Systèmes d'exploitation

Planification

Ingo Molnar

- Hongrois
- RedHat
- Completly Fair Scheduler

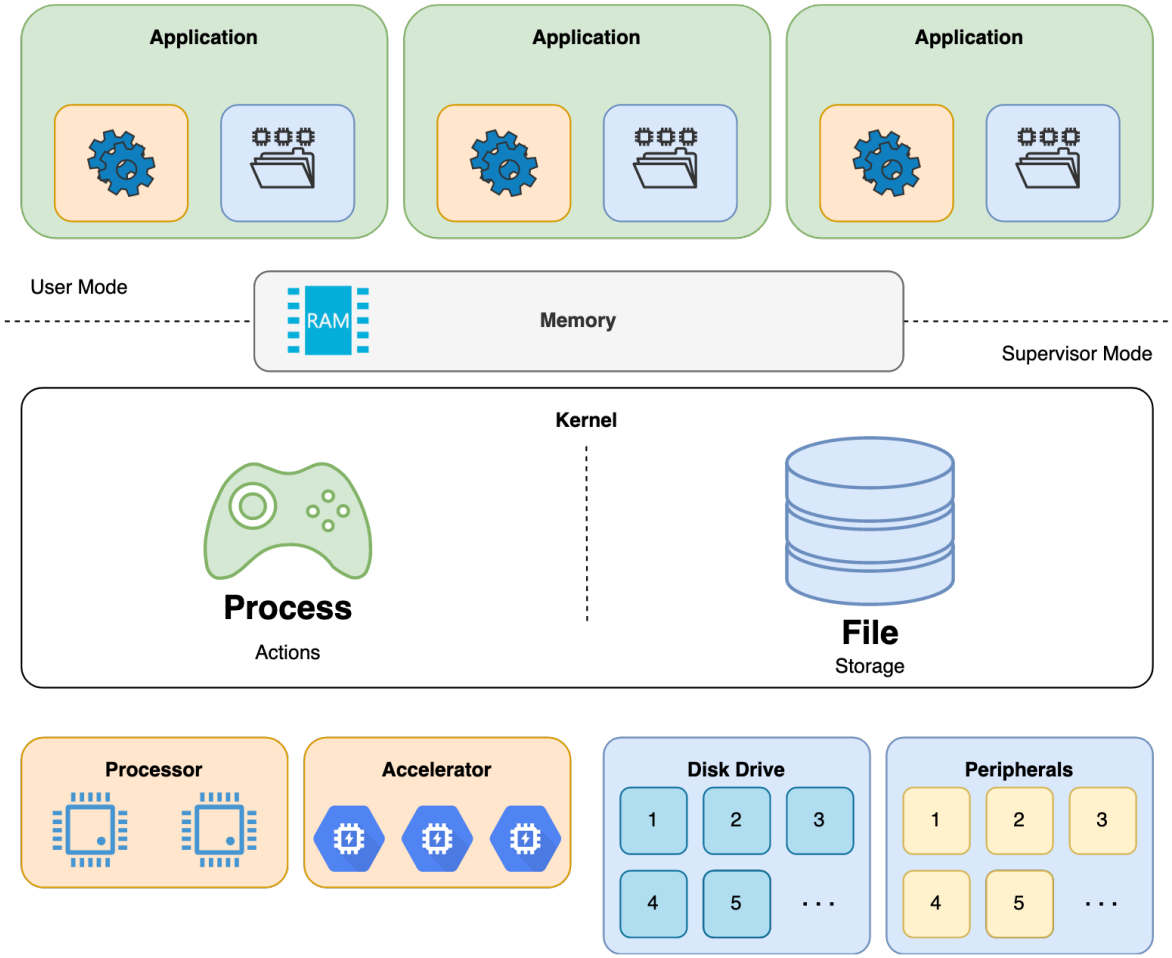
Contenu

- Planification
  - Non-préemptif
  - Préemptif

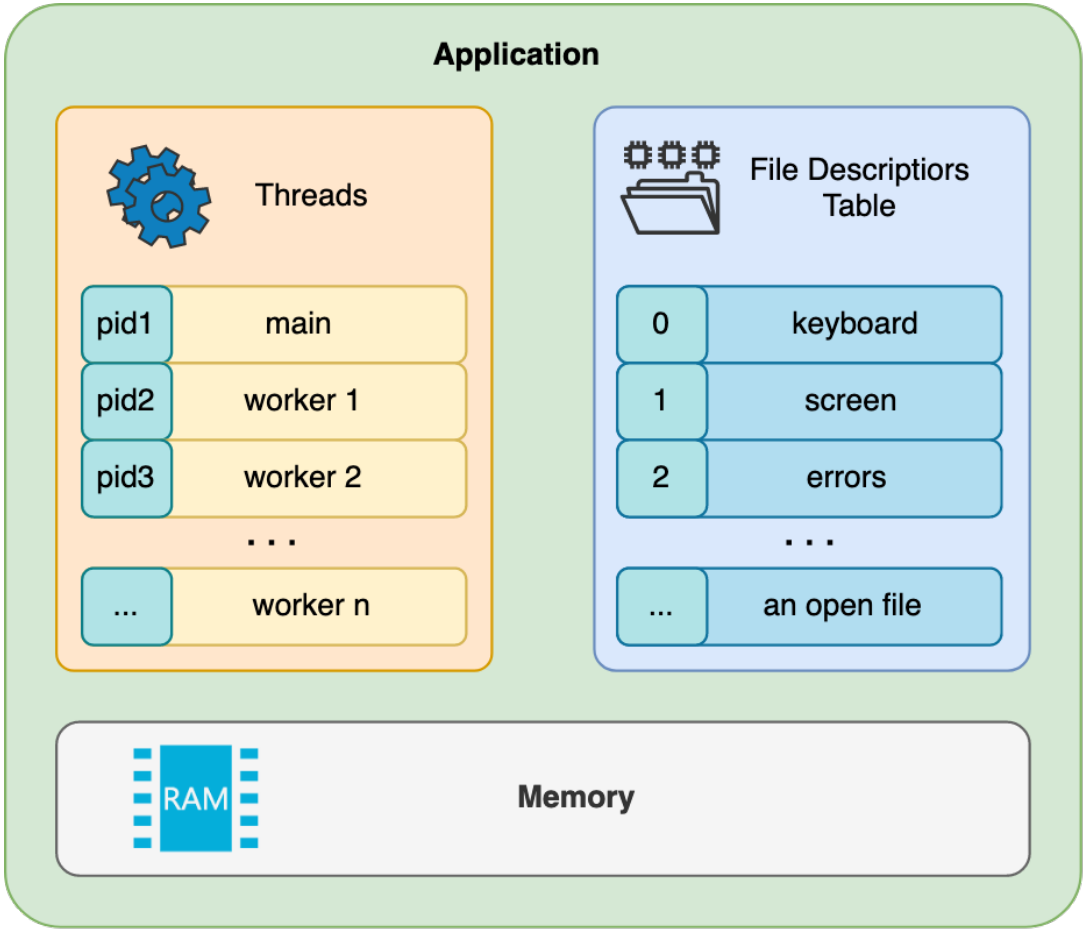
Bibliographie pour aujourd'hui

- Modern Operating Systems
  - Chapitre 2
    - 2.3
    - 2.5
- Operating Systems Concepts
  - Chapitre 6

Abstractions - Idée General



API utilisée par les processus



PLANIFICATION

Multitasking

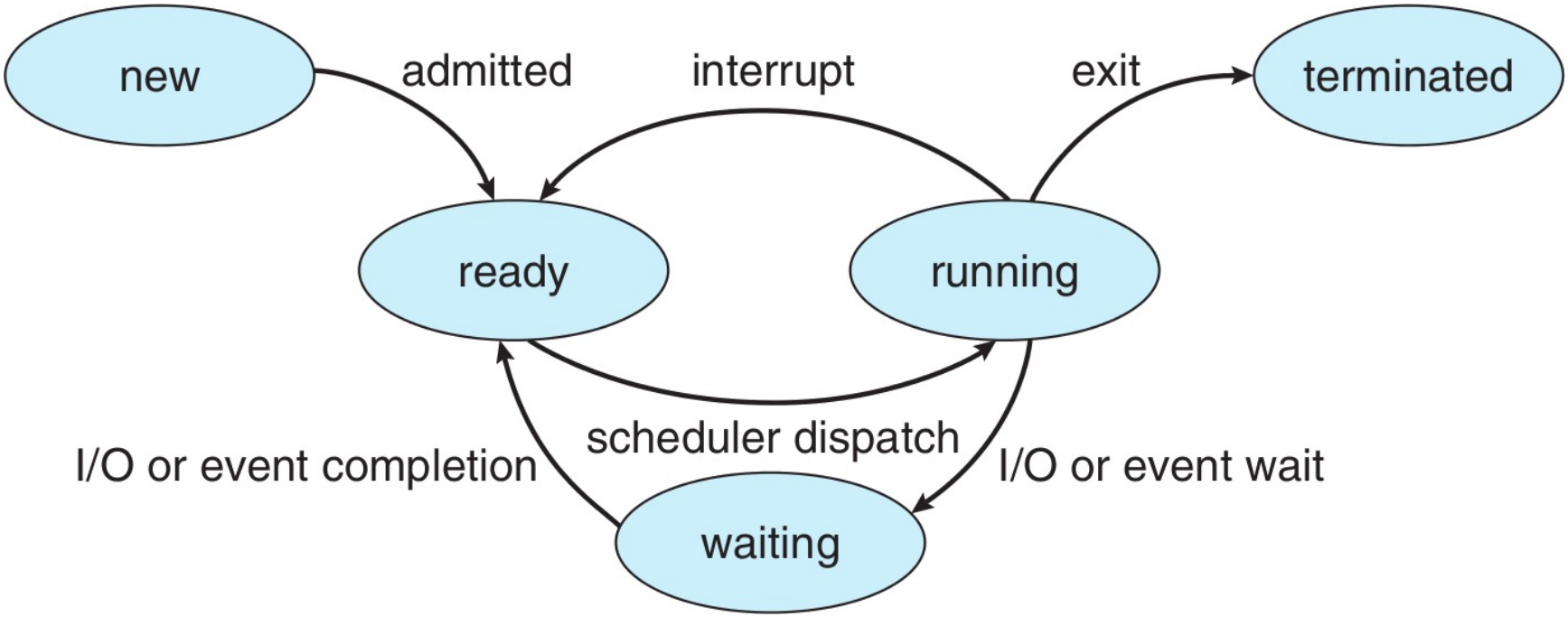
- SE traite rapidement les processus sur les processeurs
- Après un certain temps (quantum, time slice), un processus est supprimé du processeur et mis en place un autre.
  - on dit que "le quantum expire"
  - l'action s'appelle "**changement de contexte**"
- Le quantum est millisecondes
  - les processus changent très rapidement
  - l'impression de courir simultanément

Le statut d'un processus

- exécution (**RUNNING**)
  - Le processus s'exécute sur un processeur
- attendre (**WAITING**)
  - le processus a exécuté une action bloquante (par exemple, lecture d'E/S) et attend l'arrivée des données; ne peut pas s'exécuter
- prêt d'exécution (**READY**)
  - le processus peut s'exécuter sur le processeur

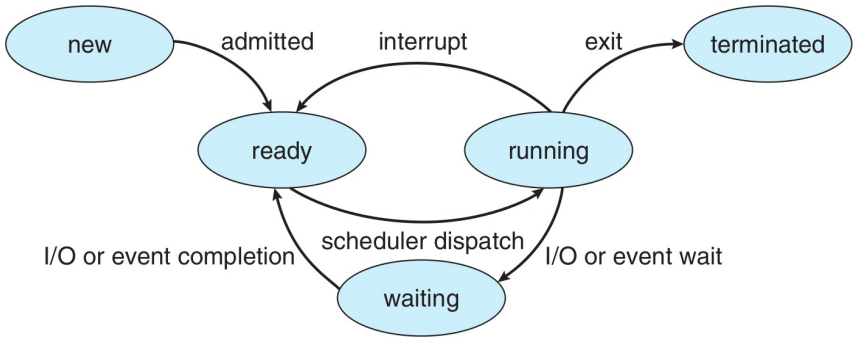
- Combien de processus peuvent être trouvés dans les chaque état?
- Comment assureriez-vous la gestion des processus dans les trois états?

Transitions entre les états

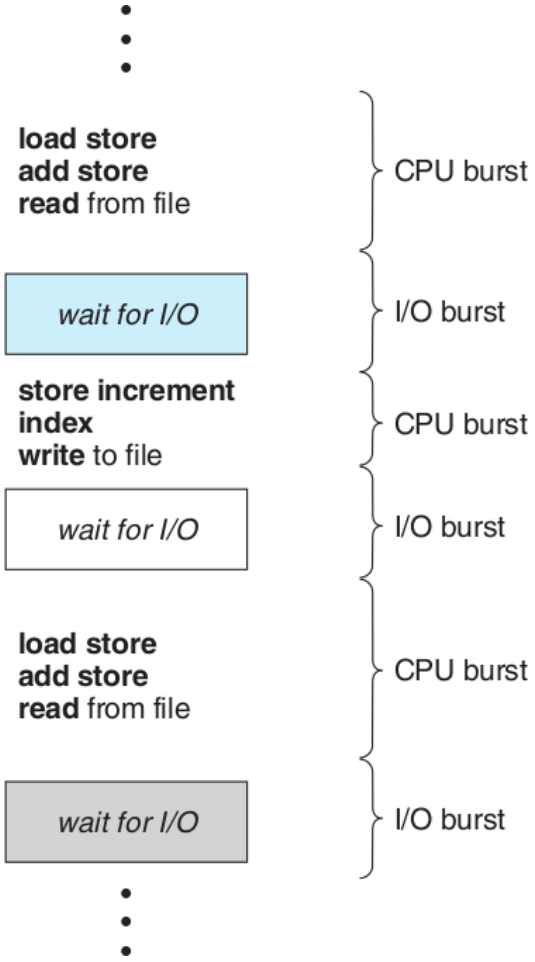


Transitions entre les états

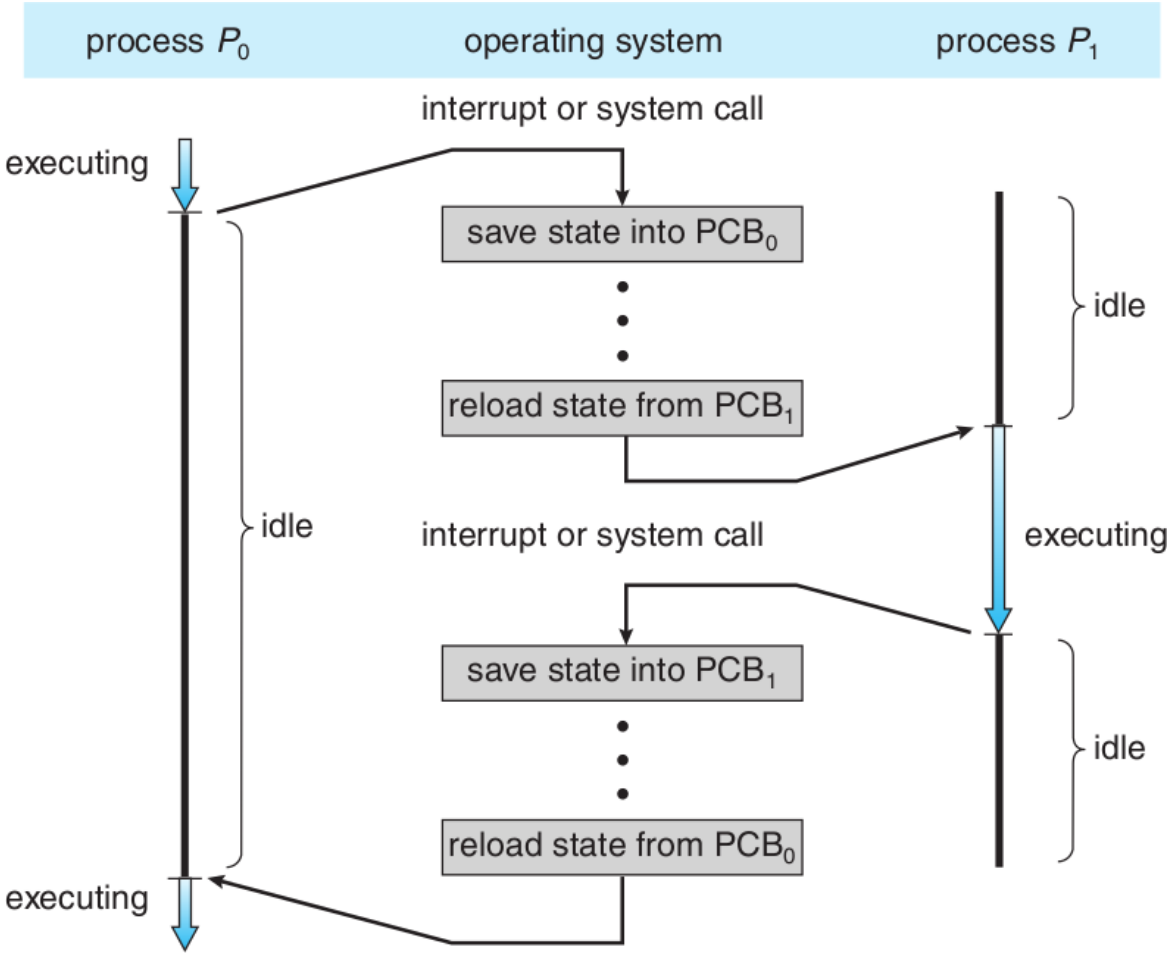
- RUNNING -> READY
  - le quantum de processus a expiré
  - il y a un autre processus dans l'état READY avec une priorité supérieure
- RUNNING -> WAITING
  - le processus a exécuté une opération bloquante
- WAITING -> READY
  - l'événement de processus attendu s'est produit
- READY -> RUNNING
  - un processeur a été libéré
  - le processus est le premier dans la file d'attente des processus READY



Fonctionnement d'un processus

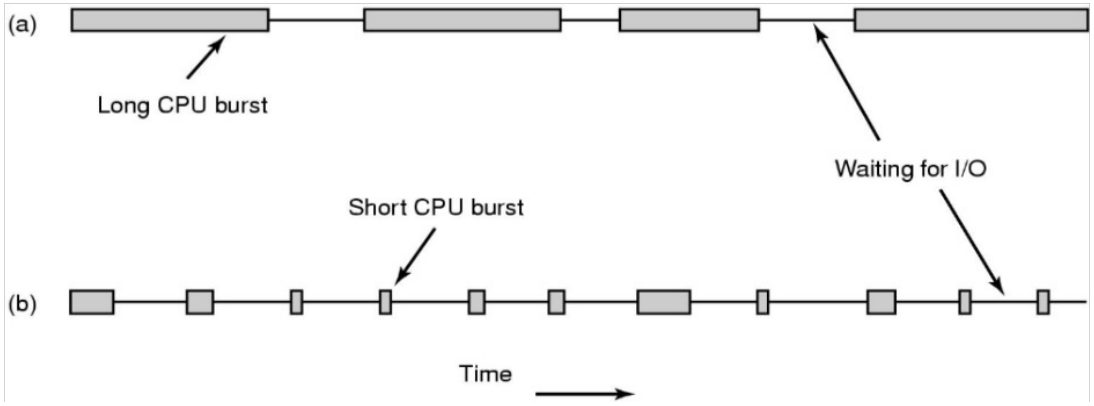


Changement de contexte



Types de processus (CPU)

- CPU bound (CPU intensive)
  - utilise beaucoup le processeur
- I/O bound (I/O intensive)
  - utilise rarement le processeur
  - faire des opérations d'E / S -> se bloque



Types de SE (dpdv. planificateur)

- Préemptive
  - les processus sont arrêtés à l'expiration de la quantum de temps
    - Windows, macOS, Linux, UNIX, Minix etc.
- Non-préemptive (coopératif)
  - les processus doivent s'arrêter (**s'ils le souhaitent**) à l'expiration de la quantum de temps
    - mac OS <= 9

Definition

	Definition	Description
WT	Wait Time	Temps d'attente pour recevoir le CPU
MWT	Mean Wait Time	Temps d'attente moyen pour la réception du CPU
TT	Turnaround Time	Temps d'exécution total du processus
MTT	Mean Turnaround Time	Le temps total moyen d'exécution du processus
J	Job	Processus en background
P	Process	Processus interactif

Systeme ideal

- Wait Time Minimale
  - les processus attendent le moins possible en READY
  - interactif
- Turnaround Minimale
  - processus à s'exécuter le plus rapidement possible
  - productif
- Fairness
  - toutes le processus doit avoir accès aux CPU

Types de systemes

- Procession en background
  - Batch Job
  - N’a pas d’interaction avec l’utilisateur
- Interactif
  - Process
  - Interaction avec l’utilisateur
- Temps réel

Sd'E
system crunch

PROCESSION EN BACKGROUND

Procession en background

- Paramètres optimisé
  - Throughput
    - Numéro de processus fini / unité de temps
  - Turnaround time
  - Utilisation de CPU

- Algorithmes
  - First Come First Served (FCFS)
  - Shortest Job First (SJF)
  - Shortest Remaining Time First (SRTF)

First Come First Served

- Simple d'implémenter

- Les processus CPU Bound ralentissent les processus I/O Bound
- Grand MWT et TT

Exemple

Job	Temp d'execution
J1	24
J2	3
J3	3
J4	5

- Calculez
  - TT
  - MTT
- J1, J2, J3, J4

- J2, J1, J4, J3

Shortest Job First

- Simple d’implémenter

- Le temps d'exécution de chaque processus doit être connu

Exemple

Job	Temp d'execution
J1	12
J2	9
J3	5
J4	20

- Calculez
  - TT
  - MTT
- FCFS
  - J1, J2, J3, J4
- SJF
  - J3, J2, J1, J4

Shortest Remaining Time First

- Version préemptif pour SJF
- Simple d’implémenter
- Le temps d'exécution de chaque processus doit être connu

Exemple

Job	Temp d'execution	Temp d'arrive
J1	8	0
J2	4	1
J3	9	2
J4	4	3

- Calculez
  - TT
  - MTT
- FCFS
  - J1, J2, J3, J4
- SJF
  - J3, J2, J1, J4
- SRTF

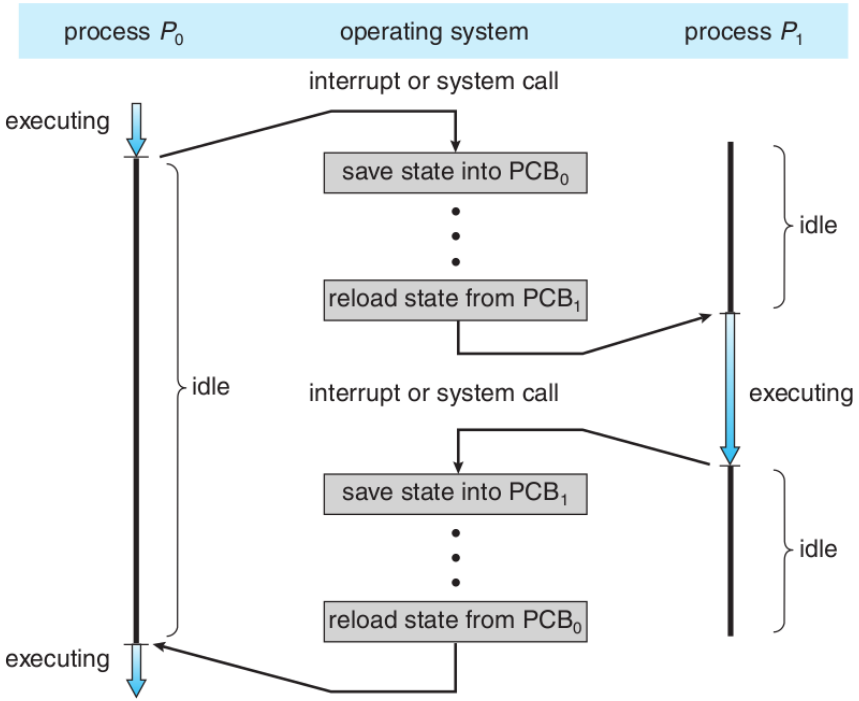
SYSTÈMES INTERACTIFS

Systèmes Interactifs

- Paramètres optimisé
  - Wait Time
  - Fairness
- Algorithmes
  - Round Robin (RR)
  - Classes des Priorisées
  - Shortest Process Next (SPN)

Round Robin

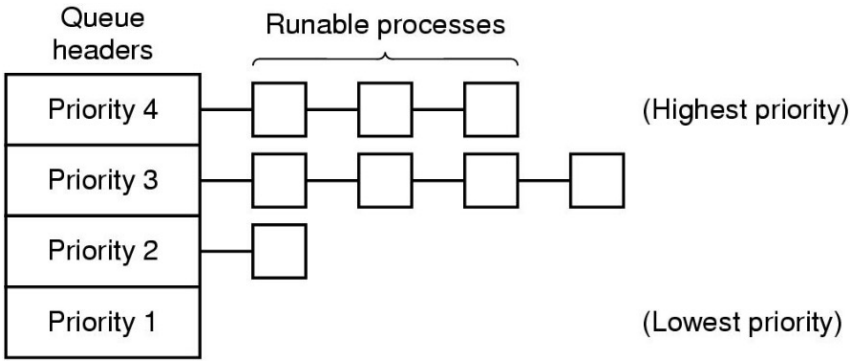
- FCFS Préemptif
  - Quantum de temps
- Grand Quantum de temps
  - Grand throughput
  - Grand Wait Time
- Pietit Quantum de temps
  - Petit throughput
    - Temps pur le changement de contexte este grand
  - Petit Wait Time



Classes des Priorisées

- Round Robin
  - avec de processus plus important que autres

- Plusieurs files d'attente



Classes des Priorités

Windows

- Priority
  - 0 Système
  - 1 a 15 User
  - 16 a 30 temps réel

Linux

- Priority
  - 0 a 99 temps reel
  - 100 a 139 user
- Nice
  - -20 a 19
- Priority effective
  - 100 + 20 + NI

Classes des Priorités (Linux)

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3540	root	15	0	470m	183m	29m	S	1	9.7	66:03.44	Xorg
27935	razvan	15	0	364m	49m	19m	S	1	2.6	6:36.10	amule
32666	haldaemo	15	0	29148	4132	3148	S	0	0.2	0:27.70	hald
1	root	18	0	10328	768	640	S	0	0.0	0:01.89	init
2	root	11	-5	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	RT	-5	0	0	0	S	0	0.0	0:00.02	migration/0
4	root	34	19	0	0	0	S	0	0.0	0:00.01	ksoftirqd/0
5	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/0
9	root	10	-5	0	0	0	S	0	0.0	0:32.93	events/0
11	root	10	-5	0	0	0	S	0	0.0	0:00.00	khelper
31	root	10	-5	0	0	0	S	0	0.0	0:00.02	kblockd/0

Shortest Process Next

- SJF Préemptif
- Estimation basée sur le comportement précédent
- T0 - quantum de temps estimée
- T1 - quantum de temps réel
- a - âgée
- Quantum de temps suivante
  - a * T1 + (1-a) * T0

TEMPS RÉEL

Temps Reel

- Limite maximale
  - Turnaround
  - Wait Time
- Hardware
  - ressources alloue
- Software
  - Les processus de temps réel ont la priorité maximale
  - Les autres processus sont retardé

Priority Inversion

- Un processus avec un priorité petit (P1) prend un ressource
- Un autre processus avec un priorité plus grand (P2) a besoin de cette ressource
- Processus P2 a la même priorité avec le processus P1

Priority Boosting

- Augmenter la priorité pendant un temps limité
- À la réception des données d'E/S
  - En dépendant de Wait Time
- Un autre processus avec un priorité plus grande attend un ressource bloque par le processus

Starvation

- Un processus ne prend pas le CPU pour un long temps

FERMER UN PROCESSUS

Fermer un processus

- Terminer l'exécution
- Fermé par le SE

Terminer l'exécution d'un processus

- Le processus termine sa exécution
  - arrivé la fin du programme
  - appelé **exit**
- La décision c’est au processus

Fermé par le SE

- Le SE ferme le processus
  - à la demande d'un autre processus (kill)
  - le processus a effectué une erreur

- Envoyer un signal (Unix) ou une exception (Windows)

Effacer le processus

- Le SE efface toute les informations du processus
  - PCB
  - PID
  - Tableau des descripteurs
  - Mémoire alloue
  - …
- Le SE envoie un signal (Unix) / message (Windows) aux processus parent

Attendre le fin du processus

- Synchroniser l'action de certains processus
  - tu fais X après Y
  - le processus qui fait l'action X attend la fin de l'exécution du processus qui fait l'action Y
- Cela s'appelle "attendre un processus"

- Informations utiles sur le fin du processus
  - la valeur de retour

La valeur de retour

```c
int main ()
{
    // do some work ...
    if (error)
    {
        // exit with an error code, here -120
        exit (-120);
    }
    // exit without error, success
    return 0;
}
```

Exemple

```
int fd, status;
pid_t pid = fork ();
if (pid == 0)
{
    // child
    if (!execl ("/bin/ls", "/bin/ls", "-l", NULL))
    {
        perror ("Error loading /bin/ls\n");
        abort ();
    }
}
else if (pid > 0) {
    // parent
    waitpid (pid, &status, 0);
    printf ("/bin/ls exited with %d\n", WEXITSTATUS (status));
}
```

Processus orphelin

- Un processus dont le parent a terminé sa exécution
- Il perd sa connexion dans la hiérarchie des processus
- Il est adopté par un processus dédié (*init* sur Unix)

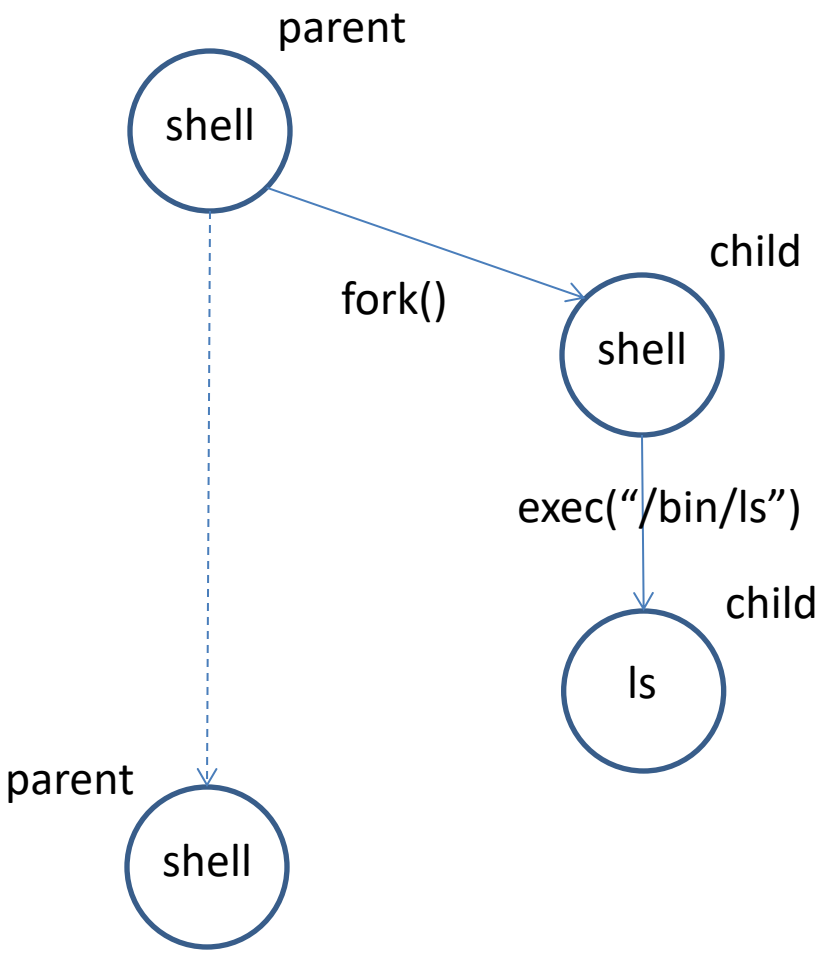
Processus zombie

- Un processus qui s'est terminé mais n'était pas attendu
  - Les informations résiduelles restent (elles prouvent être utilisé par le processus parent)
- Si un processus zombie reste orphelin, il est adopté par *init* puis il est terminé
- Les processus zombies plus longs occupent (en vain) les ressources de système

Le operateur shell &

- En règle générale, le shell attend le fin des processus créés
- L'opérateur &: shell n'attend plus la fin du processus
  - le processus s'exécute en background
  - le shell contrôle le stdin

Exemple

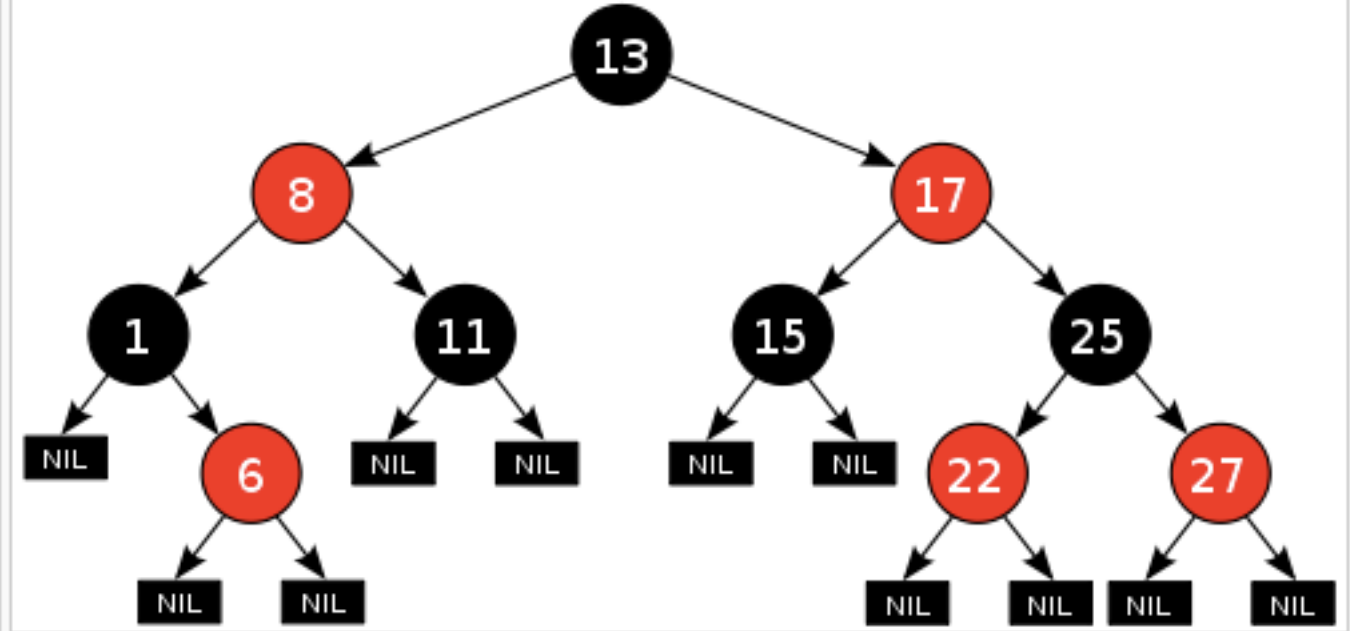

parent
shell
child
fork()
shell
exec(“/bin/ls”)
child
ls
parent
shell

COMPLETELY FAIR SCHEDULER

Completly Fair Scheduler

- Virtual Runtime (vruntime)
  - Temp d'utilisation de CPU
- Arbre Rouge - Noir
  - ordonné par la vruntime
- Processus nouveau
  - vruntime = vruntime minimale

Completly Fair Scheduler



An example of a red–black tree

Mot clés

- Planification
- Procession en background
- Job
- Systèmes interactifs
- Processus
- Orphelin
- Zombie
- Temps réel
- Shortest Job First
- Shortest Remaining Job
- First
- Round Robin
- Shortest Process Next
- Priorité
- Priority Inversion
- Priority Boosting
- Starvation
- Completly Fair Scheduler

Questions