R. Namyst
LaBRI, bureau 322

le 8 avril 2005

Licence informatique — Programmation Système
Devoir Surveillé

Durée : 1h30 — Sans document

Lisez l'intégralité du sujet *avant de commencer à répondre aux questions. Certaines peuvent être traitées même si vous n'avez pas répondu aux précédentes.*

1 Appels système (exercice d'échauffement)

Question 1 Rappelez brièvement le principe de fonctionnement d'un "appel système" (sur un exemple concret de votre choix). Dans un système d'exploitation multi-utilisateurs, les intéractions entre les applications et le noyau s'effectuent principalement au moyen de ce mécanisme. Pourquoi n'est-il pas envisageable d'utiliser de simples appels procéduraux ? Même en utilisant des morceaux de code écrits en langage machine, pourquoi une application ne peut-elle contourner ce mécanisme, par exemple pour accéder directement aux structures du noyau ?

Question 2 Les entrées/sorties proposées par la bibliothèque standard C introduisent un type `FILE` et des primitives associées (`fopen`, `fprintf`, etc.). Expliquez les avantages d'utiliser ces primitives plutôt que d'utiliser directement les appels systèmes manipulant des descripteurs de fichiers.

2 Lecture/écriture dans un fichier

On souhaite disposer d'un programme qui puisse modifier un fichier `toto` contenant une suite d'entiers sous forme binaire (de type `uint32`). Plus précisément, le programme (appelons-le `modifier`) doit simplement permettre d'incrémenter certaines valeurs, dont la position est spécifiée en ligne de commande. Pour illustrer ce fonctionnement, considérons un fichier contenant les entiers suivants :

`12 76 3 0 734329 37`

Après l'exécution de la commande `modifier 0 4 5`, le fichier `toto` doit maintenant contenir :

`13 76 3 0 734330 38`

(c'est-à-dire que les entiers aux positions logiques 0, 4 et 5 ont été augmentés de 1).

Pour simplifier, on considèrera dans un premier temps que les positions n'excèdent jamais la taille du fichier...

Question 1 Donnez le code du programme `modifier`. NB : on pourra utiliser `lseek(fd, offset, SEEK_SET)` pour se déplacer dans le fichier avec élégance...

Question 2 Dans le cas où certaines positions sont en dehors du fichier, on souhaite agrandir le fichier et écrire 1 aux positions concernées. Donnez la nouvelle version de la fonction `modifier`.

Question 3 Expliquez pourquoi il est préférable d'utiliser les fonctions `htonl` et `ntohl` lors des accès au fichier.

3 Communication par tubes

Voici le source d'un programme `mystere.c` (TSVP).

```
#define MSG "coucou"
#define NB  strlen(MSG)

unsigned chain(unsigned nb)
{
  unsigned i;
  int tube[2];

  for (i=0; i<nb-1; i++) {
    pipe(tube);
    if(fork() != 0) {
      close(tube[1]); dup2(tube[0], STDIN_FILENO); close(tube[0]);
      break;
    } else {
      close(tube[0]); dup2(tube[1], STDOUT_FILENO); close(tube[1]);
    }
  }

  return i;
}

void output(unsigned rank)
{
  char c;
  int n;

  if(rank == NB-1)
    printf("%s%c", MSG, '\0');
  else
    for (n=0; read(STDIN_FILENO, &c, 1) == 1; n++) {
      if(rank != 0 && rank == n)
        c = toupper(c); // en majuscule
      write(STDOUT_FILENO, &c, 1);
    }
}

int main(int argc, char *argv[])
{
  unsigned i;

  i = chain(NB);

  output(i);

  return 0;
}
```

Question 1 La fonction `chain` crée un certain nombre de processus et un certain nombre de tubes. Combien ?

Faites un dessin montrant les processus et leurs liens en lecture et/ou écriture avec chaque tube. Indiquez les liens de parenté entre les processus et indiquez, pour chacun, la valeur retournée (i.e. celle de la variable `i`) par la fonction `chain`.

Question 2 Discutez, pour chaque processus, du traitement effectué au sein de la fonction `output` (Qui écrit quoi ? Qui lit quoi ?). Donnez l'affichage produit par le programme `mystere`.

Question 3 Est-ce que tous les processus se terminent (i.e. est-il possible que certains restent bloqués en lecture sur un tube) ? Expliquez précisément pourquoi.