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

Contrôle terminal – Session 2 – 19 juin 2017

Durée : 1h45 **Aucun document autorisé**

Attention : On veillera à la présentation et aux commentaires

Exercice 1 : Processus UNIX

On se propose de simuler un système (très éloigné de la réalité, heureusement) dans lequel deux types de processus s'exécutent en parallèle : une tour de contrôle et NA instances d'avions (NA peut changer d'une exécution à l'autre et avoir une valeur élevée).

À intervalles de temps réguliers, un avion envoie à la tour de contrôle à quelle distance (une valeur entière) il se trouve. Cette dernière attend d'avoir reçu toutes les positions des avions encore en vol pour décider lequel a priorité pour atterrir (le plus proche, et, en cas d'égalité, le premier à avoir transmis son information) et ceux qui doivent continuer à voler en l'informant. La tour avertit ensuite chaque avion de ce qu'il doit faire. Lorsqu'il est averti, un avion agit en conséquence (être prioritaire pour atterrir consiste à terminer son traitement après avoir affiché un message).

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 **avion** et du **processus** tour ainsi que le code de la fonction **main** qui initialise l'application comportant la tour et NA avions.

Exercice 2 : Synchronisation par sémaphores

On considère les trois processus cycliques P1, P2 et P3 suivants :

```
Processus P1 {
    Boucler {
        Afficher 'A' ;
        Afficher 'B' ;
    }
}
```

```
Processus P2 {
    Boucler {
        Afficher 'C' ;
    }
}
```

```
Processus P3 {
    Boucler {
        Afficher 'D' ;
    }
}
```

Questions

1. Donnez les affichages possibles après le lancement parallèle de ces processus.
2. En utilisant un ou plusieurs sémaphores, **synchronisez** ces processus de manière à **restreindre** les affichages possibles, à chaque cycle, à ACDB **ou** ADCB.

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] */
```