TD 2 : processus et ordonnancement

Université Paris 7 — L3MI & EIDD — Systèmes et Réseaux

1 Création de processus par fork()

Exercice 1 : forks

Que fait le programme suivant ? En particulier, combien de processus crée-t-il ? Dessiner l'arbre de paternité des processus.

```
int main()
{
  int i = 1 ;
  while (fork() == 0 && i <= 10)
      i++;
  printf("%d\n",i);
  return 0;
}
```

Exercice 2 : fork again

1. Et celui-ci, que fait-il ? Combien de processus crée-t-il ? Dessiner l'arbre de paternité des processus.
2. Que se passe-t-il si l'on remplace le premier `fork()` par `execvp(argv[0],argv)` ?
3. Et le deuxième ?
4. Et le troisième ?

```
int main()
{
  fork();
  fork();
  fork();
  return 0;
}
```

Exercice 3. Des fork partout ! (examen 2007)

Commentez l'execution des trois programmes suivants. Vous insisterez sur l'utilisation des ressources du système.

```
// Programme 1
void main( ){
  while(1)
    fork( );
}

// Programme 2
void main( ){
while ( fork ( ) == 0 )
        ; // instruction vide
}
```

```
// Programme 3
void main( ){
  int i;
  for(i = 0 ; i < 100 ; i++)
      fork( );
  sleep(1); // patienter une seconde
  printf("*");
}
```

2 Ordonnancement

Exercice 4 : Ordonnancement *batch* (temps différé non préemptif)

Il s'agit de faire du *batch sheduling* : sur un ordinateur donné (mono-processeur) vous avez une série de tâches à effectuer, dont la durée est connue à l'avance. Il s'agit d'écrire un ordonnancement, c'est-à-dire de déterminer quelle tâche le processeur effectue à un moment donné.

Les tâches sont numérotés de 1 à k. La durée de la tâche i est D_i. Quand la tâche est restée D_i secondes sur le processeur, elle est *terminée*, ce qui défini sa *date de terminaison* F_i.

Des utilisateurs (différents) attendent le résultats de leur calculs. Ils sont plus ou moins pressés. On définit donc pour chaque tâche une priorité p_i (entier naturel, une valeur élevée exprime l'urgence).

Pour quantifier l'attente des utilisateurs selon l'ordonnancement, on définit la *pénalité* d'un ordonnancement comme étant

$$\sum p_i F_i$$

Il s'agit d'écrire un algorithme qui minimise la pénalité.

1. Trouver le meilleur ordonnancement pour trois tâches de durées respectives 2, 5, 8 et de priorité respectives 1, 3, 2.
2. Soient deux tâches a et b telles que $\frac{p_a}{D_a} < \frac{p_b}{D_b}$. Si on ordonnance les deux tâches *consécutivement*, quelle doit être la première ? C'est-à-dire, comment évolue la pénalité de l'ordonnancement quand on échange a et b ?
3. En déduire un algorithme

Exercice 5. Ordonnancement préemptif (examen 2007)

On considère un système d'exploitation où les tâches suivantes arrivent au cours du temps.

Tous les temps (durée de la tâche et date de lancement) sont exprimés en seconde. La durée est le temps processeur (exprimé en secondes) dont a besoin le processus pour se terminer, et la date d'arrivée (exprimée en secondes) représente le moment ou l'utilisateur lance le processus.

Nom du processus	Durée	Date d'arrivée	Priorité
A	9	0	1
B	6	2	3
C	5	4	2
D	2	5	1
E	2	8	2

Question 1 :

Pour chacun des algorithmes d'ordonnancement suivants, décrivez l'ordonnancement (quel processus sur le processeur à quelle date). Pour les algorithmes préemptifs, on suppose que le processus actif peut être changé toutes les secondes (un processus reste donc un nombre entier de secondes sur le processeur).

- **PEPS** (Premier Entré Premier Sorti) synonyme de **FIFO** (First In, First Out)
- **PCTER** (Plus Court Temps d'Execution Restant)
- **Round Robin** (tourniquet)
- **Priority** : le processus de plus forte priorité est ordonnancé (ordonnancement préemptif)

Si à un instant donné plusieurs choix sont possibles, faites celui que vous voulez.

Question 2 :

Quels lourds problèmes soulève la mise en œuvre, dans un système d'exploitation multitâches, des algorithmes d'ordonnancements PEPS et PCTER ?