Examen INFO604 – mai 2018

Documents Autorisés : Cours, TD, TP du module uniquement

---- partie I (5 pt) -----

Q1 (1pt) Expliquer la différence entre un système d'exploitation préemptif et un système d'exploitation non- préemptif

Q2 (1pt) Expliquer la différence entre un processus et un processus dit « léger » (thread)

Q3 (1pt) Expliquer le rôle principal des « tubes » entre processus.

Q4 (1pt) Expliquer le principe de l'exclusion mutuelle

Q5 (1pt) Expliquer le rôle des fonctions sem_wait() et sem_post()

---- partie II (5 pt) -----

Q6 (2,5 pt) Sur votre système Linux, vous disposez de différents programmes de surveillance de votre système :

- Le pgm **watchmem** (/usr/bin/watchmem) qui surveille le taux d'usage de votre mémoire et qui envoie un email lorsque le taux d'occupation dépasse un certain seuil.
  Exemple d'usage :
  $ watchmem -seuil 90 -mail toto@gmail.com → envoi un email si mémoire est pleine à 90 %

- Le pgm **watchtcp** (/opt/watchtcp) qui comptabilise les connexions TCP effectuées sur votre machine
  Exemple d'usage :
  $ watchtcp

Compléter le programme suivant afin que chaque commande (watchmem et watchtcp) soit lancée par un **processus** fils (ceci afin d'éviter que le non lancement de l'un des programmes ne puisse impacter le lancement de l'autre).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
   /* identifiant des processus */
   pid_t id1, id2;

   /* Liste arguments pour la commande "watchmem" */
   char *args[5] = { "watchmem","-seuil","90","-email", "toto@gmail.com"};

   /* **** a completer **** */

   printf("fin du processus \n");
}
```

Q7 (1 pt) Soit le programme suivant, donnez le nombre de fois que le message va s'afficher

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void main() {
    if (fork()) {
        fork();
        fork();
        printf("hello!\n");
    }
    return 0;
}
```

Q8 (1,5 pt) Le programme suivant permet de paralléliser un calcul. Compléter le programme pour que le père puisse récupérer le calcul du fils avec un tube

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void main() {
        /* identifiant des processus */
        pid_t id, id2;
        double res1, res2;

        /* création du clone */
        id= fork();

        if (id == -1 ){
                printf("erreur creation fork\n");
                exit(EXIT_FAILURE);
        } else if (id == 0) {
                printf("lancement calcul 2 \n");
                res2=calcul2();
                printf("resultat 2 = %f \n",res2);
                exit(EXIT_SUCCESS);
        }

        /* je suis le père */
        printf("lancement calcul 1 \n");
        res1=calcul1();
        printf("attente fin calcul2\n");
        wait(NULL);
        printf("resultat 1 = %f \n",res1);
}
```

---- partie III (5 pt) -----

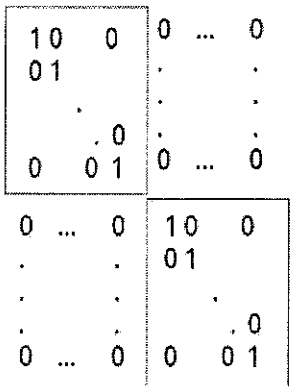
Q9 (3,5 pt) En supposant que vous disposez de la variable globale suivante

int mat[40][40] ;

et des 2 fonctions suivantes :

int estMatrcieIdentite(int c1, int l1, int c2, int l2) → indique si la sous matrice de **mat** (définie par les colonnes c1,c2 et les lignes l1,l2) est une matrice identité
int estMatriceNulle(int c1, int l1, int c2, int l2) → indique si la sous matrice de **mat** (définie par les colonnes c1,c2 et les lignes l1,l2) est une matrice nulle

Vous souhaitez paralléliser le calcul de **mat** afin de savoir s'il s'agit d'une matrice identité. Pour cela vous allez lancer un thread de calcul par zone de la matrice correspondant aux sous-matrices (0,19,0,19) (20,39,20,39) (0,19,20,39) (20,39,0,19) et afficher le résultat final dans le processus principal



Q10 (1,5 pt) On suppose que l'on généralise la démarche avec une matrice de très grande dimension. Toutefois pour connaître l'avancement au fur et à mesure des calculs, on vous propose d'ajouter une variable globale (**int avancement**) qui aura des valeurs entières de 0 à 100 et qui **sera incrémentée au fur et à mesure par chaque thread** de calcul. Expliquez ce qui ne vas pas dans cette démarche et comment vous feriez pour corriger le problème

---- partie IV (5 pt) -----

Q11 (3 pt) On veut simuler la gestion des accès à un parking souterrain contenant deux barrières, une pour entrer, et une pour sortir. On suppose que le parking peut contenir MAX voitures.
Après avoir identifié un problème classique et à l'aide des sémaphores, définissez les threads suivants :

void * gestionBarièreEntrée(void * arg)
void * gestionBarrièreSortie(void * arg)

Q12 (2 pt) A chaque ouverture de barrière, un « flag » est envoyé à un serveur qui garde les traces des ouvertures. Ce flag (une chaîne de 10 caractères) est envoyé par une socket en mode connecté. Écrire la fonction d'envoi de ce flag(connexion socket, envoi, close socket) :

int envoiFlag(adresse,port,flag) (on suppose que le serveur de socket répond correctement à vos demandes)

Le mode non connecté n'aurait-il pas été plus adéquat pour ce contexte ?