TP2 : Entrées / Sorties

RS : Réseaux et Systèmes

Deuxième année

Quelques conseils pour programmer en C (Rappels de CSH)

Dans ce TP, vous allez être amené à écrire des programmes C de taille moyenne. Il est conseillé d'appliquer les règles suivantes

- *Compilez avec* `-Wall`. Le compilateur détectera les erreurs d'inattention.
- *Utilisez gdb ou valgrind au moindre problème.* Si vous ajouter `-g` aux options de compilation, les messages de valgrind sont encore plus utiles.
- *Indentez votre code.* Quand le programme fait plus de 50 lignes, c'est indispensable.
- *Fermez tous les descripteurs inutiles.* Sans cela, il devient très difficile de détecter les fins de flux.

★ Exercice 1: Créer et utiliser un tube.

Les tubes permettent habituellement à deux processus d'échanger des données. Dans un premier temps, et à but pédagogique, nous allons écrire un programme se parlant à lui-même.

▷ **Question 1:** Créez un tube, puis écrivez la chaine `"Bonjour, moi"` dans l'entrée du tube. Lisez ensuite le contenu du tube à sa sortie, et constatez que la chaîne est inchangée.

★ Exercice 2: Déterminer la taille d'un tube.

Pour permettre à l'écrivain de continuer avant même que le lecteur n'ai lu ces données, le système d'exploitation attache une zone de stockage à chaque tube. Les données écrites y sont stockées en attendant que le lecteur ne les réclame. L'objectif est de déterminer expérimentalement la taille de cette zone.

▷ **Question 1:** Écrire un programme ouvrant un tube et écrivant dedans le nombre d'octets en paramètre.

▷ **Question 2:** Lancez votre programme avec un argument de plus en plus grand pour trouver la taille des tampons des tubes sur votre machine.

Réponse

prog

```c
#include <stdio.h>
#include <unistd.h>

main(int argc, char *argv[]) {
  char buff[102400];

  int fd[2];
  long int ecrit;
  long int n=atoi(argv[1]);

  pipe(fd);

  ecrit = write(fd[1],buff,n);
  if (ecrit < 0)
    perror("plop");

  printf("J'ai écrit %lu octets (de %lu, manque %lu)\n",
         ecrit,n,n-ecrit);
}
```

```
$ gcc E1-soliloque.c && ./a.out 65536
J'ai écrit 65536 octets (sur 65536,
manque 0)
$ gcc E1-soliloque.c && ./a.out 65537
<bloqué>
```

Fin réponse

★ Exercice 3: Emettre des mails depuis vos programmes.

On veut écrire en C l'équivalent de la commande shell suivante : `echo coucou | mail <votre login>@localhost`

▷ **Question 1:** Écrire un programme vous envoyant un mail contenant la chaîne "coucou". Vous exécuterez pour cela le programme `mail`, après avoir redirigé son entrée standard sur un tube. Référez-vous au transparent du cours numéro 93 (qui détaille l'exemple du shell). Vous pouvez consulter les mails reçus avec la commande `mutt`.

▷ **Question 2:** Faire en sorte que le programme de la question précédente vous envoie un message toutes les 3 secondes (sans utiliser `sleep(3)`, mais en utilisant une alarme).

Réponse

Q1 mailer

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

main() {
  int p[2];

  pipe(p);
  if (fork() != 0) {
    int date = time(NULL);

    write(p[1],"coucou", 7);

    close(p[0]); close(p[1]);
  }
  else {
    /* Rediriger l'entree sur le pipe
       et lancer mail */
    dup2(p[0], 0);
    close(p[0]); close(p[1]);
    execlp("mail", "mail",
           "mquinson@localhost", NULL);
  }

  wait(NULL);
  fprintf(stderr,"Message parti\n");
  return 0;
}
```

Q2 chrono

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

void mail(int i) {
  int p[2];

  pipe(p);
  if (fork() != 0) {
    int date = time(NULL);

    write(p[1],"coucou", 7);
    close(p[0]); close(p[1]);
  } else {
    /* Rediriger l'entree sur le pipe
       et lancer mail */
    dup2(p[0], 0);
    close(p[0]); close(p[1]);
    execlp("mail", "mail",
           "mquinson@localhost", NULL);
  }
  wait(NULL);
  fprintf(stderr,"Message parti\n");
}

main() {
  /* man sigaction */
  struct sigaction sig,old;

  sig.sa_handler = &mail;
  sigaction(SIGALRM,&sig,&old);

  while (1) {
    alarm(5);
    pause();
  }
}
```

Attention, si on inverse le père et le fils dans la question 2, on ferme l'entrée standard, ce qui démonize le processus. Après, pour le tuer, faut sortir kill (c'est du vécu).

Fin réponse

★ **Exercice 4: Réimplémenter popen.**

La fonction `popen` permet de lire la sortie standard d'une commande (ou d'écrire sur son entrée standard) comme s'il s'agissait d'un fichier. Exemple d'ouverture en lecture : `fich = popen("ls -lR","r");`

▷ **Question 1:** Implémentez une fonction permettant de lire la sortie d'une commande (ie, en supposant que le second argument est "r").

REMARQUE : popen() retourne un `FILE*` utilisable avec `fscanf`. Votre version devra retourner un descripteur pour `read` (c'est plus simple ainsi).

Réponse

```c
int mypopen(char *cmd) {
  int fd[2];
  pipe(fd);
  if (fork() > 0) { /* père */
    close(fd[1]);
    return (fd[0]);
  } else { /* fils */
    close(fd[0]);
    dup2(fd[1],1);
    close(fd[1]);
    execlp("sh","sh","-c",cmd,NULL);
  }
}
```

Fin réponse

★ **Exercice 5: Communication inter-processus.** (mini-projet)

Observez le programme `/home/depot/2A/RS/distributeur.c`. Il lit le flux de caractères arrivant sur l'entrée standard en séparant les chiffres des lettres, effectue l'opération appropriée en fonction du type de caractère (sommer les chiffres ; réaliser un spectre de fréquence pour les lettres) et enfin affiche le résultat.

Pour cela, le programme est composé de trois entités :

— un distributeur (en charge de la répartition des caractères) ;
— un additionneur (opérant sur les chiffres) ;
— un compteur (opérant sur les lettres).

▷ **Question 1:** Adapter ce programme pour que les fonctions d'additionneur et de compteurs soient assurées par des processus enfants d'un processus père qui assure la fonction de distributeur. Les communications entre les processus se font par tube.

Réponse

```c
/******************************************************************************
**
** Communication par tubes.
** ------------------------
** Ce programme se compose de trois processus et quatre tubes. Le processus
** parent est chargé de lire les caractères en provenance de l'entrée standard,
** d'envoyer les chiffres au premier enfant (par l'intermédiaire du premier
** tube) et les lettres au second (par l'intermédiaire du troisième tube).
** Une fois la fin de fichier atteinte, les enfants renvoient leurs résultats
** au parent (par l'intermédiaire respectivement des deuxième et troisième
** tubes). Le parent affiche ensuite les résultats.
**
** © 2002, Éric Renault pour l'Institut National des Télécommunications.
** © 2003, Denis Conan
**
******************************************************************************/

#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define TUBE_DC 0               /* Distributeur -> Chiffre */
#define TUBE_CD 1               /* Chiffre -> Distributeur */
#define TUBE_DL 2               /* Distributeur -> Lettre */
#define TUBE_LD 3               /* Lettre -> Distributeur */
#define TUBE_NOMBRE     4

#define TUBE_LIT        0
#define TUBE_ECRIT      1
#define TUBE_TOTAL      2

/*
** Écriture d'un entier.
*/
void ecrit_entier (int fichier, int entier) {
  int retour;

  do {
    retour = write(fichier, &entier, sizeof(int));
    assert(retour != -1);
  } while (retour == 0);
}

/*
** Lecture d'un entier.
*/
int lit_entier(int fichier) {
  int retour, entier;

  do {
    retour = read(fichier, &entier, sizeof(int));
    assert(retour != -1);
  } while(retour == 0);
  return entier;
}

/*
** Somme des chiffres.
*/
void chiffre(int lit, int ecrit){
  int car, somme = 0;

  /* Lecture des chiffres et somme. */
  while((car = lit_entier(lit)) != EOF) {
    somme += (car - '0');
  }
  /* Renvoi du résultat. */
  ecrit_entier(ecrit, somme);
}

/*
** Fréquence des lettres.
*/
void lettre(int lit, int ecrit){
  int car, frequence['z' - 'a' + 1], i;

  /* Initialisation. */
  for (i = 'a' ; i <= 'z' ; i ++) {
    frequence[i - 'a'] = 0;
  }
```

```
83    /* Lecture des lettres et incrément des fréquences. */
84    while ((car = lit_entier(lit)) != EOF) {
85      frequence[tolower(car) - 'a'] ++ ;
86    }
87    /* Renvoi du résultat. */
88    for (i = 'a' ; i <= 'z' ; i ++) {
89      ecrit_entier(ecrit, frequence[i - 'a']);
90    }
91  }
92
93  /*
94  ** Distribution des caractères et affichage des résultats.
95  */
96  void distributeur(int ecrit_c, int lit_c, int ecrit_l, int lit_l) {
97    int car, i;
98
99    /* Distribution des caractères. */
100   do {
101     car = getchar();
102     if (isalpha(car) || (car == EOF)) {
103       ecrit_entier(ecrit_l, car);
104     }
105     if (isdigit(car) || (car == EOF)) {
106       ecrit_entier(ecrit_c, car);
107     }
108   } while(car != EOF);
109
110   /* Lecture et affichage de la somme. */
111   printf("Somme : %d", lit_entier(lit_c));
112
113   /* Lecture et affichage des frequences. */
114   for (i = 'a' ; i <= 'z' ; i ++) {
115     printf(" ; %c : %d", i, lit_entier(lit_l));
116   }
117   printf("\n");
118 }
119
120 /*
121 ** Mise en place des tubes et des processus.
122 */
123 int main(int argc, char* argv[]) {
124   int tube[TUBE_NOMBRE][TUBE_TOTAL];
125   int i, retour;
126
127   /* Initialisation des tubes. */
128   for(i = 0 ; i < TUBE_NOMBRE ; i ++) {
129     retour = pipe(tube[i]);
130     assert(retour != -1);
131   }
132
133   /* Création des enfants et lancement des taches associées. */
134   switch(fork()) {
135   case -1 :
136     exit(1);
137   case 0 :
138     chiffre (tube[TUBE_DC][TUBE_LIT], tube[TUBE_CD][TUBE_ECRIT]);
139     break;
140   default :
141     switch(fork()) {
142     case -1 :
143       exit(2);
144     case 0 :
145       lettre(tube[TUBE_DL][TUBE_LIT], tube[TUBE_LD][TUBE_ECRIT]);
146       break;
147     default :
148       distributeur(tube[TUBE_DC][TUBE_ECRIT], tube[TUBE_CD][TUBE_LIT],
149                    tube[TUBE_DL][TUBE_ECRIT], tube[TUBE_LD][TUBE_LIT]);
150     }
151   }
152
153   /* Fermeture des tubes. */
154   for(i = 0 ; i < TUBE_NOMBRE ; i ++) {
155     retour = close(tube[i][TUBE_LIT]);
156     assert(retour != -1);
157     retour = close(tube[i][TUBE_ECRIT]);
158     assert(retour != -1);
159   }
160
161   /* Tout s'est bien terminé. */
162   exit(0);
163 }
```


Fin réponse