

# Systèmes centralisés : TD5

- **Dernier exercice du TD4 :** Combien de processus ? Liens de parenté ?

```
pid1 = fork ();
if ( pid1 != 0 ) {      // père
    pid2 = fork ();
    if ( pid2 != 0 ) {  // père
        ...
    } else {            // fils2 de pid = pid2
        ...
    }
}
else {                  // fils1 de pid = pid1
    pid3 = fork ();
    if ( pid3 != 0 ) {  // fils1
        ...
    } else {            // petit-fils 1-1 de pid = pid3
        ...
    }
}
```

# Systèmes centralisés : TD5

```
void H1 ( int sig ) {           // associé à SIGUSR1 et SIGQUIT pour tous
if ( message1NonAffiche == 1 ) { // première entrée
    printf ( " Message 1:: P1 =%d a reçu S1 =%d \n " , ( int ) getpid () , sig );
    message1NonAffiche = 0;
} else {                       // entrées suivantes
    if ( sig == SIGQUIT ) {
        printf ( " Message 2:: P2 =%d a reçu S2 =%d \n " , ( int ) getpid () , sig );
        exit (3);
    }
}
paritelBoucle = iBoucle %2;
printf ( " iBoucle =%d paritelBoucle =%d \n " , iBoucle , paritelBoucle );
write ( p [1] , &paritelBoucle , sizeof ( int ));
iBoucle =0;
}

void H2 ( int sig ) {           // associé à SIGUSR2 pour tous
    printf ( " Message 3:: P3 =%d a reçu S3 =%d \n " , ( int ) getpid () , sig );
```

# Systèmes centralisés : TD5

2 pipes p et q partagés par tous

**Père** // lecteur sur q[0]

```
for (i=0; i<NBTESTS; i++) {
    sleep(1);
    kill(pid1, SIGUSR1);
}
kill(pid2, SIGUSR2);
ret=wait(&n);
if (WIFSIGNALED(n)) {
    printf("Message_5::_P5=%d_et_R5=%d\n", ret, WTERMSIG(n));
} else {
    printf("Message_6::_P6=%d_et_R6=%d\n", ret, WEXITSTATUS(n));
}
kill(pid1, SIGINT);
ret=wait(&n);
if (WIFSIGNALED(n)) {
    printf("Message_7::_P7=%d_et_R7=%d\n", ret, WTERMSIG(n));
} else {
    printf("Message_8::_P8=%d_et_R8=%d\n", ret, WEXITSTATUS(n));
}
bilan=-999;
ret=read(q[0], &bilan, sizeof(float));
if (ret>0) {
    bilan = bilan * 2;
}
printf ( "_bilan=%f\n", bilan);
```

# Systèmes centralisés : TD5

**Pid1** // lecteur sur q[0] - // lecteur sur p[0] et rédacteur sur p[1]

**Pid2** // lecteur sur p[0] et lecteur sur q[0]

```
    } else { /* -----> pid2 == 0 */
        close(p[1]); /* Ligne 92 */
        close(q[1]);
        printf ("Message_9::P9=%d de pere PP9=%d\n", (int) getpid(), getppid());
        pause();
        execlp("ps", "ps", NULL);
        ret=wait(&n); /* Ligne 97 */
    }
} else { /* -----> pid1 == 0 */
    pid3=fork();
    if (pid3 != 0) {
        close(q[1]); /* Ligne 102 */
        printf(".....pid3=%d\n", (int) pid3);
        printf ("Message_10::P10=%d de pere PP10=%d\n", (int) getpid(), getppid);
        for (;;) { /* boucle infinie, Ligne 105 */
            iBoucle ++;
        }
    }
}
```

# Systèmes centralisés : TD5

**Pid3** // lecteur sur p[0] – rédacteur sur q[1]

```
/* Ligne 108 */
} else { /* -----> pid3 == 0 */
    close(p[1]);
    close(q[0]);
    nbpair = 0;
    nbtotal=0;
    nbpairmoyen = -1;
    nbtotal=0;
    while ( read (p[0], &i, sizeof(int)) > 0 ) { /* Ligne 116*/
        nbtotal = nbtotal + 1;
        if (i==0) nbpair++;
    }
    if (nbtotal!=0) nbpairmoyen = nbpair/nbtotal;
    printf("nbpair=%f\n", nbpair);
    printf("nbpairmoyen=%f\n", nbpairmoyen);
    write (q[1], &nbpairmoyen, sizeof(float));
}
```

# Systèmes centralisés : TD5

## Père :

envoie 5 USR1 à pid1

envoie USR2 à pid2

wait pid2, message exit

envoie INT à pid1

wait pid1, message signaled

lecture bilan sur q[0]

affichage biln\*2

arrêt

**pid1** : seul le 1<sup>er</sup> USR1  $\Rightarrow$  message H1

5 écritures sur p[1]

**pid3** : 5 lectures sur p[0]

**pid2** : message H2, execlp, arrêt

**pid1** : tué

**pid3** : 1 écriture sur q[1], arrêt

# Systèmes centralisés : TD5

## ➤ Un exemple plus simple

**père**

boucle

read (p1[0], buf, sizeof(buf))

...

...

... p2

read (p2[0], buf, sizeof(buf))

...

...

... p3

read (p3[0], buf, sizeof(buf))

...

...

p1



**Fils1**

boucle

sleep (100)

write (p1[1], buf, sizeof(buf))



**Fils2**

boucle

sleep (50)

write (p2[1], buf, sizeof(buf))



**Fils3**

boucle

sleep (1)

write (p3[1], buf, sizeof(buf))



# Systèmes centralisés : TD5

## ➤ Rendre la lecture non bloquante avec fcntl

La commande `fcntl` permet de consulter ou d'affecter des attributs relatifs au descripteur d'un fichier (`F_GETFD`/`F_SETFD`), ou à son mode d'ouverture (`F_GETFL`/`F_SETFL`)

**`int fcntl (int descripteur, int commande, int valeur);`**

<b><code>F_GETFD</code></b>	lecture des attributs descripteur
<b><code>F_SETFD</code></b>	affectation de la valeur fournie en 3ème paramètre aux attributs descripteur
	Attribut (pré)défini : <b><code>FD_CLOEXEC</code></b> , fermeture sur recouvrement ( <code>exec</code> )
<b><code>F_GETFL</code></b>	lecture des attributs propres au mode d'ouverture
<b><code>F_SETFL</code></b>	affectation de la valeur fournie en 3ème paramètre aux attributs du mode d'ouverture
	Attributs (pré)définis :
	<b><code>O_APPEND</code></b> , écriture en fin de fichier
	<b><code>O_SYNC</code></b> , écriture directe, sans cache
	<b><code>O_NONBLOCK</code></b> , E/S non bloquantes
	(retour = -1 et <code>errno</code> = <code>EAGAIN</code> si situation de blocage)

**Note** : les attributs peuvent être combinés au moyen du ou (`|`)

### Exemple

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

permet de rendre non bloquantes les E/S sur le descripteur `desc`



# Systèmes centralisés : TD5

## Père

```
fcntl (p1[0], F_SETFL, fcntl (p1[0], F_GETFL) | O_NONBLOCK) ;
fcntl (p2[0], F_SETFL, fcntl (p1[0], F_GETFL) | O_NONBLOCK) ;
fcntl (p3[0], F_SETFL, fcntl (p1[0], F_GETFL) | O_NONBLOCK) ;
boucle
    nlus = read (p1[0], buf, sizeof(buf)) ;
    if (nlus > 0) {
        // traiter les données lues
    } else if (nlus == 0) {
        // le pipe est fermé en écriture
    } else if (nlus == -1 && ERNO == EAGAIN) {
        // rien dans le pipe
    } else {
        // erreur
    }
}
```

# Systèmes centralisés : TD5

## ➤ **Select : vérifier un ensemble de descripteurs**

La commande `select` permet de gérer la scrutation sur un ensemble de descripteurs.

```
int select (int nbdesc,  
            fd_set *readfds, fd_set *writefds, fd_set *exceptfds,  
            struct timeval *timeout)
```

0 : délai écoulé  
> 0 : nombre de  
descripteurs prêts

majorant du plus  
grand numéro de  
descripteur scruté

Délai de garde  
pour la scrutation  
(NULL : attente infinie)

**Entrée** : ensembles des descripteurs scrutés  
pour la disponibilité de caractères en lecture, en  
écriture, ou « hors bande » (sockets TCP)  
**Retour** : ensembles des descripteurs trouvés prêts

## Notes

- Les ensembles sont des tableaux de bits, de même dimension que les tables de descripteurs
- Des macros (définies dans `<types.h>`) permettent de manipuler ces ensembles (`fd_set`) :
  - ◇ **FD\_ZERO** (&fds) initialise l'ensemble de descripteurs `fds` à 0 (ensemble vide)
  - ◇ **FD\_SET**(fd, &fds) ajoute le descripteur `fd` à l'ensemble `fds`
  - ◇ **FD\_CLR** (fd, &fds) supprime le descripteur `fd` de `fds`
  - ◇ **FD\_ISSET**(fd, &fds) teste si `fd` appartient à `fds` (non nul si `fd` appartient à `fds`)

# Systèmes centralisés : TD5

boucle

```
fd_set readfds;
FD_ZERO(&readfds);
// select modifie readfds ⇒ il faut le réinitialiser à chaque passage
FD_SET(p1[0], &readfds);
FD_SET(p2[0], &readfds);
FD_SET(p3[0], &readfds);
res = select(NBDESC,&readfds,NULL, NULL, NULL);
if (res>0) {
    if (FD_ISSET(p1[0],&readfds)) {           // données à lire
        nlus = read (p1[0], buf, sizeof(buf)) ;
        ....
    }
    ...
}
```