UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

Licence - Mention Informatique

Programmation multitâche

Mercredi 17 décembre 2014

Durée: 1 heure 30 minutes

Aucun document autorisé

ATTENTION: on veillera à la présentation et aux commentaires.

On se propose de simuler l'activité (simplifiée) d'un aéroport équipé de deux pistes d'atterrissage. Les avions qui arrivent en approche de l'aéroport doivent se manifester auprès du contrôleur lui signifiant leur demande d'atterrissage puis attendre l'autorisation d'atterrir sur la piste 1 ou la piste 2 tel que le précisera le contrôleur. Deux catégories d'avions se présentent : des avions passagers et des avions cargos. Les avions passagers sont prioritaires sur les avions cargos, les avions d'une même catégorie étant traités par le contrôleur dans l'ordre de leurs demandes.
Après autorisation, l'avion se pose sur la piste indiquée puis sort de cette piste pour rejoindre sa place de parking. Lorsque l'avion sort de cette piste, il prévient le contrôleur (après un temps de `NB_SECONDES`) que la piste est de nouveau libre pour un nouvel atterrissage.
Le nombre d'avions considérés est fixé à `NB_AVIONS_P` avions passagers et `NB_AVIONC_C` avions cargos.
Les avions et le contrôleur seront représentés par des processus.

Études préliminaires

- Décrire en quelques lignes (schéma autorisé) le scénario de la solution adoptée.
- Indiquer quelles sont les variables qui seront partagées entre le processus `controleur` et les processus `avion` ?
  - Justifier ces choix.
- Quels outils de communication (tubes de communication anonymes) va-t-on mettre en place pour assurer la communication entre les processus ?
  - Justifier ces choix.
  - Indiquer, pour chacun d'eux, quels sont les lecteurs et les rédacteurs ainsi que et les messages transmis et les conditions d'émission de ces messages. Expliquer.
- Quels outils de synchronisation (signaux) va-t-on utiliser ?
  - Justifier ces choix.
  - Indiquer quels sont les émetteurs et les récepteurs ainsi que les conditions d'émission de ces signaux et les traitements qui seront mis en place chez les destinataires. Expliquer.

Programmation

- Donner le programme C correspondant à cette simulation et contenant :
  - Le code permettant d'initialiser la simulation,
  - Le code du processus `controleur`,
  - Le code d'un processus `avion`.

Remarques

- Lorsque tous les avions ont atterri, la simulation est stoppée.
- On pourra utiliser une bibliothèque virtuelle permettant de gérer une file d'attente adaptée aux besoins de ce problème sans avoir à la définir (les noms des services devront être compréhensibles).
- Il sera possible aussi d'utiliser toute fonction générale qui allégerait la programmation sans à avoir à la décrire mais en la spécifiant seulement.

Barème (provisoire)

- Études préliminaires : 8 points (réflexion à mener et analyse à produire **précisément**)
- Programmation : 12 points

Quelques définitions UNIX

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <fcntl.h>
```

```
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 Compte_rendu);
void  _exit(int Compte_rendu);

pid_t wait   (int *Circonstances);
pid_t waitpid(pid_t Pid_attendu, int *Circonstances, int Options);

/* WIFEXITED, WIFSIGNALED, WEXITSTATUS, WTERMSIG */
```

```
int execl (char *Chemin_Acces,
char *Arg0, char *Arg1, ..., char *ArgN, NULL);
int execv (char *Chemin_Acces, char *Tab_Arg[]);

int execlp(char *Chemin_Acces,
char *Arg0, char *Arg1, ..., char *ArgN, NULL);
int execvp(char *Chemin_Acces, char  *Tab_Arg[]);

int execle(char *Chemin_Acces,
char *Arg0, char *Arg1, ..., char *ArgN, NULL,
char *Env[]);
int execve(char *Chemin_Acces,
char *Tab_Arg[],
char *Env[]);
```

```
int creat(char *Chemin_Acces, mode_t Droits);

int open (char *Chemin_Acces, int Mode, mode_t Droits);
/*Mode: O_RDONLY, O_WRONLY, O_RDWR,
   O_APPEND, O_CREAT, O_EXCL, O_TRUNC */

int write(int Numero_interne,
char *Adresse, unsigned Nombre_transmis);
int read (int Numero_interne,
char *Adresse, unsigned Nombre_demande);

int close(int Numero_interne);

int dup (int Numero_interne);
int dup2(int Origine, int Destination);

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

```
int sigemptyset (sigset_t *Ens_Signaux);
int sigaddset   (sigset_t *Ens_Signaux, int Un_Signal);
int sigdelset   (sigset_t *Ens_Signaux, int Un_Signal);
int sigfillset  (sigset_t *Ens_Signaux);
int sigismember (sigset_t *Ens_Signaux, int Un_Signal);

int sigprocmask (int Action_souhaitée,
sigset_t *Ens_Signaux,
sigset_t *Ancien_Ens_Signaux);
 /* Action_souhaitée: SIG_SETMASK, SIG_BLOCK, SIG_UNBLOCK */

int sigpending (sigset_t *Ens_Signaux);

typedef void (*Traitement) (int Signal);

struct sigaction /* predefinie */
  { Traitement sa_handler;
    sigset_t   sa_mask;
    int        sa_flags;
  };
 /* sa_handler: SIG_IGN, SIG_DFL fonction */

Traitement signal(int Sig_intercepte, Traitement Mon_Traitement);
int pause();

int sigaction(int Sig_intercepte,
               struct sigaction *Nouvelle_Action,
               struct sigaction *Ancienne_Action);
int sigsuspend(sigset_t *Ens_Signaux);

int kill(int Pid_Destinataire, int Sig_emis);

unsigned alarm(unsigned Duree);
```