UE Programmation Unix

Contrôle Continu & Examen final

Partie 2
APPLICATION DU COURS

Gestion des Processus

CC 2019 – 4 points

Ecrire un programme `auto_exec.c` qui se recouvre N fois lui-même.
A chaque recouvrement le programme affiche :

```
./auto_exec : recouvrement i.
```

Avant de se terminer, il affiche :

```
./auto_exec : terminé
```

Par exemple : `./auto_exec 3` aura pour résultat :

```
./auto_exec recouvrement 1.
./auto_exec recouvrement 2.
./auto_exec recouvrement 3.
./auto_exec : terminé
```

```c
#include <stdio.h> // fprintf(), printf(), sprint()
#include <stdlib.h> // atoi(), exit(), EXIT_FAILURE
#include <unistd.h> // execl()
int main(int argc, char *argv[]) {
    if(argc < 2){
       fprintf(stderr, "Usage : %s N \n", argv[0]);
       exit(EXIT_FAILURE);
    }

    if( argc == 2 ) {
       execl(argv[0], argv[0], argv[1], "1", NULL);
    }else  {
       int N = atoi( argv[1] );
       int i = atoi( argv[2] );
       printf("%s recouvrement %d.\n", argv[0], i);

       if( i != N) {
          char next_i[10];
          sprintf(next_i, "%d", i + 1);
          execl(argv[0], argv[0], argv[1], next_i, NULL);
       } else {
          printf("%s : terminé \n", argv[0]);
       }
    }
    return 0;
} /* main */
```

Gestion des Signaux

CC 2019 – 3 points

Sachant que le temps d'écriture d'un octet dans un tube ordinaire est inférieur à 1 seconde, écrire un programme qui calcule et affiche la capacité maximale d'un tube ordinaire puis se termine

```c
#include <stdio.h> // printf()
#include <stdlib.h> // exit(), EXIT_SUCCESS
#include <signal.h> // sigaction(), sigemptyset(), alarm()
#include <unistd.h> // pipe(), write()
long count = 0;
void handler(int signo);
int main(int argc, char *argv[]) {
    //Structure pour la mise en place des gestionnaires
    struct sigaction action;

    /* Remplissage de la structure */
    // Adresse du gestionnaire
    action.sa_handler = handler;
    // Mise a zero du champ sa_flags théoriquement ignoré
    action.sa_flags = 0;
    /* int sigemptyset(sigset_t *set);
     * initialise a VIDE l'ensemble de signaux pointé par set
     * retourne 0 en cas de succès ou -1 en cas d'erreur
     */

    // On ne bloque pas de signaux spécifiques
    sigemptyset(&action.sa_mask);
    /* int sigaction(int sigo, const struct sigaction *act,
       struct sigaction *oldact); */
    // Mise en place du gestionnaire pour le signal SIGALRM
    sigaction(SIGALRM, &action, NULL);
    int fd[2];
    pipe(fd);
    alarm(1);
    while(1){
         if( write(fd[1], "i", 1) >= 0 ) {
           count++;
           alarm(1);
       }
    }
    return 0;
}
void handler(int signo) {
    printf("Capacite maximale du tube : %ld\n", count);
    exit(EXIT_SUCCESS);
}
```

Gestion des Threads

Examen 2016 – 6 points

Le programme `game_of_life` crée un automate cellulaire sur une grille à 2 dimensions X, Y. Chaque case de la grille est appelée *cellule*.

A chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines selon les règles suivantes :

- **R1 : Une cellule morte possédant exactement trois voisines vivantes devient vivante (elle nait).**
- **R2 : Une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.**

- **La grille sera implémentée par un tableau à deux dimensions de caractères.**
- **L'état de chaque cellule sera géré par un thread `cellule` qui appliquera la règle R1 ou R2 à sa cellule.**
- **Il y aura donc X*Y threads `cellule`.**
- **Pour une étape `i`, chaque thread `cellule` détermine le nouvel état de sa cellule et attend que la grille soit affichée par un thread d'affichage `affichage`.**
- **Lorsque le thread `affichage` a terminé l'affichage de la grille, il informe tous les autres threads `cellule` qu'il peuvent passer à l'étape `i + 1`.**
- **Ce qui précède est itéré indéfiniment.**

L'appel du programme se fera de la façon suivante :

```
game_of_life X Y x1 y1 x2 y2 … xN yN
```

ou `X Y` sont les dimensions de la grille et `xi yi` sont les coordonnée d'une cellule vivante au démarrage de l'automate.

Une cellule vivante sera représentée par le caractère `'*'`, une cellule morte par *un blanc*.

a Expliquez en quoi consiste la solution que vous allez implémenter pour la synchronisation entre les threads `cellule` lors de l'acces a la grille.

b Expliquez en quoi consiste la solution que vous allez implémenter pour la synchronisation entre les threads `cellule` et le thread `affichage`.

c Les threads `cellule` devront-ils être créés joignable ? Justifiez votre réponse.

d Ecrivez le code de `game_of_life`.