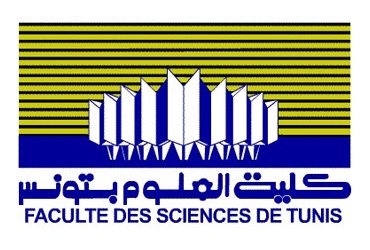
***Gestion Avancée des Processus et des Signaux en C sous linux***

**Réalisée par : Haddad Narmine**

**Année universitaire : 2023-2024**

[](http://www.utm.rnu.tn/)

***I-Introduction***

**L'objectif de ce projet est de développer un programme en C sous Linux qui illustre la gestion avancée des processus et des signaux. Le programme doit créer plusieurs processus fils à partir d'un processus père, gérer la communication entre ces processus à l'aide de signaux, et mettre en œuvre des mécanismes de synchronisation pour coordonner les tâches des processus fils.**

***II-Objectifs***

 **Créer un processus père qui génère quatre processus fils.**

 **Le processus père doit envoyer des signaux aux processus fils pour leur demander d'exécuter des tâches spécifiques.**

 **Les processus fils doivent traiter les signaux reçus, effectuer des tâches complexes, et répondre en envoyant des signaux de confirmation au processus père.**

 **Le processus père doit afficher un message à chaque fois qu'il reçoit une confirmation d'un processus fils.**

 **Implémenter une synchronisation entre les processus fils pour qu'ils ne commencent leurs tâches qu'une fois qu'ils ont tous reçu un signal du père.**

 **Utiliser un mécanisme de sémaphore pour gérer la synchronisation.**

***III- contraintes***

** Utiliser les appels système fork(), kill(), signal(), et les sémaphores POSIX (sem\_open, sem\_wait, sem\_post).**

** Gérer les signaux SIGUSR1, SIGUSR2, SIGTERM pour la communication entre le père et les fils.**

** Utiliser sleep() pour simuler des tâches dans les processus fils**

***IV- code***

Voici le code pour chaque objectif :

**Objectif 1 : Créer un processus père qui génère quatre processus fils**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main() {

pid\_t pid;

for (int i = 0; i < 4; i++) {

pid = fork();

if (pid < 0) {

perror("fork échoué");

exit(1);

}

if (pid == 0) {

printf("Processus fils %d créé\n", getpid());

exit(0);

}

}

// Le processus père attend que tous les processus fils aient terminé

while (wait(NULL) > 0);

printf("Tous les processus fils ont été créés et terminés.\n");

return 0;

}

#### Objectif 2 : Le processus père envoie des signaux aux processus fils

#include <signal.h>

void handle\_signal(int signal) {

printf("Processus fils %d a reçu le signal %d\n", getpid(), signal);

exit(0);

}

int main() {

pid\_t pids[4];

for (int i = 0; i < 4; i++) {

pids[i] = fork();

if (pids[i] < 0) {

perror("fork échoué");

exit(1);

}

if (pids[i] == 0) {

signal(SIGUSR1, handle\_signal);

signal(SIGUSR2, handle\_signal);

while (1) pause(); // Attendre les signaux

}

}

sleep(1); // S'assurer que les processus fils sont prêts à recevoir des signaux

for (int i = 0; i < 4; i++) {

kill(pids[i], SIGUSR1);

sleep(1);

}

while (wait(NULL) > 0);

printf("Tous les processus fils ont traité leurs signaux et ont terminé.\n");

return 0;

}

#### Objectif 3 : Les processus fils traitent les signaux, effectuent des tâches et répondent

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/wait.h>

pid\_t ppid;

void handle\_signal(int signal) {

printf("Processus fils %d a reçu le signal %d\n", getpid(), signal);

sleep(2); // Simuler une tâche

kill(ppid, SIGUSR2); // Envoyer la confirmation au père

exit(0);

}

void parent\_signal\_handler(int signal) {

printf("Le processus père a reçu le signal de confirmation %d\n", signal);

}

int main() {

pid\_t pids[4];

ppid = getpid();

signal(SIGUSR2, parent\_signal\_handler);

for (int i = 0; i < 4; i++) {

pids[i] = fork();

if (pids[i] < 0) {

perror("fork échoué");

exit(1);

}

if (pids[i] == 0) {

signal(SIGUSR1, handle\_signal);

while (1) pause(); // Attendre les signaux

}

}

sleep(1); // S'assurer que les processus fils sont prêts à recevoir des signaux

for (int i = 0; i < 4; i++) {

kill(pids[i], SIGUSR1);

pause(); // Attendre la confirmation du fils

}

while (wait(NULL) > 0);

printf("Tous les processus fils ont traité leurs signaux et ont terminé.\n");

return 0;

}

#### Objectif 4 : Le processus père envoie un message aux processus fils pour confirmation

Le code ci-dessus satisfait déjà cet objectif, car le processus père attend un signal de confirmation de chaque processus fils

#### Objectif 5 : Synchronisation avec des sémaphores

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

#include <semaphore.h>

#include <fcntl.h>

sem\_t \*sem;

void handle\_signal(int signal) {

printf("Processus fils %d a reçu le signal %d\n", getpid(), signal);

sleep(2); // Simuler une tâche

sem\_post(sem); // Signaler l'achèvement de la tâche

exit(0);

}

int main() {

pid\_t pids[4];

sem = sem\_open("/sem\_example", O\_CREAT, 0644, 0);

if (sem == SEM\_FAILED) {

perror("sem\_open échoué");

exit(1);

}

for (int i = 0; i < 4; i++) {

pids[i] = fork();

if (pids[i] < 0) {

perror("fork échoué");

exit(1);

}

if (pids[i] == 0) {

signal(SIGUSR1, handle\_signal);

while (1) pause(); // Attendre les signaux

}

}

sleep(1); // S'assurer que les processus fils sont prêts à recevoir des signaux

for (int i = 0; i < 4; i++) {

kill(pids[i], SIGUSR1);

sem\_wait(sem); // Attendre que le processus fils termine sa tâche

}

while (wait(NULL) > 0);

sem\_close(sem);

sem\_unlink("/sem\_example");

printf("Tous les processus fils ont traité leurs signaux et ont terminé.\n");

return 0;

}

#### Objectif 6 : Utiliser sleep() pour simuler des tâches

Le code ci-dessus utilise sleep() pour simuler des tâches dans le gestionnaire de signaux des processus fils.

Ces implémentations respectent les contraintes spécifiées, en utilisant les appels système, les signaux et les sémaphores POSIX pour gérer la synchronisation et la communication entre les processus.