

	EX - Examen n°1 (corrigé)	
Thomas Baldaquin - Juan Angel Lorenzo - Son Vu	Programmation Système et Réseau.	
ING2-GSI	Année 2017-2018	

Modalités

- Durée : 2 heures.
- Vous devez rédiger votre copie à l'aide d'un stylo à encre exclusivement.
- Toutes vos affaires (sacs, vestes, trousse, etc.) doivent être placées à l'avant de la salle.
- Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.
- Aucune machine électronique ne doit se trouver sur vous ou à proximité, même éteinte.
- Aucune sortie n'est autorisée avant une durée incompressible d'une heure.
- Aucun déplacement n'est autorisé.
- Aucun échange, de quelque nature que ce soit, n'est possible.

Questions courtes (8 points)

- 1 | Expliquez la différence entre un processus et un programme produit par compilation.

- 2 | Énumérez les états du cycle de vie d'un processus.

- 3 | Quelle est la différence entre les appels système `wait()` et `waitpid()` ?

- 4 | Expliquez qu'est-ce que le *recouvrement* d'un processus.

- 5 | Expliquez à quoi sert les signaux dans le contexte des communications inter-processus.

- 6 | Que veut dire qu'un signal, comme SIGKILL, est *non déroutable*?

- 7 | Expliquez la différence entre les tubes anonymes et nommés (*unnamed* et *named pipes*).

Les tubes anonymes n'apparaissent pas sur le système de fichiers et ne peuvent être utilisés que pour communiquer des processus sur lesquels il y a une relation père-fils. Ils sont automatiquement terminés lorsque les processus finissent.

Les tubes nommés apparaissent sur le système de fichiers et peuvent être utilisés pour communiquer processus qui n'ont pas des relations père-fils (s'ils ont les bons droits d'accès).

- 8 | Donnez au moins un avantage des files de messages par rapport à l'utilisation des tubes pour l'envoi d'information entre processus.

Une possible réponse : Avec la file des messages, un processus peut émettre des messages vers plusieurs processus, par l'intermédiaire d'une même file de message (multiplexage).

Exercice 1 (3 points)

9

Dans le programme suivant, un processus père est censé créer 4 fils qui vont générer et retourner une valeur aléatoire au père.

- Pourquoi faut-il ajouter **break** dans la ligne 15? (1 pt)
- Expliquez le contenu du tableau `pid_list` du père à partir de la ligne 18 (1 pt).
- Expliquez quel sera le contenu du tableau `res` à la fin du programme (1 pt).

□

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6
7 int main (int argc, char ** argv) {
8     int nfils = 4;
9     pid_t pid_list[nfils];
10    int i, res[nfils];
11
12    for (i = 0; i < nfils; i++){
13        pid_list[i] = fork();
14        if (pid_list[i] == 0) {
15            break;
16        }
17    }
18
19    if ((pid_list[i] == 0) && (i != nfils)) {
20        return i*rand();
21    }
22    else {
23        for (i = 0; i < nfils; i++)
24        {
25            waitpid(pid_list[i], res+i, 0);
26            printf("res: %d\t", WEXITSTATUS(res[i]));
27        }
28    }
29 }
```

Exercice 2 (3 points)

10

La fonction `sigaction()` permet de spécifier ou de connaître le comportement d'un processus à la réception d'un signal donné. Sa syntaxe est la suivante :

```
int sigaction(int signum, const struct sigaction *act, struct
              sigaction *oldact);
```

où `signum` est le numéro de signal, `act` est la fonction *handler* à exécuter lorsque `signum` est reçu, et `oldact` est une sauvegarde de l'ancienne action, que dans notre cas sera toujours `NULL`.

`SIGINT` est le numéro de signal qui correspond à la séquence «CTRL+C» sur le clavier du terminal de contrôle. `SIG_DFL` permet de restaurer la réaction par défaut d'un signal. Complétez le code suivant (à partir des lignes 14 et 22) pour que, chaque fois que l'utilisateur appuie sur «CTRL+C», le programme détecte cette séquence, montre un message à l'utilisateur et lui demande s'il souhaite rétablir la fonction par défaut de la séquence. Dans l'affirmatif, le programme devra rétablir la fonction par défaut.

□

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5
6 void MonHandler(int);
7 struct sigaction act;
8
9 void MonHandler(int sig) {
10     printf("Vous avez appuyé sur Ctrl-C\n");
11     printf("Voulez-vous rétablir la fonction par défaut de CTRL-C ?\n");
12     char c = getchar();
13     if (c == 'y' || c == 'Y'){
14         /* Complétez */
15         printf("Fonction par défaut rétabli\n");
16     }
17 }
18
19 int main(int argc, char *argv[]) {
20     act.sa_handler = MonHandler;
```

```
21     /* Complétez */
22
23     while (1) {
24         // On attend CTRL-C
25     }
26 }
```

Réponse :

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <stdlib.h>
4  #include <unistd.h>
5
6  void MonHandler(int);
7  struct sigaction act;
8
9  void MonHandler(int sig) {
10     printf("Vous avez appuyé sur Ctrl-C\n");
11     printf("Voulez-vous rétablir la fonction par défaut de CTRL-C ?\n");
12     char c = getchar();
13     if (c == 'y' || c == 'Y'){
14         act.sa_handler = SIG_DFL;
15         sigaction(SIGINT, &act, NULL);
16         printf("Fonction par défaut rétabli\n");
17     }
18 }
19
20 int main(int argc, char *argv[]) {
21
22     act.sa_handler = MonHandler;
23     sigaction(SIGINT, &act, NULL);
24
25     while (1) {
26         // On attend CTRL-C
27     }
28 }
```

Exercice 3 (3 points)

11

Écrivez deux programmes qui communiquent par tube nommé. Le premier effectue les actions suivantes :

- Création d'un tube nommé `monTube` en lecture et en écriture pour le user et le groupe.
- Demande d'un descripteur en écriture.
- Écriture d'un message.
- Libération du descripteur.

Le deuxième effectue les actions suivantes:

- Demande d'un descripteur en lecture sur le tube nommé.
- Lecture et affichage du message.
- Libération du descripteur.

□

Exercice 4 (3 points)

12

Le code suivant est un extrait d'un programme qui démarre un serveur dans le port 8000.

- Dans la ligne 18, que signifie l'affectation de la variable `s_addr` avec la valeur `INADDR_ANY`? (1 pt)
- À quoi sert la fonction `bind()` dans la ligne 23? (1 pt)
- Pourquoi faut-il fermer le socket `socket_serv` dans la ligne 43? (1 pt)

□

```
1 #define PORT 8000
2 ...
3
4 int main(int argc, char **argv){
5 ...
6 int socket_serv, socket_conex;
7 socklen_t socket_length;
8
```

```
9  struct sockaddr_in socket_remote;
10 socklen_t socket_remote_length;
11
12 if( (socket_serv=socket(PF_INET, SOCK_STREAM, 0)) < 0 ) {
13     perror("Le socket serveur ne peut pas être créé.");
14     exit ( EXIT_FAILURE );
15 }
16
17 socket_address.sin_family = AF_INET;
18 socket_address.sin_addr.s_addr = htonl(INADDR_ANY);
19
20 socket_address.sin_port = htons(PORT);
21 socket_length = sizeof(struct sockaddr_in);
22
23 if( (bind( socket_serv, (struct sockaddr *) &socket_address, socket_length )) < 0 ) {
24     perror("Erreur bind");
25     exit ( EXIT_FAILURE );
26 }
27
28 if( (listen( socket_serv, N_CONEX )) < 0) {
29     perror("Erreur listen");
30     exit ( EXIT_FAILURE );
31 }
32
33 socket_remote_length = sizeof(struct sockaddr_in);
34 while(1) {
35     if( (socket_conex=accept( socket_serv, (struct sockaddr *)
36         &socket_remote, &socket_remote_length)) < 0 ) {
37         perror("Erreur accept");
38         exit ( EXIT_FAILURE );
39     }
40
41     pid_t child_proc = fork();
42     if(child_proc == 0) {
43         if( (close( socket_serv )) < 0)
44             exit ( EXIT_FAILURE );
45
46         /* Processement de l'information */
47
48         exit( EXIT_SUCCESS );
49     }
50     else if(child_proc > 0) {
```

```
51         if ( (close( socket_conex )) < 0) {
52             exit ( EXIT_FAILURE );
53         }
54     }
55 }
56 ...
57 exit( EXIT_SUCCESS );
58 }
```

INADDR_ANY : socket attaché à toutes les interfaces locales.