Université Toulouse III – Paul Sabatier
118 route de Narbonne
31062 Toulouse cedex 9

Contrôle terminal – Session 1 – 12 décembre 2016

Durée : 2h **Aucun document autorisé**

Attention : On veillera à la présentation et aux commentaires

Exercice 1 : Processus UNIX

On se propose de simuler un jeu, du type « jeu de fléchettes », conçu sur le principe de la recherche d'une valeur cible. Deux catégories de processus sont impliquées dans cette simulation : un arbitre et NJ joueurs. La partie se déroule en plusieurs tours.

À chaque tour, l'arbitre décide d'une valeur à atteindre (la valeur cible). Chaque joueur, encore présent dans le jeu, choisit une valeur, puis transmet cette valeur à l'arbitre. Le joueur qui propose la valeur la plus éloignée de la valeur cible est éliminé. Dans le cas où plusieurs joueurs auraient proposé une même valeur éliminatoire, tous ces joueurs sont éliminés, sauf si cela doit conduire à l'élimination de tous les joueurs encore présents. Dans ce dernier cas, le tour est rejoué et aucun joueur n'est éliminé.

Lorsque l'arbitre a reçu les valeurs proposées par tous les joueurs encore présents, il décide qui doit être éliminé et il invite le(s) joueur(s) éliminé(s) à quitter la partie. Ces processus joueurs doivent alors se terminer.

Exemple de partie à 5 joueurs :

Arbitre	**Joueur 1**	**Joueur 2**	**Joueur 3**	**Joueur 4**	**Joueur 5**
20	7	5	10	**40**	3
Le joueur 4 est éliminé.					
500	52	**22**	**22**		104
Les joueurs 2 et 3 sont éliminés.					
10	**7**				**7**
Le tour est annulé.					
9	11				**4**
Le joueur 5 est éliminé.					
Le joueur 1 est déclaré vainqueur de la partie.					

L'arbitre signale le début de chaque tour.

La partie est finie lorsqu'il ne reste plus qu'un seul joueur. L'arbitre affiche alors l'identité du joueur vainqueur (son PID par exemple). Le dernier joueur ainsi que l'arbitre doivent alors se terminer.

Remarques :

- Le nombre NJ de joueurs est une constante de l'application.
- Les valeurs sont choisies dans un intervalle dont on fixera arbitrairement les bornes.

Questions

1. Faire une **description schématique** claire, précise et commentée de la solution proposée qui utilisera signaux et tubes.
2. Écrire le **code** d'un processus joueur et du processus arbitre ainsi que le code de la fonction main qui initialise l'application.

Exercice 2 : Synchronisation par sémaphores

On désire synchroniser les accès de NN nageurs à une piscine disposant de NP paniers et NC cabines (on suppose que : $1 \leq NP < NN$, $1 \leq NC < NN$ et $NP \geq NC$). Un panier est utilisé par un nageur pour déposer ses vêtements, et une cabine pour se déshabiller/rhabiller. Avant d'accéder au bassin, un nageur doit disposer d'un panier puis se déshabiller dans une cabine. Lorsqu'il a fini de nager, un nageur doit se rhabiller dans une cabine puis rendre le panier utilisé.

Chaque nageur effectue donc les opérations suivantes :

```
Nageur {
        Se déshabiller ;
        Nager ;
        Se rhabiller ;
}
```

On assimile les nageurs à des activités parallèles ; les casiers et cabines à des ressources partagées.

Questions

1. Écrire l'algorithme (pseudo-code) d'une activité Nageur afin de **synchroniser**, à l'aide de **sémaphores**, les accès des nageurs à la piscine.
2. On considère maintenant que les nageurs entrants sont **prioritaires** pour l'acquisition des cabines. Réécrire l'algorithme permettant de synchroniser les accès des nageurs aux ressources partagées.

Quelques définitions UNIX

```
/* On suppose que les #include seront faits de la bonne manière,
inutile de les préciser dans votre code */
```

```
int main(int argc, char **argv, char **env);
```

```
pid_t getpid (void);
pid_t getppid (void);
uid_t getuid (void);
uid_t geteuid (void);
```

```
pid_t fork (void);

void exit (int compteRendu);
void _exit (int CompteRendu);

pid_t wait (int *circonstances);
pid_t waitpid (pid_t pidAttendu, int *circonstances, int options);
/* options: WIFEXITED, WIFSIGNALED, WEXITSTATUS, WTERMSIG */
```

```
int execl (char *cheminAcces,
           char *arg0, char *arg1, ..., char *argN, NULL);
int execv (char *cheminAcces, char *tabArg[]);
int execlp (char *cheminAcces,
           char *arg0, char *arg1, ..., char *argN, NULL);
int execvp (char *cheminAcces, char *tabArg[]);
int execle (char *cheminAcces,
           char *arg0, char *arg1, ..., char *argN, NULL, char *env[]);
int execve (char *cheminAcces, char *tabArg[], char *env[]);
```

```
int creat (char *cheminAcces, mode_t droits);
int open (char *cheminAcces, int mode, mode_t droits);
     /* mode:   O_RDONLY, O_WRONLY, O_RDWR,
     O_APPEND, O_CREAT, O_EXCL, O_TRUNC */

int write (int numeroInterne, char *adresse, unsigned nombreTransmis);
int read (int numeroInterne, char *adresse, unsigned nombreDemande);

int close(int numeroInterne);

int dup (int numeroInterne);
int dup2 (int origine, int destination);

int pipe(int tube[2]);
```

```
int sigemptyset (sigset_t *ensSignaux);
int sigaddset (sigset_t *ensSignaux, int unSignal);
int sigdelset (sigset_t *ensSignaux, int unSignal);
int sigfillset (sigset_t *ensSignaux);
int sigismember (sigset_t *ensSignaux, int unSignal);

int sigprocmask (int actionSouhaitee,
                 sigset_t *ensSignaux, sigset_t *ancienEnsSignaux);
/* actionSouhaitee: SIG_SETMASK, SIG_BLOCK, SIG_UNBLOCK */

int sigpending (sigset_t *ensSignaux);

typedef void (*traitement) (int leSignal);

traitement signal(int sigIntercepte, traitement monTraitement);

struct sigaction /* prédéfinie */ {
     traitement sa_handler; /* SIG_IGN, SIG_DFL, fonction */
     sigset_t   sa_mask;
     int        sa_flags;
};
int sigaction (int sigIntercepte,
     struct sigaction *nouvelleAction, struct sigaction *ancienneAction);

int pause();
int sigsuspend (sigset_t *ensSignaux);

int kill(int pidDestinataire, int sigEmis);
unsigned alarm(unsigned duree);
```

```
void srand(unsigned int seed);
/* Initialise le générateur de nombres pseudo-aléatoires avec la graine
donnée en paramètre */
int rand(void);
/* Retourne un nombre pseudo-aléatoire dans l'intervalle [0, RAND_MAX] */
```