R. Namyst
LaBRI, bureau 322

le 8 avril 2005

Licence informatique — Programmation Système
Correction du Devoir Surveillé du 8 avril 2005

1 Appels système (exercice d'échauffement)

Question 1 Dans un environnement multi-tâches (et *a fortiori* dans un environnement multi-utilisateurs), chaque processus se voit attribuer un nombre fini de *droits* (permissions). L'accès limité à son propre espace d'adressage en est un exemple. Les mécanisme des droits affectés aux fichiers en est un autre. Pour assurer le respect de ces droits et, plus généralement, afin d'empêcher l'accès direct à certaines fonctionnalités offertes par le matériel, bon nombre d'opèrations délicates doivent être exécutées sous-contrôle du système. C'est justement le rôle d'un appel système. Son principe est le suivant :

- Chaque opération nécessitant un accès contrôlé n'est pas exécutable directement par les processus. Un tel service est proposé par le noyau du système et est identifié par un numéro d'appel système.
- Un processus désirant exécuter un appel système (ex : `read`) déclenche une interruption logicielle (ex : `syscall` sur un processeur MIPS) après avoit stocké dans un registre le numéro correspondant (ex : 3 sous Linux). Les paramètres suivants sont également passés dans des registres, ou placés sur la pile lorsqu'ils sont nombreux.
- L'interruption provoque le basculement en mode privilégié et le déroutement de l'exécution vers une routine prédéfinie du noyau. Cette routine aiguille l'exécution vers la bonne routine de service (ex :`sys_read`) en utilisant une table. Le processus exécute donc, en mode privilégié, un traitement sur lequel il n'a pas le contrôle, qui est supposé fiable et sécurisé. Ce traitement peut donc vérifier les permissions du processus et refuser l'opération en cas de problème.
- Au retour de l'appel système (c'est-à-dire au retour de l'interruption), le processeur bascule de nouveau en mode utilisateur, et le code de retour est contenu dans un des registres.

Ce type de mécanisme est incontournable car l'utilisateur n'a pas accès à toutes les instructions et certaines ne peuvent s'executer qu'en mode superviseur. Ainsi, même si l'utilisateur disposait d'une copie du code de `sys_read`, il ne pourrait pas l'exécuter car le processeur refuse d'effectuer certaines opérations ou certains accès mémoire lorsqu'il n'est pas en *mode noyau*.

Question 2 Le type `FILE` sert à utiliser des entrées/sorties tamponnées, c'est-à-dire que des tampons sont utilisés en espace utilisateur pour limiter le nombre d'appels systèmes effectués pour lire ou écrire des octets. Il est donc généralement plus performant d'utiliser les primitives associées au type `FILE` que les appels système `open`, `read`, `write`, etc.

2 Lecture/écriture dans un fichier

Question 1 & 2 Cf figure 1.

Question 3 Sans utiliser les fonctions `htonl` et `ntohl`, le fichier `toto` ne sera pas lisible sur des machines adoptant une représentation différente des entiers en mémoire...

3 Communication par tubes

Question 1 La fonction `chain` provoque la création de `nb-1` nouveaux processus (donc au total il y a *nb* processus). Chaque processus possède une valeur de `i` différente (elle vaut 0 pour le père, 1 pour le premier fils, etc.). Chaque processus P_i de rang $i > 0$ est relié par un tube au processus P_{i-1}, et sa sortie standard est redirigée dessus. Chaque processus de rang $i < nb - 1$ est relié au processus P_{i+1} par un autre tube, et son entrée standard est redirigée vers lui.

```
int main(int argc, char *argv[])
{
  int fd, pos, i = 1;
  uint32_t val = 0;

  fd = open("toto", O_RDWR | O_CREAT, 0666);
  if(fd == -1) { perror("open"); abort(); }

  while(i < argc) {
    pos = atoi(argv[i]);
    lseek(fd, pos*sizeof(uint32_t), SEEK_SET);
    if(read(fd, &val, sizeof(uint32_t)) == -1)
      val = 0;
    else
      val = ntohl(val);
    val = htonl(val++);
    lseek(fd, pos*sizeof(uint32_t), SEEK_SET);
    write(fd, &val, sizeof(uint32_t));
    i++;
  }

  close(fd);
  return 0;
}
```

FIG. 1 – Source du programme `modifier`

Question 2 P_{nb-1} écrit le message `coucou` dans le tube à destination de P_{nb-2}. Tous les autres processus P_i lisent ce qu'ils reçoivent du processus P_{i+1} caractère par caractère et, après avoir éventuellement appliqué un traitement au caractère, l'écrivent dans le tube à destination de P_{i-1}. Chaque processus P_i de rang $i > 0$ met en majuscule le $i^{\text{ème}}$ caractère qu'il voit passer.

L'affichage final est donc précisément : `cOUCOu`.

Question 3 Tous les processus se terminent, car on a bien veillé à fermer les extrémités des tubes en écriture de façon à ce que chaque tube ne soit accessible en écriture qu'à un seul processus. Donc lorsque P_{nb-1} se termine, il ferme le tube qui va conduire le processus P_{nb-2} à se terminer à son tour, et ainsi de suite...