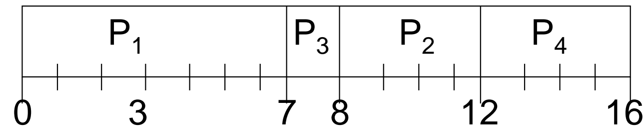
Politiques de choix : PCTU

- Politique PCTU (Plus Court Temps d'Utilisation d'abord)
- Allocation selon ordre croissant de durée d'utilisation prévue
- Inconvénients
  - Pas réaliste : exige la connaissance a priori des durées d'utilisation
  - Famine (privation) : les tâches dont la durée d'exécution estimée est longue peuvent attendre leur tour indéfiniment …
- Avantages
  - Temps d'attente faible pour entités à courte durée d'utilisation
  - Temps moyen d'attente minimal
- Il est optimal – donne un temps moyen minimal pour un ensemble de processus donnés

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Exemple de pctu

Processus	Tps d'Arrivée	Tps CPU
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- Temps moyen d'attente = (0 + 6 + 3 + 7)/4 = 4

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Politique de choix : FIFO avec priorité

- Politique FIFO avec priorités
  - Chaque entité a une priorité
  - Une file FIFO par niveau de priorité
  - Ressource allouée à une entité ssi
    - FIFOs de priorités supérieures vides & la ressource est en tête de sa FIFO
  - Inconvénients
    - Tout le monde veut la plus haute priorité…
    - Famine pour entités de faible priorité
  - En pratique
    - utilisation parcimonieuse des priorités élevées
    - modification dynamique des niveaux de priorité

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Description

- Motivations
  - politiques sans réquisition mal adaptées, voire inadaptées, à certaines activités
    - temps réel
    - interactivité
- La réquisition permet:
  - de forcer le partage du temps d'utilisation (modulo les contraintes de priorités)
  - de diminuer le temps de traitement maximum
  - mais cela détériore le temps de traitement moyen (overhead = frais de gestion, temps passé dans le noyau), donc diminue le débit

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Mise en oeuvre

- Le détenteur de la ressource peut être interrompu avant d'avoir terminé :
  - lorsqu'un délai maximal expire
  - lorsqu'un processus de priorité plus élevée demande la ressource
- La politique d'ordonnancement choisit le nouveau processus
- Le processus interrompu est mis «en sommeil» (état prêt)
- C'est la politique utilisée dans les systèmes « à temps partagé » (time-sharing) : Unix, NT…

Problème des fonctions non réentrantes

- Un processus peut être interrompu alors qu'il exécute une fonction de l'exécutif
  - Problème des fonctions non ré-entrantes dans la même mémoire :
  - Le nouveau/futur élu peut demander à son tour l'exécution de la même fonction :
    - Réutilisation d'une même variable globale
    - Insertion non terminée dans une liste chaînée
    - …
  - Solution : retarder la commutation jusqu'à ce que l'exécution atteigne un point de commutation, c.-à-d. contrôler les sections critiques du noyau.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

La politique du tourniquet

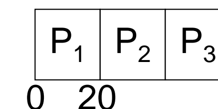
- Politique du Tourniquet (Round Robin / RR)
  - idée : Fournir à l'une quelconque des *n* entités en attente, *1/n* ème du temps d'utilisation de la ressource
  - pb : *n* varie au cours du temps
  - Solution : le temps est découpé en tranches de taille (durée)
  - identique, appelées quantum de temps
    - Les entités sont placées dans une file
    - La ressource est allouée à l'entité en tête de la file, pour une durée d'au maximum un quantum
    - Lorsque le quantum est épuisé, l'entité est interrompue et replacée à la fin de la file d'attente

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Exemple de RR avec Q = 20

Processus	Temps CPU
P_1	53
P_2	17
P_3	68
P_4	24

Le diagramme de Gantt est:

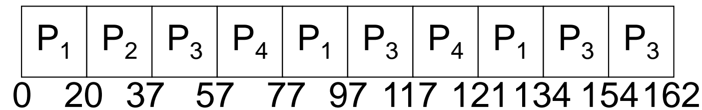


Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Exemple de RR avec Q = 20

Processus	Temps CPU
P_1	53
P_2	17
P_3	68
P_4	24

- Le diagramme de Gantt est:



- Typiquement, une moyenne de temps de rotation plus grande que pctu, mais un meilleur *temps de réponse*

Tourniquet : choix du quantum

- Durée ni trop courte …
  - Lorsque la durée du quantum est écoulée, il faut déclencher un changement de contexte
  - Un changement de contexte prend du temps : Plus le quantum de temps est petit, plus on perd souvent du temps à changer de contexte !
- … ni trop longue :
  - L'illusion d'exécution parallèle s'estompe
  - Lorsque la durée est trop longue, l'interactivité diminue
    - Exemple : quantum = 1s, 3 tâches de longue durée sont présentes et attendent le processeur. Elles comptent utiliser systématiquement tout leur quantum.
    - Une tâche interactive ne peut obtenir le processeur au mieux que toutes les 3 secondes (délai entre frappe clavier et affichage d'au moins 3 secondes …)

Tourniquet : partage du temps CPU

- Horloge programmée
  - Déclenche une interruption à intervalles de temps réguliers
    - interruption appelée « tic d'horloge » : tic, tac, …
  - Le traitant de l'interruption :
    - 1. Décompte du temps d'occupation CPU pour l'entité courante (initialisé à la valeur de quantum)
    - 2. Si temps restant = 0 : lancer algorithme d'ordonnancement pour choisir un nouveau processus
    - 3. Autres actions non liées à l'ordonnancement
    - 4. actions liées à l'ordonnancement effectuées à des tics principaux (versions évoluées du tourniquet)
    - 5. Commutation de contexte vers entité élue

Politique d'ordonnancement préemptif

- En théorie : à tout instant la ressource est détenue par le processus de plus haute priorité
  - il faut donc retirer la ressource au processus qui la possède lorsqu'elle n'est plus la plus prioritaire
- En pratique :
  - il suffit que l'exécutif regarde la priorité d'un processus qui naît ou se réveille
  - si la commutation est retardée (cf. pb ré-entrance), on parle d'inversion de priorité

Préemption : problème de famine

- Problème : famine des entités de faible priorité
  - Solution : ajustement dynamique des priorités
    - Plus une entité attend longtemps, plus sa priorité augmente
    - Lorsqu'une entité obtient (enfin) la ressource, sa priorité redescend au niveau initial
- Conséquence : provoque de nombreux changements dans les files de priorités
  - Car les processus de même niveau de priorité sont placés sur une même file (généralement FIFO)
  - La gestion des files doit être efficace !

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels
  - Introduction
  - Tâches périodiques
  - Tâches apériodiques

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Introduction

- Deux types de systèmes
- Synchrone
  - Existence d'une base de temps commune,
  - Les évènements n'arrivent pas n'importe quand
- Asynchrone
  - Pas d'hypothèse sur les instants où les évènements peuvent se produire
- Le monde synchrone est plus « simple », le monde réel est plutôt asynchrone

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Déterminisme

- Pouvoir garantir que le système respectera ses spécifications, notamment temporelles, pendant sa durée de vie
  - Ré-exécution donne des résultats identiques
- Méthodologie
  - Déterminer les cas pires
  - Conditions de faisabilité (CF)
  - Déterminer valeurs numériques CF
  - Vérifier

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Algorithme déterministe

- Temps maximum d'exécution garanti
- Indépendant du contexte courant
- Indépendant de la valeur des arguments
- Principe valable pour toutes les fonctions d'un même service
  - allocation / libération pour gestion mémoire
- S'applique aux séquences d'exclusion mutuelle
  - Pas d'effet(s) de bord sur reste du système

Caractéristiques temporelles

- Durée maximum (pire cas) d'un thread TH_i : C_i
  - Thread seul sans interruption
  - Par analyse ou par mesure
- (Pire) temps de réponse d'un thread TH_i : R_i
  - Temps entre demande activation et réponse
  - Prend en compte délai dans exécution induit par les autres threads et l'overhead OS
- $R_i \geq C_i$

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Contraintes Temporelles

- Échéance de terminaison au plus tard
- Thread TH_i activé à instant t_i doit être terminé au plus tard à instant $t_i + D_i$
- D_i : échéance relative
- $t_i + D_i$: échéance absolue

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Plan

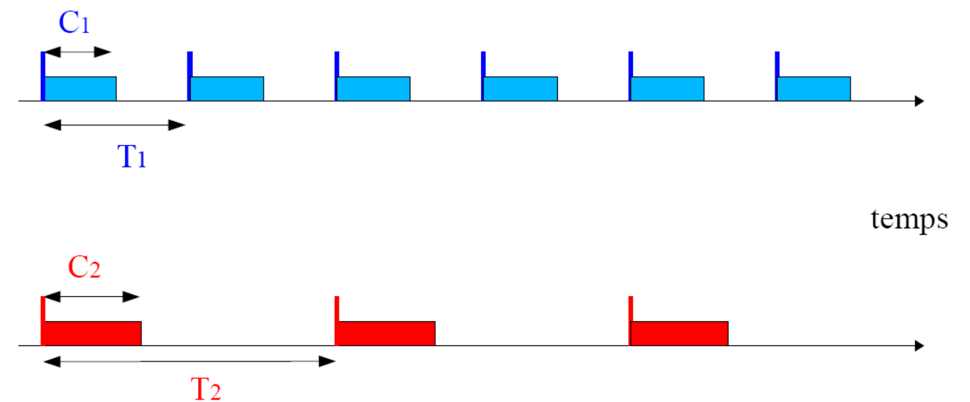
- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels
  - Introduction
  - Tâches périodiques
  - Tâches apériodiques

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Systèmes Périodiques

- Chaque thread THi activé périodiquement
  - Période activation Ti
  - Échéance relative Di (en général Ti = Di)
- La kième instance du thread THi
  - Est activée à (k-1) Ti
  - Doit être terminée au plus tard à k Ti
- Hyper période = PPCM(Ti) i = 1,...,n

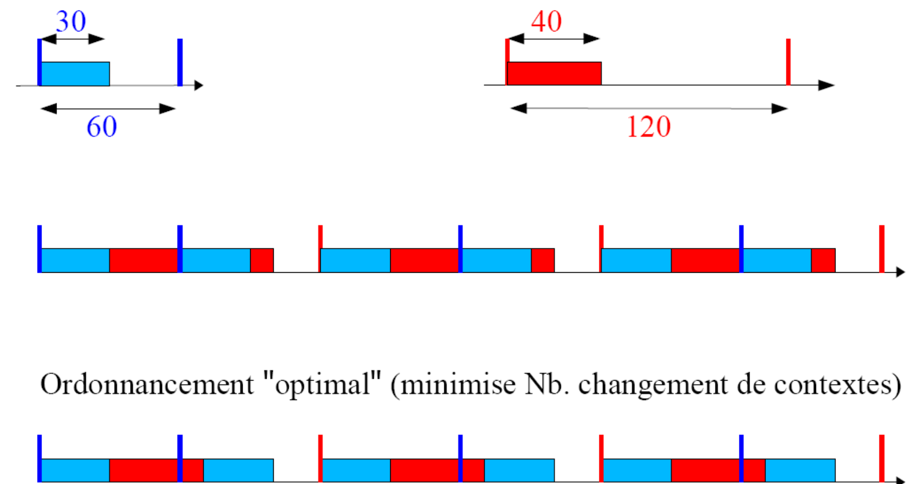
Systèmes Périodiques



RMA

- Priorité déterminée en fonction de la période
- Plus la période est petite, plus la priorité est élevée
- Optimal
  - Pour systèmes périodiques
  - Avec ordonnancement préemptif

Ordonnancement Rate Monotonic



Test d'ordonnancement

- *Liu and Layland* ont démontré que lorsque la condition suivante est rencontrée on aura toujours un résultat d'ordonnancement **:**

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i}\right) < N(2^{1/N} - 1)$$

- Quand N -> ∞ le terme de droite tend vers 69.3%

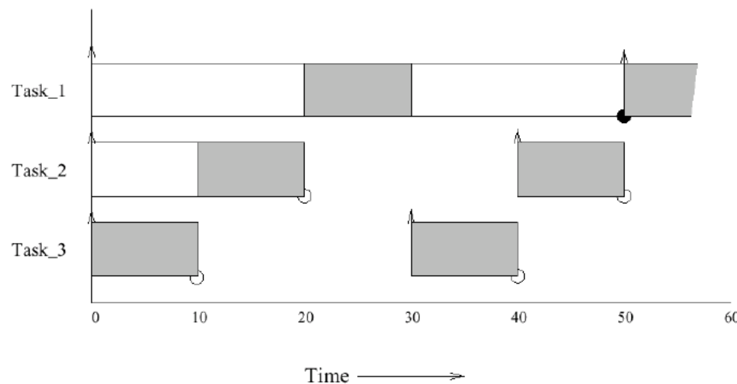
Condition d'ordonnancement

Exemple 1: Soit l'ensemble de tâches suivant à ordonnancer:

	Period T	Computation Time, C	Priority P	Utilization U
Task_1	50	12	1	0.24
Task_2	40	10	2	0.25
Task_3	30	10	3	0.33

Que peut on dire ?

RMA



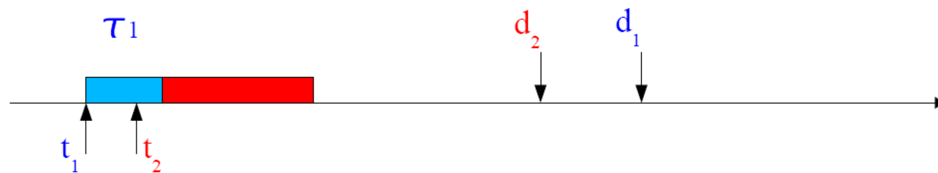
On voit graphiquement que l'ordonnancement est impossible.

Plan

- Introduction : ordonnancement de processus
- Algorithmes classiques
  - Sans préemption
  - Avec Préemption
- Systèmes temps réels
  - Introduction
  - Tâches périodiques
  - Tâches apériodiques

Tâches Sporadiques

- Chaque thread THi
  - Activée à un temps ti
  - Durée maximum d'exécution ci
  - Échéance absolue di = ti + Di
- Ordonnancement possible



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Tâches Sporadiques

- Ordonnancement statique
  - Ensemble (ti ,ci ,di) i = 1,...,n connu avant exécution
  - Construire un ordonnancement qui respecte les échéances de chaque thread (ordonnancement faisable)
- Ordonnancement dynamique
  - A chaque "moment d'ordonnancement", déterminer la prochaine thread à exécuter
  - Satisfaire les échéances de toutes les threads

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

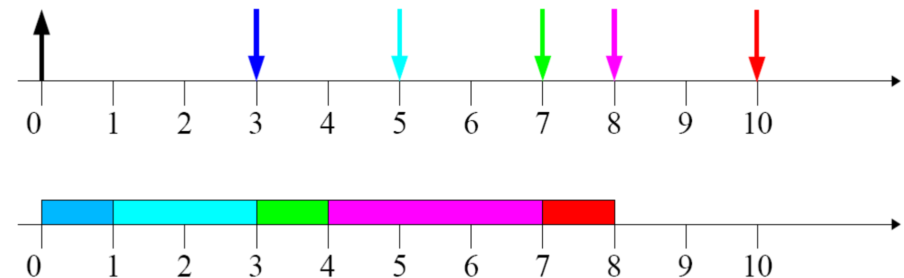
Ordonnancement EDF

- **E**arliest **D**eadline **F**irst
- Ordonnanceur dynamique
- Donne priorité à la thread la plus proche de son échéance de terminaison
- Doit trier les threads en fonction de leurs échéances
- Optimal si système non surchargé
- Comportement non-prédictible en cas de surcharge

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

EDF / même temps d'activation

5 threads : (0,1,3), (0,1,10), (0,1,7), (0,3,8), (0,2,5)

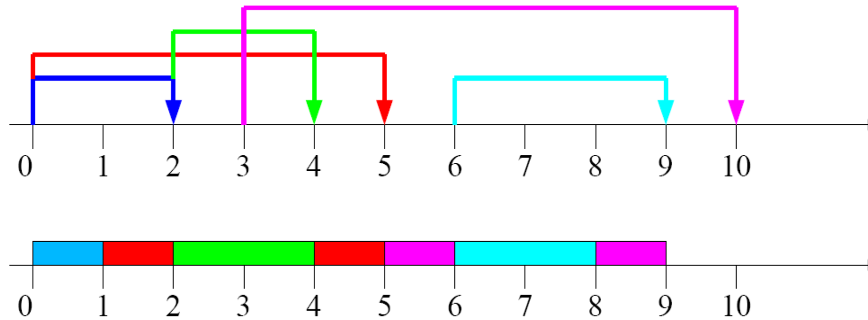


préemption pas nécessaire

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

EDF / temps d'activation différents

5 threads : (0,1,2), (0,2,5), (2,2,4), (3,2,10), (6,2,9)



=> Préemption nécessaire pour satisfaire échéance de (2,2,4)

Inconvénients

- Ordonnancements basés sur le temps
  - Supposent CPU est la seule ressource partagée
- Dépendants
  - Caractéristiques matériel
    - CPU (instructions, fréquence, cache(s), etc...)
    - Performances bus (mémoire, I/O, etc)
  - Compilateurs
- Très sensibles aux évolutions du logiciel
  - Correction de bugs
  - Ajout de tâches

Inconvénients

- Décomposition des activités en blocs d'exécution synchrones :
- thread = (activation, durée maximum, échéance)
  - Figé
  - [relativement] simple
- Décomposition en étapes asynchrones de priorités différentes :
- étape = interruption / thread
  - Plus souple
  - Plus complexe

Any question ?

Système d'exploitation
Allocation - désallocation mémoire

Licence Informatique

Jean-Louis Lanet / Guillaume Bouffard
Jean-louis.lanet@unilim.fr

Plan

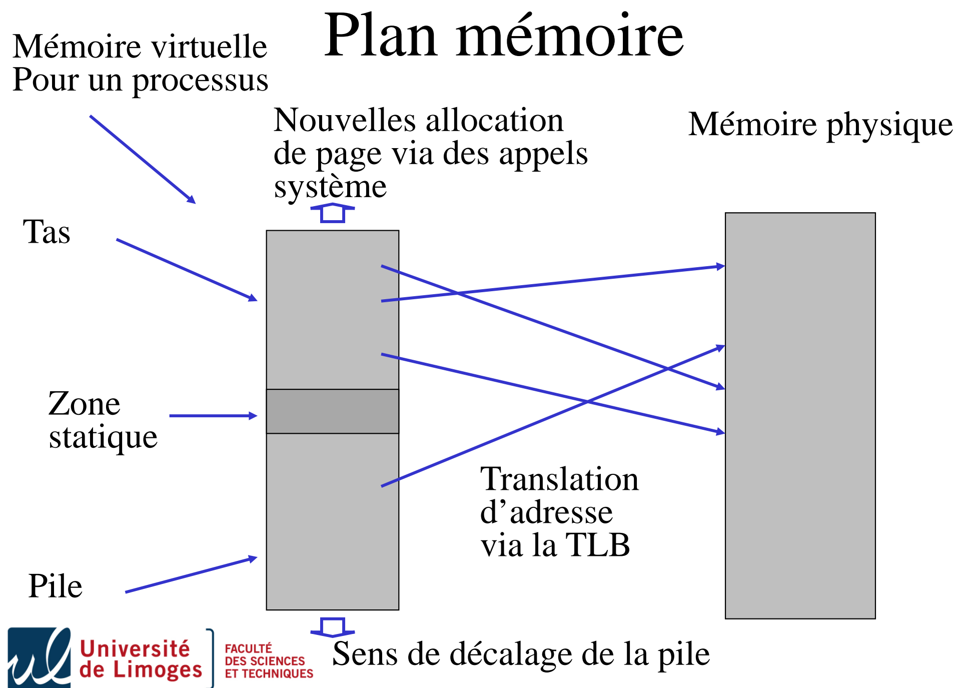
- Garbage collector
  - Garbage collector
    - Garbage collector
      - Garbage collector
        - » Garbage collector
          - » Garbage collector
            - » Garbage collector

Gestion de la mémoire

- Lorsque le programme s'exécute il peut avoir besoin de mémoire dynamiquement,
  - Allocation d'objet, création de tableau, extension de tableau etc.
  - La mémoire allouée n'est pas infinie et il faut la gérer proprement,
  - La gestion manuelle est source d'erreur, mauvaise libération fuite de mémoire etc.
  - La gestion manuelle se fait à l'aide d'appel à la librairie (`free`,..),
  - Déléguer au système cette gestion soulage le programmeur et permet une plus grand efficacité.

Différentes façons de faire

- Tous les langages de programmation moderne autorisent les programmeurs à allouer de l'espace de stockage dynamiquement,
  - Nouveaux enregistrements, des tableaux, des objets, etc.
- De même, ces langages ont besoin d'un mécanisme pour recycler la mémoire abandonnée,
- C'est généralement l'aspect le plus difficile des run time de tout langage moderne (Java, ML, Lisp, Scheme, Modula, …)



Garbage = détritus

- C'est quoi un détritus ?
  - une cellule mémoire est considérée comme détritus si elle ne sera plus usée par le programme quelque soit le chemin utilisé par le programme.
- Est il simple de décider si une cellule mémoire doit être considérée comme un détritus ?

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Garbage = détritus

- C'est quoi un détritus ?
  - une cellule mémoire est considérée comme détritus si elle ne sera plus utilisée par le programme quelque soit le chemin utilisé par le programme.
- Est il simple de décider si une cellule mémoire doit être considérée comme un détritus ?
- C'est un problème indécidable (cf. théorème de Rice)

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Détritus

- Puisque déterminer si une cellule mémoire est un détritus est indécidable, on utilise des techniques approchées.
  - Soit on laisse le travail au programmeur par allocation et dés allocation explicite
  - On évite en les détritus en faisant le ménage grâce à la pile,
  - On compte les références sur un objet,
  - On utilise un GC automatique : Mark-sweep, copying collection, Generational GC

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Type d'allocation

- Allocation statique
  - Le plus simple si tout est connu à la compilation
  - Pas de récursivité, taille fixe des données
  - Fortran 77
- Allocation via la pile
  - À chaque appel de la mémoire est allouée,
  - Taille des données fonction de paramètres d'appel,
  - Pas de rémanence des données,
  - Algol 58
- Allocation du tas
  - Au-delà de la durée de vie d'un appel, rémanence, objet dynamique,
  - Gestion explicite Pascal, C, C++, implicite … Java, Scheme,…

Gestion mémoire en pile

- Exemple:

```
void les_20_premiers_premiers(){
int premiers[] = new int[20];
// variable premiers en pile,

for (int i=0;i<20;i++)
  premiers[i]=calculer_premier(i);
afficher_tableau_entiers(20,premiers);

} // ici, premiers est dépilé, elle peut
// être automatiquement recyclée/libérée
```

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

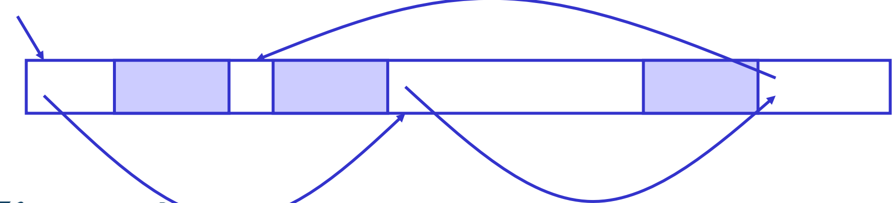
Le programmeur fait le travail

- Le programmeur utilise la librairie pour choisir quand allouer et dés allouer :
  - `void* malloc(long n)`
  - `void free(void *addr)`
  - Le système gère le besoin en mémoire en demandant des pages supplémentaires si nécessaire,
  - L'avantage c'est que le programmeur est intelligent ,
  - L'inconvénient c'est que le programmeur est généralement un peu bête , oublie de dés allouer et n'a pas envie de s'ennuyer avec ces détails.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Gestion explicite de la mémoire

- Fonctionnement de `malloc/free` ?
  - Les blocs de mémoire libres sont stockés dans une liste chainée,
  - `malloc`: recherche dans cette liste des blocs de mémoire libre,
  - `free`: insère en tête de liste le bloc libéré

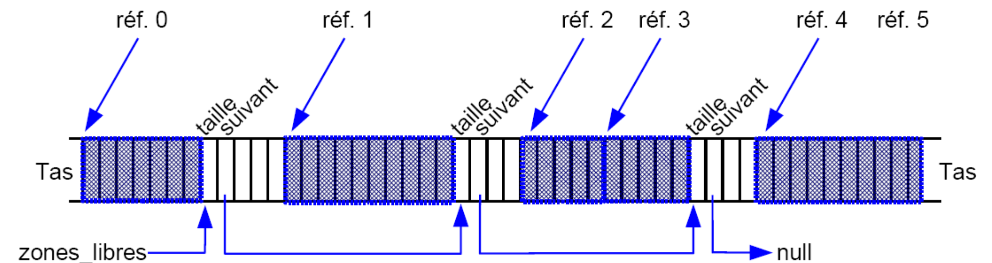


Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Gestion explicite de la mémoire

- Solutions:
  - Utilisation de liste multiple par taille de bloc,
    - `Malloc` and `free` sont alors en O(1)
    - Mais si le besoin est de 4 blocs et que cette liste est vide on est en famine alors que des blocs plus grands existent.
  - Les blocs sont de taille en puissance de 2
    - Subdiviser les blocs pour obtenir la bonne taille
    - Les blocs adjacent sont réunis pour former des blocs plus gros,
    - Néanmoins on a 30% d'espace mémoire perdu en moyenne.
  - La gestion mémoire a toujours un coût.

Liste des blocs libres



Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Gestion mémoire automatique

- Système automatique de gestion mémoire = ramasse-miettes
- Doit pouvoir garder (ou retrouver) les données/objets encore actifs, et du coup être capable de libérer les autres
- Connaît les zones mémoire libres (allouables) et celles qui sont occupées,
  - Avant très consommateur de temps,
  - Surcout dépend de la façon de programmer, du type d'objet etc.

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Gestion automatique

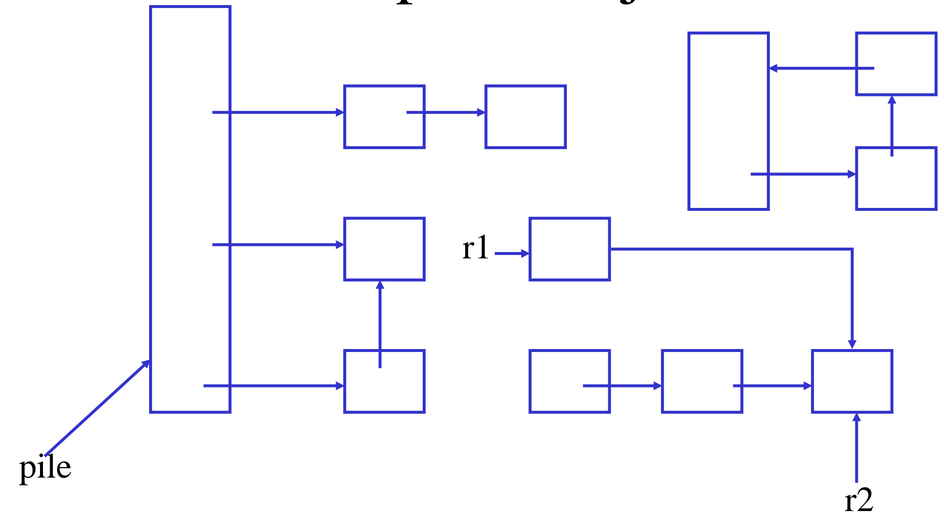
- Un système remplace le développeur en allouant et surtout dés allouant automatiquement
  - impossible d'oublier de dés allouer (mais possible de continuer de référencer à tort)
  - plus difficile (si pointeurs) voire impossible (si références) d'utiliser une donnée déjà libérée
  - Détritus : plus de référence avant la dés allocation: la mémoire est perdue (*memory leakage*),
  - Pointeur flottant (*dangling reference*) deux pointeurs sur le même objet, l'un dés alloue la mémoire et l'autre pointe sur du vide…

Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Gestion automatique

- Quand peut on décider qu'il est temps de dés allouer la mémoire ?
  - On ne le sait pas précisément,
  - Donc on prend une approche conservative
  - Solution évidente lorsqu'il devient inatteignable depuis une **racine**,
    - Les racines: les registres, la pile, les données statiques
    - Si à partir d'une racines l'objet est inatteignable alors c'est un détritus.

Graphe d'objet



– Comment évaluer l'atteignabilité ?

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Algorithmes de GC

- Comptage de références
- Marquage - balayage
- Copie - compactage

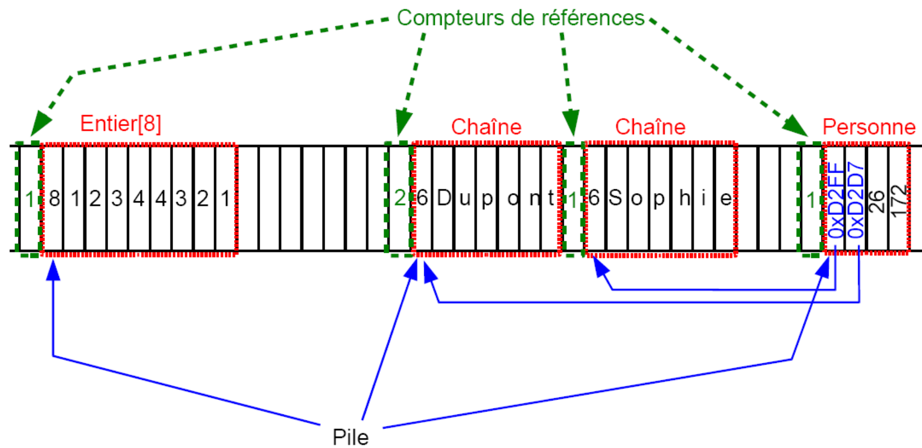
Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Comptage de référence

- À chaque objet (ou donnée, structure, zone mémoire) alloué est associé un compteur (entier) indiquant le nombre de références sur cet objet.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Comptage de référence



Comptage de référence

- A chaque affectation, on met à jour les compteurs concernés:

```
a = new X();   // nb_réf(X1)=1
b = a;         // nb_réf(X1)=2
a = new X();   // nb_réf(X1)=1; nb_réf(X2)=1
b = null;      // nb_réf(X1)=0: X1 libérable
```

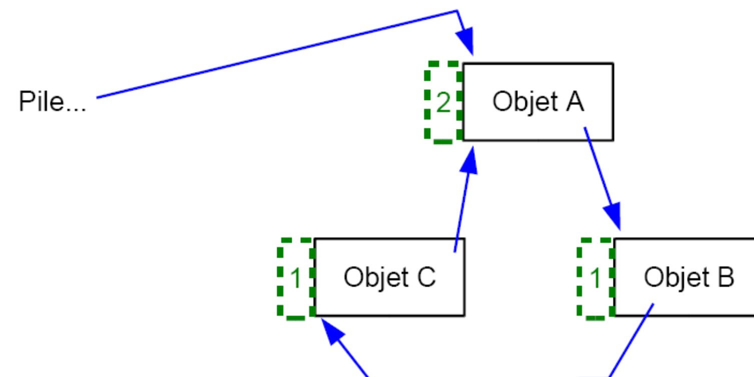
- Libération peut être immédiate (+ simple) ou différée

Comptage par référence

- Libérer un objet X1:
  - récupérer la mémoire de X1 (remise en liste libre)
  - diminuer les compteurs de références des objets pointés par X1
  - libérations en cascade possibles (peut prendre du temps)

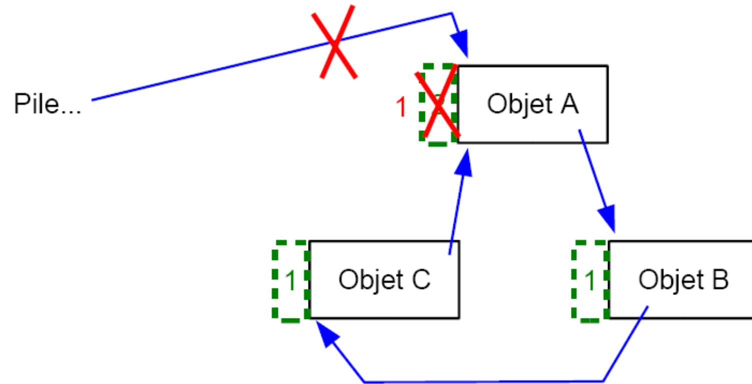
Comptage de référence

- Le problème des cycles



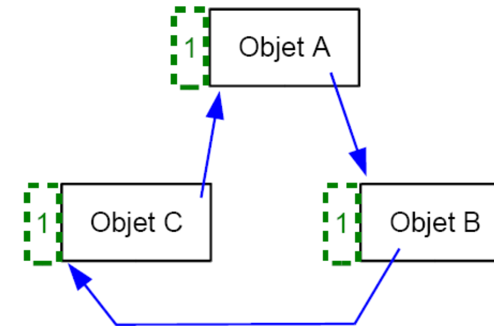
Comptage de référence

- Le problème des cycles



Comptage de référence

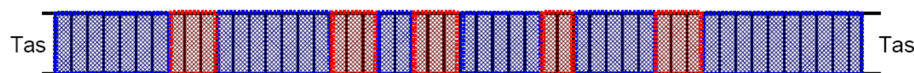
- Cycles non détectés
  - Besoin en plus d'un système de détection de cycles
  - Le cycle A/B/C n'est plus référencé, mais ses compteurs sont >0, donc les objets ne sont pas collectés...



Comptage de référence

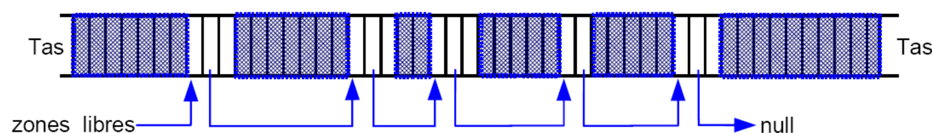
- Problème de la fragmentation

A l'instant T1:



Le programme s'exécute, des objets sont libérés

A l'instant T2 > T1:

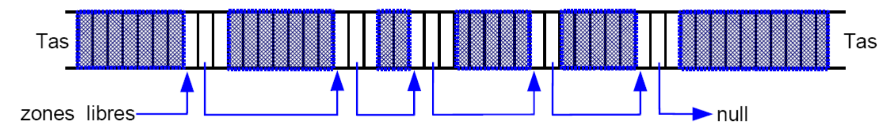


Comptage de référence

- Fragmentation possible

A l'instant T2, on veut allouer 5:

Mais pas de zone mémoire assez grande disponible:



Pourtant il y a de la mémoire libre (14 en tout): c'est la fragmentation.

Comptage de référence

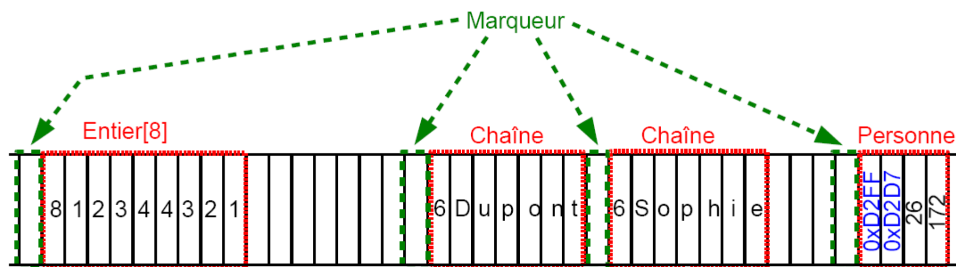
- Bilan
  - Simple
  - Exécution répartie le long de celle du programme.
  - Coûteux au total: MAJ de compteur(s) à chaque affectation
  - Pas de gros délai si objets libérés un par un.
  - Délais si cascades...
  - Problème des cycles
  - Fragmentation possible

Marquage balayage

- Le ramasse-miettes se déclenche par intermittence
  - Exécution du ramasse-miettes arrête le programme temporairement
- Lorsque ramasse-miettes se déclenche:
  - phase de marquage : trouver les vivants
  - phase de balayage : recycler les morts

Marquage balayage

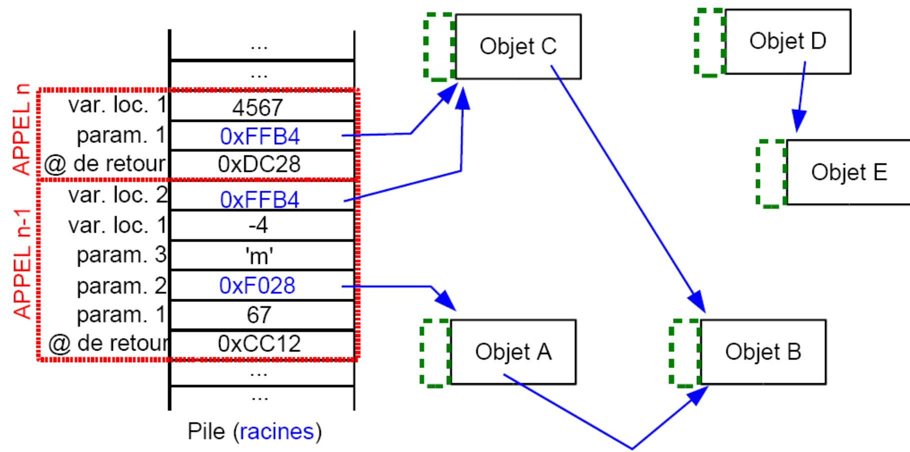
- A chaque objet alloué est associé un marqueur (ou drapeau, ou *mark flag*)



Algorithme de marquage

- Partir des racines (piles, zone statique) du graphe d'objets
- Pour chaque objet rencontré
  - s'il est déjà marqué, rien à faire
  - sinon le marquer et propager l'algorithme sur tous les objets qu'il référence
- Quand le marquage se termine, on a tous les actifs. Les autres sont morts.

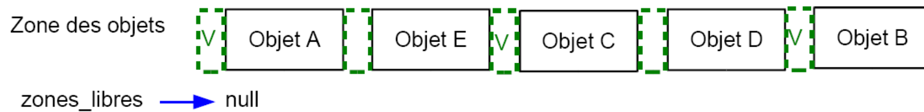
Algorithme de marquage



Algorithme de balayage

- On parcourt la liste de toutes les zones mémoires.
- Si marqué, on conserve (on démarque pour le coup suivant),
- Si pas marqué, on intègre la zone dans la liste libre.

Algorithme de balayage



Balayage: drapeaux vidés, zones libres en liste

Marquage - balayage: bilan

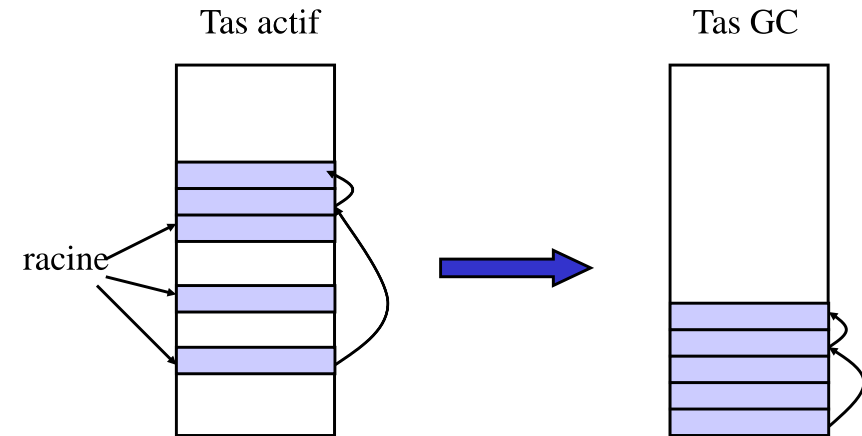
- Pauses longues: marquage + balayage
  - durée du marquage dépend de la taille du graphe d'objets (surtout les vivants)
- Amélioration: incrémental (pauses fractionnées)
- Les cycles ne sont pas un problème
- Fragmentation possible

Copie compactage

- Principe: parcours du graphe d'objets (comme marquage-balayage)
- Recopie des vivants dans nouvel espace mémoire (de façon contigüe), les morts sont abandonnés sur le champs de bataille,
- Le nouveau tas est le recopié,
- L’autre devient le futur tas du GC.

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

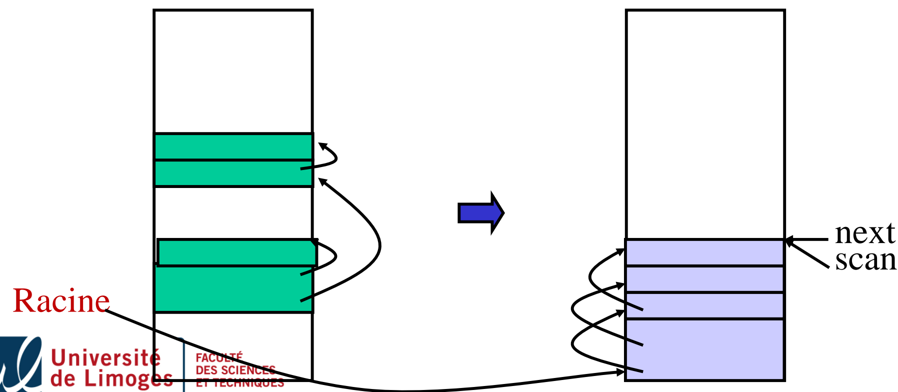
Copie compactage



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Copie

- Algorithme de Cheny’s
  - Traversée en largeur, copie du tas



Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

Copie - compactage: bilan

- Parcours comme marquage
- Recopie coûteuse
- Gestion de « forwarding pointers »
- Double espace mémoire
- Pas de fragmentation

Université de Limoges FACULTÉ DES SCIENCES ET TECHNIQUES

GC générationnel

- Observation 1 : un objet vieux et vivant a une très forte probabilité de le rester,
- Observation 2 : de nombreux objets sont créés avec une durée de vie très courte,
- Conclusion : la durée de vie d'un objet est le signe de non détritus.

GC générationnel

- Assigner aux objets différentes générations G0, G1,…
  - G0 contient des objets récents fortte probabilité à recycler
  - G0 parcouru plus souvent que G1
  - Cas général utilisation de deux générations
  - Racines de G0 inclus tous les objets de G1 en plus de la pile et des registres.

GC générationnel

- Comment éviter de scanner de vieux objets ?
  - Observation: les vieux pointent rarement les jeunes
    - Après la création d'un objet, son initialisation pointera vers des objets plus vieux
    - Si un objet ancien est modifié longtemps après sa création alors ceci peut arriver.
- Quand est on assez vieux pour changer de génération ?
    - Généralement après la passe de scan si il survit il est promu
- On peut utiliser des algorithmes différents suivant les générations,
  - Copie compactage pour les jeunes
  - Mark and sweep pour les vieux.

Any question ?