TD 1 - LIF12 système d'exploitation

Passage d'informations sur les pipes

29 janvier 2018

I Envoie et lecture de données

Vous devez écrire 2 fonctions

— void envoie_donnees(int fd, const std::vector<char> &data) : Cette fonction doit envoyer le contenu du vecteur data à travers le *file descriptor* fd. Cette fonction doit s'assurer que toutes les données sont envoyées.

— std::vector<char> lire_donnees(int fd, int taille) : Cette fonction lit les données sur le *file descriptor* fd et les renvoie sous la forme d'un vector. Cette fonction doit s'assurer de bien lire toutes les données demandées. Elle doit donc se bloquer jusqu'à leur arrivée. Si le flux est fermé trop tôt, elle doit remplir les octets suivant avec des 0.

En cas d'erreur sur le flux, les 2 fonctions, doivent lancer une exception ou sortir du programme. Pour simplifier le traitement d'erreur, vous pouvez utiliser la macro :

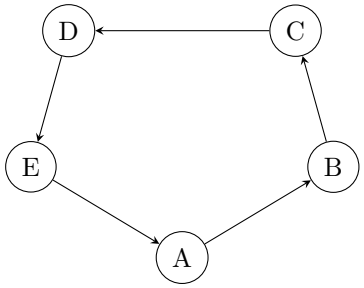
```
exit_error (const char *msg, bool cond, int errnum);
```

Qui sort du programme en affichant le message msg et le code d'erreur errnum si la condition cond est vraie.

II Élections

Dans certaines circonstances, différents programmes doivent s'organiser pour résoudre un problème et pour cela désigner un *leader*, c'est à dire un programme qui sera en charge soit de coordonner les tâches soit d'envoyer les informations aux autres. En général, on procède à une élection, c'est à dire que chacun tire une valeur au hasard et celui qui a la plus forte valeur devient le *leader*. Pour que cela fonctionne, il faut que la plus grande valeur soit bien transmise à tous les autres processus.

Vous devez effectuer les élections dans le cas d'un réseau de procéssus relié en anneau. Cela signifie que chacun peut envoyer une valeur à son voisin de droite et lire celle du voisin de gauche.



Q.II.1) - Vous devez proposer un algorithme qui permet aux processus de désigner un leader.

L'idée est de faire circuler la valeur des processus le long de l'anneau. Le premier processus envoie sa valeur au second, qui la compare avec la sienne et transmet la plus grande au suivant ... Une fois que 2 tours complets ont été effectués, on peut montrer que la plus grande valeur est passée partout.

Données : val, id, $nbprocs$
début
  $valcour = val$
  $idcour = id$

  si $id \neq 0$ **alors**
    $(valtemp, idtemp) = lireprecedant()$
    si $valtemp \geq valcour$ **alors**
      $valcour = valtemp$
      $idcour = idtemp$
  envoiesuivant($valcour$, $idcour$)
  $(valtemp, idtemp) = lireprecedant()$
  si $valtemp \geq valcour$ **alors**
    $valcour = valtemp$
    $idcour = idtemp$
  si $id \neq nbprocs - 1$ **alors**
    envoiesuivant($valcour$, $idcour$)
fin
retourner $idcour$

Q.II.2) - Comment allez-vous assurer un passage correcte des valeurs entre les processus.

voir la solution dans le code.
Il y a deux valeurs à transmettrent (la valeur courante du processus et l'identifiant du leader courant. Donc il y a 2 entiers. On peut les envoyer sous la forme de 2 tableaux de 16 octets (15 caractères pour écrire l'entier et 1 pour la fin de la chaine). Pour cela on utilise les fonctions de la question précédante.

Q.II.3) - Écrire la fonction :

```
int election(int num, int nb_proc, int entree, int sortie);
```

Où num est le numéro du processus, nb_proc le nombre totale de processus, entree et sortie les *files descriptors* permettant de lire ce qu'envoie le voisin de gauche et d'écrire au voisin de droite. Pour cela chaque processus doit lire la valeur du précedant et l'envoyer au suivant 2 fois. Sauf le premier qui ne lit pas au début et le dernier qui n'envoie rien à la fin.

```
//#########################################
//##  solution election
//#########################################

int election(int num, int nb_proc, int entree, int sortie) {
  int leader = num;
  std::ostringstream c;
  std::vector<char> data(32);

  srand((num+1)*time(NULL));
  int val = rand()%100;

  c.str("");
  c << "Processus " << num
    << " ma valeur est " << val << std::endl;
  std::cerr << c.str();

  if (num != 0) {
    std::vector<char> mess = lire_donnees(entree, 16);
    int val_autre = atoi(mess.data());
    mess = lire_donnees(entree, 16);
    int num_autre = atoi(mess.data());

    if (val_autre>=val) {
      val = val_autre;
      leader = num_autre;
    }

    c.str("");
    c << "Processus " << num << " recu " << val_autre
      << " ma valeur est " << val << " le leader est " << leader
      << std::endl;
    std::cerr << c.str();

  }

  snprintf(data.data(), 32, "%15d %15d", val, leader);
  envoie_donnees(sortie, data);

  std::vector<char> mess = lire_donnees(entree, 16);
  int val_autre = atoi(mess.data());
  mess = lire_donnees(entree, 16);
  int num_autre = atoi(mess.data());

  if (val_autre>=val) {
    val = val_autre;
    leader = num_autre;
  }
  c.str("");
  c << "Processus " << num << " recu " << val_autre
    << " ma valeur est " << val << " le leader est " << leader
    << std::endl;
  std::cerr << c.str();

  if (num != nb_proc-1) {
    snprintf(data.data(), 33, "%15d%16d", val, leader);
    envoie_donnees(sortie, data);
  }
  return leader;
}
```



```
//#########################################
//##  Fonctions pour tester
//#########################################
```