UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

Licence - Mention Informatique

Programmation multitâche

Mardi 15 décembre 2015
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'une production d'énergie renouvelable par un parc de N éoliennes.
Les éoliennes envoient à intervalles de temps réguliers l'énergie qu'elles produisent à une centrale de gestion. La centrale de gestion stocke l'énergie reçue des N éoliennes. Si l'énergie envoyée par une éolienne est inférieure à un seuil S_PROD, elle demande à l'éolienne de s'arrêter.
Toutes les heures, un central météo communique la prévision de la force du vent à la centrale de gestion. Si cette prévision est supérieure ou égale à un seuil S_VENT, la centrale de gestion doit demander aux éoliennes arrêtées de redémarrer.

On supposera données les fonctions :
`void fonctionnementEolienne(void)` : qui simule le fonctionnement d'une éolienne et met à jour la valeur de sa production dans une variable globale que vous déclarerez,
`int fonctionnementMeteo(void)` : qui simule le fonctionnement du central météo et met à jour l'estimation de la force du vent dans une variable globale que vous déclarerez.

Étude préliminaire

- Donner un schéma de communication, entre les processus mis en jeu, bien commenté.

Programmation

- Donner le programme C correspondant à une simulation mettant en jeu un nombre donné (paramètre) d'éoliennes transmettant leur information de production selon un délai donné (paramètre). Ce programme contiendra :
  - Le code permettant d'initialiser la simulation
  - Le code du processus `centraleGestion`
  - Le code d'un processus `centralMeteo`
  - Le code d'un processus `eolienne`

Barème (provisoire)

- Étude préliminaire : 6 points
- Programmation : 14 points

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);
struct sigaction /* prédéfinie */ {
      traitement sa_handler; /* SIG_IGN, SIG_DFL, fonction */
      sigset_t   sa_mask;
      int        sa_flags;
};
traitement signal(int sigIntercepte, traitement monTraitement);
int pause();
int sigaction (int sigIntercepte,
      struct sigaction *nouvelleAction, struct sigaction *ancienneAction);
int sigsuspend (sigset_t *ensSignaux);
int kill(int pidDestinataire, int sigEmis);
unsigned alarm(unsigned duree);
```