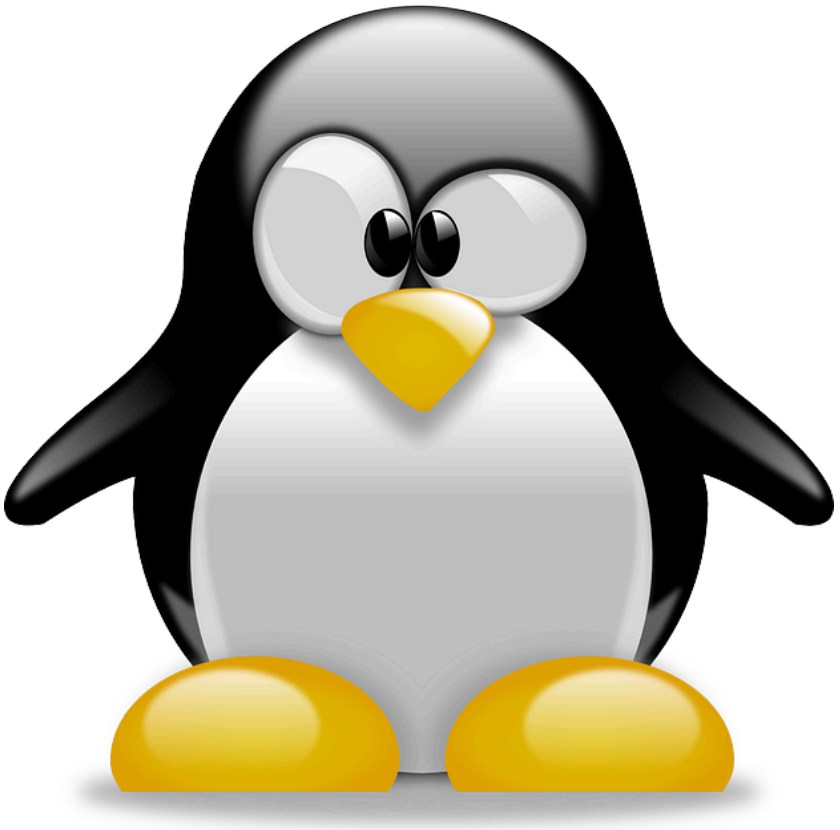
Fusi Belaid
Thomas
4A

Projet 3
ProgSys



1. Introduction

Ce projet de Programmation Système, dans sa troisième et dernière partie, vise à implémenter le fonctionnement de la politique d'ordonnancement **Round Robin** en Python. Cette politique, couramment utilisée dans les systèmes d'exploitation à temps partagé, permet d'allouer des portions égales de temps (quantuums) à chaque processus en file d'attente. Le projet met en œuvre un système dans lequel un ensemble de processus fils est géré par un processus parent, qui les ordonne et contrôle leur exécution en fonction de tranches de 15 secondes. À travers cette simulation, nous avons manipulé des signaux pour suspendre et reprendre l'exécution des processus, en utilisant les signaux UNIX tels que **SIGSTOP** et **SIGCONT**, ainsi que des alarmes pour gérer l'ordonnancement.

2. Exemple d’implémentation Round Robin

Nous allons essayer ici d’implémenter la gestion Round Robin des processus sur python:

```
import os
import signal
import time
import sys


liste_processus = []
processus_actuel = 0

def fin_programme(signum, frame):  # tue tous les processus et
quitte
   os.write(1, b"Fin du programme\n")
   global liste_processus
   for pid in liste_processus:
       os.kill(pid, signal.SIGKILL)
   sys.exit(0)



def creation_processus(signum, frame):
   global liste_processus

   pid = os.fork()
   if pid == 0:  # fils
       os.kill(os.getpid(), signal.SIGSTOP)  # on arrete le fils
       while True:
           os.write(1, f"Processus {os.getpid()} en cours
d'exécution\n".encode())
           time.sleep(3)

   else:  # père
       liste_processus.append(pid)
       message_ajout = f"Liste des processus après ajout :
{liste_processus}\n"
       os.write(1, message_ajout.encode())
       if len(liste_processus) == 1:
           # on démarre le premier processus
```

```
            os.kill(liste_processus[0], signal.SIGCONT)
            signal.alarm(15)



def changement_processus(signum, frame):
    global processus_actuel
    global liste_processus

    if liste_processus:
        # on arrete le processus actuel
        message_arret = f"Arrêt du processus
{liste_processus[processus_actuel]}\n"
        os.write(1, message_arret.encode())
        os.kill(liste_processus[processus_actuel], signal.SIGSTOP)

        # on change de processus
        processus_actuel += 1
        if processus_actuel >= len(liste_processus):
            processus_actuel = 0

        # on commence le nouveau processus
        message_demarrage = f"Démarrage du processus
{liste_processus[processus_actuel]}\n"
        os.write(1, message_demarrage.encode())
        os.kill(liste_processus[processus_actuel], signal.SIGCONT)

        # on relance l'alarme
        signal.alarm(15)



# omodif gestionnaires de signaux
signal.signal(signal.SIGINT, creation_processus)
signal.signal(signal.SIGALRM, changement_processus)
signal.signal(signal.SIGTERM, fin_programme)


os.write(1, f"PID du père : {os.getpid()}\n".encode())
os.write(1, b"Utilisez 'kill -SIGINT <PID>' pour envoyer un signal
qui va creer des processus\n")
```

```
# Boucle principale
while True:
    os.write(1, b"En attente de signaux\n")
    time.sleep(100)
```

3. Tests

Nous allons donc ouvrir 2 terminaux afin d'essayer notre programme, le premier servira à lancer le programme et à contrôler les affichages, le 2e servira à envoyer les signaux.

```
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet3$ python3 projet3.py
PID du père : 5618
Utilisez 'kill -SIGINT <PID>' pour envoyer un signal qui va creer des processus
En attente de signaux
Liste des processus après ajout : [5619]
Arrêt du processus 5619
Démarrage du processus 5619
Processus 5619 en cours d'exécution
Processus 5619 en cours d'exécution
Processus 5619 en cours d'exécution
Liste des processus après ajout : [5619, 5629]
Processus 5619 en cours d'exécution
Processus 5619 en cours d'exécution
Arrêt du processus 5619
Démarrage du processus 5629
Processus 5629 en cours d'exécution
Processus 5629 en cours d'exécution
Processus 5629 en cours d'exécution
Processus 5629 en cours d'exécution
Processus 5629 en cours d'exécution
Arrêt du processus 5629
Démarrage du processus 5619
Processus 5619 en cours d'exécution
Processus 5619 en cours d'exécution
Liste des processus après ajout : [5619, 5629, 5636]
Processus 5619 en cours d'exécution
Processus 5619 en cours d'exécution
Processus 5619 en cours d'exécution
Arrêt du processus 5619
Démarrage du processus 5629
Processus 5629 en cours d'exécution
Processus 5629 en cours d'exécution
Fin du programme
```

1er terminal

```
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet3$ kill -SIGINT 5618
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet3$ kill -SIGINT 5618
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet3$ kill -SIGINT 5618
perceval@pc-perceval-u:~/Documents/ProgSys/ProgSys-S3/Projets/Projet3$ kill -SIGTERM 5618
```

2e terminal

On envoie donc des signaux SIGINT au processus père qui en recevant ce signal va créer un fils et l'inscrire dans la file d'attente.

4. Conclusion

Ce projet nous a permis de simuler de manière pratique une politique d'ordonnancement **Round Robin**, en approfondissant notre compréhension des mécanismes d'ordonnancement des processus et des signaux sous UNIX. L'implémentation de cette politique en Python a été l’occasion d'apprendre à gérer la suspension et la reprise des processus à travers des signaux, tout en organisant efficacement une file d’attente. En suivant cette approche, nous avons développé une meilleure compréhension de la gestion des processus concurrents dans les systèmes d’exploitation, tout en pratiquant la gestion de signaux et d’alarmes, des concepts cruciaux pour la maîtrise des systèmes temps partagé.