Contrôle des interactions – Synthèse

Thèmes traités

— schémas d'interaction : scrutation, synchronisation et publier/s'abonner
— API de contrôle des flots d'E/S : `fcntl, select`

1 Interaction entre processus

Situation (abstraite) : 2 processus (un émetteur, un récepteur) doivent échanger une information. Le récepteur ne contrôle pas l'émetteur. Le récepteur doit obtenir l'information produite par le récepteur « dès que possible ».

Deux schémas d'interaction sont alors possibles.

Communication asynchrone

le récepteur s'exécute indépendamment de l'émetteur. Lorsque l'émission a lieu, le récepteur est (momentanément) interrompu pour traiter le message émis, avant de poursuivre son exécution.
Dans ce schéma d'interaction, le récepteur ne contrôle pas le point de son flot d'exécution où le message sera traité.
C'est le schéma des signaux, ou des interruptions matérielles. On parle de schéma *publier/s'abonner* : le récepteur s'*abonne* (primitive `sigaction(...)`) à la réception de messages *publiés* (primitive `kill(...)`) au rythme de l'émetteur.

Communication synchrone

Le récepteur choisit le point de son exécution où la réception sera traitée. À ce point, il **attend** que le message soit émis et lui parvienne. Cette attente peut être réalisée de deux manières :

① la scrutation (E/S non bloquantes) : le récepteur exécute une boucle consistant à tester si le message a été reçu, jusqu'à la réception effective du message.
Dans le cas de la communication par lecture/écriture de flots d'octets (E/S Unix), ce type d'attente peut être réalisé en positionnant en mode non bloquant (`O_NONBLOCK`) le descripteur associé au flot, grâce à la primitive `fcntl`.

② le blocage (E/S bloquantes) : si le message n'est pas immédiatement disponible, le récepteur est mis en veille. C'est l'émetteur qui provoquera son réveil, au moment de l'émission.
Dans le contexte des entrées-sorties Unix, ce type d'attente est le mode usuel : par défaut les primitives d'accès aux fichiers (`read`, `write`, ...) sont bloquantes.

2 Exercice : rendre un programme plus interactif

Dans le programme 1, un processus père crée un tube puis un fils. Le **fils** transmet un texte (contenu dans un fichier) **via le tube**. Le père reçoit les commandes sur l'entrée standard (clavier), traite le texte (filtrer les voyelles, mettre en majuscules, ne rien afficher...) reçu sur le

tube (en fonction de la dernière commande reçue sur l'entrée standard), et affiche le résultat du traitement sur la sortie standard.

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <stdbool.h>

#define BUFSIZE 512

void traiter(char tampon [], char cde, int nb) {
    int i;
    switch(cde) {
    case 'X' :
        break;
    case 'Q' :
        exit(0);
        break;
    case 'R' :
        write(1,tampon,nb);
        break;
    case 'M' :
        for (i=0; i<nb; i++) {
            tampon[i]=toupper(tampon[i]);
        }
        write(1,tampon,nb);
        break;
    case 'm' :
        for (i=0; i<nb; i++) {
            tampon[i]=tolower(tampon[i]);
        }
        write(1,tampon,nb);
        break;
    default :
        printf("????\n");
    }
}

int main (int argc, char *argv[]) {
    int p[2];
    pid_t pid;
    int d, nlus;
    char buf[BUFSIZE];
    char commande = 'R'; /* mode normal */
    if (argc != 2) {
        printf("utilisation : %s <fichier source>\n", argv[0]);
        exit(1);
    }

    if (pipe(p) == -1) {
        perror ("pipe");
        exit(2);
    }

```

```
55      pid = fork();
56      if (pid == -1) {
57          perror ("fork");
58          exit(3);
59      }
60      if (pid == 0) {   /* fils  */
61          d = open (argv[1], O_RDONLY);
62          if (d == -1) {
63              fprintf (stderr, "Impossible d'ouvrir le fichier ");
64              perror (argv[1]);
65              exit (4);
66          }
67
68          close(p[0]); /* pour finir malgre tout, avec sigpipe */
69          while (true) {
70              while ((nlus = read (d, buf, BUFSIZE)) > 0) {
71                  /* read peut lire moins que le nombre d'octets demandes
    , en
72                   * particulier lorsque la fin du fichier est atteinte.
    */
73                  write(p[1], buf, nlus);
74                  sleep(5);
75              }
76              sleep(5);
77              printf("on recommence...\n");
78              lseek(d, (off_t) 0, SEEK_SET);
79          }
80
81      } else {   /* pere */
82          close(p[1]);
83          system("stty -icanon min 1"); // saisie entrees clavier sans
    tampon
84
85          /* a completer */
86
87          while (true) {
88
89              /* debut code a completer et adapter */
90
91              read(0,&commande,sizeof(char));
92              printf("-->%c\n",commande);
93
94              bzero(buf, BUFSIZE); // nettoyage
95              if ((nlus = read(p[0],buf,BUFSIZE))>0) {
96                  traiter(buf,commande,nlus);
97              }
98
99              /* fin code a completer et adapter */
100
101             sleep(1);
102         }
103     }
104     return 0;
105 }
```

Listing 1 – Programme à compléter

La saisie des commandes est aussi simple que possible : chaque caractère saisi correspondra à une commande, et est pris en compte sans attendre de retour chariot [1]. Les commandes implantées sont 'M' (majuscules), 'm' (minuscules), 'X' (ne rien afficher)', 'R' (rétablir un affichage sans filtre) et 'Q' (quitter).

Dans la version du programme 1, le père alterne en séquence la saisie d'une commande et le traitement (puis l'affichage) d'une partie du texte reçu sur le tube. L'objectif est d'adapter le programme fourni, afin de découpler la saisie des commandes et l'affichage du texte traité. Deux solutions seront alors proposées :

- rendre les E/S non bloquantes et consulter en permanence si au moins l'un des descripteurs est prêt ;
- utiliser la primitive `select` pour attendre qu'un ensemble de descripteurs soit prêt puis à traiter les données de chaque descripteur prêt.

> **Remarque :** Les modifications du programme 1 demandées se situent exclusivement dans le code du père à la ligne 85 et entre les lignes 89 et 99.

2.1 Rendre les E/S non bloquantes

```
1 int fcntl(int desc, int cmd, int args);
```

La primitive `fcntl` permet de contrôler et changer les paramètres (`O_RDONLY`, `O_WRONLY`, ...) d'un descripteur de fichiers (associé à un fichier ou à un tube). La primitive retourne -1 en cas d'échec. `cmd` est la commande à appliquer au descripteur de fichiers `desc`. Parmi les commandes possibles, nous utiliserons :

- `F_GETFL` : permet de récupérer les paramètres du descripteur de fichiers.
- `F_SETFL` : permet de modifier les paramètres du descripteur de fichier ; les nouveaux paramètres à utiliser sont dans `arg`.

Les valeurs de `arg` peuvent être `O_RDONLY`, `O_WRONLY`, `O_RDWR`, ou encore `O_NONBLOCK` (il y en a d'autres).

Pour rendre les E/S non bloquantes, la primitive s'écrit alors :

```
1 fnctl(desc, F_SETFL, fcntl(desc, F_GETFL)|O_NONBLOCK);
```

L'exécution ajoute l'option `O_NONBLOCK` aux paramètres existants.

Question. *Modifiez le code du programme 1 de manière à rendre la lecture du caractères au clavier et la lecture dans le tube non bloquantes.*

2.2 Attendre un ensemble de descripteurs de fichiers

```
1 int select(int fds, fdset *readfds, fd_set *writefds, fd_set *errorfds,
2           struct timeval *timeout);
```

La primitive `select` permet d'attendre un changement d'un descripteur de fichiers stocké dans un ensemble :

- `readfs` indique les descripteurs de fichiers prêts en lecture ;
- `writefs` indique les descripteurs de fichiers prêts pour l'écriture, c'est-à-dire pour lesquels il y a de la place dans les tampons d'écriture ;

1. L'appel `system("stty -icanon min 1")` en ligne 83 permet de configurer le terminal dans ce mode de saisie.

— `exceptfs` contient les descripteurs de fichiers en attente sur une condition exceptionnelle, par exemple l'attente d'un message urgent.

La primitive `select` examine les `fds` descripteurs de fichiers stockés dans chaque ensemble. En d'autres termes, `select` examine l'état des descripteurs de 0 à `fds-1`.

Choix de la valeur de `fds` La valeur de `fds` doit être suffisamment grande pour prendre en compte tous les numéros de descripteurs considérés. Par exemple, si nous souhaitons examiner les descripteurs de fichiers 1 et 25, il sera nécessaire de choisir `fds==26`. Pour généraliser (et simplifier), il est possible d'utiliser la constante `FD_SETSIZE` qui correspond à la valeur maximale des descripteurs de fichiers du système.

Résultat de l'exécution de `select` En retour, `select` remplace les ensembles de descripteurs par des sous-ensembles constitués des descripteurs de fichiers prêts. En conséquence, toute lecture ou écriture d'un descripteur de fichiers de ces sous-ensembles se fera sans blocage. Enfin, la primitive retourne le nombre total de descripteurs prêts dans tous les ensembles, ou -1 en cas d'erreur.

Manipulation des ensembles de descripteurs Le type `fd_set` se manipule à l'aide des macros suivantes :

— `FD_ZERO(&fdset)` : `fdset` $\leftarrow \emptyset$

— `FD_SET(fd, &fdset)` : `fdset` $\leftarrow$ `fdset` $\cup \{$ `fd` $\}$

— `FD_CLR(fd, &fdset)` : `fdset` $\leftarrow$ `fdset` $\setminus \{$ `fd` $\}$

— `FD_ISSET(fd, &fdset)` : retourne $\neq 0$ si `fd` $\in$ `fdset`, 0 sinon.

où `fdset` est défini par `fd_set fdset;` et `fd` est un descripteur de fichiers préalablement ouvert.

Délai d'attente d'une modification Le changement d'état d'un descripteur de fichiers est conditionné par un délai d'attente spécifié par l'argument `timeout`. Plusieurs possibilités :

— si `timeout == NULL`, la primitive `select` restera bloquée jusqu'à ce qu'un descripteur de fichiers devienne prêt ;

— si `timeout != NULL`, `select` restera bloquée jusqu'à ce que la durée définie par `timeout` n'est pas écoulée ou qu'un descripteur de fichier devienne prêt ;

— si `timeout` représente une durée nulle, `select` sortira immédiatement.

Question. *Modifiez le code du programme 1 de manière à attendre à la fois que les descripteurs de fichiers 0 et `p[0]` soient prêts. Nous considérerons que le délai d'attente n'est pas défini.*

3 Exercice de synthèse : processus communiquant par signaux et tubes

On considère le programme suivant :

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <signal.h>
5  #include <stdlib.h>
6
7  #define NBTESTS 5
8  static int iBoucle = 0; /* compteur de boucle incrémenté sans fin par
       le fils pid1
9                           * et remis à zéro par le déclenchement de H1
       */
10 static int pariteIBoucle = 0; /* 0 si iBoucle est pair */
11 static int message1NonAffiche = 1;  /* indique si H1(_) n'a jamais été
       déclenché */
12
13 static int p[2];
14
15 void H1 (int sig) {
16     if (message1NonAffiche==1) {
17         printf ("Message 1:: P1=%d a recu S1=%d\n", (int) getpid(), sig
       );
18         message1NonAffiche = 0;
19     } else {
20         if (sig==SIGQUIT) {
21             printf ("Message 2:: P2=%d a recu S2=%d\n", (int) getpid(),
       sig);
22             exit(3);
23         }
24     }
25     pariteIBoucle=iBoucle%2;
26     printf ("iBoucle=%d pariteIBoucle=%d \n", iBoucle, pariteIBoucle);
27     write(p[1],&pariteIBoucle, sizeof(int));
28     iBoucle=0;
29 }
30
31 void H2 (int sig) {
32     printf ("Message 3:: P3=%d a recu S3=%d\n", (int) getpid(), sig);
33 }
34
35 int main (int argc, char *argv[]) {
36     int pid1, pid2, pid3, ret, i, n;
37     int q[2];
38     float nbtotal, nbpair;
39     float bilan, nbpairmoyen;
40     struct sigaction sa1,sa2;
41
42     printf ("Message 4:: P4=%d de pere PP4=%d\n", (int) getpid(),
       getppid());
43
44     pipe(p);
45     pipe(q);
```

```
46
47      sa1.sa_handler = H1;
48      sa1.sa_flags = 0;
49      sigemptyset(&sa1.sa_mask);
50      sigaction(SIGQUIT, &sa1, NULL);
51      sigaction(SIGUSR1, &sa1, NULL);
52
53      sa2.sa_handler = H2;
54      sa2.sa_flags = 0;
55      sigemptyset(&sa2.sa_mask);
56      sigaction(SIGUSR2, &sa2, NULL);
57
58      pid1=fork();
59      if (pid1 != 0) {
60          printf("..... pid1=%d \n ",(int) pid1);
61          pid2=fork();
62          if (pid2 != 0) {
63              close(p[1]);        /* Ligne 63 */
64              close(q[1]);        /* Ligne 64 */
65              close(p[0]);
66              printf("..... pid2=%d \n ", (int) pid2);
67              for (i=0; i<NBTESTS; i++) {
68                  sleep(1);
69                  kill(pid1,SIGUSR1);
70              }
71              kill(pid2, SIGUSR2);     /* Ligne 71 */
72              ret=wait(&n);       /* Ligne 72 */
73              if (WIFSIGNALED(n)) {
74                  printf("Message 5:: P5=%d et R5=%d \n", ret, WTERMSIG(n
    ));
75              } else {
76                  printf("Message 6:: P6=%d et R6=%d \n", ret,
    WEXITSTATUS(n));
77              }
78              kill(pid1,SIGINT);       /* Ligne 78 */
79              ret=wait(&n);       /* Ligne 79 */
80              if (WIFSIGNALED(n)) {
81                  printf("Message 7:: P7=%d et R7=%d \n", ret, WTERMSIG(n
    ));
82              } else {
83                  printf("Message 8:: P8=%d et R8=%d \n", ret,
    WEXITSTATUS(n));
84              }
85              bilan=-999;
86              ret=read(q[0], &bilan, sizeof(float));
87              if (ret>0) {
88                  bilan = bilan * 2;
89              }
90              printf ( " bilan=%f\n",bilan);
91          } else {  /* ------------------------------------> pid2 == 0
    */
92              close(p[1]);        /* Ligne 92 */
93              close(q[1]);
94              printf ("Message 9:: P9=%d de pere PP9=%d \n", (int) getpid
    (), getppid());
95              pause();
```

```
96              execlp("ps","ps",NULL);
97              ret=wait(&n);      /* Ligne 97*/
98          }
99      } else {    /* ------------------------------------> pid1 == 0 */
100         pid3=fork();
101         if (pid3 != 0) {
102             close(q[1]);       /* Ligne 102*/
103             printf("..... pid3 = %d \n ", (int) pid3);
104             printf ("Message 10:: P10=%d de pere PP10=%d \n", (int)
    getpid(), getppid());
105             for (;;) {      /* boucle infinie, Ligne 105*/
106                 iBoucle ++;
107             }
108             /* Ligne 108 */
109         } else  { /* ------------------------------------> pid3 == 0
    */
110             close(p[1]);
111             close(q[0]);
112             nbpair = 0;
113             nbtotal=0;
114             nbpairmoyen = -1;
115             nbtotal=0;
116             while ( read (p[0], &i, sizeof(int)) > 0 ) { /* Ligne 116*/
117                 nbtotal = nbtotal + 1;
118                 if (i==0) nbpair++;
119             }
120             if (nbtotal!=0) nbpairmoyen = nbpair/nbtotal;
121             printf("nbpair=%f\n", nbpair);
122             printf("nbpairmoyen=%f \n", nbpairmoyen);
123             write (q[1], &nbpairmoyen, sizeof(float));
124         }
125     }
126     return 0;
127 }
```

Questions

Une exécution du code `exam` avec `NBTESTS=5` donne la sortie suivante (notez que certaines valeurs ont volontairement été remplacées par des ????)

```
>exam
Message 4, P4=???? de pere PP4=????
 .......... pid1 = 2504
  .......... pid2 = 2505
 .......... pid3 = 2506
  Message 10, P10=???? de pere PP10=????
  Message 9, P9=???? de pere PP9=????
Message 1, P1=???? a recu S1=30
 iBoucle=241534614 pariteIBoucle=0
 iBoucle=242681100 pariteIBoucle=0
 iBoucle=485825588 pariteIBoucle=0
 iBoucle=242802438 pariteIBoucle=0
 Message 3, P3=???? a recu S3=????
 iBoucle=485986617 pariteIBoucle=1
  PID TTY          TIME CMD
 2130 pts/0    00:00:00 bash
 2503 pts/0    00:00:00 exam
 2504 pts/0    00:00:04 exam
 2505 pts/0    00:00:00 ps
 2506 pts/0    00:00:00 exam
 Message 6, P6=???? et R6=????
 nbpair=4.000000,
 nbpairmoyen=0.800000
Message 7, P7=???? et R7=????
 bilan=1.600000
```

1. Décrire l'architecture de communication entre processus et indiquer/représenter les descripteurs ouverts par processsus.
2. Quels sont les descripteurs connectés au processus qui effectue la boucle infinie (Ligne 105). Comment ce processus sort-il de cette boucle ?
3. Compléter et expliquer l'affichage des valeurs `Pi` et (selon les cas) `Si` ou `Ri` ou `PPi`, pour i=1,10 Expliquer en particulier pourquoi les messages 2, 5 et 8 ne sont pas affichés.
4. Expliquer comment le processus qui effectue le `while` en ligne 116 sort de cette boucle.
5. Que se passe t-il si on supprime le `close(p[1])` en Ligne 63 ?
6. Si on supprime l'instruction `kill(pid2, SIGUSR2)` en Ligne 71, que se passe-t-il ?
7. Dans quel cas atteint-on la ligne 97 ?
8. Si on ajoute l'instruction `signal(SIGUSR1,SIG_IGN)` avant la boucle infinie ligne 105 que se passe-t-il ? Indiquer la valeur de bilan.
9. Que se passe-t-il si on supprime la ligne 78 : `kill(pid1,SIG_INT)` ?
10. Que se passe-t-il si on remplace `kill(pid1,SIGINT)` en ligne 78 par `kill(pid1,SIGQUIT)` ? Indiquer notamment l'effet sur l'affichage du message 7.