

Découverte des processus. Dans ce compte-rendu, n'hésitez pas à faire des phrases, enrichir vos explications de copie de code dans un terminal.

```
$ ./ma-commande avec des arguments
```

```
Elle fonctionne
```

```
$
```

### 1 Donnez un exemple de copie d'écran d'un terminal qui contient le résultat de la commande ls

```
omergs@omerpc:~$ ls
Bureau  Images  Musique  snap      Vidéos
Documents  Modèles  Public  Téléchargements
```

### 2 Présentez au moins 3 façons mettre un processus en arrière plan

On peut mettre un processus en arrière plan avec les touches CTRL+Z, mettre « & » à la fin de la commande de lancement ; utiliser nohup puis écrire la commande « nohup sleep 60 & »

### 3 Présentez au moins 3 façons de stopper un processus

CTRL+C ; killall nom\_du\_processus ; kill PID

### 4 Présentez au moins 3 façons de reprendre un processus

Les commandes sont : « fg », « bg », la commande disown permet de reprendre un processus suspendu et de le détacher

### 5 Créez un zombie (donner le code et une trace)

```
#!/bin/bash

# Créer un processus enfant
(
    echo "Enfant (PID=$$) se termine."
    sleep 1
    exit 0
) &
```

```
# Récupérer l'ID du processus enfant
CHILD_PID=$!
```

```
# Ne pas attendre le processus enfant (ne pas utiliser wait ici)
echo "Parent (PID=$$) ne va pas attendre l'enfant (PID=$CHILD_PID)."
```

```
# Boucle infinie pour laisser le processus enfant devenir un zombie
while true; do
    sleep 10
done
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
rayanne	4231	0.0	0.0	0	0	?	Z	14:04	0:00	[sd_espeak-ng-mb] <defunct>
rayanne	5850	0.0	0.0	9216	2560	pts/0	S+	14:24	0:00	grep --color=auto Z

### 6 Créez un processus « démon » (donner le code et une trace)

```
#!/bin/bash
```

```
# Fonction appelée lorsque le processus reçoit un signal de fin
cleanup() {
    echo "Processus arrêté, nettoyage en cours..."
    # Placez ici les commandes de nettoyage nécessaires
    exit 0
}
```

```
# Intercepte les signaux SIGINT (Ctrl+C) et SIGTERM (kill)
trap cleanup SIGINT SIGTERM
```

```
# Fonction principale du daemon
daemon_function() {
    while true; do
        # Remplacez ceci par le travail réel que le daemon doit effectuer
        echo "Daemon en cours d'exécution..."
        sleep 5
    done
}
```

```
# Démarrer la fonction principale en arrière-plan
daemon_function &
```

```
# Obtenir le PID du processus en arrière-plan
DAEMON_PID=$!
```

```
echo "Daemon démarré avec le PID $DAEMON_PID"
```

```
# Attendre que le processus enfant se termine  
wait $DAEMON_PID
```

## 7 Récupérez ou recopiez le code des programmes C : mkNamedSem, namedSemV, namedSemP et rmNamedSem

On a copié le code à partir du fichier donné. Pour l'exécuter on a utilisé la commande suivante :

```
chmod u+x compile.sh
```

```
./nameSemP
```

## 8 Donnez des exemples de création et d'utilisation de sémaphores

On peut par exemple créer des sémaphores avec le langage C++. Pour cela il faut la bibliothèque <semaphore.h>.

On doit ensuite créer l'objet semaphore en créant une variable de ce type.

Puis on peut l'initialiser avec : **int sem\_init(sem\_t \*semaphore, int pshared, unsigned int valeur);**

Pour bloquer le semaphore on va pouvoir utiliser la méthode suivante ; **int sem\_wait(sem\_t \*semaphore);**

On reçoit ensuite un nombre 0 si tout est ok, -1 si une erreur survient.

A cet instant le semaphore est bloqué, pour le débloquent on doit utiliser la méthode suivante : **int sem\_post(sem\_t \*semaphore);**

On reçoit ensuite un nombre 0 si tout est ok, -1 si une erreur survient.

## 9 Créez d'un carrousel

```
#!/bin/bash
```

```
LOCKFILE="/tmp/mylockfile.lock"
```

```
MAX_ATTEMPTS=10  
WAIT_TIME=1
```

```
# Function to create a lock file  
create_lock() {  
    ./mkNamedSem "/tmp/mylockfile.lock" 1  
}
```

```
# Function to check if the lock file is held by another process  
check_lock() {  
    if [ -f "$LOCKFILE" ]; then  
        local pid  
        pid=$(cat "$LOCKFILE")  
        if [ -d "/proc/$pid" ]; then  
            echo "used"  
            return 1 # Lock is held  
        else  
            echo "not used"  
            return 0 # Lock is stale  
        fi  
    else  
        return 0 # Lock file does not exist  
    fi  
}
```

```
# Try to acquire the lock  
attempts=0  
while ! check_lock; do  
    if [ $attempts -ge $MAX_ATTEMPTS ]; then  
        echo "Failed to acquire lock after $MAX_ATTEMPTS attempts, exiting..."  
        exit 1  
    fi  
    echo "Lock is held by another process. Waiting..."  
    sleep $WAIT_TIME  
    attempts=$((attempts + 1))  
done
```

```
create_lock  
echo "Lock acquired."
```

```
# Critical section  
echo "Performing critical operations..."  
sleep 20 # Simulate some work
```

```
# Release the lock  
./rmNamedSem "$LOCKFILE"  
echo "Critical operations done. Lock released."
```

## 10 Créez une application lecteurs/écrivains

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

sem_t rw_mutex; // Sémaphore pour les écrivains
sem_t mutex;    // Sémaphore pour les lecteurs
int read_count = 0;

void* reader(void* arg) {
    sem_wait(&mutex);
    read_count++;
    if (read_count == 1) {
        sem_wait(&rw_mutex); // Bloque les écrivains
    }
    sem_post(&mutex);

    // Section critique
    printf("Lecteur %ld lisant\n", (long)arg);

    sem_wait(&mutex);
    read_count--;
    if (read_count == 0) {
        sem_post(&rw_mutex); // Permet aux écrivains d'écrire
    }
    sem_post(&mutex);
    return NULL;
}

void* writer(void* arg) {
    sem_wait(&rw_mutex);

    // Section critique
    printf("Écrivain %ld écrivant\n", (long)arg);

    sem_post(&rw_mutex);
    return NULL;
}

int main() {
    pthread_t readers[5], writers[2];

    sem_init(&rw_mutex, 0, 1);
    sem_init(&mutex, 0, 1);

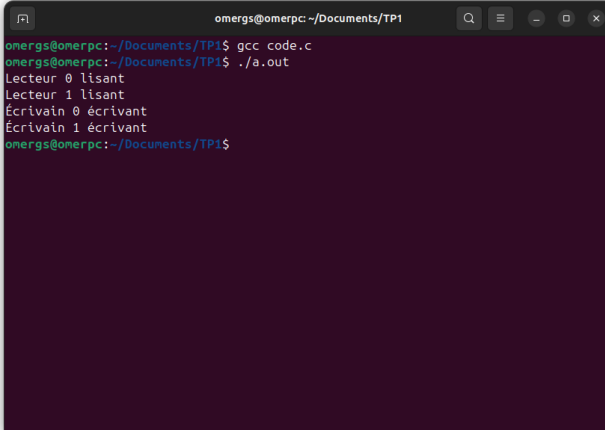
    for (long i = 0; i < 2; i++) {
        pthread_create(&readers[i], NULL, reader, (void*)i);
    }
    for (long i = 0; i < 2; i++) {
        pthread_create(&writers[i], NULL, writer, (void*)i);
    }
}
```

```
}

for (int i = 0; i < 5; i++) {
    pthread_join(readers[i], NULL);
}
for (int i = 0; i < 2; i++) {
    pthread_join(writers[i], NULL);
}

sem_destroy(&rw_mutex);
sem_destroy(&mutex);

return 0;
}
```



```
omer@omerpc: ~/Documents/TP1
omer@omerpc:~/Documents/TP1$ gcc code.c
omer@omerpc:~/Documents/TP1$ ./a.out
Lecteur 0 lisant
Lecteur 1 lisant
Écrivain 0 écrivant
Écrivain 1 écrivant
omer@omerpc:~/Documents/TP1$
```