Système d'exploitation II

TP 2 : Héritage et communication entre Processus

Le présent TP a pour objectif l'étude des mécanismes de communication entre les processus hiérarchiques.

Exercice 1 : Passage des variables :

Dans cet exercice nous proposons d'étudier le passage de variable entre le processus père et fils.

- Implémenter un programme (prog2.c) qui permet de créer un processus fils et père.
- Dans le même programme, déclarer une variable **m** de type entier, avant d'accéder aux zones de code des deux processus. Afficher l'adresse et le contenu de la variable dans les deux processus. Qu'est ce que vous remarquez ?
- Exécuter le programme suivant (prog3.c) :

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

int n=1000;

int main(){
    int m=1000, pid;
    printf("Adresse de n dans le père: %p\n", &n);
    printf("Adresse de m dans le père: %p\n", &m);
    printf("Valeurs de m et n dans le père : %d %d\n ", m, n);
    switch(pid=fork()){
        case    -1:
            perror("fork");
            exit(2);
        case    0 :     /* on est dans le processus fils*/
            printf("Adresse de n dans le fils: %p\n", &n);
            printf("Adresse de m dans le fils: %p\n", &m);
            printf("Valeurs de m et n dans le fils : %d %d\n", m, n);
            m*=2; n*=2;
            printf("Valeurs de m et n dans le fils : %d %d\n", m, n);
            sleep(3);
            exit(0);
        default:    /*on est dans le processus père*/
            sleep(2);
            printf("Valeurs de m et n dans le père : %d %d\n", m, n);
            m*=3; n*=3;
            printf("Valeurs de m et n dans le père : %d %d\n", m, n);
            sleep(2);
            exit(0);
    }
    return 0;
}
```

- Suivez l'évolution des valeurs des variables m, n, qu'est ce que vous remarquer ?

Solution :

Voici un exemple d'exécution du programme :

```
Adresse de n dans le père: 0x601054
Adresse de m dans le père: 0x7fff02d50948
Valeurs de m et n dans le père : 1000 1000
Adresse de n dans le fils: 0x601054
Adresse de m dans le fils: 0x7fff02d50948
Valeurs de m et n dans le fils : 1000 1000
Valeurs de m et n dans le fils : 2000 2000
Valeurs de m et n dans le père : 1000 1000
Valeurs de m et n dans le père : 3000 3000
```

Chaque processus possède sa propre copie des variables après le fork(). Les modifications faites dans un processus ne sont pas visibles dans l'autre. Ceci illustre le principe d'isolation entre processus dans les systèmes d'exploitation utilisant des mémoires virtuelles.

Les adresses affichées pour n et m sont identiques entre le parent et le fils. Cela ne signifie pas que les variables sont partagées, mais plutôt que leur disposition en mémoire virtuelle est identique avant que les modifications ne les séparent physiquement (via Copy-On-Write).

Exercice 2 : Les Pipes

La présente partie a pour objectif la mise en place d'un moyen de communication entre les processus appartenant à la même famille. Pour cela nous proposons l'utilisation des pipes vus en cours.

Exécuter le programme suivant. Avant de répondre aux questions qui suivent**.**

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

int main() {

    int fils1, fils2, p[2]; char buf[5];
    pipe(p);                /* création de pipe */
    if (fils1=fork()==0)    /* création du premier fils */
    {
        printf("je suis le fils producteur \n");
        printf("j'écris 5 caractères dans le pipe \n");
        write(p[1],"ABCDE",5);
        printf("fin d'écriture dans le pipe \n");
        exit(3);
    }
    else                    /* le père crée le fils consommateur */
    {
```

```
        if (fils2=fork()==0)         /* création du deuxième fils */
        {
            printf("je suis le fils consommateur \n");
            read(p[0],buf,5);   /* lecture du pipe */
            printf("voici les caractères lus \n");
            write(1,buf,5);/*affichage des caractères sur output standard*/
            printf("\n");
            exit(3);
        }
        else{
            while wait(NULL)!=-1;
            printf("processus père c'est fini …. \n");
        }
    }
    return 0;
}
```

1. Combien de processus existent dans le code. Identifier pour chaque processus la partie d'exécution.
2. Remplir le pipe jusqu'à dépasser sa taille. Qu'est-ce que vous remarquez ? Expliquer.
3. Modifier le code afin de lire à partir d'un pipe vide. Qu'est-ce que vous remarquez ? Expliquer.
4. Modifier le code pour lire deux fois les mêmes données du pipe. Qu'est-ce que vous remarquez ? Expliquer.
5. Modifier le programme pour que le père envoie un message au premier fils.
6. Ecrire un programme (prog4.c) qui permet :
    - La création de deux fils (fils 1 et fils 2).
    - La création d'un processus fils (fils 3) de fils1.
    - Le fils 3 crée un pipe avant d'envoyer un message au grand père (main). Qu'est ce que vous remarquez ? Expliquer.

Solution :

Lorsque vous créez un pipe en utilisant l'appel système pipe(), il vous fournit deux descripteurs de fichier :
Pipe pour la lecture : Le premier descripteur de fichier (habituellement p[0]) est utilisé pour lire les données du pipe. Il représente l'extrémité de lecture du pipe. Un processus utilise ce descripteur pour lire les données envoyées dans le pipe par un autre processus.
Pipe pour l'écriture : Le deuxième descripteur de fichier (habituellement p[1]) est utilisé pour écrire des données dans le pipe. Il représente l'extrémité d'écriture du pipe. Un processus écrit des données dans le pipe en utilisant ce descripteur, qui peuvent ensuite être lues par un autre processus à partir de l'extrémité de lecture.

1. Il y a trois processus, le processus parent et deux fils crées avec deux appels de fork.

2. Si le nombre de bytes que vous demandez à write d'écrire est plus petit que la taille réelle du message (par exemple, le nombre de caractères dans une chaîne de caractères), alors seuls les premiers n bytes du message seront écrits, où n est le nombre de bytes que vous avez spécifié. Les données restantes ne seront pas écrites.

   Exemple : write(p[1], "Bonjour", 3) ne va écrire que "Bon".

3. Il suffit de ne pas écrire le message dans le pipe. Faut commenter ou supprimer l'instruction d'écriture :
   `// write(p[1],"ABCDE",5);`

   Il est observé que le second processus fils est dans un état de blocage, car il attend une valeur à partir du pipe. Cette situation est similaire à celle où l'exécution d'un programme est suspendue, en attente de la saisie d'une valeur par l'utilisateur. Le processus parent est aussi en pause car il attend à son tour la fin du second processus fils.

4. Voici la modification à faire :

```
printf("je suis le fils consommateur \n");
read(p[0],buf,5);   /* lecture du pipe */
printf("voici les caractères lus \n");
printf("%s\n",buf); /*affichage des caractères sur output standard*/
// Effectuer une deuxième lecture
read(p[0],buf,5);   /* lecture du pipe */
printf("voici les caractères lus \n");
printf("%s\n",buf); /*affichage des caractères sur output standard*/
```

Remarque : Le second processus fils va lire le premier message envoyé, mais il se bloque encore une fois à cause de l'attend d'un deuxième message de la pipe.

5. Le programme :

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

int main() {

    int fils1, fils2, p[2]; char buf[5];
    pipe(p);                /* création de pipe */
    if (fils1=fork()==0)    /* création du premier fils */
    {
        printf("je suis le fils consommateur \n");
        read(p[0],buf,5);   /* lecture du pipe */
        printf("voici les caractères lus \n");
        write(1, buf, 5); /*affichage des caractères sur output standard*/
```

```
        printf("\n");
        exit(3);
    }
    else                    /* le père crée le fils consommateur */
    {
        if (fils2=fork()==0)         /* création du deuxième fils */
        {
            exit(3);
        }
        else{
            printf("je suis le père producteur \n");
            write(p[1],"ABCDE",5);  /* Le père envoie le message */
            while wait(NULL)!=-1;
            printf("processus père c'est fini …. \n");
        }
    }

    return 0;
}
```

6. Le programme prog4.c :

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

int main() {

    int fils1, fils2, fils3, p[2]; char buf[5];

    if (fils1=fork()==0)    /* création du premier fils */
    {
        if (fils3=fork()==0)         /* création du troisième fils */
        {
            pipe(p);                /* création de pipe */
            printf("je suis le fils producteur \n");
            write(p[1],"ABCDE",5);  /* Le fils envoie le message */
            exit(3);
        }

        exit(3);
    }
    else                    /* le père crée le fils consommateur */
    {
        if (fils2=fork()==0)         /* création du deuxième fils */
        {
```

```
            exit(3);
        }
        else{
            printf("je suis le père consommateur \n");
            read(p[0],buf,5);   /* lecture du pipe */
            printf("voici les caractères lus \n");
            printf("%s\n",buf); /*affichage des caractères sur output standard*/

            while wait(NULL)!=-1;
            printf("processus père c'est fini …. \n");
        }
    }
    return 0;
}
```

Le pipe(p) est créé après la bifurcation (fork()) du troisième fils, ce qui signifie que seul ce processus fils (le producteur) aura accès à ce pipe. Les autres processus (le père et le deuxième fils) ne seront pas au courant de l'existence de ce pipe et donc ne pourront pas lire à partir de celui-ci. Les messages d'affichage du père pourraient ne pas s'afficher correctement ou conduire à un comportement indéfini à cause de l'accès à un pipe non initialisé.

Exercice 3: Les Pipes

Déterminer la taille d'un tube. Pour permettre à l'écrivain de continuer avant même que le lecteur n'a 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 :
Ecrire un programme « prog.c » 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.

Solution:

Question 1:

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

int 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 %d octets (de %d, manque %d)\n", ecrit, n, n-ecrit);

    return 0;
}
```

Question 2:

$ gcc prog.c -o prog && ./prog 65536
J'ai ´ecrit 65536 octets (sur 65536, manque 0)

$ gcc prog.c -o prog && ./prog 65537
Bloquage

Exercice 4: Les Pipes

Créer un programme en C qui initialise une chaîne de communication entre plusieurs processus à l'aide de pipes. Le programme doit créer trois processus (en comptant le processus parent) qui communiquent en séquence : le parent écrit un message au premier enfant, le premier enfant modifie ce message et le passe au second enfant, et le second enfant affiche le message final.

Étapes :

1. Le processus parent crée un pipe et un processus enfant (Enfant 1).
2. Le processus parent envoie un message "Hello from Parent" à l'Enfant 1 via le pipe.
3. L'Enfant 1 lit ce message, y ajoute " and Child 1", et envoie le nouveau message à l'Enfant 2 via un second pipe.
4. L'Enfant 2 lit le message final et l'affiche sur la sortie standard.
5. Chaque processus doit fermer les extrémités du pipe qu'il n'utilise pas.

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUFFER_SIZE 100

int main() {

    int pipe1[2], pipe2[2];
```

```
    int pid1, pid2;
    char buf[BUFFER_SIZE];

    // Créer le premier pipe pour la communication Parent -> Enfant 1
    pipe(pipe1);

    // Créer le second pipe pour la communication Enfant 1 -> Enfant 2
    pipe(pipe2);

    // Créer le premier enfant
    pid1 = fork();

    if (pid1 == 0) { /* Enfant 1 */
        // Lire le message du parent
        read(pipe1[0], buf, BUFFER_SIZE);

        // Modifier le message
        strcat(buf, " and Child 1");

        // Envoyer le message modifié à l'Enfant 2
        write(pipe2[1], buf, strlen(buf) + 1);

        exit(EXIT_SUCCESS);
    } else {
        // Créer le second enfant
        pid2 = fork();

        if (pid2 == 0) { /* Enfant 2 */

            // Lire le message modifié de l'Enfant 1
            read(pipe2[0], buf, BUFFER_SIZE);

            // Afficher le message
            printf("Le fils 2 a reçu : %s\n", buf);

            exit(EXIT_SUCCESS);
        } else { /* Nous sommes dans le processus parent */

            // Envoyer un message à l'Enfant 1
            char message[] = "Hello from Parent";
            write(pipe1[1], message, strlen(message) + 1);

            // Attendre les enfants pour terminer
            while wait(NULL)!=-1;
        }
    }

    return 0;
}
```

Après exécution vous aurez ce message :
Le fils 2 a reçu : Hello from Parent and Child 1