Ressource R3.05 - TD 3

Processus

Ordonnancement

1. On s'intéresse à l'exécution (en temps partagé) de trois tâches T_1, T_2 et T_3 :
   - La tâche T_1 dure 200ms (hors E/S) et réalise une unique E/S au bout de 110ms ;
   - La tâche T_2 dure 50ms et réalise une unique E/S au bout de 5ms ;
   - La tâche T_3 dure 120ms sans faire d'E/S.

   Les tâches T_1 et T_2 sont créées à l'instant 0, la tâche T3 est créée à l'instant 140. Chaque E/S dure 10ms. La valeur du quantum est de 100ms. Représentez sur un diagramme de temps l'exécution des 3 tâches.

2. On considère une variante de l'algorithme d'ordonnancement circulaire ou tourniquet. Les tâches sont rangées dans une file unique. Le processeur est donné à la première tâche prête de la file. La tâche perd le processeur en cas d'E/S ou quand elle a épuisé le quantum de temps. Elle est alors mise en fin de la file d'attente des tâches. Si une tâche arrive au début de la file dans l'état bloqué (attente d'une E/S), elle reste en début de file et on parcourt la file pour trouver une tâche prête. Toute nouvelle tâche est mise à la fin de la file.

   On considère les tâches suivantes :

Tâche	Temps CPU	E/S	Durée E/S
T_1	300ms	aucune	
T_2	30 ms	toutes les 10ms	250ms
T_3	200ms	aucune	
T_4	40ms	toutes les 20ms	180ms

   On considérera un quantum de 100ms, et on supposera que les tâches sont initialement toutes prêtes et rangées dans l'ordre de leur numéro et que les E/S des tâches T2 et T4 se font sur des disques différents.

   Décrire précisément l'évolution du système : instant, nature de l'événement (commutation, demande d'E/S, fin E/S, etc.), état de la file et la tâche élue. Donner pour chacune des tâches l'instant ou elle se termine.

Cycle de vie

1. On considère le programme suivant :

```c
int main(int argc, char * argv[]) {
  int a,e;
  a=10 ;
  if (fork() == 0){
    a=a*2;
    if (fork() == 0) {
      a=a+1;
      exit(2);
    }
    printf("%d\n",a);
    exit(1);
  }
  wait(&e);
  printf("a :%d, e :%d\n", a, WEXITSTATUS(e));
  return (0);
}
```

   (a) Donnez le nombre de processus créés, ainsi que les affichages effectués par chacun.
   (b) On supprime l'instruction `exit(2)`, reprenez la question précédente.
   (c) Modifiez le programme initial pour créer un processus zombie pendant 30 secondes.

2. (a) Combien de processus le programme engendre-t-il ?
   (b) Affecter des PID aux processus et dire ce qu'affiche le programme (Utiliser un graphe).

```c
int main(void){
  pid_t tab[4]={-1,-1,-1,-1};;
  int i;
  for (i=0;i<2;i++) tab[i]=fork();
  printf("Process: %d -- Parent: %d --",getpid(),getppid());
  printf("tab-->(%d,%d,%d,%d)\n",tab[0],tab[1],tab[2],tab[3]);
}
```

3. Dessinez le graphe de processus correspondant à l'exécution du programe suivant (en admettant que chaque appel `fork()` réussisse)

```c
#include <unistd.h>
int main(void){
fork() && (fork() || fork());
while(1);
}
```