Séance 7 : Les tubes

Sommaire

- [x] Exercice 1
- [x] Exercice 2
- [x] Exercice 3
- [ ] Exercice 4
- [ ] Exercice 5
- [ ] Exercice 6
- [ ] Exercice 7

Exercice 1

On écrit le code suivant qui fonctionne correctement même lorsque le père attend avant le dédbut de son exécution.

```
int main()
{
    pid_t pid_fils = -1;

    int tube[2];
    pipe(tube);
    char buffer;

    pid_fils = fork();

    if (pid_fils != 0)
    {
        /* Code du père */
        //sleep(10);
        close(tube[0]);
        char *text = "Lorem ipsum dolor sit amet consectetur adipisicing
elit.";
        printf("<père> Écriture en cours...\n");
        write(tube[1], text, strlen(text));
        close(tube[1]);
        wait(NULL);
    }
    else
    {
        /* Code du fils */
        close(tube[1]);
        while (read(tube[0], &buffer, 1) > 0)
        {
            sleep(1);
            printf("<fils> Caractère lu : %c\n", buffer);
        }
        close(tube[0]);
```

```
        exit(EXIT_SUCCESS);
    }
}
```

Exercice 2

On procède de la même façon qu'à l'exercice précédent à l'exception qu'ici on ne lit pas une chaine de caractères, caractère par caractère mais un entier d'un seul coup d'où l'appel à la primitive `sizeof` qui prend en argument le type voulu (i.e. `int`).

```
int main()
{
    pid_t pid_fils = -1;
    int tube[2];
    pipe(tube);

    pid_fils = fork();

    if (pid_fils != 0)
    {
        /*Code du père*/
        close(tube[0]);
        int nb;
        printf("<Père> Veuillez entrer un nombre : ");
        scanf("%d", &nb);
        write(tube[1], &nb, (int)sizeof(int));
        close(tube[1]);
        wait(NULL);
    }
    else
    {
        /*Code du fils */
        int nb;
        read(tube[0], &nb, (int)sizeof(int));
        printf("<Fils> Le résultat est : %d\n", nb * nb);
        close(tube[0]);
        exit(EXIT_SUCCESS);
    }
}
```

Exercice 3

On réutilise le code de l'exercice 1, sauf qu'ici il y a deux consommateurs; et vu qu'aucun système de sémaphore n'a été mis en place la zone critique de chaque fils n'est pas protégée ce qui fait que, selon les exécutions, l'ordonnanceur fait des choix différents ce qui résulte en des affichages différents.

```
int main()
{
```

```c
    pid_t pid_fils1 = -1;
    pid_t pid_fils2 = -1;

    int tube[2];
    pipe(tube);
    char buffer;

    pid_fils1 = fork();

    if (pid_fils1 != 0)
    {
        pid_fils2 = fork();

        if (pid_fils2 != 0)
        {
            /* Code du père */
            close(tube[0]);
            char *text = "Lorem ipsum dolor sit amet consectetur
adipisicing elit.";
            printf("<Père> Écriture en cours...\n");
            write(tube[1], text, strlen(text));
            close(tube[1]);
            wait(NULL);
        }
        else
        {
            /* Code du fils 2*/
            close(tube[1]);
            while (read(tube[0], &buffer, 1) > 0)
            {
                sleep(1);
                printf("<Fils2> Caractère lu : %c\n", buffer);
            }
            close(tube[0]);
            exit(EXIT_SUCCESS);
        }
    }
    else
    {
        /* Code du fils 1 */
        close(tube[1]);
        while (read(tube[0], &buffer, 1) > 0)
        {
            sleep(1);
            printf("<Fils1> Caractère lu : %c\n", buffer);
        }
        close(tube[0]);
        exit(EXIT_SUCCESS);
    }
}
```

Exercice 4