Université de Perpignan — Année 2004-2005
Licence d'Informatique L3 — B. Goossens, D. Defour

Examen de système

Juin 2005, durée 3h00, tous documents autorisés

Exercice 1. Soit le fragment de programme suivant:

```
#define L 10
total = 0; d = open("g",O_RDONLY);
while ((lu=read(d,buf,L))>0) total += lu;
printf("%d\n",total);
if (lu==-1) { close(d); exit(EXIT_FAILURE); }
sleep(10); // 10 secondes d'attente...
lu = read(d,buf,L); printf("%d\n",lu); close(d);
```

On suppose qu'avant l'exécution le fichier g contient 103 caractères:

Q1 : que produira le premier appel à printf() ?

Q2 : si rien ne se passe dans le système pendant les 10 secondes d'attente, que produira le second appel à printf() ?

Q3 : si pendant les 10 secondes d'attente, 7 caractères ont été ajoutés par ailleurs au fichier g, que produit le second appel à printf() ? et pourquoi ?

Q4 : si pendant les 10 secondes d'attente on supprime g (rm g) que se passe t-il pour chacun des deux cas précédents ?

Exercice 2. Soit le fragment de code:

```
void truc() {
  while ((lu=read(STDIN_FILENO,buf,L))>0) {
    write(STDOUT_FILENO,buf,lu);
    write(STDERR_FILENO,buf,lu);
  }
  return;
}
```

Q1 : que réalise la fonction truc() ?

On souhaite réutiliser la fonction truc() sans la modifier afin de recopier le contenu du fichier $f1$ à la fois sur la sortie standard et dans un fichier de nom $f2$:

Q2 : quelle(s) fonctionnalité(s) POSIX peut-on utiliser ?

Q3 : écrire le code devant précéder l'appel à truc() pour réaliser ce qu'on désire.

Q4 : décrire sous la forme d'un graphique et pour chaque étape l'évolution des différentes tables du système.

Q5 : est-il possible de rétablir l'état des descripteurs STDIN_FILENO, STDOUT_FILENO et STDERR_FILENO ? si oui écrivez le code correspondant, si non expliquez pourquoi.

Exercice 3. Soit l'extrait de code suivant:

```
d = open("f",ouverture,0); c = 'a'; write(d,&c,1);
lseek(d,(off_t)100000,SEEK_END); write(d,&c,1);
lseek(d,(off_t)50000,SEEK_SET); read(d,&c,1);
printf("J'ai lu le caractère ASCII %d\n",c); close(d);
```

On supposera qu'avant la première exécution il n'existe pas dans le répertoire courant de lien de nom f.

Q1 : quel mode d'`ouverture` doit être spécifié dans l'appel à open() pour que le programme puisse fonctionner comme attendu ?

Q2 : à quelles conditions l'appel à open() réalisera-t-il la création du fichier ?

Q3 : quelle sera la longueur du fichier après exécution (les conditions d'une exécution normale étant réunies) ?

Q4 : que produira l'appel à printf() ?

Q5 : que contiendra le fichier ?

Q6 : que se passera-t-il lors d'une seconde exécution ?

Exercice 4. Soient les deux programmes suivant:

```
/* Programme 1 */ int main(int argc,char *argv[]) {
  if (fork()==0) main(argc,argv);
}

/* Programme 2 */ int main(int argc,char *argv[]) {
  if (fork()==0) execvp(argv[0],argv);
}
```

Q1 : décrivez le comportement de chacun des programmes (en termes de processus, parenté, terminaison, etc).

Soit le code suivant:

```
#include <toucekilfo.h>
int main(int argc,char *argv[]) {
  int i; pid_t p;
  printf("Bonjour je m'appelle %d\n",getpid());
  for (i=0; i<argc; i++)
    switch(fork()) {
      case 0: printf("Je suis %d et mon père %d\n",getpid(),getppid());
              main(argc-1,NULL); printf("J'ai terminé: %d\n",getpid());
              exit(0);
      case -1: printf("raté\n"); exit(1);
    }
  for (i=0; i<argc; i++) {
    p = wait(NULL); printf("%d a terminé\n",p);
  }
  exit(0);
}
```

Q2 : qu'afficherait-on si on l'exécutait sans arguments en ligne de commande ?

Q3 : même question si on l'exécutait avec un argument en ligne de commande ?

Q4 : combien de processus seraient créés avec n arguments en ligne de commande ?