### GESTION DES PROCESSUS : Définition

Un processus est un programme en **cours d'exécution** auquel est associé un contexte à savoir un environnement processeur et un environnement mémoire.

Un processus est l’instance dynamique d’un programme et incarne le fil d’exécution de celui-ci.

**États d’un processus**

Lors de son exécution, un processus est caractérisé par un état :

Prêt

Bloqué

Elu

Blocage

Election

Déblocage

Fin

Réveil

**En attente du processeur**

En attente de ressources

**En exécution**

Diagramme d’états d’un processus

|  |
| --- |
| Identificateur processus |
| États du processus |
| Compteur ordinal  Bloc de contrôle de processus  Contexte reprise (registre et pointeurs, piles… = contexte processeurs) |
| Chainage selon les files de l’ordonnanceur priorité (ordonnancement) |
| Information mémoire (limites et tables pages / segments) |
| Informations sur les ressources utilisées fichiers ouverts, outils de synchronisation, entrées- sorties |
| Informations de comptabilisation ( info pour stat performance) |

### Opérations sur les processus

Le système d‘exploitation offre généralement les opérations suivantes pour la gestion des processus (via l’API (Application Programming Interface) : Application : Interface de Programmation pour accéder aux primitives du système d’exploitation) :

* Création, destruction de processus, suspension et reprise de l’exécution.
* Ordonnancement sur l’unité centrale

La fonction d’ordonnancement du SE(système d’exploitation) gère le partage du processeur entre les différents processus en attente pour s’exécuter.

***Ordonnancement préemptif et non préemptif***

Si l’ordonnancement est non préemptif, la transition de l’état élu vers l’état prêt est interdite : un processus quitte le processeur s’il a terminé son exécution ou s’il se bloque

Si l’ordonnancement est préemptif, la transition de l’état élu vers l’état prêt est autorisée : un processus qui le processeur s’il a terminé son exécution, s’il se bloque ou si le processeur est réquisitionné

**Politiques d’ordonnancement**

La politique d’ordonnancement détermine quel sera le prochain processus élu.

* Politique Premier Arrivé, Premier Servi (FIFO)
* Politique Plus court d'abord
* Politique du tourniquet (round robin) : c’est celle mise en œuvre dans les systèmes en temps partagé. Le temps est découpé en tranches nommées quantum de temps.

Élection

Exemple de Windows XP (tourniquet + priorités)

Les processus sous Windows

Temps

Occupation

CPU

**Priorité élevée**

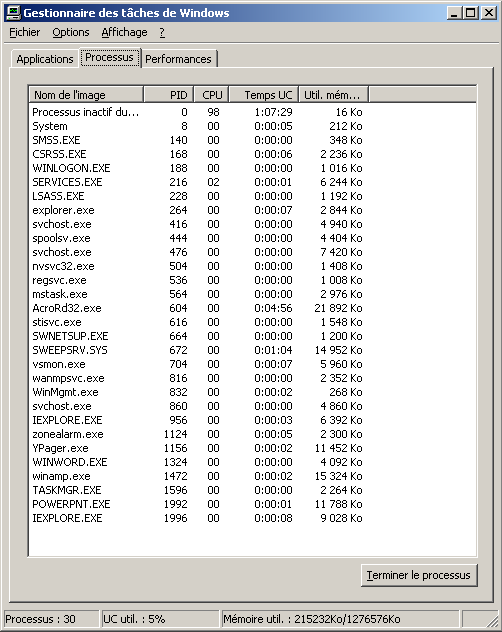
**Priorité moyenne**

**Priorité faible**

Processus 1

Processus 2

Processus 3



**Processus par défaut**

Exemple :

explorer.exe

SMSS.exe

**Processus lancés par l’utilisateur**

Exemple :

WINWORD.exe

zonealarm.exe

## Programmation multitâche





Figure 1: avantage multitâche : réactivité serveur

Figure 2:avantage multitâche : réactivité des IHM

**A. Problématique des processus « lourds »**

Les processus classiques d’UNIX possèdent des ressources séparées (espace mémoire, table des fichiers ouverts...). Lorsqu’un nouveau fil d’exécution (processus fils) est créé par fork(), il se voit attribuer une copie des ressources du processus père.

Il s’ensuit deux problèmes :

Problème de performances, puisque la duplication est un mécanisme coûteux

Problème de communication et de synchronisation entre les processus, qui ont des variables séparées.

**B. Définition du « processus léger »**

Il existe des moyens d’atténuer ces problèmes par exemple l’utilisation **de segments de mémoire partagée** (IPC) pour mettre des données en commun. Il est cependant apparu utile de définir un mécanisme permettant d’avoir plusieurs fils d’exécution (threads) dans un même espace de ressources non dupliqué : c’est ce qu’on appelle les processus légers. Ces processus légers peuvent se voir affecter des priorités. Leur création est de 10 à 100 fois plus rapide qu’un processus lourd.

Ces processus légers ayant vocation à communiquer entre eux, la norme Posix (norme spécifique aux systèmes d’exploitation) 1003.1c définit également des mécanismes de synchronisation : exclusion mutuelle (mutex : ) et sémaphores.

**C. Le mécanisme des threads**



Chaque processus « lourd » peut créer un ou plusieurs « threads » en fonction de ses besoins.

Ces threads sont alors associés au processus et partagent le même espace de travail. Ils peuvent donc manipuler des données communes moyennant des outils de synchronisation et s’exécuter en « simultané ».

La création de « thread » est beaucoup moins coûteuse en ressources que pour un processus lourd.

Les processus légers sont indépendants et possèdent leur propre espace de code et de pile. Par contre, ils peuvent partager certaines ressources comme l’espace mémoire ou les descripteurs de fichiers.

Le schéma ci-contre montre également que les threads présentent les mêmes caractéristiques que les processus lourds (pile, registres) exceptés qu’ils partagent certaines données entre eux et le processus lourd qui les a créés.

**D. Caractéristiques d’un thread**

Un thread dispose des caractéristiques suivantes :

* Un numéro d'identification (thread ID), unique et affecté à la création du processus léger.
* Des registres : pointeur de pile, pointeur d'instruction (compteur ordinal)…
* Un masque de signaux (interruptions) permettant de spécifier quels sont les signaux à intercepter.
* Une priorité utilisée au moment de déterminer quel processus léger peut s'exécuter.
* Des données privées dont l'accès ne peut être réalisé qu'à l'aide d'une clé.

**E. Le multithreading**

L’exécution de plusieurs « threads » au sein d’un processus lourd est 05580appelée le « multithreading ». L’avantage d’utiliser les processus légers est de créer un certain « parallélisme » entre les différentes tâches que réalise un programme.

On distingue :

* **Le parallélisme virtuel** (concurrence): Terme employé pour désigner un système monoprocesseur qui n’exécute qu’une seul processus/thread à la fois, donnant l’illusion de parallélisme
* **Le parallélisme réel** : Terme employé pour définir un système multiprocesseur où le parallélisme est une réalité. Plusieurs processus peuvent alors s’exécuter en parallèle



### Synchronisation et communication inter- processus

Lorsque plusieurs processus doivent accéder à un objet commun, celui-ci constitue une ressource critique. Ils sont dits en exclusion mutuelle pour cette ressource lorsqu'ils doivent en prendre l'usage de façon exclusive.

L'exemple le plus évident est l'usage du processeur, mais cela se produit également dans bien d'autres cas, par exemple lors d'entrées-sorties pour accéder à un périphérique.

Deux processus qui coopèrent peuvent également se trouver en exclusion mutuelle pour une ressource interne commune : imaginons un mécanisme de transfert où un premier processus écrit dans un tampon qu'un deuxième lit.

Ils ne doivent pas y accéder simultanément : cette ressource est donc critique. Les parties de code qui correspondent à l'accès au tampon sont dites des zones critiques et doivent être étudiées avec soin pour éviter tout problème.

Exemple simple

*Réservation :*

*Si nbplaces > 0*

*Alors*

*Réserver une place ;*

*Nbplace = nbplace -1 ;*

*finsi*

Ce programme peut être exécuté par plusieurs processus Client\_1 et Client\_2. Imaginons qu’on démarre Client\_1 et qu’il y a commutation juste après le test, client 2 se lance et effectue à son tour le teste et effectue la réservation. Ensuite Client\_1 reprend la main on se retrouve alors avec nbplace négatif.

Donc pour résoudre ce Pb il faut mettre en place un mécanisme de synchronisation des processus : une exclusion mutuelle qui se manifeste par la délimitation d’une zone ou section critique.

Il faudra donc un préalable en entrant dans cette section et un postule en sortant de celle-ci :

Prélude = une protection pour empêcher les autres d’accéder à la ressource critique

Postule = enlever la protection pour permettre à un autre processus d’accéder.

### Le sémaphore

Le **sémaphore** est un des moyens qu’offre le système d’exploitation pour effectuer des exclusions mutuelles. (un moyen de synchronisation, un moyen de gérer des ressources critiques)

C’est une structure système composé d’une file d’attente d’un compteur (val). Cette structure peut être manipulée que par 3 opérations :

P(sem), V(sem), *init*(sem,val). Un sémaphore peut être comparé à un distributeur de jetons dont le nombre est fixé par l’opération *init*.

* *Opération* ***P*** *= attribuer un jeton*
* *Opération* ***V****= rendre un jeton*

Donc si on utilise le sémaphore pour résoudre le Pb de l’exemple :

*Init (mutex,1) ; // création du sémaphore avec 1 jeton*

*// Réservation :*

*P(mutex) ; // prélude de section critique , instruction bloquante*

*Si nbplaces > 0*

*Alors*

*Réserver une place ;*

*Nbplace = nbplace -1 ;*

*finsi*

*V(mutex) ; //postule de section critique, on rend le jeton*