ReDS
Reconfigurable & Embedded Digital Systems

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Systèmes d'exploitation
Gestion des processus

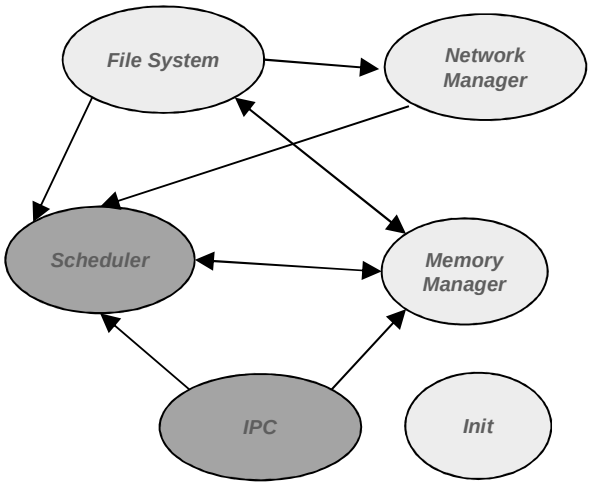
Cours SYE

Prof. Daniel Rossier

Version 2.3 (2009-2010)

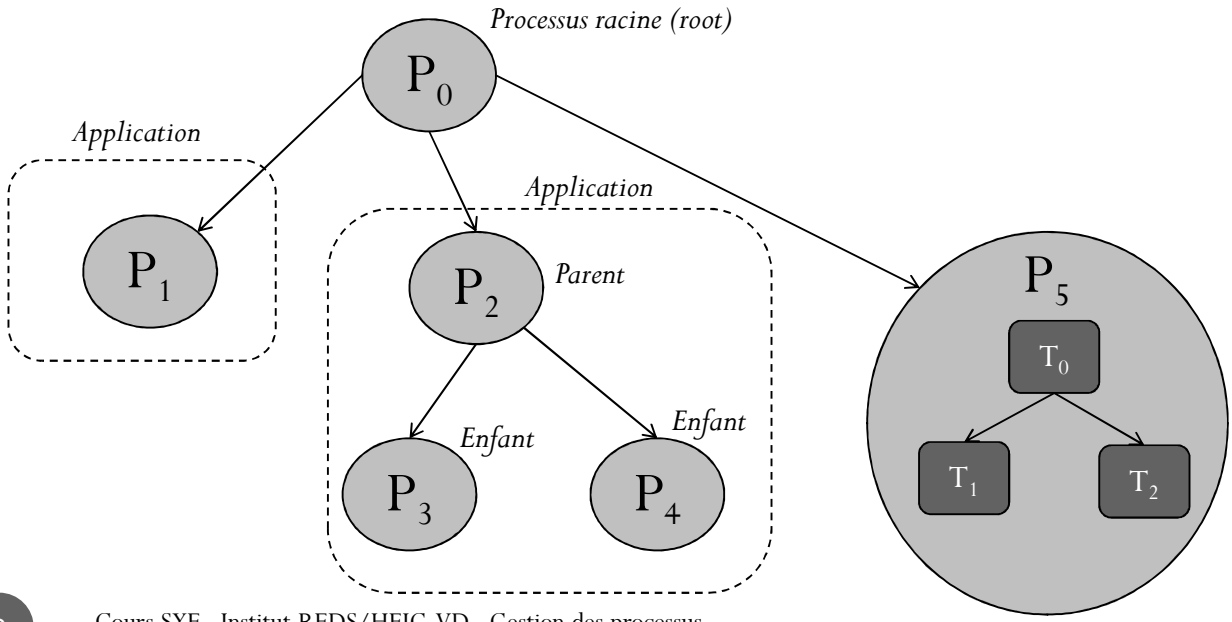
Plan

- Processus
- Changement de contexte
- Threads
- IPC (Inter-Process Communication)
- Algorithmes d'ordonnancement



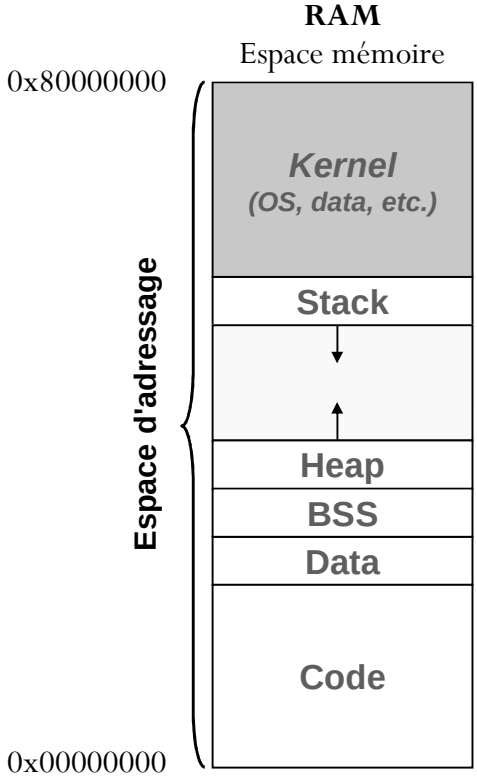
Applications, processus, tâches, ...

- **Seul la notion de processus (et par la suite de *thread*) est pertinente au niveau d'un OS (noyau).**
  - Les autres termes sont propres aux langages, environnements, "habitudes", etc.



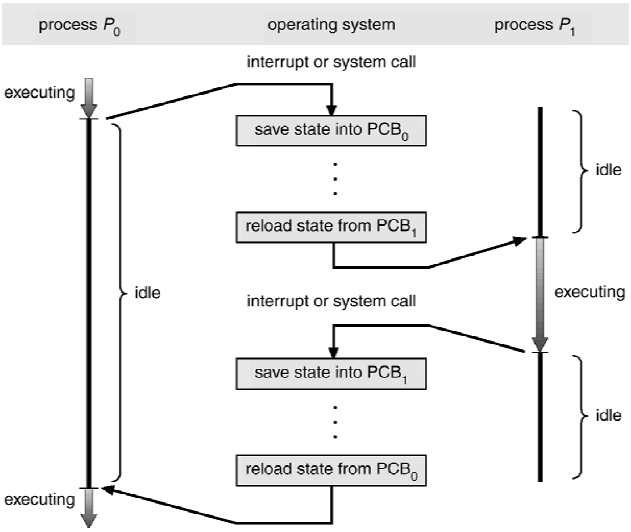
Processus - Introduction

- Un processus est caractérisé par:
  - Une **image binaire**
    - Un fichier exécutable pouvant être chargé en mémoire.
  - Un **contexte d'exécution**
    - Code
    - Valeurs de registres, caches processeur
  - Un **espace mémoire**
    - Données (variables globales, locales, constantes)
    - Espace mémoire virtuel
  - Un **ensemble de ressources**
    - Fichiers ouverts
    - Ressources de communications interprocessus
    - Ressources matériels
    - Alertes/signaux en attente
    - …



Processus – Changement de contexte

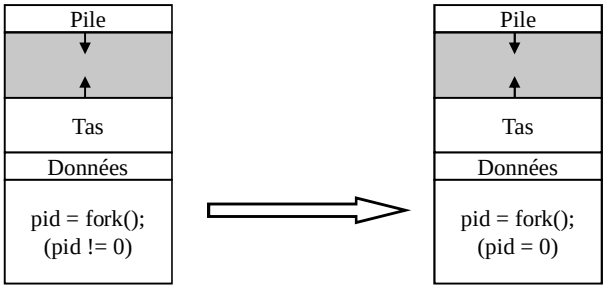
- Le basculement d'un processus à l'autre est géré par le noyau.
  - **Suspendre** le processus P0
  - **Mettre à jour** le PCB du processus P0
  - **Restaurer** le PCB du processus P1
  - **Reconfigurer** l'espace mémoire
    - Reconfiguration du Memory Management Unit (MMU)
  - **Démarrer** processus P1
- Cette opération est généralement décidée et réalisée par l'ordonnanceur.
  - C'est une opération **très coûteuse** !
    - O(ms)



Création de processus – fork()

- L'appel système ***fork***() permet de créer un nouveau processus.
  - Le nouveau processus devient un **processus fils** du processus qui exécute fork().
  - Le processus fils **hérite des ressources du processus parent**.
    - **L'espace mémoire est dupliqué** (inclut les descripteurs de fichiers et d'autres ressources)
  - L'espace mémoire **est copiée dans son intégralité** (code, données, etc.)
    - Au niveau du code, il faudra **différencier** celui du parent et celui du fils.

Processus parent (original) *Processus fils (nouveau)*



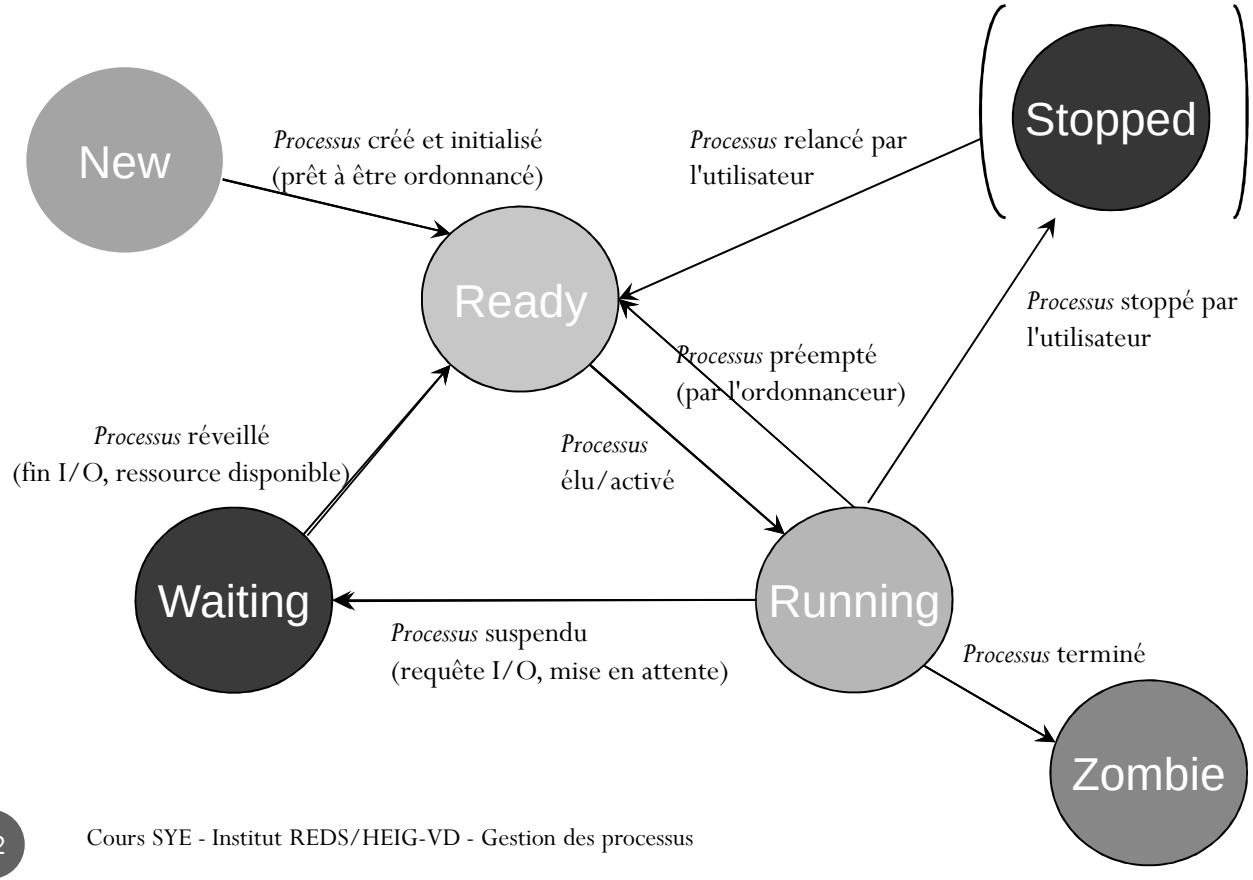
Appel système *fork()*

- Soit le code ci-dessous:

```
int main ()
{
  int pid;

  printf(" message0 \n ");
  pid = fork();
  if (pid < 0) exit(-1);
  if (pid)
  {
    printf(" message1 \n ");
    pid = fork();
    if (pid < 0) exit(-1);
    if (pid)
      printf(" message2\n ");

  } else
   printf("message3\n ");

}
```

- Combien y a-t-il de processus au maximum?
- Donnez, sous forme d'un arbre, les différents ordres possibles d'affichage des messages (chaque chemin de l'arbre devrait correspondre à un ordre possible).

Processus – Etats et transitions



Threads (1/7) - Introduction

- Décomposition d'une application en plusieurs tâches indépendantes.
  - Ces tâches peuvent s'exécuter en parallèle.
  - L'application peut s'exécuter sur plusieurs processeurs.
- Deux approches
  - Multi-processus
  - ***Multi-thread***
- **Multi-processus**
  - Les **changement de contexte** sont coûteux.
  - Chaque processus dispose de ses propres ressources.
  - La **communication inter-processus** est coûteuse (cf IPC).

Threads (2/7) - Introduction

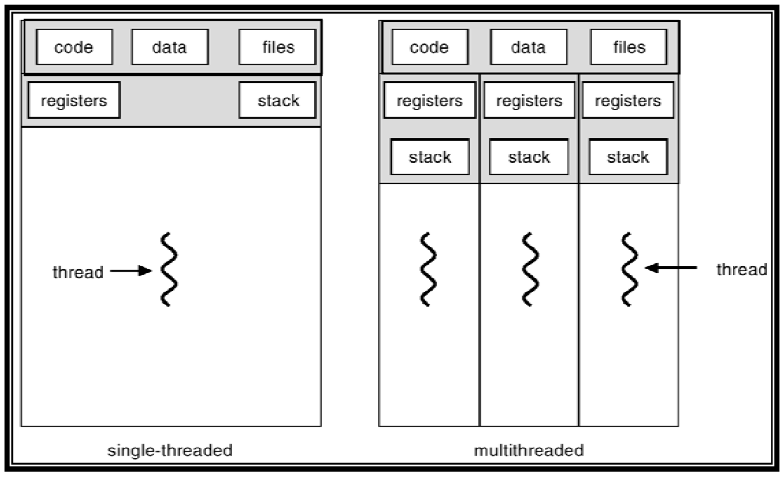
- *Multi-thread*
  - Un thread est un **processus léger** (*lightweight process*)
  - Les ressources sont partagées
    - Création rapide (env. 10-100 fois plus rapide que pour un processus)
    - Changements de contexte simplifiés et rapides
    - La communication entre les threads est "directe".
  - Très bonne performance
  - Notion *d'hyperthreading*
    - Gestion multi-threads au niveau du processeur

Threads (3/7) – Caractéristiques

- Un thread a un contexte réduit
  - Un **compteur ordinal** (PC)
  - Des **registres (*virtuels*)**
  - Une **pile**
  - Un **état**
  - Une **priorité**
- Conséquences
  - Un processus *multi-thread* a plusieurs piles !
  - Le modèle de programmation *multi-thread* peut être soit **coopératif**, soit **préemptif**

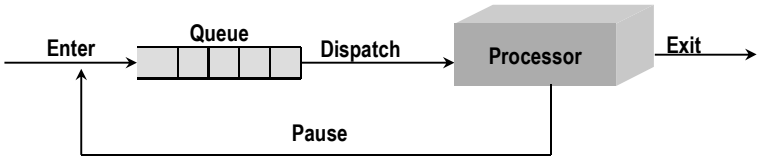
Threads (4/7) – Ressources

- Le thread utilise **l'espace d'adressage** du processus ainsi que **toutes les ressources** gérées par le processus.
  - Tous les *threads* se partagent les ressources du processus (code, variables globales, fichiers ouverts, …)
  - **La protection entre les *threads* n'est pas garantie** !
    - Les threads s'exécutent dans une même machine virtuelle.



Ordonnancement (1/12) - Processus

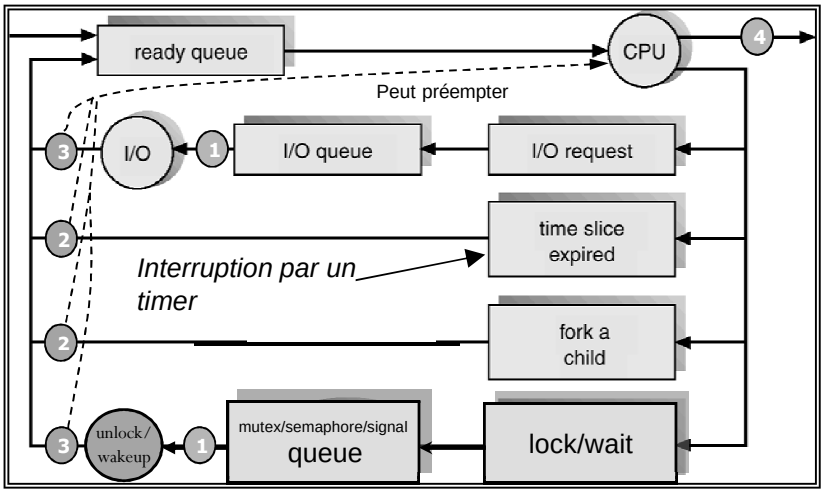
- Il faut déterminer quel processus va occuper le processeur.
  - Les processus dans l'état *ready* sont examinés.



- Cette opération peut s'effectuer à différents instants, durant l'exécution du code noyau.
  - Ces instants s'appellent *points de préemption*.
  - Typiquement, un point de préemption peut survenir à la fin du traitement d'une routine de service (ISR).

Ordonnancement (2/12) - Points de préemption

1. Transition *running* ⇒*waiting*
2. Transition *running* ⇒ *ready*
3. Transition *waiting* ⇒ *ready*
4. Transition *running* ⇒ *zombie*



- Dans le cas ① et ④ il y a ordonnancement de processus et l'ordonnancement est dit **non-préemptif**
- Dans le cas ② et ③ il peut y avoir ordonnancement de processus et l'ordonnancement est dit **préemptif**

Ordonnancement (3/12) - Critères

- Taux d'utilisation du CPU ⇒ Difficile à utiliser
  - Rapport temps CPU / temps écoulé
  - Il faut tenir compte des entrées/sorties.
    - Lié au degré de multiprogrammation
- **Délai de rotation** (*turnaround*)
  - Temps écoulé entre la soumission du processus jusqu'à sa terminaison
  - Intéressant lorsque l'on considère les I/Os
- Capacité de traitement (*throughput*) ⇒ Difficile à utiliser
  - Une bonne capacité de traitement ne garantit pas un délai de rotation minimale.
- **Temps d'attente** (*waiting time*)
  - Temps écoulé dans l'état ***ready***

Ordonnancement (4/12) - Burst

- Normalement, un processus effectue **une alternance de cycle d'activité et de cycle d'entrée-sortie (I/O)**.
  - Pour l'étude d'algorithmes d'ordonnancement, on considère uniquement des processus effectuant un cycle d'activité (sans entrée-sortie), que l'on appellera *burst* (rafale).
    - Pour l'étude des performances, un processus effectuant des I/Os peut être décomposé en plusieurs processus sans I/Os.
- Le *burst* a une certaine durée.

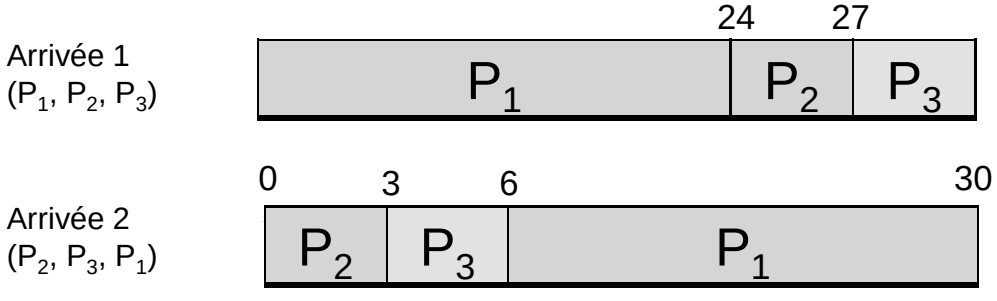
Ordonnancement (5/12) - Politiques/Algo

- **FCFS**
  - *First Come First Served*
- **SJF**
  - *Shortest Job First*
- **RR**
  - *Round-Robin*
- Ordonnancement **par priorité** avec/sans files d'attente multiples

Ordonnancement (6/12) - FCFS

- **FCFS** - *First Come First Served*
  (Premier arrivé, Premier servi)
  - Gestion des processus à l'aide d'un **FIFO**
  - Ordonnancement **non-préemptif**
  - Facile à implémenter
  - Sous-optimal

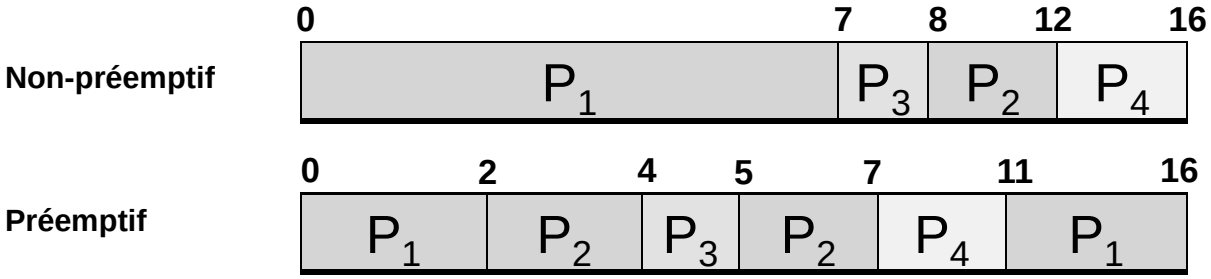
Processus	Durée
P_1	24
P_2	3
P_3	3



Ordonnancement (7/12) – SJF

- **SJF** - *Shortest Job First*
  (Job le plus court d'abord)
  - Sélection du processus nécessitant le moins de temps d'exécution (le prochain burst le plus court)
  - Ordonnancement **non-préemptif** ou **préemptif**
  - Difficile à implémenter
  - Tend à être optimal

Processus	Durée	Arrivée
P_1	7	0.0
P_2	4	2.0
P_3	1	4.0
P_4	4	5.0



Ordonnancement (8/12) – Détails SJF

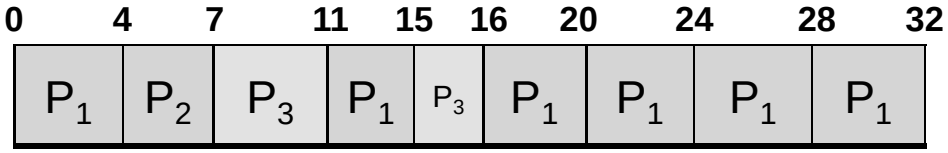
- SJF est impossible à implémenter en tant que tel.
  - Impossible de connaître la longueur du prochain *burst*.
  - On peut faire une approximation.
- La longueur du prochain *burst* peut être estimé au moyen d'une estimation basée sur les durées précédentes.
  - Ordonnancement à long terme
    - Nécessite l'observation des durées d'exécution.
    - Possibilité de **pondérer la dernière observation avec la dernière estimation**.

Ordonnancement (9/12) - RR

- **RR** - *Round Robin* (Tourniquet)
- Il s'agit d'un FCFS avec l'introduction d'une tranche ou **quantum de temps** (*time slice*)
- Ordonnancement **préemptif**
- Cet algorithme est très utilisé.

Processus	Durée
P_1	24
P_2	3
P_3	5

Quantum = 4 ms



Ordonnancement (10/12) - Priorité

- **Ordonnancement par priorité**
  - Un processus ne peut s'exécuter que si aucun processus de priorité supérieure n'est dans l'état *ready*.
  - Si tous les processus ont la même priorité, **c'est la politique FIFO** qui est appliquée.
  - Priorité dynamique
    - La priorité des processus est **augmenté graduellement**, afin d'éviter la famine du processus.
  - Ordonnancement
    - **non-préemptif** ou **préemptif**

Processus	Durée	Priorité
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

⇒ Politique **SCHED_FIFO** (défini dans la norme POSIX)