TD 1 - LIF12 système d'exploitation

Passage d'informations sur les pipes

11 février 2017

I Envoie et lecture de données

Vous devez écrire 2 fonctions :

— void envoie_donnees(int fd, 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.

L'idée est de faire circuler la valeur des processus le long de l'année. 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 tour complet on été effectué, on est sur 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 $id \neq nbprocs$ **alors**
    envoiesuivant($valcour$, $idcour$)
fin
retourner $idcour$

— 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 2 entier. On peut les envoyer sous la forme de 2 tableaux de 16 octets (15 caractère pour écrire l'entier et 1 pour la fin de la chaine). Pour cela on utilise les fonctions de la question précédante.

— 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. Elle doit donc se bloquer jusquà leur arrivée. Si le flux est fermé trop tôt, elle doit remplir les octets suivant de 0.

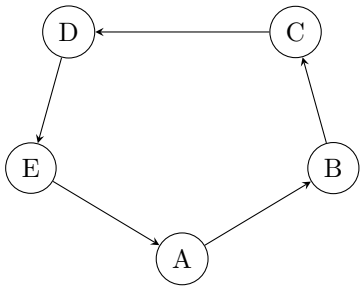
En cas d'erreur sur le flux, les 2 fonctions, doivent lancer une exception ou sortir du programme.

```cpp
//#########################################
//##  Solution lecture/ecriture
//#########################################

std::vector<char> lire_donnees(int fd, int taille) {
  std::vector<char> res(taille);

  int deja = 0;
  while(deja < taille) {
    int nb = read(fd, res.data()+deja, taille-deja);
    exit_error("read", nb == -1, errno);

    if (nb == 0) {
      std::cerr << "Problème le flux est fermé alors qu'il reste "
                << taille-deja << " octets à lire"  << std::endl;
      // je vide le contenu du tableau
      memset(res.data()+deja, 0, taille-deja);
      break;
    }

    deja += nb;
  }

  return res;
}

void envoie_donnees(int fd, std::vector<char> data) {
  int deja = 0;
  int taille = data.size();

  while (deja < taille) {
    int nb = write(fd, data.data()+deja, taille-deja);
    exit_error("write", nb == -1, errno);

    deja += nb;
  }
}

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

II Élections

Dans certaines circonstances, différents programmes doivent s'organiser pour résoidre un problème. 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 devient le *leader*.

Vous devez effectuer les élections dans le cas d'un réseau de procéssus relié en anneau. 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.

Q.II.2) - 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 *file descriptor* 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();

  // std::string osf;
  // getline(std::cin, osf);

  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);
  }
```