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**)

..……..
.....
.....
.....
.....

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)**

..……..
.....
.....

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

..……..
.....
.....

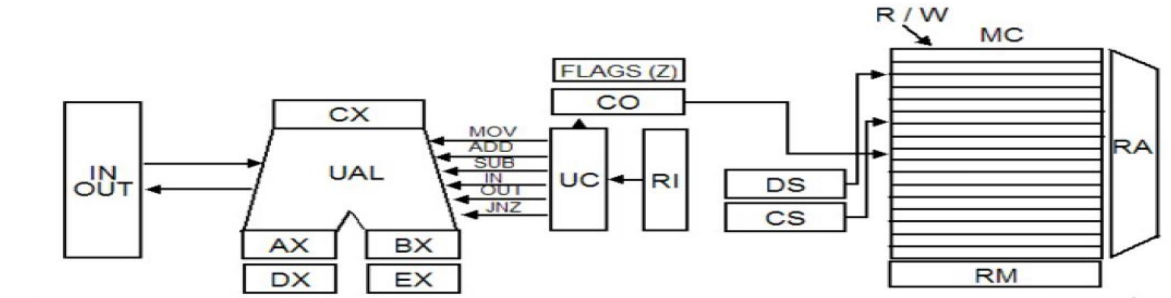
3) Un processeur ne traite que les interruptions et il a une capacité de lire ou écrire un mot mémoire de 4 octets en *10ns*. Lors d'une interruption il faut recopier 32 registres dans la pile du processus. Quel est le nombre maximal d'interruptions que ce processeur peut traiter en *2ms* ? (**1,5 points)**

..……..
.....
.....

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)**

…….. ……

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)**

…….. …… ……

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

…….. …… ……

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

…….. ……

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

……..

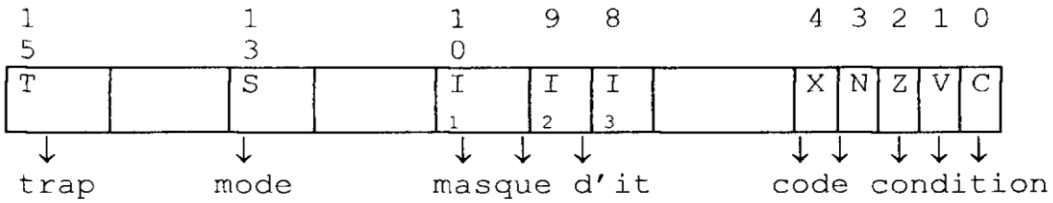
a) Citer les avantages et les inconvénients de cette architecture (**0,5 point)**

.....
.....
.....
.....

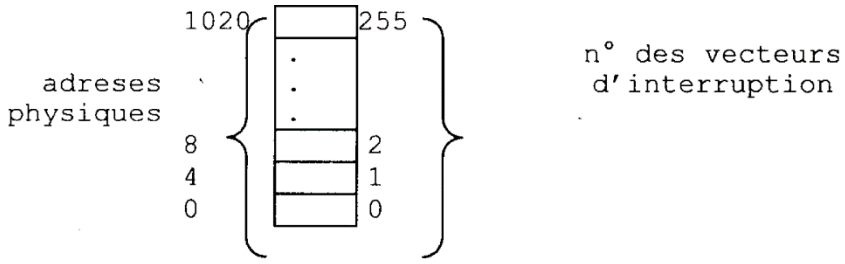
b) Proposer une autre architecture qui répond aux inconvénients de cette architecture et discuter les caractéristiques de cette nouvelle architecture ? (**1 point)**

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

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)**

.....
.....

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

.....
.....
.....
.....
.....

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

.....
.....
.....
.....
.....

Nom : …… Prénom (s) ……

Partie 2 (10 points) --- Copie 2

Exercice 1 (3.5 points) : les signaux

La suite de Fibonacci est définie par :

$$u_n = \begin{cases} 1 & n = 0,1 \\ u_{n-1} + u_{n-2}\,, & n > 1 \end{cases}$$

On souhaite compléter le code du programme donné ci-dessous. Ce programme permet de lancer deux processus qui bouclent à l'infini et ayant les fonctionnalités suivantes :

- A la réception d'un signal **SIGUSR1**, le processus fils calcule *m (m* est initialisé à 5) termes de la suite de Fibonacci et affiche le dernier terme calculé, puis il envoie le signal **SIGUSR1** au processus père avant de se mettre en pause.
- A la réception du signal **SIGUSR1**, le père fait une attente de 5 secondes et donne, à nouveau, la main fils, en émettant un signal **SIGUSR1** pour calculer les *m* termes suivants de Fibonacci.
- Pendant que le processus fils calcule des termes de la suite, le processus père se met en pause.
- Après la réception de 10 signaux de type **SIGUSR1**, le processus fils suspend l'exécution de son père en lui envoyant un signal **SIGINT,** et il se termine.

```
#include "bibio.h"
#define MAX_SIGUSR1 10
#define m  5
pid_t pid_fils;
int nb_sig = 0; // nombre de signaux reçus
 /* f1 : l'avant dernier Fibonacci déjà calculé  u_{n-2}
 f2 : le dernier terme Fibonacci déjà calculé  u_{n-1}
 n: dernier entier pour lequel on a calculé le terme Fibonacci  */
void fibonacci_m_termes(int* n, int *f1, int *f2){
        int i;
        for(i=0; i<m; i++){
                int tmp = *f2;
                *f2 = *f2 + *f1;
                *f1 = tmp;
                (*n) ++ ;
        }
        printf("\nfib(%d)=%d",n,*f2); fflush(stdout);
}
void handler_fils(int sig){ // routine de traitement du signal SIGUSR1 par le processus fils
        /*…… (1) ……..*/
        ……
        ……
        ……
        ……
        ……
        ……
        ……
}
```

```
void handler_pere(int sig){ // routine de traitement du signal SIGUSR1 par le processus père
        /*…… (2) ……*/
        ……
        ……
        ……
}
int main(){
        /*…….. (3) Installation du nouveau handler (handler_pere())…….*/
        ……
        ……
        ……
        pid_fils = fork();
        if(pid_fils==0){
                sleep(1);
                /*..... (4) Installation du nouveau handler (handler_fils()) .....*/
                ……
                ……
                ……
                long int f1 = 1; // terme u0 de Fibonacci
                long int f2 = 1; // terme u1 de Fibonacci
                int n = 1; // le dernier terme calculé est u1
                while(1){ /*…… (5) ……. */
                        ……
                        ……
                        ……
                        ……
                        ……
                        ……
                        ……
                }
                exit(0);
        }
        while(1){
                /*…… (6) ……*/
                ……
                ……
        }
        return EXIT_SUCCESS;
}
```

Questions :

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)**
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)**

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

Exercice 2 (6.5 points) : les mémoires partagées et les signaux

Nous souhaitons développer un programme qui permet de manipuler un tableau d'entier, partagé en mémoire par plusieurs processus. Le tableau est initialement stocké dans un fichier binaire par le processus parent qui le projettera ensuite en mémoire en utilisant la primitive *mmap*. Le tableau projeté en mémoire sera modifié séquentiellement par deux processus *p1* et *p2* créés par le parent. À la fin de l'exécution des processus *p1* et *p2*, le parent affiche le contenu du tableau projeté en mémoire. La synchronisation entre les différents processus est réalisée avec des signaux. Ci-dessous, les descriptions détaillées des trois processus :

Le processus parent :

- Initialiser aléatoirement un fichier binaire de *n* entiers entre 0 et 100. Le nom du fichier et la taille du tableau (*n)* sont passés en paramètres dans la fonction main.
- Projeter le fichier en mémoire avec la primitive *mmap.* Le fichier est projeté dans la variable *int *tab.*
- Créer et attacher au processus parent une mémoire partagée de deux entiers (*pid_t pids[2])* pour sauvegarder les pid du processus *p1* et le processus *p2* respectivement. Cette mémoire est partagée aussi avec les processus *p1* et *p2*.
- Créer deux processus *p1* et *p2* à l'aide de la primitive *fork()*. Les deux processus se mettent initialement en pause (exécution de la primitive *pause()*).
- Envoyer un signal **SIGUSR1** au processus *p1* (dont le pid est *pids[0]*) pour le réveiller.
- Se mettre en pause.
- À la réception du signal **SIGUSR1**, afficher le contenu du tableau *tab* (trié par le processus *p2*).
- Détacher et supprimer les segments de mémoires créés par *shmget*, *shmat* et *mmap.*

Le processus *p1* :

- Se mettre en pause.
- À la réception du signal **SIGUSR1** émit par le parent, incrémenter les valeurs paires du tableau *tab*.
- Envoyer un signal **SIGUSR1** au processus *p2* (dont son pid est *pids[1]*) pour le réveiller.

Le processus *p2 :*

- Se mettre en pause.
- À la réception du signal **SIGUSR1** émit par le processus *p1*, trier le contenu du tableau *tab*.
- Envoyer un signal **SIGUSR1** au processus parent (dont son pid est obtenu par la fonction *getppid()*) pour le réveiller.

Questions :

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

```
void initab (int n, int fd) {
        ……
        ……
        ……
        ……
        ……
        ……
        ……
}
```

2. 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**)

```
void showtab (int n, int * tab) {
        ……
        ……
        ……
        ……
        ……
}
```

3. 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**)

```
void handler_pere (int sig) {
        ……
        ……
        ……
        ……
        ……
        ……
        ……
}
```

```
void handler_fils1 (int sig) {
        ……
        ……
        ……
        ……
        ……
        ……
        ……
}
```

```
/* implémenter la fonction de tri du tableau tab ici ……*/
……
……
……
……
……
……
……
……
```

```
void handler_fils2 (int sig) {
        ……
        ……
        ……
        ……
        ……
        ……
        ……
}
```

4. 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**)

```
/*Programme principale ……*/
include "median.h" /* tous les bibliothèques *.h nécessaires sont inclues dans le fichier median.h*/
/* déclarer les variables globales ici…*/
……
……
……
……
……
……
int main (int argc, char * argv[]) {
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
        ……
```

/*Suite programme principale ………………*/

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

……………………………………………………………………………………………………

}