L3 INFORMATIQUE

Examen final de Programmation Unix – 4 janvier 2016 – session 1

Michel SOTO

AUCUN DOCUMENT AUTORISÉ

Durée: 1 H 30

Le barème est indicatif - Nombre de pages: 2

La concision de vos réponses et la propreté de votre copie seront prises en compte.

PARTIE I : CONNAISSANCE DU COURS

Question 1 (4 points)

Répondez aux affirmations suivantes uniquement par "VRAI", ou "FAUX" ou "NE SAIS PAS".

Barème : **réponse exacte : +1 point, réponse fausse : -0,5 point sur la copie, "ne sais pas" : 0 point**

a) Si un verrou partagé *mandatory* est posé sur zone d'un fichier aucun autre processus ne peut accéder à cette zone.

b) Avant l'exécution de la toute première instruction d'un processus, la première entrée libre de sa `u-ofile` est l'entrée n°3

c) La primitive `sigsuspend` permet de changer de manière provisoire le masque des signaux du processus appelant.

d) Une connexion sur un serveur s'établit et se poursuit sur la socket qui afait l'objet du `bind`.

PARTIE II : APPLICATION DU COURS

Dans les questions suivantes, les programmes devront être rédigés selon les règles de l'art : vérification du nombre de paramètres éventuels, vérification des valeurs de retour des primitives du système, indentation, propreté et **surtout commentaires** du code. Vous êtes dispensés des `includes`.

Question 2 (4 points)

En utilisant le verrouillage coopératif de fichier, écrire un programme dont l'appel est le suivant:

```
share_file file phrase
```

`share_file` crée le fichier `file` en lecture/écriture et un processus fils. **A tour de rôle**, le père écrit dans `file` les caractères en position paire de `phrase` et le fils écrit dans `file` les caractères en position impaire de `phrase`. Le premier caractère de `phrase` occupe la position 0. Le père affiche ensuite le contenu de `file`.

Question 3 (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*. À 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 naît).
- 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
```

où `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'accès à 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`.

Question 4 (6 points)

Ecrivez le code du programme `star` qui simule, très approximativement, un réseau de communication ayant une topologie en étoile. Pour cela, `star` crée N processus appelés *station*.

- `star` itère indéfiniment le comportement suivant: il envoi un signal `POLL` à la `station i` pour lui demander si elle a des données à émettre. `star` attend **exactement** `DELAI` secondes puis, à la fin de ce temps, se demande si la `station i` lui a envoyé le signal `DATA`. Si oui, il envoie le signal `DATA` à toutes les autres `station` puis le signal `ACK` à la `station i` et passe à la `station i+1`. Si non, il passe directement à la `station i+1`.

  Vous écrirez le corps de la fonction `void wait_delai (int delai)` pour implémenter cette attente.

- une `station i` itère indéfiniment le comportement suivant: si elle a des données à émettre, elle attend le signal `POLL`. Quand elle reçoit le signal `POLL`, elle envoie le signal `DATA` à `star` puis attend le signal `ACK`. Quand elle reçoit le signal `ACK`, elle affiche: `STi: données transmises`. Si elle n'a pas de données à émettre, elle ignore le signal `POLL` et affiche: `STi: pas de données`. Lorsque la `station` reçoit le signal `DATA`, elle affiche: `STi: données reçues`. Les attentes seront implémentées dans le corps de la fonction `void wait_poll_ack ()` dont vous écrirez le corps.

  Vous écrirez également le corps de la fonction `void simul_data (int dmax)` qui simule une arrivée de données à transmettre dans un délai aléatoire compris entre 0 et `dmax`. Pour chaque arrivée de données à émettre `station i` fait simplement +1 sur son compteur `cpt_data`. Pour chaque envoi de données, la `station i` fait simplement -1 sur son compteur `cpt_data`. Tant que `cpt_data` est > 0, la `station i` a des données à émettre.

L'appel du programme se fera de la façon suivante: `star N`
Vous utiliserez les `defines` suivants:

```
#define DATA            SIGUSR1 // Emission/réception de données
#define ACK             SIGUSR2 // Acquittement de données émises
#define DATA_TO_SEND    SIGALRM // Arrivée de données à émettre
```

et les primitives `POSIX` pour les signaux.

ANNEXE

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
                ------------------------------------------------
int sigaction(int signum, const struct sigaction *act,struct sigaction *oldact);
struct sigaction {
      void     (*sa_handler)(int);
      sigset_t   sa_mask;
      int        sa_flags;
};
                ------------------------------------------------
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
                ------------------------------------------------
int sigemptyset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
                ------------------------------------------------
int fcntl(int fd, int cmd, ... /* arg */ );
struct flock {
      short l_type;   /* Type de verrouillage : F_RDLCK,F_WRLCK, F_UNLCK */
      short l_whence; /* Interprétation de l_start: SEEK_SET, SEEK_CUR, SEEK_END */
      off_t l_start;  /* Décalage de début du verrouillage */
      off_t l_len;    /* Nombre d'octets du verrouillage */
      pid_t l_pid;    /* PID du processus bloquant notre verrou(F_GETLK seulement) */
};
                ------------------------------------------------
int rand(void);
      The  rand() function returns a pseudo-random integer in the range [0, RAND_MAX]
```