Cours de systèmes d'exploitation
Un exemple: UNIX

Franck Cassez

CNRS/IRCCyN (UMR CNRS 6597)
BP 92101
1 rue de la Noë
44321 Nantes Cedex 3
France

Janvier 2004
Ecole Centrale Nantes

Plan

Plan

Chronologie (1960–1970)

- **1962** : Time-sharing (CTSS), implémenté à Dartmouth (MIT) succés dans la communauté scientifique

Chronologie (1960–1970)

- **1962** : Time-sharing (CTSS), implémenté à Dartmouth (MIT)
  succés dans la communauté scientifique
- **1965** : MIT + Bell-Labs + General Electric : MULTICS
  MULTiplexed Information & Computing Services
  Projet très ambitieux, nombreuses idées nouvelles, . . . ,
  pas un succés commercial

Chronologie (1960–1970)

- **1962** : Time-sharing (CTSS), implémenté à Dartmouth (MIT)
  succés dans la communauté scientifique
- **1965** : MIT + Bell-Labs + General Electric : MULTICS
  MULTiplexed Information & Computing Services
  Projet très ambitieux, nombreuses idées nouvelles, . . . ,
  pas un succés commercial
- parallèlement : développement de micro-computers,
  ex. DEC PDP-1, . . . PDP-11

Chronologie (1960–1970)

- **1962** : Time-sharing (CTSS), implémenté à Dartmouth (MIT)
  succés dans la communauté scientifique
- **1965** : MIT + Bell-Labs + General Electric : MULTICS
  MULTiplexed Information & Computing Services
  Projet très ambitieux, nombreuses idées nouvelles, . . . ,
  pas un succés commercial
- parallèlement : développement de micro-computers,
  ex. DEC PDP-1, . . . PDP-11
- Bell Labs $\downarrow$, G.E. $\rightarrow$ Honeywell $\rightarrow$ SCO

Chronologie (1960–1970)

- **1962** : Time-sharing (CTSS), implémenté à Dartmouth (MIT)
  succés dans la communauté scientifique
- **1965** : MIT + Bell-Labs + General Electric : MULTICS
  MULTiplexed Information & Computing Services
  Projet très ambitieux, nombreuses idées nouvelles, . . . ,
  pas un succés commercial
- parallèlement : développement de micro-computers,
  ex. DEC PDP-1, . . . PDP-11
- Bell Labs $\downarrow$, G.E. $\rightarrow$ Honeywell $\rightarrow$ SCO
- **1968** : Ken Thompson (Bell Labs) développe une version
  `light` de MULTICS : UNICS (UNiplexed . . .)
  maintenant : UNIX

Chronologie (1960–1970)

- **1962** : Time-sharing (CTSS), implémenté à Dartmouth (MIT)
  succés dans la communauté scientifique
- **1965** : MIT + Bell-Labs + General Electric : MULTICS
  MULTiplexed Information & Computing Services
  Projet très ambitieux, nombreuses idées nouvelles, . . . ,
  pas un succés commercial
- parallèlement : développement de micro-computers,
  ex. DEC PDP-1, . . . PDP-11
- Bell Labs $\downarrow$, G.E. $\rightarrow$ Honeywell $\rightarrow$ SCO
- **1968** : Ken Thompson (Bell Labs) développe une version
  `light` de MULTICS : UNICS (UNiplexed . . .)
  maintenant : UNIX
- **1970** : D. Ritchie (Bell Labs) + Thompson $=$ UNIX + C

Chronologie (1970–)

- **1970** : D. Ritchie (Bell Labs) + Thompson = UNIX + C

Chronologie (1970–)

- **1970** : D. Ritchie (Bell Labs) + Thompson = UNIX + C
- **1974** : publication de *The UNIX Timesharing System*, Comm. of the ACM, july.
  Ritchie & Thompson : ACM Turing Award en 1984

Chronologie (1970–)

- **1970** : D. Ritchie (Bell Labs) + Thompson = UNIX + C
- **1974** : publication de *The UNIX Timesharing System*, Comm. of the ACM, july.
  Ritchie & Thompson : ACM Turing Award en 1984
- Code source disponible pour les universités
  Bell Labs = System III, V vs. Berkeley = 4.4BSD
  BSD : mémoire virtuelle, pagination, ...

Chronologie (1970–)

- **1970** : D. Ritchie (Bell Labs) + Thompson = UNIX + C
- **1974** : publication de *The UNIX Timesharing System*, Comm. of the ACM, july.
  Ritchie & Thompson : ACM Turing Award en 1984
- Code source disponible pour les universités
  Bell Labs = System III, V vs. Berkeley = 4.4BSD
  BSD : mémoire virtuelle, pagination, ...
- **fin des années 80** : System V R3 et 4.3BSD

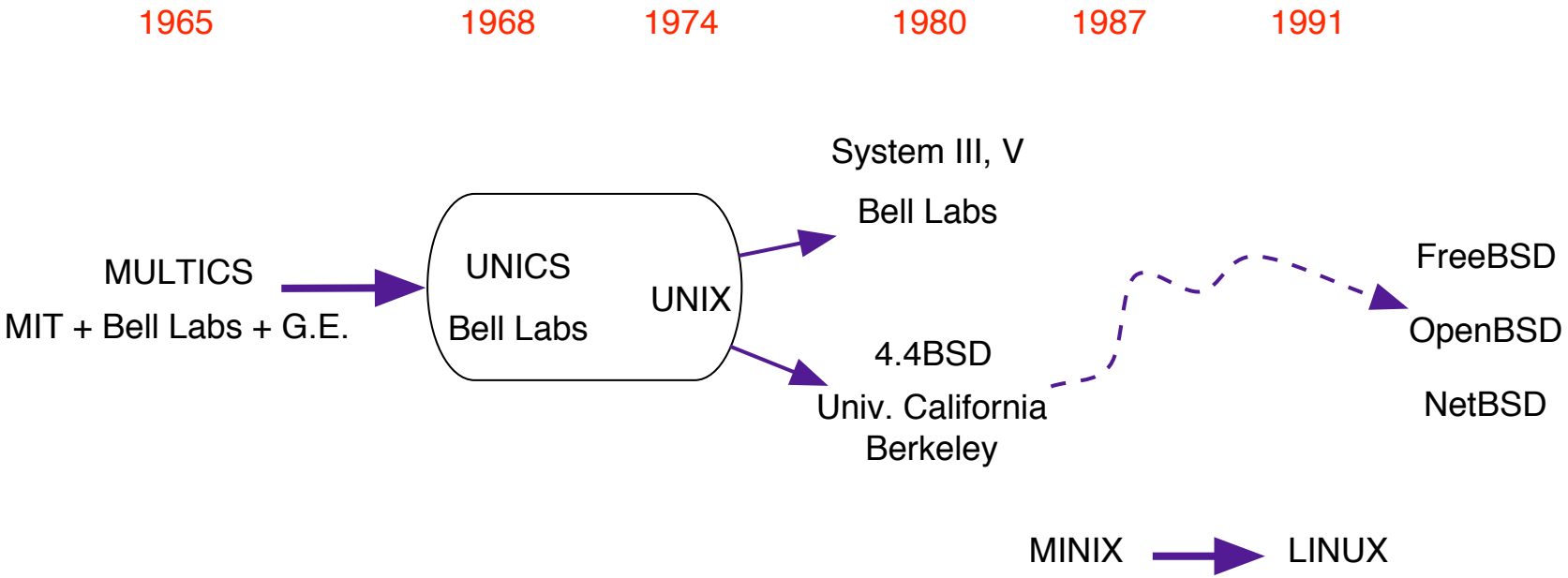
Chronologie (1970–)

- **1970** : D. Ritchie (Bell Labs) + Thompson = UNIX + C
- **1974** : publication de *The UNIX Timesharing System*, Comm. of the ACM, july.
  Ritchie & Thompson : ACM Turing Award en 1984
- Code source disponible pour les universités
  Bell Labs = System III, V vs. Berkeley = 4.4BSD
  BSD : mémoire virtuelle, pagination, ...
- **fin des années 80** : System V R3 et 4.3BSD
- **1987** : MINIX : UNIX pour l'enseignement

Chronologie (1970–)

- **1970** : D. Ritchie (Bell Labs) + Thompson = UNIX + C
- **1974** : publication de *The UNIX Timesharing System*, Comm. of the ACM, july.
  Ritchie & Thompson : ACM Turing Award en 1984
- Code source disponible pour les universités
  Bell Labs = System III, V vs. Berkeley = 4.4BSD
  BSD : mémoire virtuelle, pagination, ...
- **fin des années 80** : System V R3 et 4.3BSD
- **1987** : MINIX : UNIX pour l'enseignement
- **1991** : LINUX

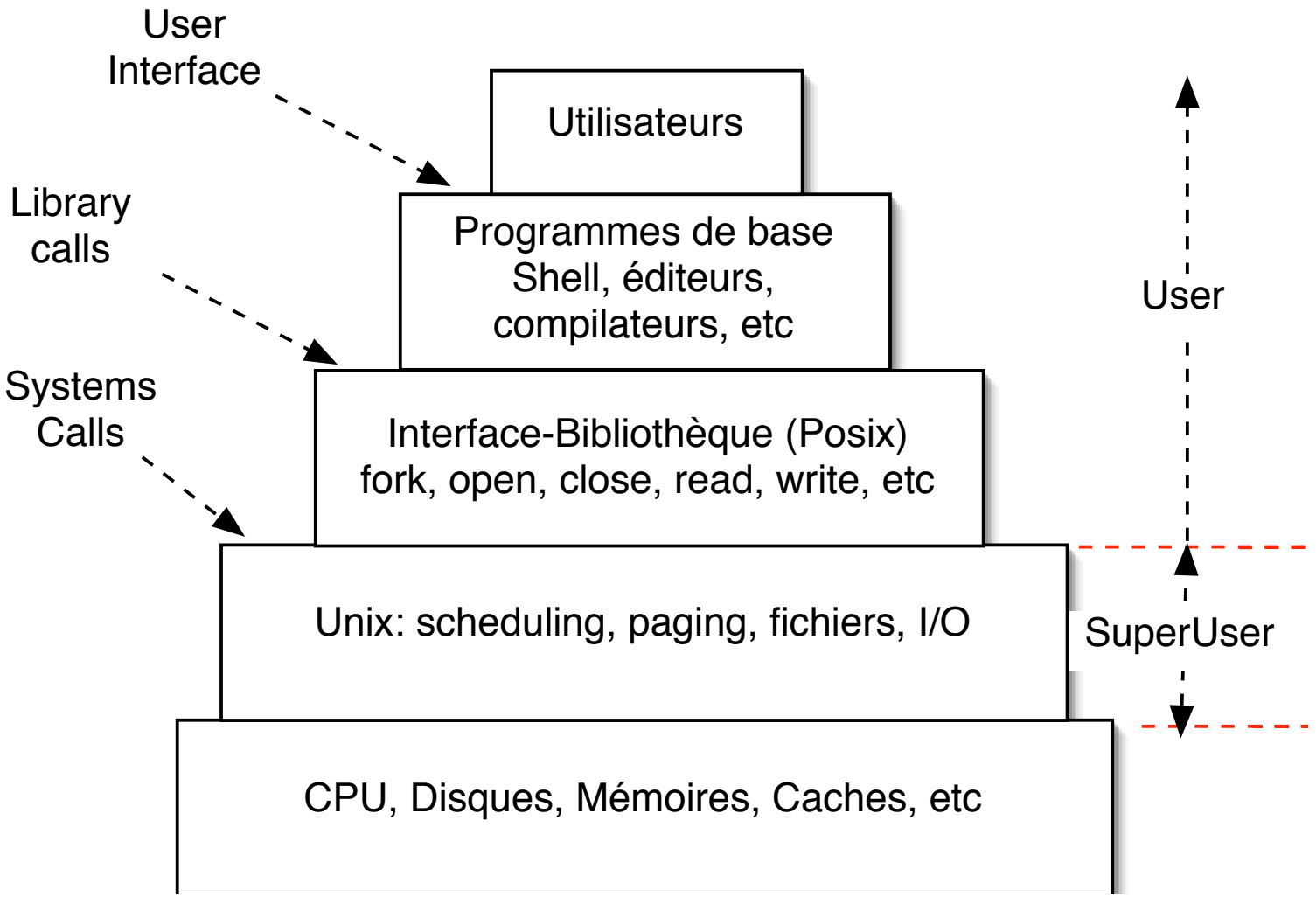
De MULTICS à Linux



Noyau UNIX standard

- Fait : il existe différentes versions d'UNIX
  System V, 4.4BSD, , FreeBSD, Solaris, LINUX, OpenBSD, NetBSD, ...
- POSIX = Portable Operating Systems (IEEE)
  Définit les System Calls que doit fournir une implémentation d'UNIX
  Dans ce cas : POSIX compliant
- but : interopérabilité
  P = programme écrit en utilisant les System Calls POSIX
  alors P est exécutable sur tout UNIX qui est POSIX compliant

Vue d'ensemble d'UNIX



Plan

Le fork pour créer des processus

Daemon : processus crées au démarrage du système
exemples : cron, sendmail, paging etc

Processus : création par System Call = fork

```
1  pid=fork();
2  if (pid<0) {
3     error_treatment(); /* si pb: pas assez place ... */
4  }
5  else if (pid>0) {
6           prog_for_parent(); /* code pour le parent */
7        }
8        else {
9           prog_for_child(); /* code pour le fils */
10       }
```

Réalisation d'un fork

Processus père

```
pid=fork();
if (pid<0) {
   error_treatment();
}
else if (pid>0) {
        prog_for_parent();
     }
     else {
        prog_for_child();
     }
```

Réalisation d'un fork

Processus père

pid=478

```
pid=fork();
if (pid<0) {
   error_treatment();
}
else if (pid>0) {
        prog_for_parent();
     }
     else {
        prog_for_child();
     }
```

Process fils: 478

pid=0

```
pid=fork();
if (pid<0) {
   error_treatment();
}
else if (pid>0) {
        prog_for_parent();
     }
     else {
        prog_for_child();
     }
```

Informations liées aux processus

Pour chaque processus, informations dans 2 tables :

Table des Processus

Info. d'ordonnancement: priorité, CPU usage, etc
Info. mémoire: pointeur vers table des pages ou segments
Signaux attendus
Etat du processus: PID, PID du père, etc

- Zone `proc`

Informations liées aux processus

Pour chaque processus, informations dans 2 tables :

Table des Processus

Info. d'ordonnancement: priorité, CPU usage, etc
Info. mémoire: pointeur vers table des pages ou segments
Signaux attendus
Etat du processus: PID, PID du père, etc

- Zone `proc`
- toujours en mémoire

Informations liées aux processus

Pour chaque processus, informations dans 2 tables :

Table des Processus

Info. d'ordonnancement: priorité, CPU usage, etc
Info. mémoire: pointeur vers table des pages ou segments
Signaux attendus
Etat du processus: PID, PID du père, etc

- Zone `proc`
- toujours en mémoire

Infos Utilisateurs

Registres CPU: reg. adresse, etc
Info sur System Call: paramètres, résultats etc
Fichiers ouverts
Infos diverses: limitations pour le processus, max. pages, etc

- Zone `u`

Informations liées aux processus

Pour chaque processus, informations dans 2 tables :

Table des Processus

Info. d'ordonnancement: priorité, CPU usage, etc
Info. mémoire: pointeur vers table des pages ou segments
Signaux attendus
Etat du processus: PID, PID du père, etc

- Zone `proc`
- toujours en mémoire

Infos Utilisateurs

Registres CPU: reg. adresse, etc
Info sur System Call: paramètres, résultats etc
Fichiers ouverts
Infos diverses: limitations pour le processus, max. pages, etc

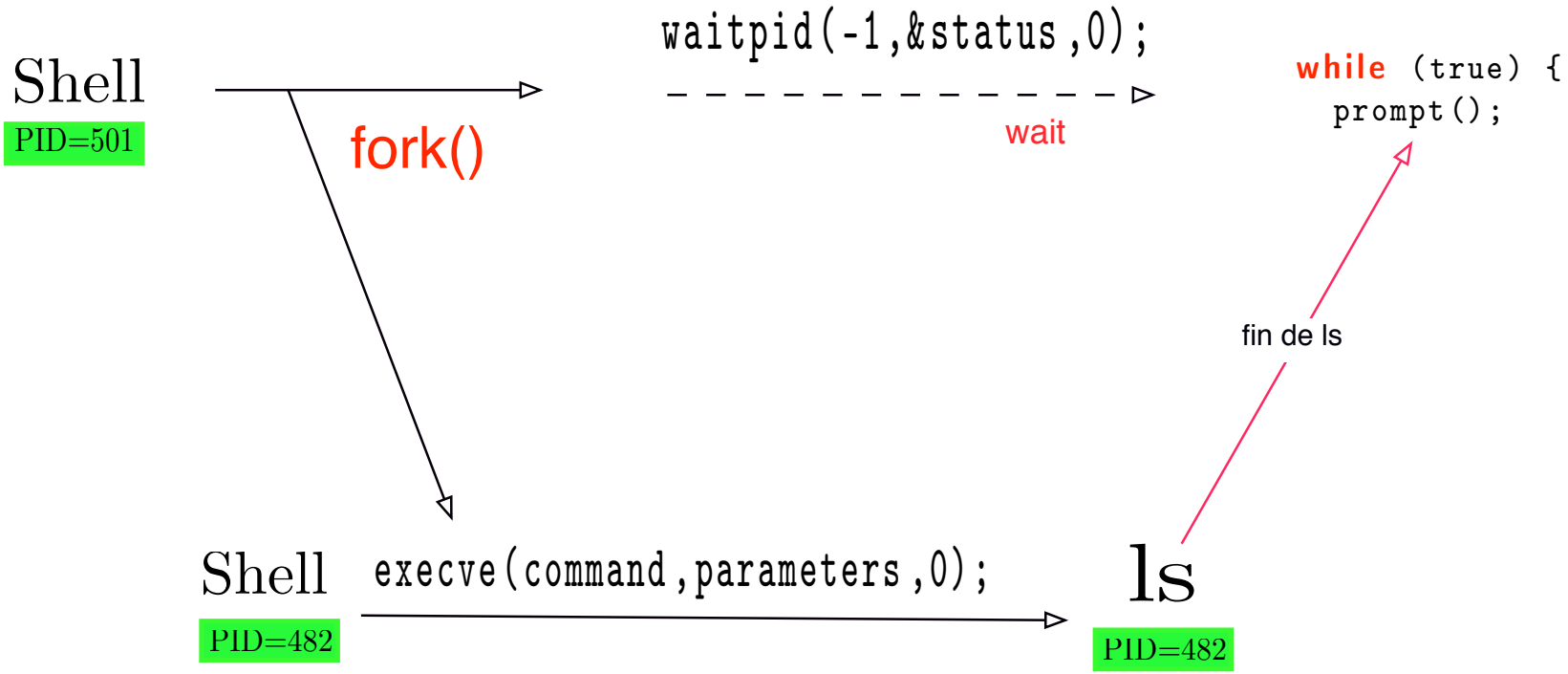
- Zone `u`
- peut être swappée

Programme pour un Shell simplifié

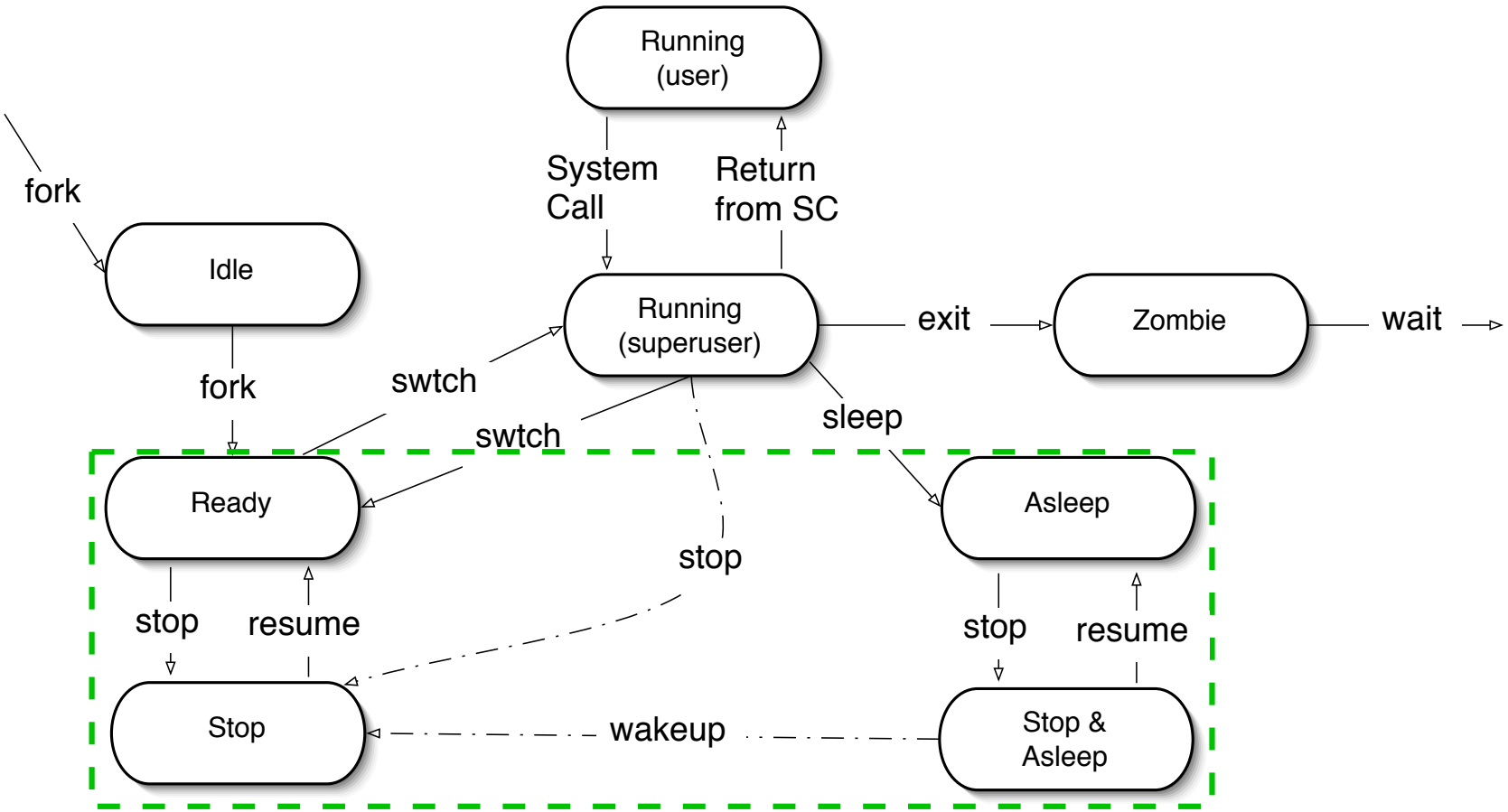
```
1  while (true) {
2    prompt();
3    lire_commande(command,parameters);
4
5    pid=fork(); /* fork pour faire la commande */
6    if (pid<0) {
7      printf(``impossible de faire la commande'');
8      break;
9    }
10
11   if (pid!=0) {
12     waitpid(-1,&status,0); /* le parent attend */
13   }
14   else { /* le fils fait la commande */
15     execve(command,parameters,0);
16   }
17 }
```

Exécution d'une commande dans le Shell

commande `ls`



Changements d'états d'un processus



Plan

Types de processus supportés

Système typique $\Longrightarrow$ différents types de processus :

Types de processus supportés

Système typique $\Longrightarrow$ différents types de processus :

- **Interactifs :** éditeurs, Shell, interface graphique ; Δ-CPU court, réponse rapide

Types de processus supportés

Système typique $\Longrightarrow$ différents types de processus :

- **Interactifs :** éditeurs, Shell, interface graphique ;
  Δ-CPU court, réponse rapide
- **Batch :** compilateurs, programmes de calcul ;
  Δ-CPU long, minimiser $\frac{\textit{temps transit}}{\textit{temps exécution}}$

Types de processus supportés

Système typique $\Longrightarrow$ différents types de processus :

- **Interactifs :** éditeurs, Shell, interface graphique ; Δ-CPU court, réponse rapide
- **Batch :** compilateurs, programmes de calcul ; Δ-CPU long, minimiser $\frac{\textit{temps transit}}{\textit{temps exécution}}$
- **Real-time :** . . . tous les autres ; video, etc ; deadlines ou besoins CPU réguliers

Types de processus supportés

Système typique $\Longrightarrow$ différents types de processus :

- **Interactifs :** éditeurs, Shell, interface graphique ; Δ-CPU court, réponse rapide
- **Batch :** compilateurs, programmes de calcul ; Δ-CPU long, minimiser $\frac{\textit{temps transit}}{\textit{temps exécution}}$
- **Real-time :** . . . tous les autres ; video, etc ; deadlines ou besoins CPU réguliers

---

Ordonnanceur UNIX :

- processus interactifs et batch
- principes : favoriser les processus interactifs + équilibrer le temps CPU donnés aux processus batchs

Algorithme à 2 niveaux

- Scheduler : ordonnance les processus qui sont en mémoire
  un processus bloqué (attente d'une E/S) n'est pas en mémoire (swappé)
- Swapper : contrôle le passage mémoire $\longleftrightarrow$ disque
  assure que tous les processus seront un jour en mémoire
  $\Longrightarrow$ éviter la famine

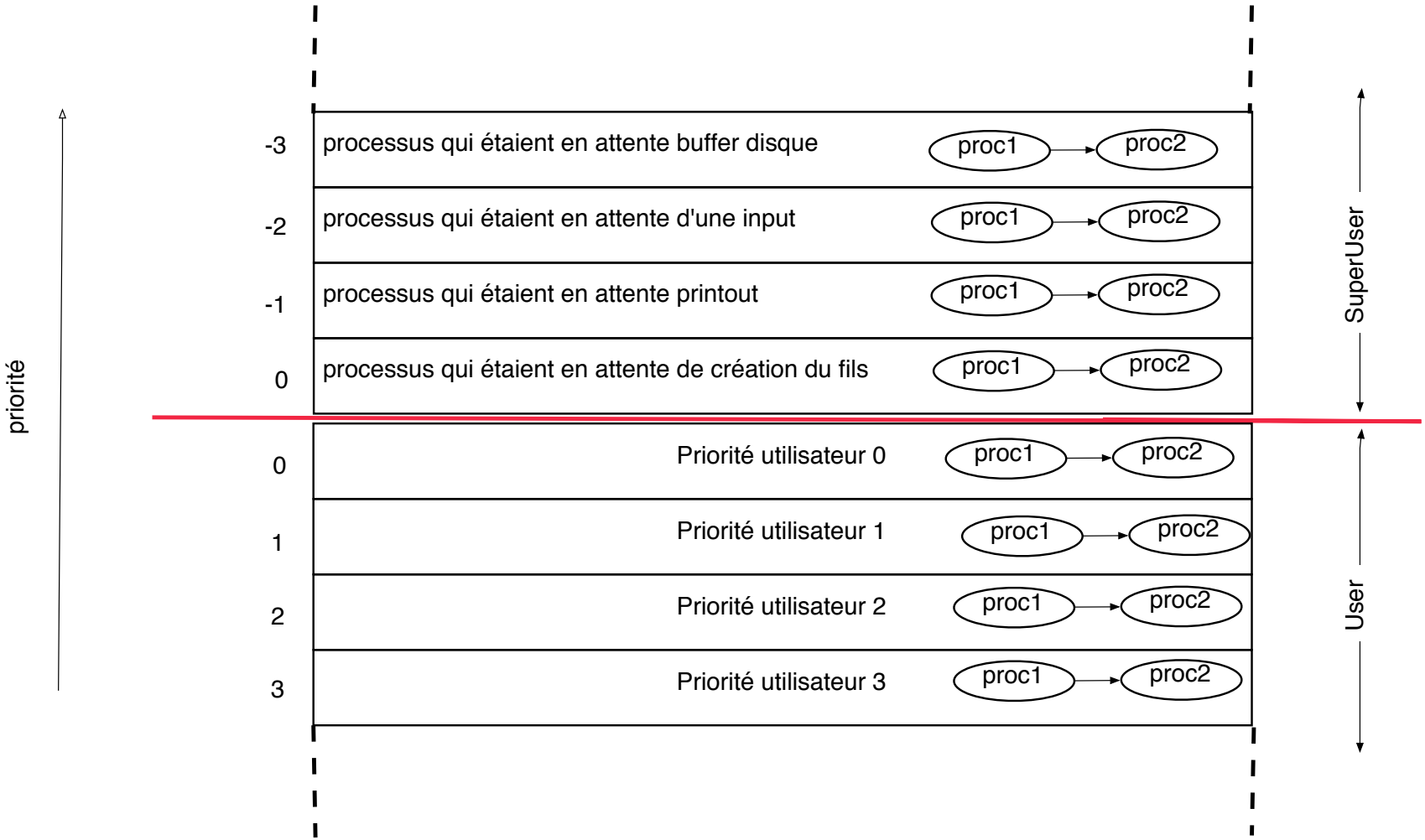
Algorithme à 2 niveaux

- Scheduler : ordonnance les processus qui sont en mémoire
  un processus bloqué (attente d'une E/S) n'est pas en mémoire (swappé)
- Swapper : contrôle le passage mémoire $\longleftrightarrow$ disque
  assure que tous les processus seront un jour en mémoire
  $\Longrightarrow$ éviter la famine

Scheduler :

- round-robin à étages ; Quantum $\equiv$ 100ms
- chaque étage $\equiv$ niveaux de priorité (disjoints)
- Priorité :
  - < 0 pour le mode SuperUser
  - ≥ 0 pour le mode User

Attribution des priorités



Calcul des priorités des processus

- toutes les secondes $\Longrightarrow$ recalcul de la priorité

Calcul des priorités des processus

- priorité calculée par :

$$prio = CPU\text{-}Usage + nice + base \tag{1}$$

Calcul des priorités des processus

- priorité calculée par :

$$prio = CPU\text{-}Usage + nice + base \quad (1)$$

  - CPU-Usage : usage CPU moyen récent (nombre de ticks) dans quelques secondes précédentes
  - nice : $\in [-20, 20]$, fixé par le processus
    `nice` = System Call
  - base : si une E/S (etc) attendue par un processus P se termine, P est débloqué et *base* = valeur associée à la cause de l'attente

Calcul des priorités des processus

- priorité calculée par :

$$prio = CPU\text{-}Usage + nice + base \quad (1)$$

- *CPU-Usage* $\Longrightarrow$ si un processus utilise beaucoup de CPU il est pénalisé

Calcul des priorités des processus

- priorité calculée par :

$$prio = CPU\text{-}Usage + nice + base \quad (1)$$

- *CPU-Usage* $\Longrightarrow$ si un processus utilise beaucoup de CPU il est pénalisé
- Facteur de dégradation dans le *CPU-Usage* :
  - soit C le dernier *CPU-Usage*
  - on définit : $k_{i+1} = \frac{k_i + C}{2}$, $k_0 =$ premier *CPU-Usage*
  - calcul itératif : $k := \frac{k+C}{2}$
  - remplacement de *CPU-Usage* par k dans equation (1)

Propriétés de l'ordonnanceur UNIX

- préemptif en round-robin multi-niveaux
  (noyau ne peut être préempté)

Propriétés de l'ordonnanceur UNIX

- préemptif en round-robin multi-niveaux (noyau ne peut être préempté)
- simple et efficace

Propriétés de l'ordonnanceur UNIX

- préemptif en round-robin multi-niveaux
  (noyau ne peut être préempté)
- simple et efficace
- favorise les processus "I/O bound" :
  - processus attendant une I/O est bloqué en mode SuperUser
  - vise à faire quitter le mode SuperUser rapidement

Propriétés de l'ordonnanceur UNIX

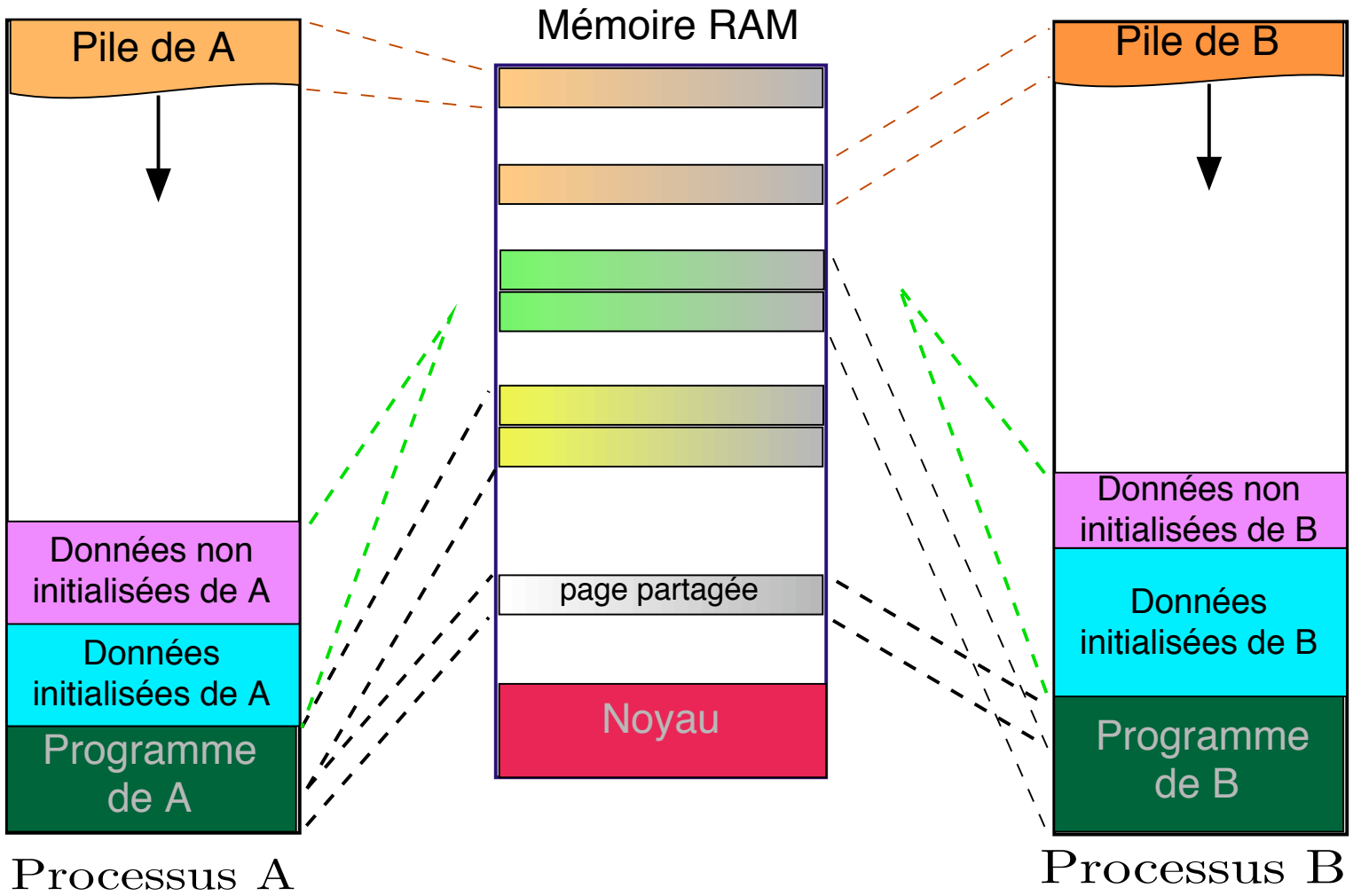
- préemptif en round-robin multi-niveaux
  (noyau ne peut être préempté)
- simple et efficace
- favorise les processus "I/O bound" :
  - processus attendant une I/O est bloqué en mode SuperUser
  - vise à faire quitter le mode SuperUser rapidement
- équilibre CPU pour les processus "CPU bound" :
  Facteur de dégradation ($\Longrightarrow$ pas de famine)

Propriétés de l'ordonnanceur UNIX

- préemptif en round-robin multi-niveaux
  (noyau ne peut être préempté)
- simple et efficace
- favorise les processus "I/O bound" :
  - processus attendant une I/O est bloqué en mode SuperUser
  - vise à faire quitter le mode SuperUser rapidement
- équilibre CPU pour les processus "CPU bound" :
  Facteur de dégradation ($\Longrightarrow$ pas de famine)
- pas de garantie de temps de réponse
  $\Longrightarrow$ pas un OS temps-réel

Plan

Partage de la mémoire par les processus



Propriétés du modèle mémoire

- possibilité de partager des segments

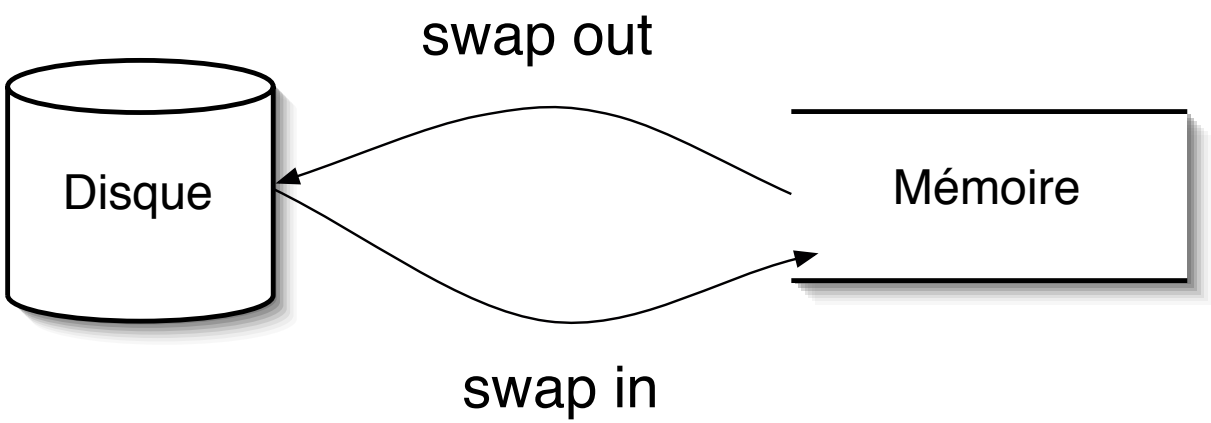
Propriétés du modèle mémoire

- possibilité de partager des segments
- `memory-mapped file` : fichier = portion de le l'espace virtuel d'un processus
  `read/write` + rapide, partage (ex : librairies partagées)
  Systems Calls : `mmap` et `munmap`

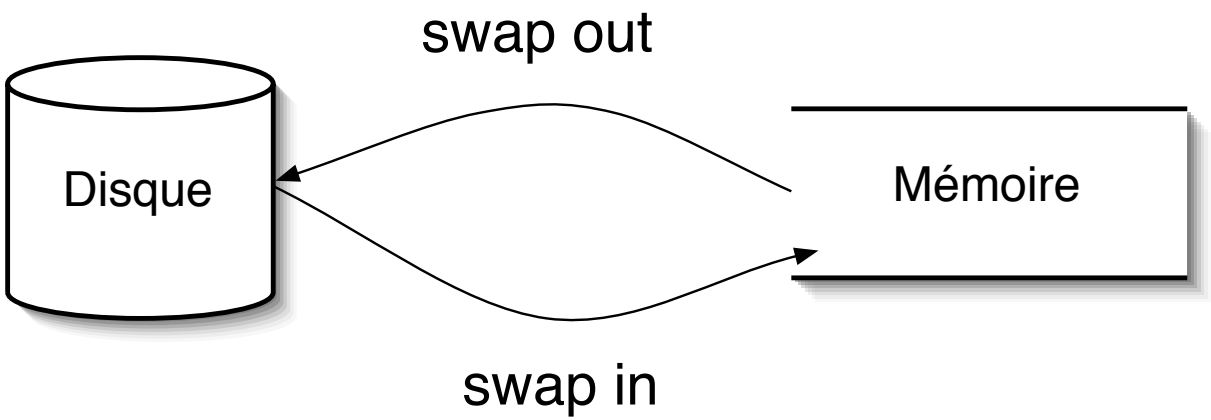
Propriétés du modèle mémoire

- possibilité de partager des segments
- `memory-mapped file` : fichier = portion de le l'espace virtuel d'un processus
  `read/write` + rapide, partage (ex : librairies partagées)
  Systems Calls : `mmap` et `munmap`
- demande mémoire par un processus :
  System Call : `brk`
  `malloc` en C utilise `brk`

Fonctionnement du Swapper



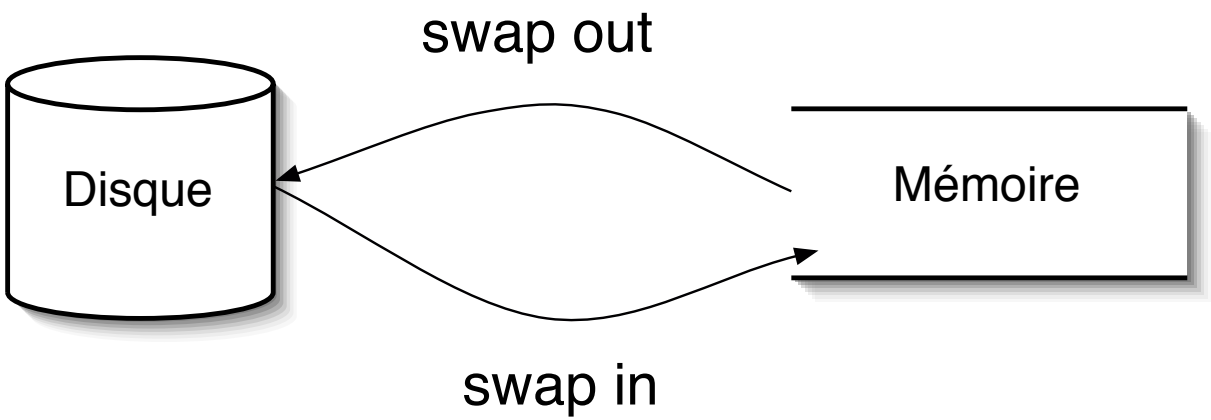
Fonctionnement du Swapper



Swap out quand manque de mémoire sur

- `fork`, `brk`, ou débordement de pile

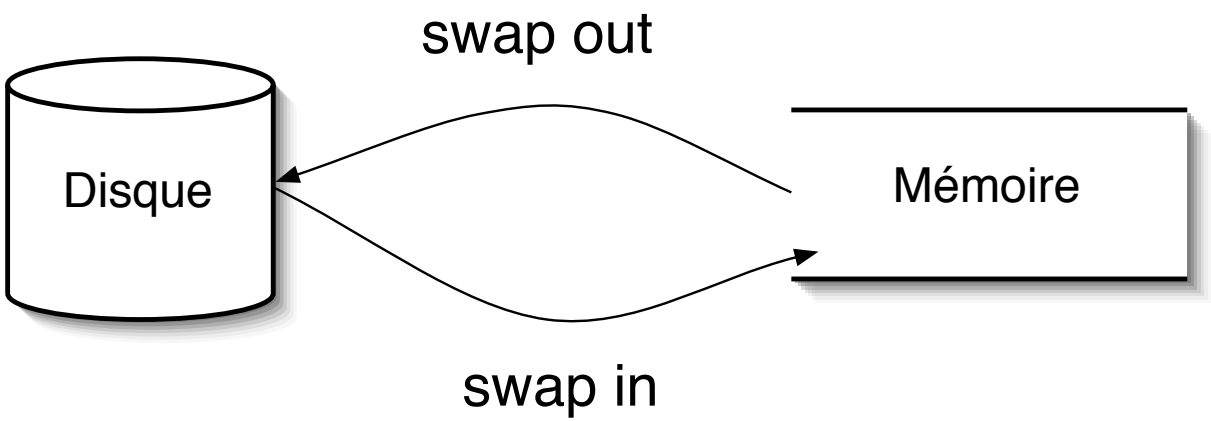
Fonctionnement du Swapper



Swap out quand manque de mémoire sur

- `fork`, `brk`, ou débordement de pile
- choix d'une victime :
  1. $\exists$ processus bloqués : critère C = *prio* + temps résidence
     victime = le plus grand C
  2. sinon : parmi les non bloqués, victime = plus grand C

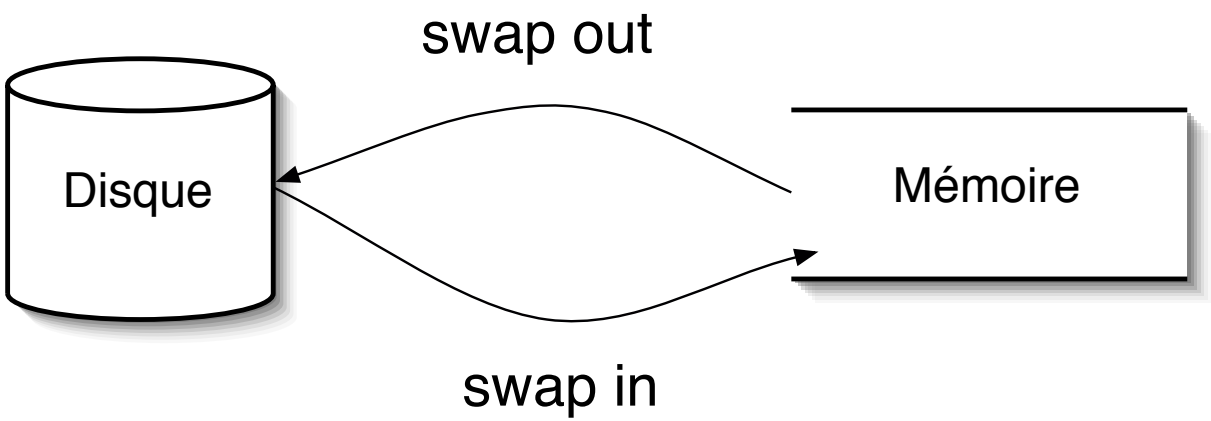
Fonctionnement du Swapper



Swap in : toutes les 4-5 secondes

- Swapper cherche un processus prêt sur le disque

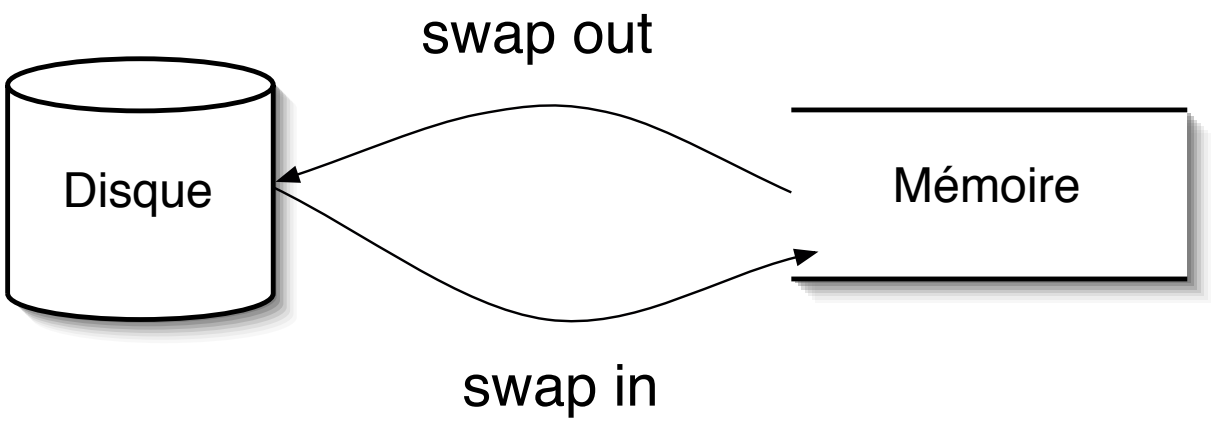
Fonctionnement du Swapper



Swap in : toutes les 4-5 secondes

- Swapper cherche un processus prêt sur le disque
- critère de choix : temps le plus long sur le disque

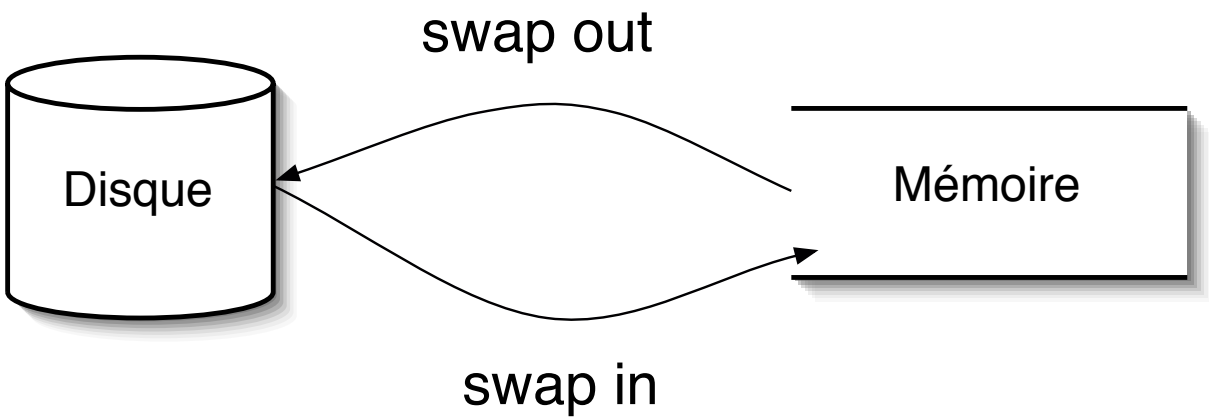
Fonctionnement du Swapper



Swap in : toutes les 4-5 secondes

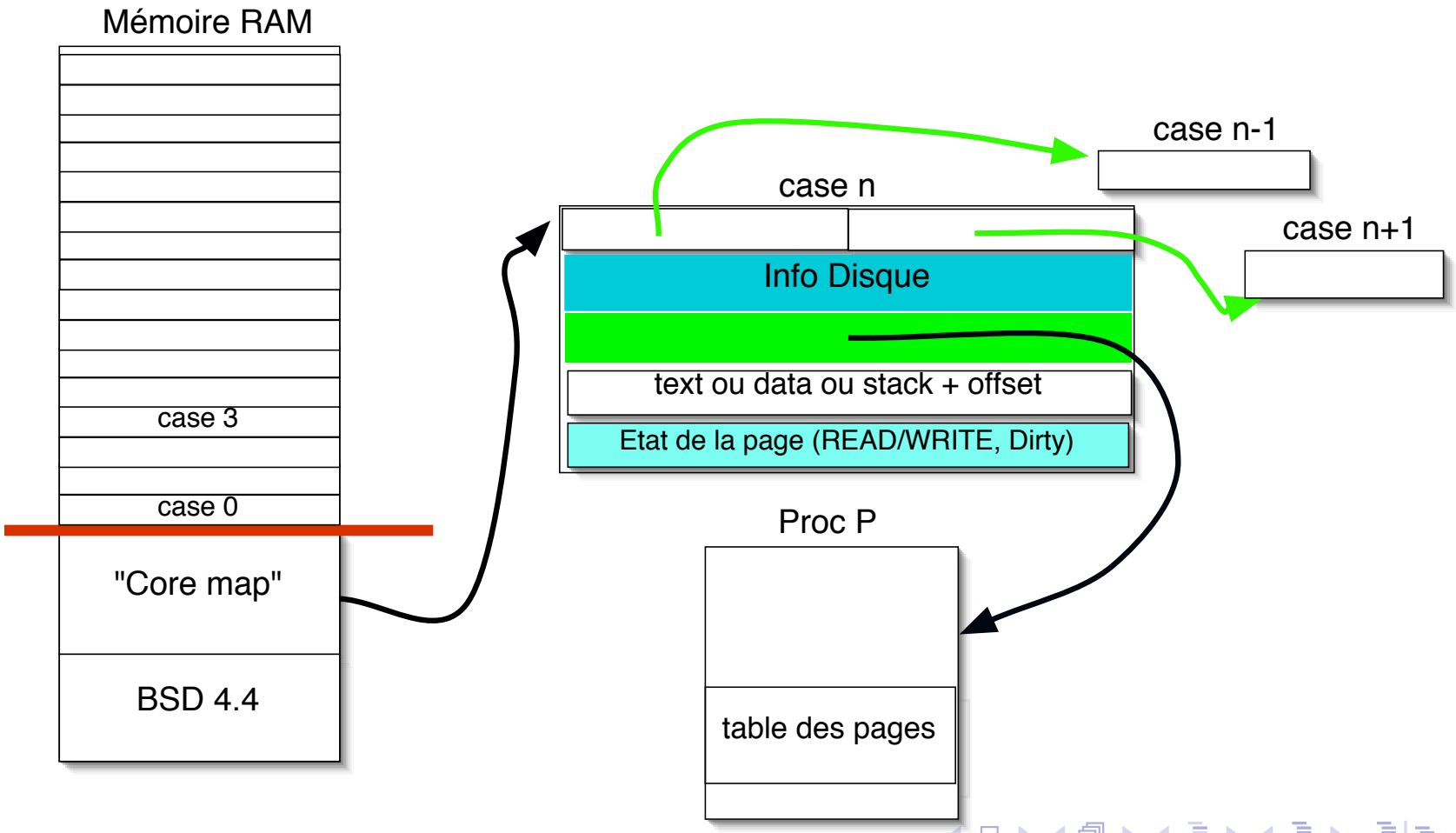
- Swapper cherche un processus prêt sur le disque
- critère de choix : temps le plus long sur le disque
- éventuellement *swap out* d'un autre processus

Fonctionnement du Swapper



- Swapper : *swap in* (toutes les 4-5 secs) + *swap out* (à la demande)
- Changement de comportement quand :
  - plus de processus swappés
  - trop de processus en mémoire (*Thrashing*)
    min. de 2 sec. en mémoire avant de swapper un processus

Fonction de pagination, informations sur les cases



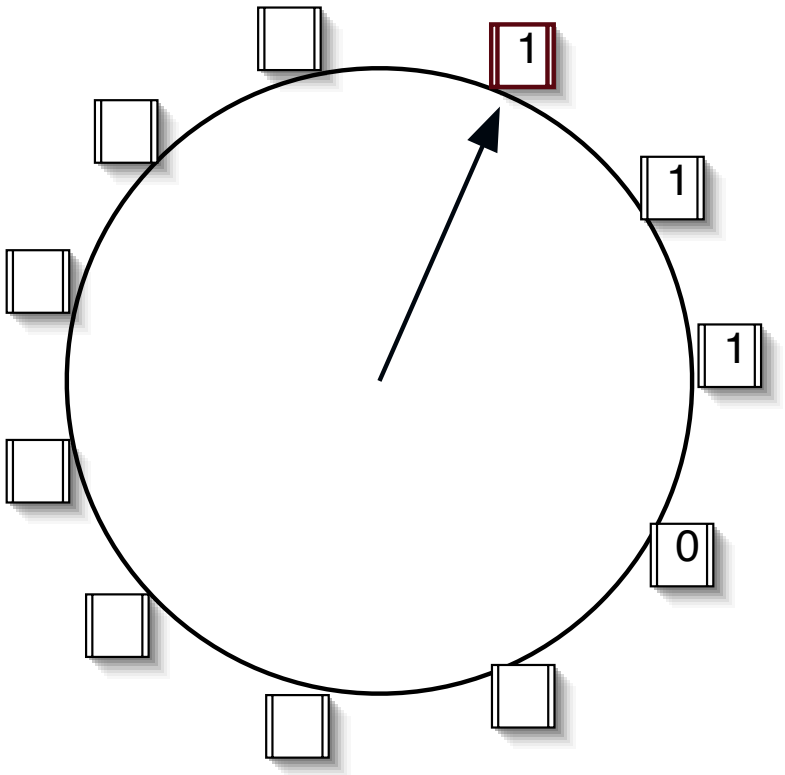
Le pagedaemon

- toutes les 250 msec. le `pagedaemon` est activé
- Algorithme :
  - nombre de cases libres $\geq$ `lotsfree` ?
  - si oui : `sleep`
  - sinon : transférer des pages sur le disque
    but : avoir des pages libres constamment

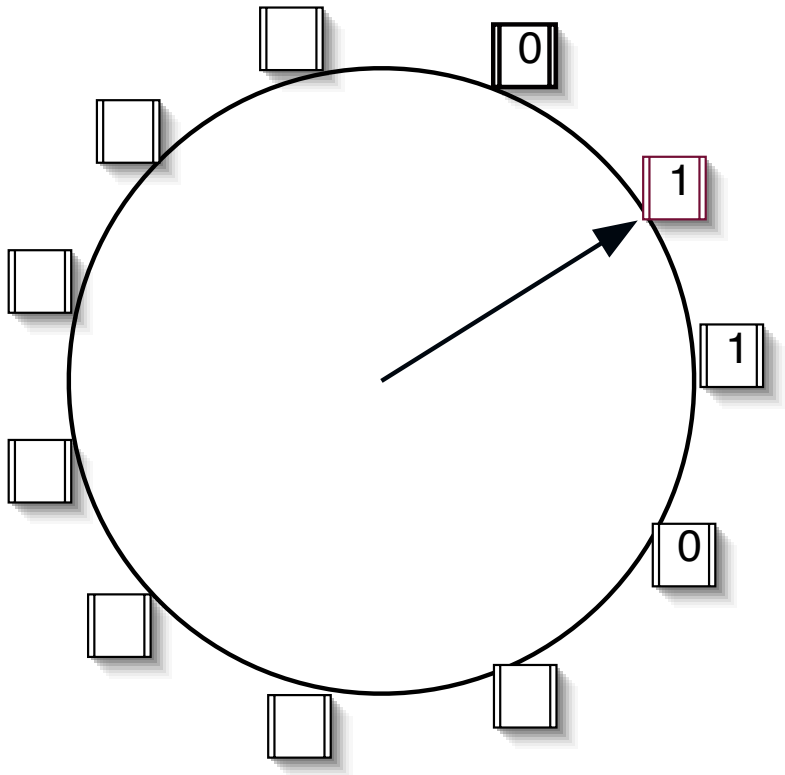
Le pagedaemon

- toutes les 250 msec. le `pagedaemon` est activé
- Algorithme :
  - nombre de cases libres $\geq$ `lotsfree` ?
  - si oui : `sleep`
  - sinon : transférer des pages sur le disque
    but : avoir des pages libres constamment
- Algorithme de traitement de défaut basé :
  sur "seconde chance" (ou "Clock Algorithm")

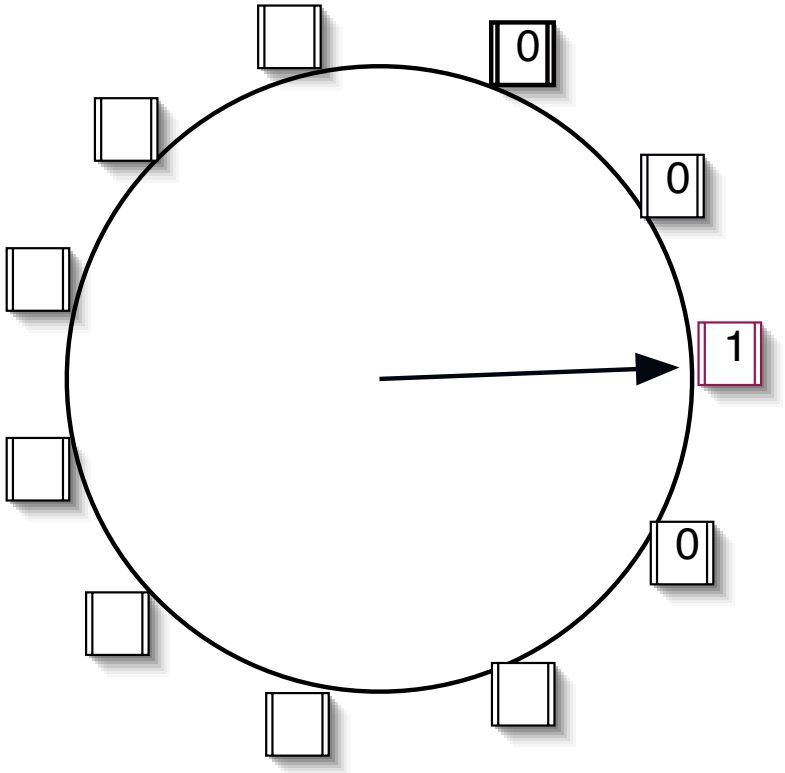
Clock Algorithm



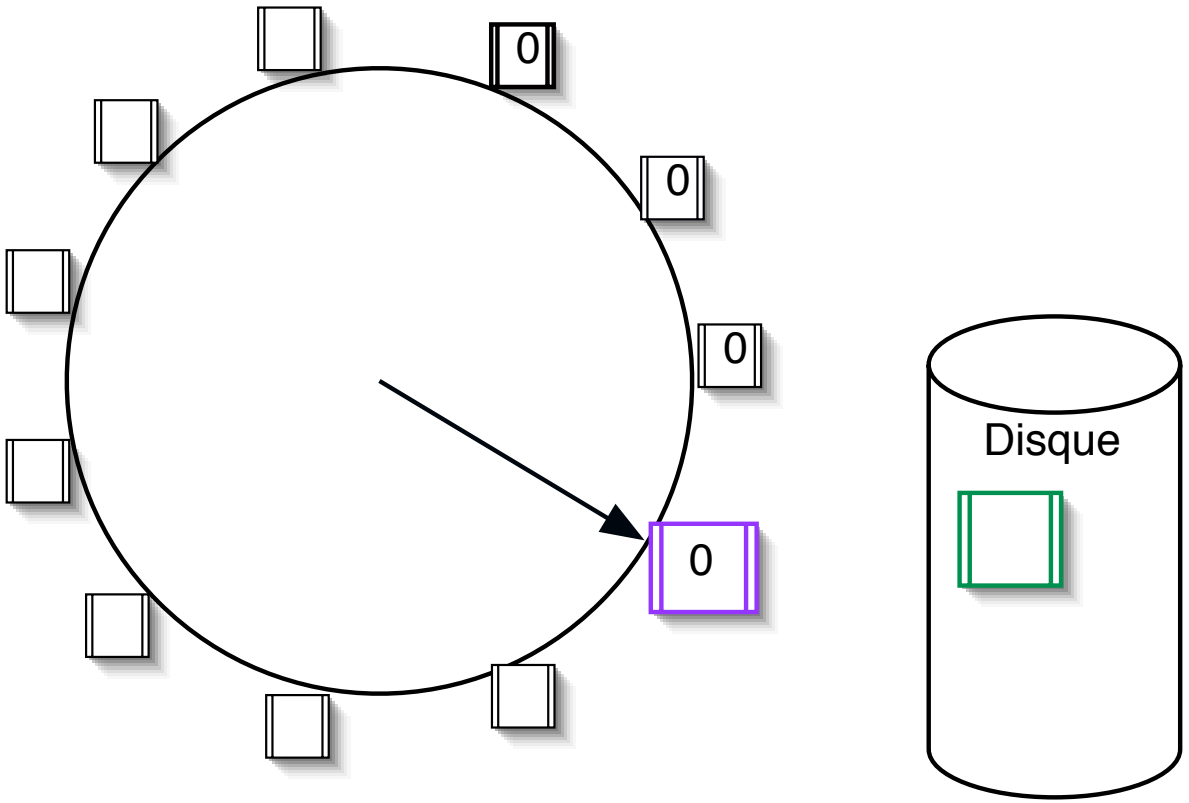
Clock Algorithm



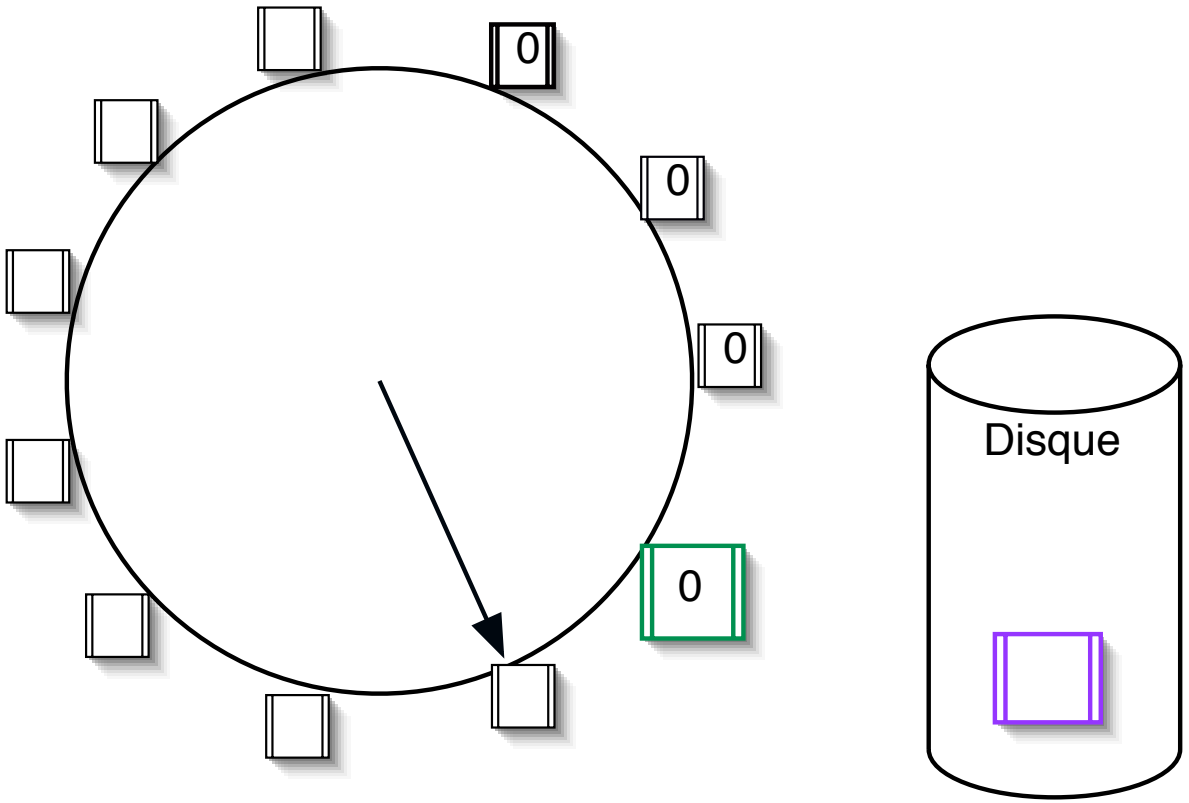
Clock Algorithm



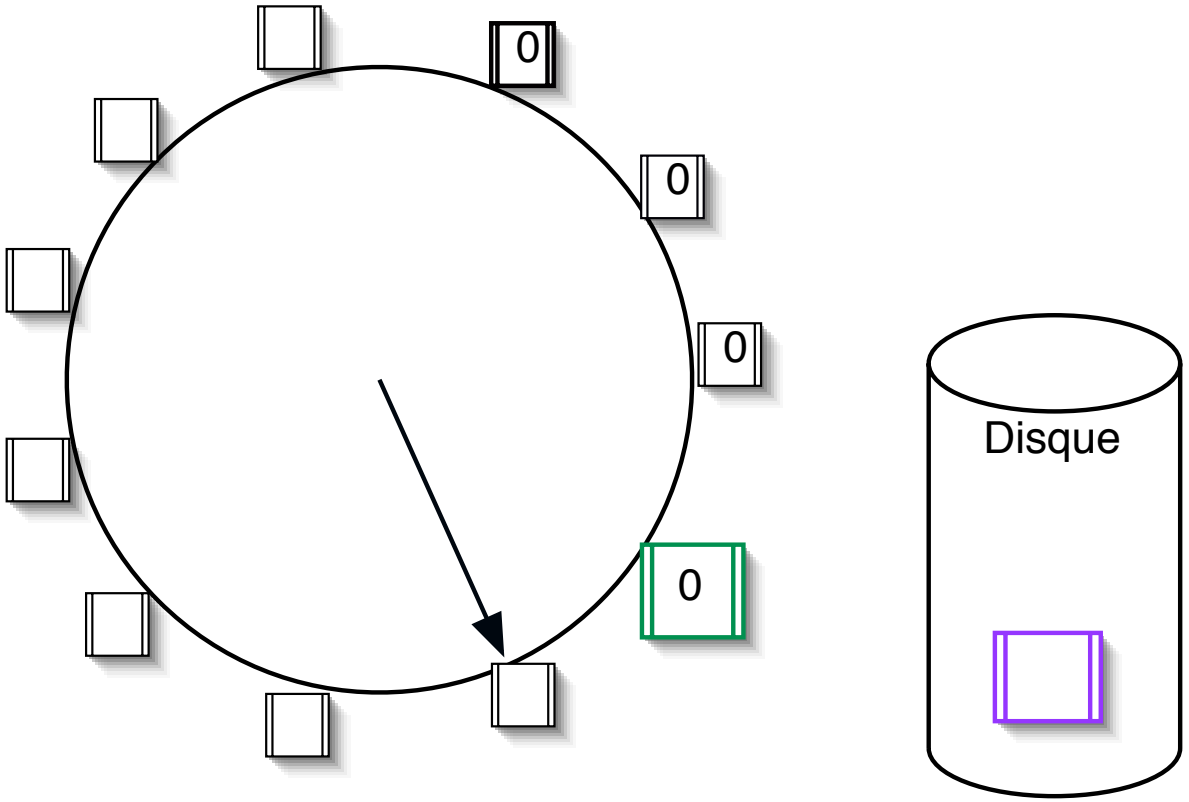
Clock Algorithm



Clock Algorithm

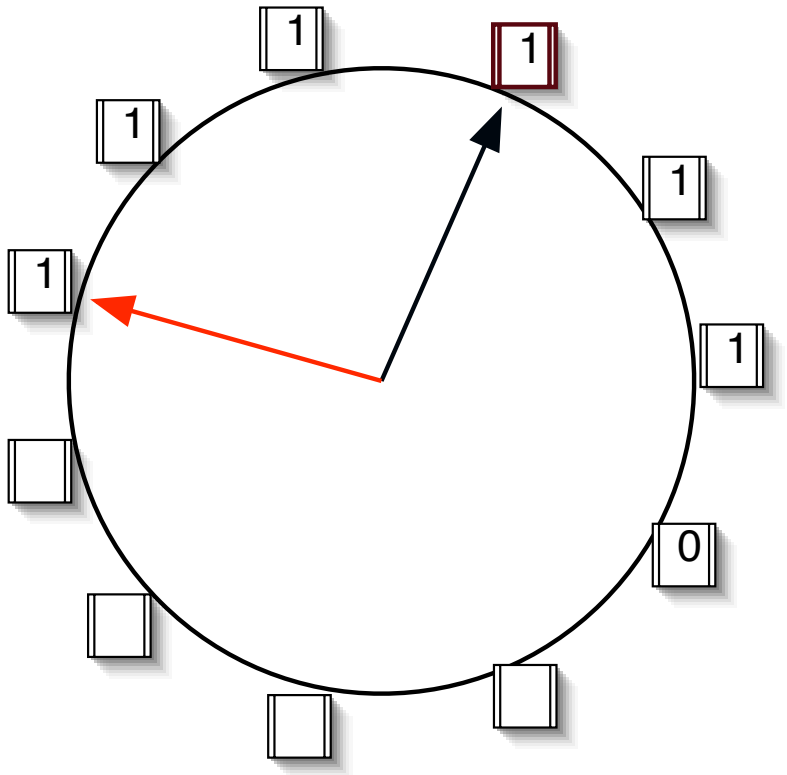


Clock Algorithm

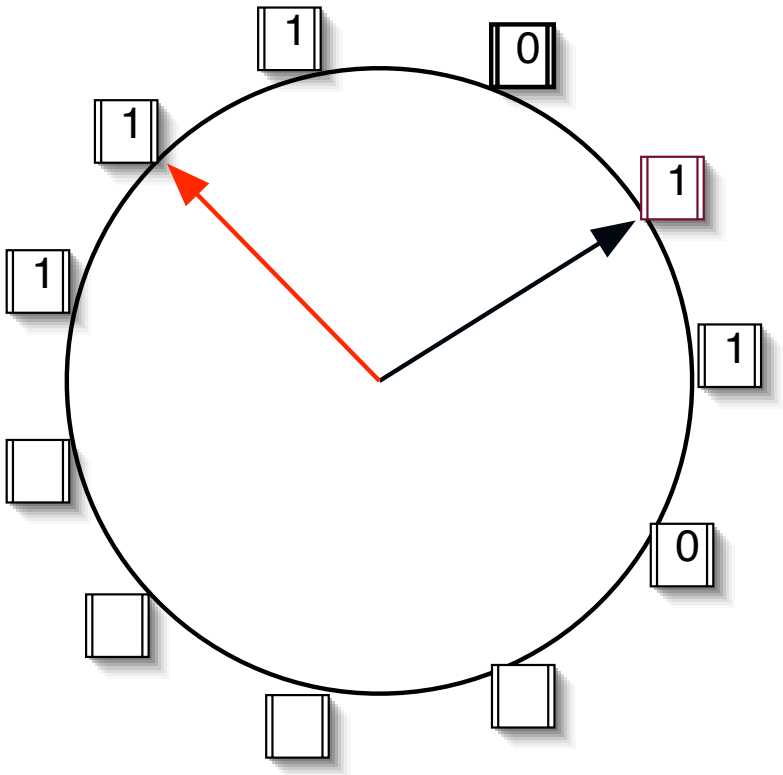


Si nombre de pages très grand ? ? ? ?

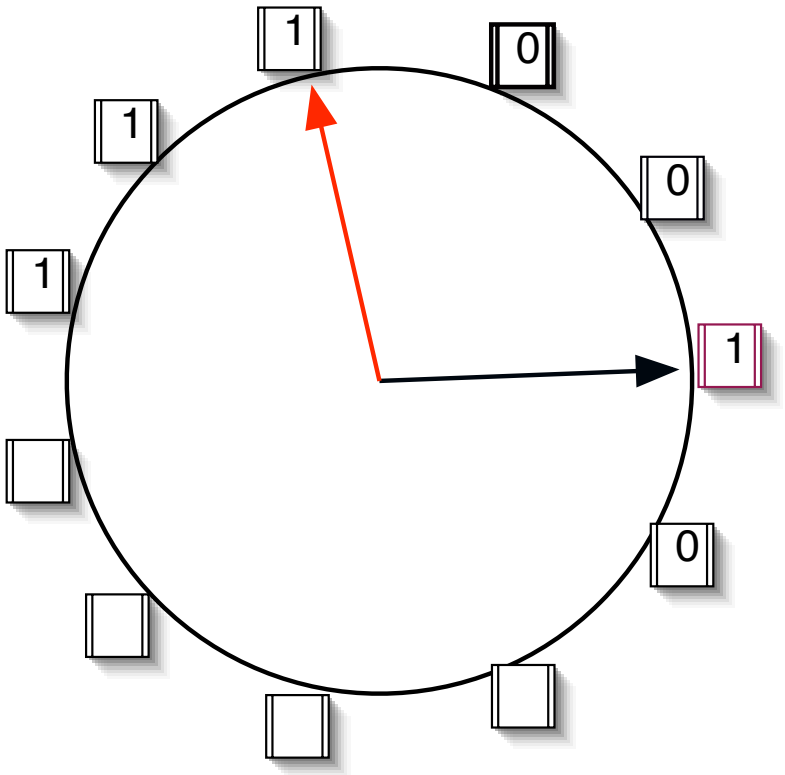
Two-Handed Clock Algorithm



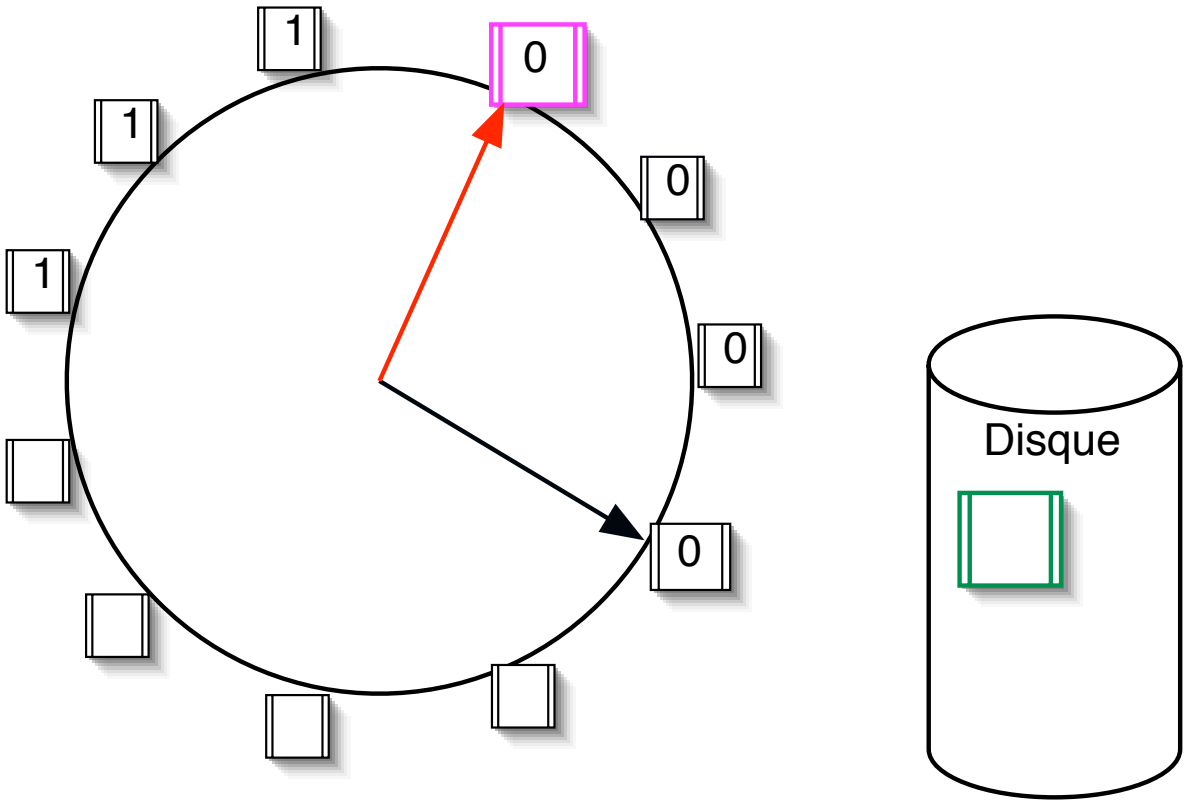
Two-Handed Clock Algorithm



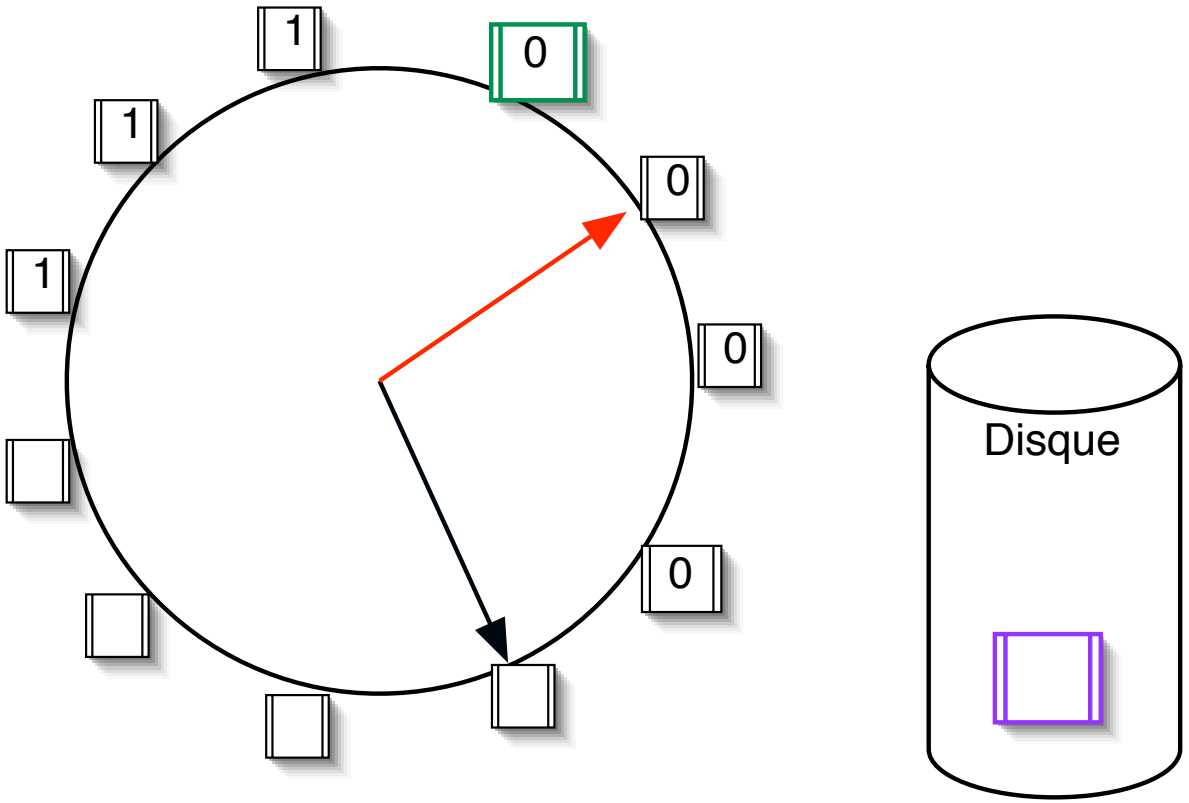
Two-Handed Clock Algorithm



Two-Handed Clock Algorithm



Two-Handed Clock Algorithm



Thrashing

Si le taux de défaut de pages est très grand : appel au swapper

- si $\exists$ des processus `idle` depuis $\Delta \geq 20sec$.
  swap out le moins actif récemment (max. `idle` time)
- sinon parmi les 4 plus gros processus (taille mémoire)
  swap out le plus vieux (en mémoire)
- si nécessaire plusieurs processus peuvent être swappés

Thrashing

Si le taux de défaut de pages est très grand : appel au swapper

- si $\exists$ des processus `idle` depuis $\Delta \geq 20sec.$
  swap out le moins actif récemment (max. `idle` time)
- sinon parmi les 4 plus gros processus (taille mémoire)
  swap out le plus vieux (en mémoire)
- si nécessaire plusieurs processus peuvent être swappés

Variantes de l'algorihtme (System V) :

- Clock Algorithm avec : *n* passages avant de libérer une page
  libère moins vite mais plus proche du Working Set
- Au lieu de `lotsfree` : deux valeurs `min` et `max`
  si # pages libres $\leq$ *min* : libère des pages jusqu'à `max` atteint
  Evite instabililté du précédent

Fonction de pagination LINUX

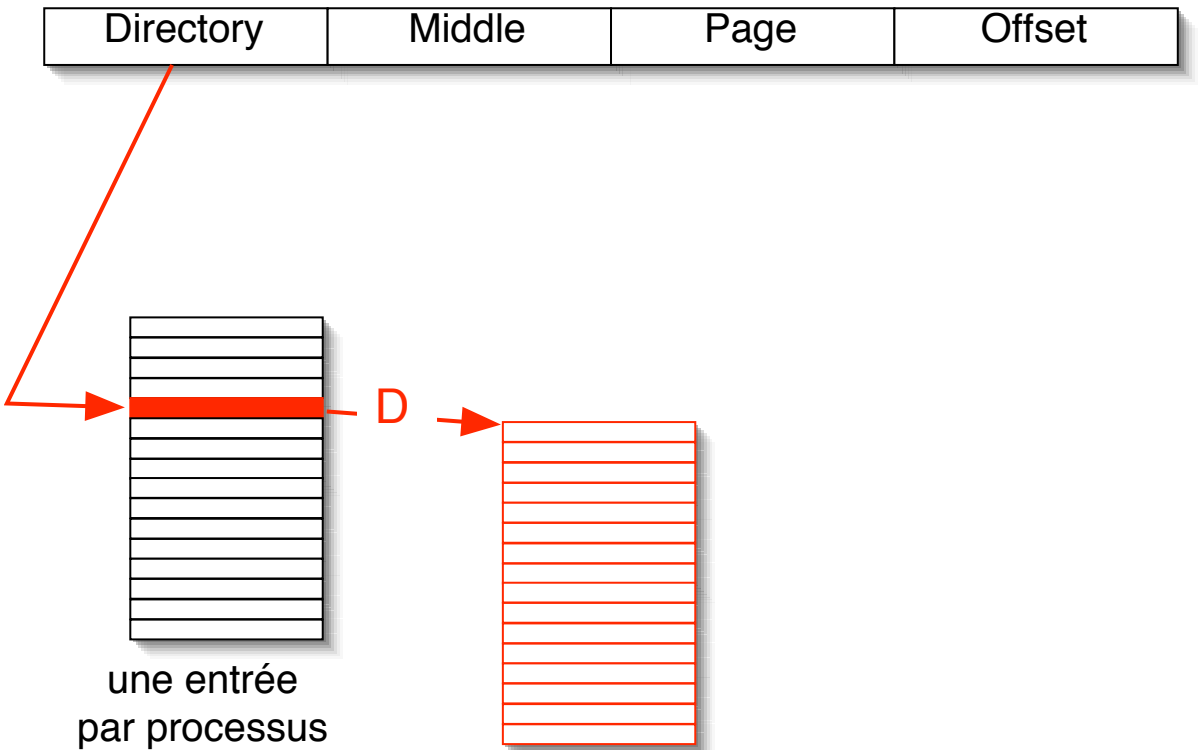
- Pagination 3 niveaux
- adresse sur 32 bits : 3GB + 1GB (SuperUser mode)
- adresse virtuelle :

Fonction de pagination LINUX

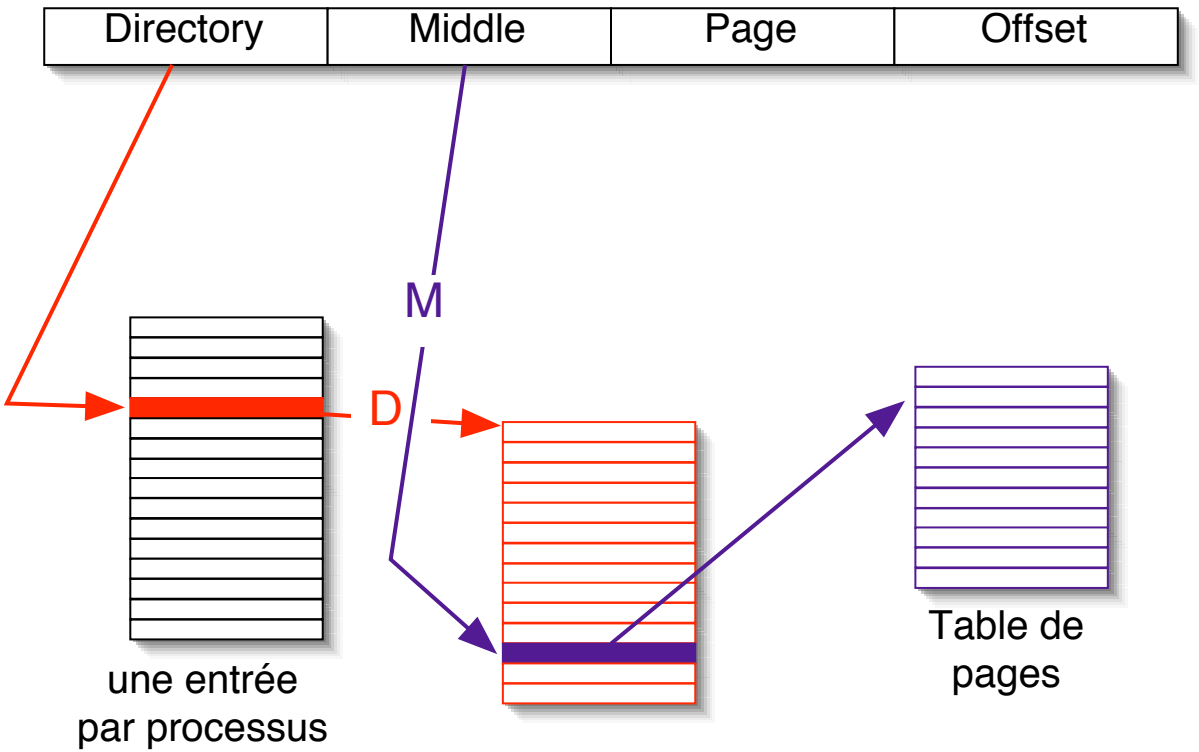
- Pagination 3 niveaux
- adresse sur 32 bits : 3GB + 1GB (SuperUser mode)
- adresse virtuelle :

Directory	Middle	Page	Offset

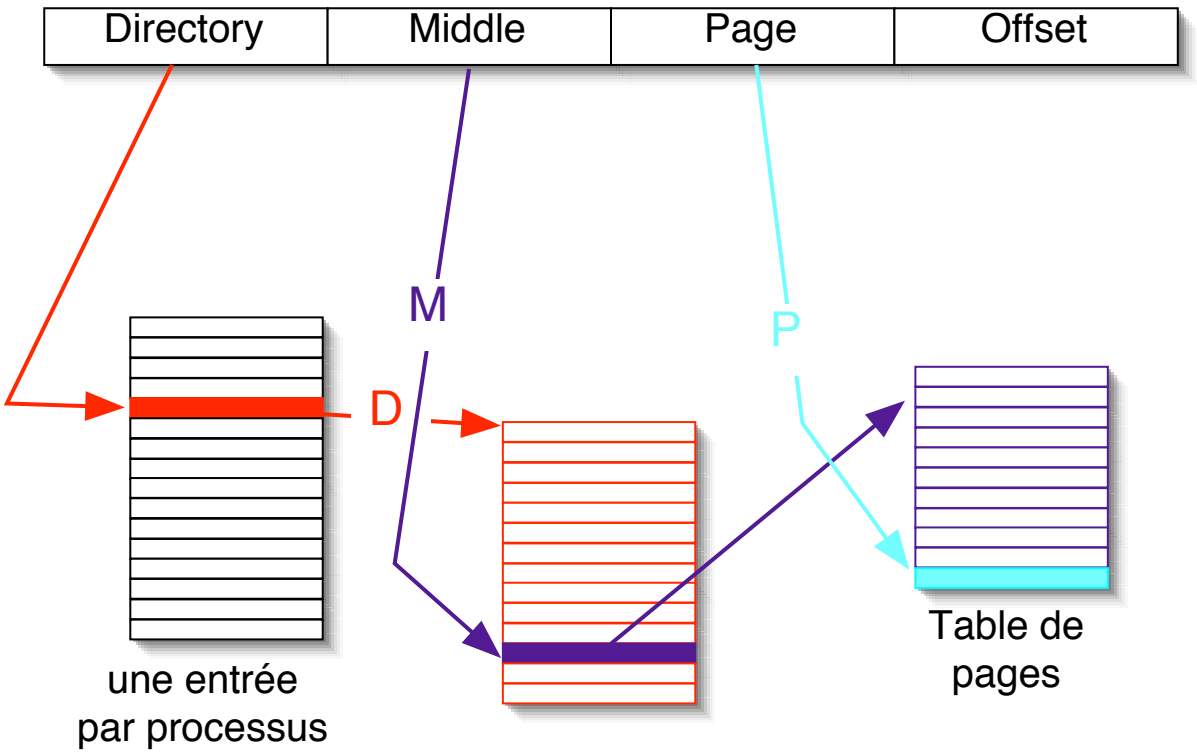
Fonction de pagination LINUX



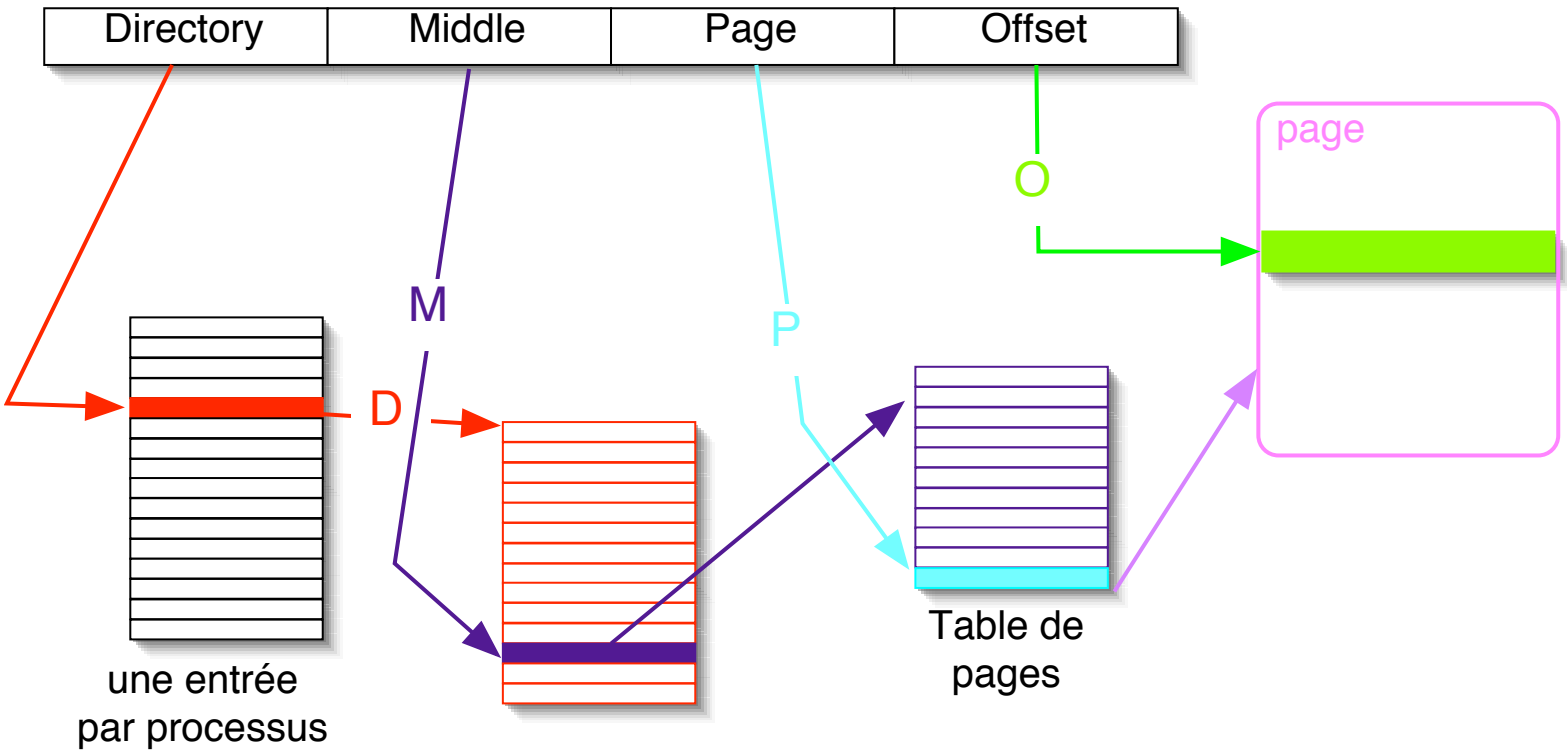
Fonction de pagination LINUX



Fonction de pagination LINUX



Fonction de pagination LINUX



Module noyau chargé dynamiquement

- LINUX supporte chargement à la demande de drivers etc

Module noyau chargé dynamiquement

- LINUX supporte chargement à la demande de drivers etc
- $\Longrightarrow$ augmentation de la taille du noyau

Module noyau chargé dynamiquement

- LINUX supporte chargement à la demande de drivers etc
- $\Longrightarrow$ augmentation de la taille du noyau
- Mémoire physique est gérée par Buddy System binaire

Module noyau chargé dynamiquement

- LINUX supporte chargement à la demande de drivers etc
- $\Longrightarrow$ augmentation de la taille du noyau
- Mémoire physique est gérée par Buddy System binaire
- Problème : fragmentation interne

Module noyau chargé dynamiquement

- LINUX supporte chargement à la demande de drivers etc
- $\Longrightarrow$ augmentation de la taille du noyau
- Mémoire physique est gérée par Buddy System binaire
- Problème : fragmentation interne
- autre niveau d'allocation mémoire ...

Le kswapd

- `kswapd` = daemon (gère les défauts de page)
- toutes les secondes le `kswpad` s'active
- si assez pages libres, `sleep`
- Algorithme de `kswpad` ..... maximum 6 "essais"
  1. renvoie d'une page qui est dans le cache des pages non récemment utilisées
     clock algorithm
  2. libération d'une page partagée non utilisée
  3. renvoie d'une page d'un processus utilisateur
     clock algorithm

Plan

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)
2. `boot` lit le répertoire `root` (peut lire le *filesystem*) et charge en mémoire basse le `kernel`

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)
2. `boot` lit le répertoire `root` (peut lire le *filesystem*) et charge en mémoire basse le `kernel`
3. `boot` effectue ensuite un `goto` pour exécuter le `kernel` (programme écrit en assembleur, dépendant de la machine) But : détecter capacité mémoire, CPU, paging system etc

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)
2. `boot` lit le répertoire `root` (peut lire le *filesystem*) et charge en mémoire basse le `kernel`
3. `boot` effectue ensuite un `goto` pour exécuter le `kernel` (programme écrit en assembleur, dépendant de la machine) But : détecter capacité mémoire, CPU, paging system etc
4. `kernel` termine par lancer la boucle `main` (OS) Les messages de `main` sont écrits dans un buffer

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)
2. `boot` lit le répertoire `root` (peut lire le *filesystem*) et charge en mémoire basse le `kernel`
3. `boot` effectue ensuite un `goto` pour exécuter le `kernel` (programme écrit en assembleur, dépendant de la machine) But : détecter capacité mémoire, CPU, paging system etc
4. `kernel` termine par lancer la boucle `main` (OS) Les messages de `main` sont écrits dans un buffer
5. allocation des structures mémoire de l'OS : table des pages, des processus, `coremap` etc etc

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)
2. `boot` lit le répertoire `root` (peut lire le *filesystem*) et charge en mémoire basse le `kernel`
3. `boot` effectue ensuite un `goto` pour exécuter le `kernel` (programme écrit en assembleur, dépendant de la machine) But : détecter capacité mémoire, CPU, paging system etc
4. `kernel` termine par lancer la boucle `main` (OS) Les messages de `main` sont écrits dans un buffer
5. allocation des structures mémoire de l'OS : table des pages, des processus, `coremap` etc etc
6. `probing` des périphériques

UNIX type FreeBSD

1. chargement du premier secteur du disque de *boot* et exécution programme de 512 octets : chargement du programme `boot` à une adresse mémoire fixe (haute)
2. `boot` lit le répertoire `root` (peut lire le *filesystem*) et charge en mémoire basse le `kernel`
3. `boot` effectue ensuite un `goto` pour exécuter le `kernel` (programme écrit en assembleur, dépendant de la machine) But : détecter capacité mémoire, CPU, paging system etc
4. `kernel` termine par lancer la boucle `main` (OS) Les messages de `main` sont écrits dans un buffer
5. allocation des structures mémoire de l'OS : table des pages, des processus, `coremap` etc etc
6. `probing` des périphériques
7. chargement des drivers des périphériques détectés

UNIX type FreeBSD

- fabrication du processus $PID = 0$
  programmation de l'horloge, `mount` du `root filesystem`
  et création des processus *init* (1) et *pagedaemon* (2)
- suivant les paramètres `init` fait :
  - mode single user : `fork` un processus qui fait un `exec` du shell et attend qu'il se termine
  - mode normal : `fork` un processus qui fait :

UNIX type FreeBSD

- fabrication du processus $PID = 0$
  programmation de l'horloge, `mount` du `root filesystem`
  et création des processus *init* (1) et *pagedaemon* (2)
- suivant les paramètres `init` fait :
  - mode single user : `fork` un processus qui fait un `exec` du shell et attend qu'il se termine
  - mode normal : `fork` un processus qui fait :
    1. un `exec` de `/etc/rc`
       But : `mount` le reste du `filesystem`, démarre des `daemons`

UNIX type FreeBSD

- fabrication du processus $PID = 0$
  programmation de l'horloge, `mount` du `root filesystem`
  et création des processus *init* (1) et *pagedaemon* (2)
- suivant les paramètres `init` fait :
  - mode single user : `fork` un processus qui fait un `exec` du shell et attend qu'il se termine
  - mode normal : `fork` un processus qui fait :
    1. un `exec` de `/etc/rc`
       But : `mount` le reste du `filesystem`, démarre des `daemons`
    2. lit `/etc/ttys` : pour chaque `terminal` :
       `fork` et exécution du programme `getty`
       (prompte avec `login :`)

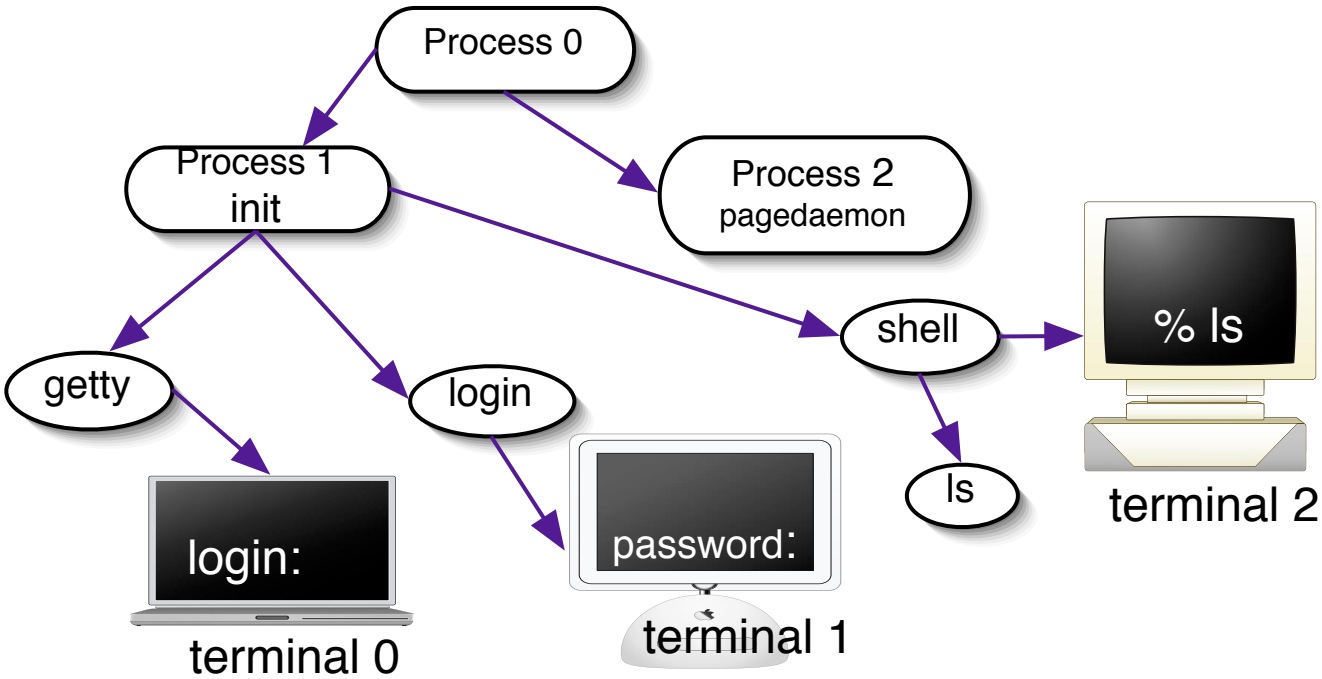
UNIX type FreeBSD

- fabrication du processus $PID = 0$
  programmation de l'horloge, `mount` du `root filesystem`
  et création des processus *init* (1) et *pagedaemon* (2)
- suivant les paramètres `init` fait :
  - mode single user : `fork` un processus qui fait un `exec` du shell et attend qu'il se termine
  - mode normal : `fork` un processus qui fait :
    1. un `exec` de `/etc/rc`
       But : `mount` le reste du `filesystem`, démarre des `daemons`
    2. lit `/etc/ttys` : pour chaque `terminal` :
       `fork` et exécution du programme `getty`
       (prompte avec `login :`)
    3. si un `login` est tapé : `getty` termine en faisant un `exec` de `/bin/login` (qui demande le password)

UNIX type FreeBSD

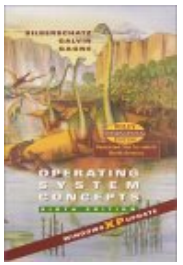
- fabrication du processus $PID = 0$
  programmation de l'horloge, `mount` du `root filesystem`
  et création des processus *init* (1) et *pagedaemon* (2)
- suivant les paramètres `init` fait :
  - mode single user : `fork` un processus qui fait un `exec` du shell et attend qu'il se termine
  - mode normal : `fork` un processus qui fait :
    1. un `exec` de `/etc/rc`
       But : `mount` le reste du `filesystem`, démarre des `daemons`
    2. lit `/etc/ttys` : pour chaque `terminal` :
       `fork` et exécution du programme `getty`
       (prompte avec `login :`)
    3. si un `login` est tapé : `getty` termine en faisant un `exec` de `/bin/login` (qui demande le password)
    4. si *login* correct, `login` fait un `exec du shell`

Après le boot

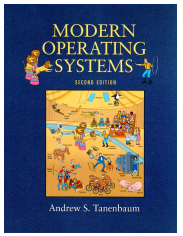


Livres Sur les systèmes d'exploitation et UNIX

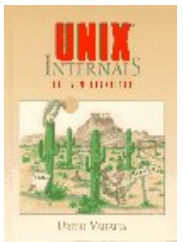
- James L. Perterson and Abraham Silberschatz.
  Operating Systems Concepts.
  John Wiley & Sons Inc, 6th edition, April 2002.
  952 pages.
  ISBN : 0471262722.



- Andrew S. Tanenbaum.
  Modern Operating Systems.
  Prentice-Hall, second edition, 2002.



- Uresh Vahalia.
  Unix Internals – The New Frontiers.
  Prentice-Hall, 1996.
  ISBN :0-13-101908-2.



Unix

Dennis Ritchie and Ken Thompson.
The Unix Timesharing System.
Communications of the ACM, pp. 225–233, july 1974.
http://www.bell-labs.com/history/unix/