Nom : …… Prénom (s) ……

Université de Technologie de Compiègne Examen Médian SR02, Printemps 2019

Introduction

- Seuls les documents de cours sous format papier sont autorisés
- Répondrez sur le sujet de l'examen
- Les réponses doivent être **claires** et **concises** : cela sera pris en compte lors de l'évaluation des réponses
- La durée de l'examen est 120 minutes.

Partie 1 (10 points) --- Copie 1

1) Citer deux inconvénients de la multiprogrammation et deux avantages des systèmes à temps partagé. (**1 point**)

- Inconvénients de la multiprogrammation : Gestion de la mémoire, le prob de sécurité des processus, il faut un ordonnanceur … ect.
- Avantages des systèmes à temps partagé : Utilisation efficace du CPU, Temps de traitement petit pour les programmes courts.

2) L'accès direct à la mémoire est utilisé pour les périphériques d'E/S à grande vitesse afin d'éviter la surcharge du processeur.
   a) Comment le processeur communique-t-il avec le périphérique pour coordonner le transfert ? (**0,75 point)**

La CPU peut lancer une opération DMA en écrivant des valeurs dans des registres spéciaux auxquels le périphérique peut accéder indépendamment.

   b) Comment le processeur sait-il que les opérations d'E/S sont terminées? (**0,75 point)**

Lorsque le périphérique a terminé son opération, il interrompt la CPU pour indiquer la fin de l'opération.

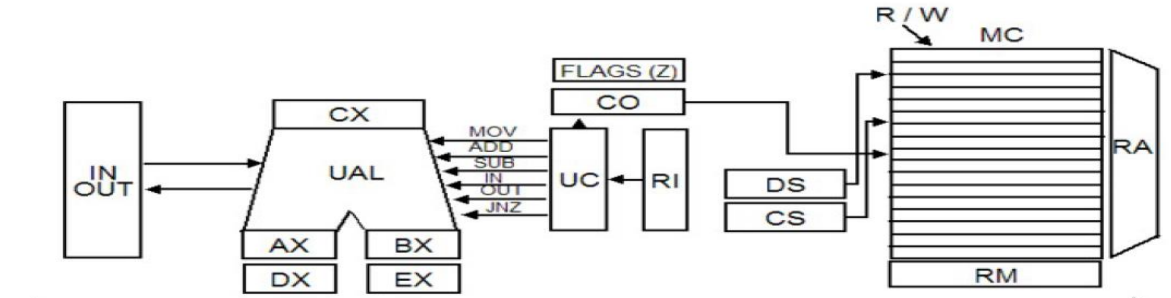
3) Un processeur ne traite que les interruptions et il a une capacité de lire ou écrire un mot mémoire de 4 octets en *10ns*. Lors d'une interruption il faut recopier 32 registres dans la pile du processus. Quel est le nombre maximal d'interruptions que ce processeur peut traiter en *2ms* ? (**1,5 points)**

Le temps d'une interruption est 2x32x10ns= 640 ns. Le nombre d'interruptions qu'il peut traiter en 2ms est 2000000/640 =31225.

4) Que pouvez-vous dire d'un système dans lequel lorsqu'un processus *P* crée un processus fils, l'exécution du processus *P* est suspendu jusqu'à la terminaison du processus fils ? (**1 point)**

Système mono-programmé

5) Soit l'architecture suivante :



CX: registre d'accueil du résultat de l'opération UAL
AX,BX,DX, EX: registres généraux
RI: registre instruction en cours ([CS+CO])
CO: compteur ordinal. Initialisé à 0, s'incrémente de 1 après chaque instruction
DS: registre pointant sur le début du segment data. Initialisé par le chargeur.
CS: registre pointant sur le début du segment code. Initialisé par le chargeur.
UC: unité de contrôl
FLAGS: registre des flags. Z: bit mis à zero quand résultat CX=0
RM: registre mémoire
IN/OUT: lecture dans DX / écriture contenu de DX
MOV R1, R2: R1:=R2
MOV R1, @x: R1:=[DS+@x]
MOV @x, R1: [DS+@x]:=R1
ADD X,Y: CX:=X+Y
SUB X,Y: CX:=X-Y
JNZ etiq: allez à CS+etiq si flag Z<>0

a) Dans cette architecture, peut-on exécuter un programme sans stocker ses données en mémoire centrale (MC) ? (**0,5 point)**

Oui : on peut utiliser les registres si le nombre de données est petit.

b) Dans cette architecture, peut-on exécuter un programme sans stocker ses instructions en mémoire centrale (MC) ? (**0,5 point)**

Non. Le processus est installé dans la MC.

c) Citer deux types d'entité de cette architecture qui peuvent envoyer un signal (**0,5 point)**

Processus, processeur.

d) Citer un exemple d'entité de cette architecture qui peut générer une interruption (**0,5 point)**

E/S

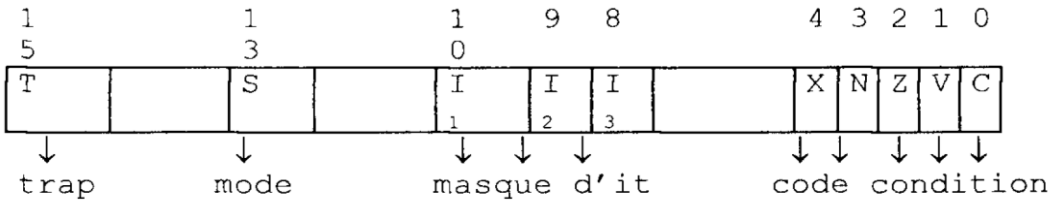
a) Citer les avantages et les inconvénients de cette architecture (**0,5 point)**

Il s'agit d'un type de l'architecture de Von Neumann. L'inconvénient est les goulots d'étranglements.

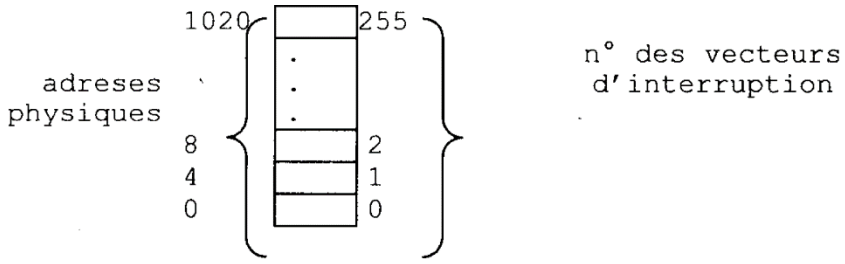
b) Proposer une autre architecture qui répond aux inconvénients de cette architecture et discuter les caractéristiques de cette nouvelle architecture ? (**1 point)**

Architecture de Harvard.

6) Le contexte d'un programme s'exécutant sur une machine X est constitué de 16 registres généraux (R0 à R15), du compteur ordinal (CO) et du PSW dont la structure est la suivante :



Cette machine reconnaît 256 causes d'interruption (it) dont les vecteurs sont stockés dans la table suivante:



Le vecteur d'interruption contient le compteur ordinal et la routine d'interuption correspondante. Parmi ces 256 causes d'interruption huit causes sont dues à des organes externes. Chacune de ces huit causes possède un numéro d'interruption qui correspond à sa priorité (de 0 à 7). Une interruption de ce type arrive de la manière suivante :

i. L'unité centrale ne prend en considération un signal d'interruption que si le numéro de celle-ci est supérieur à la valeur du masque d'it qui se trouve dans le PSW
ii. L'unité centrale envoie un signal d'acquittement au composant qui a déclenché l'interruption
iii. Le composant envoie alors le numéro de son vecteur d'it
iv. L'unité centrale récupère ce numéro et empile dans la pile système le CO et le PSW du programme courant
v. L'unité centrale positionne le bit S du PSW à 1 et le masque d'it à la valeur correspondante au numéro d'it récupéré en I
vi. L'unité centrale se branche sur la routine d'it correspondante

La machine dispose de l'instruction RTI qui charge le PSW et le CO à partir des deux mots de tête de la pile système.

a) Comment le système se branche vers la routine d'it *i*, $0 \leq i \leq 255$ (**0,5 point)**

Co= i x 4;

b) Donner le corps d'une routine d'it quelconque (**0,5 point)**

SR0=R0, …… SR15=R15 ;
Traitement ;
R0=SR0, …… R15=SR15;
RTI;

c) Donner les valeurs du masque d'it du PSW pour les 8 causes d'it (**0,5 point)**

000
001
010
011
101
110
100
111

Correction de la partie 2 du median SR02-P2019

1 Solution Exercice 1 (3.5 points)

1.1 Question 1

Donner les bouts de code (3) et (4) nécessaires permettant d'installer des nouveaux handlers pour le traitement d'un signal SIGUSR1. **(1 point)**

1.2 Solution 1

```
struct sigaction old, new;
new.sa_handler=handler_pere;
sigaction(SIGUSR1,&new,&old);
```

Listing 1: installation du handler pour le père

```
struct sigaction new_fils;
new_fils.sa_handler = handler_fils;
sigaction(SIGUSR1,&new_fils,NULL);
```

Listing 2: installation du handler pour le fils

1.3 Question 2

Donner les bouts de code (5) et (6) nécessaires pour que les deux processus puissent synchroniser l'exécution tel qu'il est décrit dans l'énoncé. **(1 point)**

1.4 Solution 2

```
while(1){ // ........... (5)
  fibonacci(&n, &t1, &t2);
  kill(getppid(), SIGUSR1); // envoi du signal SIGUSR1 au parent.
  pause(); // se mettre en pause
}
```

Listing 3: Code du processus fils

```
while(1){ // .......... (6)
  pause();
}
```

Listing 4: Code du processus père

1.5 Question 3

Donner les bouts de codes (1) et (2) correspondants aux fonctions handler_pere() et handler_fils() qui serviront des handlers pour le traitement du signal SIGUR1. **(1.5 point)**

1.6 Solution 3

```
1  void handler_fils(int sig){
     nb_sig ++;
3    if (nb_sig == MAX_SIGUSR1){
       kill(getppid(), SIGINT);
5      exit(0);
     }
7  }

9  void handler_pere(int sig){
     sleep(5);
11   kill(pid_fils,SIGUSR1);
   }
```

Listing 5: Exercice 3

2 Solution Exercice 2 (6.5 points)

2.1 Question 1

Expliquer pourquoi il faut partager le pid du processus p2 dans une mémoire partagée **(0.5 points)**

2.2 Solution 1

Le processus p1 a besoin du PID du processus p2 pour lui envoyer un signal. Le seul moyen pour que p1 connaisse le PID de p2 c'est d'utiliser une mémoire partagée pour stocker les PID de p2.

2.3 Question 2

Ecrire la fonction initab(int n, int fd) qui permet d'initialiser aléatoirement un tableau de n entiers et copier son contenu dans le fichier représenté par le descripteur fd. **(0.5 points)**

2.4 Solution 2

```
void initab(int n, int fd){
  srand(time(NULL));
  int i;
  for(i=0; i<n; i++){
    int a = rand()%30+10;
    write(fd,&a,sizeof(a));
  }
}
```

Listing 6: initab()

2.5 Question 3

Ecrire la fonction showtab(int n, int * tab) qui permet d'afficher le contenu du tableau tab (projeté en mémoire par la primitive mmap). **(0.5 points)**

2.6 Solution 3

```
void showtab(int n, int *tab){
  int i;
  for(i=0; i<n; i++){
    printf("\ntab[%d]=%d",i, tab[i]); fflush(stdout);
  }
}
```

Listing 7: showtab()

2.7 Question 4

Ecrire les routines de traitement du signal SIGUSR1 handlers_pere(), handler_fils1(), et handlers_fils2() liées aux processus parent, p1 et p2 respectivement. **(2.5 points)**

2.8 Solution 4

```
void handler_fils1(int sig){
  showtab(n, tab);
  int i;
  for(i=0; i<n; i++){
    if (!(tab[i]%2)) tab[i]+= 1;
  }
  kill(pids[1],SIGUSR1);
}
```

```c
/* fonction utilisateur de comparaison fournie a qsort() */
static int compare (void const *a, void const *b)
{
    int const *pa = a;
    int const *pb = b;
    return *pa - *pb;
}

void handler_fils2(int sig){
  showtab(n, tab);
  qsort (tab, n, sizeof(int), compare);
  kill(getppid(),SIGUSR1);
}

void handler_pere(int sig){
  showtab(n, tab);
  shmdt(pids); // detacher le segment memoire du processus parent
  shmctl(shmid, IPC_RMID, NULL);  // supprimer le segment memoire
  munmap((void *) tab, n*sizeof(int) );
}
```

Listing 8: Les handlers

2.9 Question 5

Ecrire le programme principal permettant de créer les deux processus p1 et p2 et réaliser l'objectif décrit ci-dessus, en faisant appel aux différentes fonctions : initab, showtab, etc. **(2.5 points)**

2.10 Solution 5

```c
// Les variables globales
int *tab, *pids, n, fd, shmid;

int main(int argc, char* argv[]){
  if(argc < 3) {perror("\nUsage: <filepath> <n>");exit(-1);}
  struct sigaction new;
  new.sa_handler=handler_pere;
  sigaction(SIGUSR1,&new,NULL);
  n = atoi(argv[2]);
  fd = open(argv[1], O_RDWR|O_CREAT, 0666);
  initab(n, fd);
  tab = (int *)mmap(NULL, n*sizeof(int), PROT_READ|PROT_WRITE,
   MAP_SHARED, fd, 0);
  if(tab == (int*)MAP_FAILED){
    perror("Erreur mmap!!"); exit(1);
  }
  close(fd);
```

```
17    showtab(n,tab);
      int p;
19    if ((shmid = shmget(IPC_PRIVATE, 2*sizeof(int), PERM)) == -1) {
        perror("Echec de creation du segment");
21    return 1;
      }
23    if ((pids = (int *)shmat(shmid, NULL, 0)) == (void *)-1) {
        perror("Echec de l'attachement du segment");
25    }
      if((p=fork())==-1){ perror("\n(parent) Echec de creaction du
       processus p1"); return 1;}
27    if(p > 0){ //parent
        pids[0] = p; // sauvegarder le pid du fils 1 dans la memoire
       partagee.
29      if((p=fork())==-1){ perror("\nEchec de creaction du processus p2"
       ); return 1;}
        if(!p){ //fils 2
31        struct sigaction new;
          new.sa_handler=handler_fils2;
33        sigaction(SIGUSR1,&new,NULL);
          sleep(3);
35        pause();
          exit(0);
37      }
        pids[1]=p;
39      sleep(1);
        kill(pids[0], SIGUSR1);
41      int i;
        pause();
43    }
      else{ //fils 1
45      struct sigaction new;
        new.sa_handler=handler_fils1;
47      sigaction(SIGUSR1,&new,NULL);
        sleep(3);
49      pause();
        exit(0);
51    }
      return 0;
53  }
```

Listing 9: Programme principal