Université de Technologie de Compiègne Examen Médian SR02, Printemps 2018

Réponses aux questions

Partie I

1) Les problèmes de gestion des entrées/sorties, gestion de la mémoire, et gestion des processus, etc. se poseraient.

2) a) A l'état bloqué, on ne sait pas si la cause du blocage a été entièrement traitée par le SE. Si entièrement traitée, on ne sait pas si le processeur est libre et le cas échéant, si le processus a la priorité d'exécution.

b) Le CO indique l'adresse de la prochaine instruction à exécuter. Le PSW indique l'état ou une image du processeur.

3) Un système monoprogrammation sera généralement plus facile à implémenter qu'un système multiprogrammation

4) Interruption : évènement extérieur provenant généralement d'une E/S (interruption matérielle). Déroutement : évènement interne lié à une exception provoquée par le programme en cours d'exécution. Appel au superviseur : évènement interne à un processus demandant au SE la réalisation d'une tâche en mode superviseur.

Note : Tous les trois entrainent des commutations de contexte.

5) a) Le DMA est en concurrence avec le processeur sur le bus de données.

b) Le processeur via le pilote du périphérique initialise le DMA qui lance l'E/S. Le DMA signale la fin d'entrée/sortie par le déclenchement d'une interruption que le processeur traite en exécutant une routine d'interruption.

6) Cela facilite la gestion modulaire des périphériques par le SE. Au niveau physique le contrôleur de périphérique assure l'interopérabilité matérielle. Le niveau logique est la couche de programmation utilisant des primitives de la couche physique.

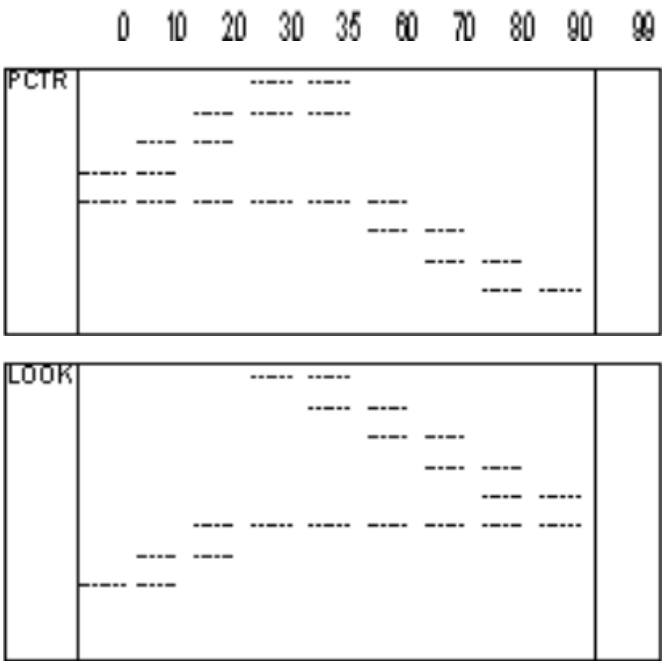
7) Capacité maximale théorique = nombre de faces*nombre de pistes par faces*nombre de secteur par piste*taille secteur.

Taille maximale d'un fichier = (10 + 256 + 256*256)*4 Ko

8) Interruptions, DMA, etc.

9)

a) PCTR : 130
b) LOOK : 150



10)

a) faux (c'est possible mais cela peut être complexe)
b) faux (ca dépend du contexte d'utilisation)
c) faux (ca dépend du masque d'interruption et la priorité)
d) faux (taille maximale fichier = 10 + 2^8 + 2^{16} + 2^{24} Ko > 2^4 * 2^{20} Ko)

Partie II

1)

(i) *désactiver toutes les interruptions* : mode noyau, bien que ce soit un moyen basique de faire de la synchronisation, si un utilisateur le fait mal, alors le système d'exploitation cessera de fonctionner et n'aura aucun moyen de sortir.

(ii) *lire l'horloge du jour* : mode utilisateur, il s'agit d'une opération en lecture seule, et un utilisateur ne peut pas faire de mal. Cependant, certains systèmes d'exploitation le conservent dans le noyau

(iii) *régler l'horloge du jour* : mode noyau, si l'utilisateur a changé le temps, les autres utilisateurs seraient affectés négativement

(iv) *tuer un processus* : mode noyau, les utilisateurs ne peuvent pas tuer les processus arbitrairement. Ils peuvent tuer leurs propres processus, mais le noyau doit vérifier que le paramètre appartient effectivement à l'utilisateur appelant.

2) Un signal est généré par le processeur ou un processus. Il est envoyé à un processus. Une interruption est générée par une dispositif d'E/S. Elle est traitée par le processeur.
3) Si un processus attend une opération qui se produira bientôt, cela ne vaut pas la peine de faire le changement de contexte impliqué dans le blocage d'une interruption. Généralement, lorsque le temps d'attente est inférieur au temps système pour le changement de contexte, il est préférable d'interroger.

4) La séquence de Fibonacci est la série de nombres 0, 1, 1, 2, 3, 5, 8, ... Formellement, elle peut être exprimée comme suit:

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n) = fib(n\text{-}1) + fib(n-2),\ n>1$$

a)

```
#include <stdio.h>
#include <stdlib.h>
  int main(int argc,char *argv[])
        {
           int k, pid,i=0,j=0;
           int sum=1;
           int num=atoi(argv[1]);
          if (argc == 0)
           {
              printf ("Entrer un nombre à la séquence de fibonacci ");
           }

           pid = fork();
           if (pid == 0)
           {
              for(k=1;k<num;k++)
              {
                    i = j;
                    j= sum;
                    sum = i + j;
                    printf("Fibonacci (%d) = %d\n",k,sum);
              }
           }
           else
          {
            wait();
            printf ("Le parent a attendu son fils \n");
          }
           return 0;
        }
```

b) Le fils passe à l'état Zombie

c) En utilisant la communication interprocessus : pipe ou segment de mémoire partagé ou file de message ou socket.

5)

a) 256 couleurs ==> 8 bits par point ==> 1 octet par point

b) Les processus doivent partager leurs PID. *struct shared_data* permet de les partager. Contre-exemple : Si on utilise des variables globales, il se peut que ces variables ne soient pas initialisées avant un fork, donc le premier processus créé ne saura pas le PID de chacun des deux autres processus créés après sa création.

c) (4) shmid=shmget(IPC_PRIVATE,sizeof(struct shared_data),IPC_CREAT|0666);
(5) shared_d=(struct shared_data*)shmat(shmid,0,0);

d) (6) sigact_lecteur.sa_handler=handler_lecteur;
sigaction(SIGUSR1,&sigact_lecteur,NULL);

(7) sigact_rotation.sa_handler=handler_rotation;
sigaction(SIGUSR1, &sigact_rotation, NULL);

(8) sigact_redacteur.sa_handler=handler_redacteur;
sigaction(SIGUSR1, &sigact_redacteur, NULL);

e)

```
void handler_lecteur(int x){
        int i,j;
image=mmap(0,size*size,PROT_READ|PROT_WRITE,MAP_SHARED,shared_d->fd,0);
        for(i=0;i<size;i++){
                for(j=0;j<size;j++)
                        printf("%c",*(image+i*size+j));
                printf("\n");
        }
        kill(shared_d->p_rotation, SIGUSR1);
}

void handler_rotation(int x){
        struct iovec vec[size];
        int i,j;
        close(shared_d->piped[0]);
image=mmap(0,size*size,PROT_READ|PROT_WRITE,MAP_SHARED,shared_d->fd,0);
        for(j=0;j<size;j++){
                for(i=0;i<size;i++){
                        vec[i].iov_base=image+i*size+j;
                        vec[i].iov_len=1;
                }
                writev(shared_d->piped[1],vec,size);
        }
        close(shared_d->piped[1]);
        kill(shared_d->p_redacteur,SIGUSR1);
}
void handler_redacteur(int x){
        struct iovec vec[size];
        int i;
        close(shared_d->piped[1]);
```

```
        image=mmap(0,size*size,PROT_READ|PROT_WRITE,MAP_SHARED,shared_d->fd,0);

                for(i=0;i<size;i++){
                        vec[i].iov_base=image+i*size;
                        vec[i].iov_len=size;
                }
                readv(shared_d->piped[0],vec,size);
                close(shared_d->piped[0]);
        }
```