PRACTILAL 1

import multiprocessing

import os

def execute\_same\_code():

print(f"Process {os.getpid()} executing same code")

def execute\_different\_code():

print(f"Process {os.getpid()} executing different code")

def child\_process():

execute\_same\_code()

def main():

print(f"Parent process ID: {os.getpid()}")

# a) Same program, same code

process\_a = multiprocessing.Process(target=child\_process)

process\_a.start()

process\_a.join()

# b) Same program, different code

process\_b = multiprocessing.Process(target=execute\_different\_code)

process\_b.start()

process\_b.join()

print("Parent process finished.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

PRACTILAL 4

import os

import sys

import stat

import time

def get\_file\_details(file\_path):

try:

# Get file stat information

file\_stat = os.stat(file\_path)

# Extract owner access permissions

permissions = stat.filemode(file\_stat.st\_mode)

# Get file access time

access\_time = time.ctime(file\_stat.st\_atime)

# Print file details

print(f"File: {file\_path}")

print(f"Owner access permissions: {permissions}")

print(f"File access time: {access\_time}")

except FileNotFoundError:

print(f"Error: File '{file\_path}' not found.")

except Exception as e:

print(f"Error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Check if a file name is provided as a command-line argument

if len(sys.argv) != 2:

print("Usage: python file\_details.py <file\_name>")

sys.exit(1)

file\_name = sys.argv[1]

get\_file\_details(file\_name)

PRACTILAL 5

import subprocess

import sys

def copy\_file(source\_path, destination\_path):

try:

# Using system call to copy file (Linux/Unix)

subprocess.run(["cp", source\_path, destination\_path])

# For Windows, you can use the following system call:

# subprocess.run(["copy", source\_path, destination\_path])

print(f"File copied from {source\_path} to {destination\_path}")

except Exception as e:

print(f"Error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Check if source and destination paths are provided as command-line arguments

if len(sys.argv) != 3:

print("Usage: python copy\_file.py <source\_path> <destination\_path>")

sys.exit(1)

source\_path = sys.argv[1]

destination\_path = sys.argv[2]

copy\_file(source\_path, destination\_path)

PRACTILAL 6

class Process:

def \_\_init\_\_(self, process\_id, arrival\_time, burst\_time):

self.process\_id = process\_id

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

def fcfs\_scheduler(processes):

processes.sort(key=lambda x: x.arrival\_time) # Sort processes based on arrival time

completion\_time = [0] \* len(processes)

turnaround\_time = [0] \* len(processes)

waiting\_time = [0] \* len(processes)

completion\_time[0] = processes[0].burst\_time

turnaround\_time[0] = completion\_time[0] - processes[0].arrival\_time

waiting\_time[0] = turnaround\_time[0] - processes[0].burst\_time

for i in range(1, len(processes)):

completion\_time[i] = completion\_time[i - 1] + processes[i].burst\_time

turnaround\_time[i] = completion\_time[i] - processes[i].arrival\_time

waiting\_time[i] = turnaround\_time[i] - processes[i].burst\_time

print("Process\t Arrival Time\t Burst Time\t Completion Time\t Turnaround Time\t Waiting Time")

for i in range(len(processes)):

print(f"{processes[i].process\_id}\t\t{processes[i].arrival\_time}\t\t{processes[i].burst\_time}\t\t"

f"{completion\_time[i]}\t\t\t{turnaround\_time[i]}\t\t\t{waiting\_time[i]}")

if \_\_name\_\_ == "\_\_main\_\_":

# Example processes

processes = [

Process(1, 0, 6),

Process(2, 2, 4),

Process(3, 4, 8),

Process(4, 6, 5),

]

fcfs\_scheduler(processes)

PRACTILAL 7

def optimal\_page\_replacement(pages, capacity):

page\_faults = 0

page\_frames = [-1] \* capacity

next\_use = [-1] \* capacity

for i, page in enumerate(pages):

if page not in page\_frames:

page\_faults += 1

if -1 in page\_frames:

# If there is an empty slot, insert the page

empty\_slot = page\_frames.index(-1)

page\_frames[empty\_slot] = page

next\_use[empty\_slot] = find\_next\_use(pages, i)

else:

# Find the page in page\_frames with the farthest next use

idx = next\_use.index(max(next\_use))

page\_frames[idx] = page

next\_use[idx] = find\_next\_use(pages, i)

print("Page Faults:", page\_faults)

def find\_next\_use(pages, current\_index):

for i in range(current\_index + 1, len(pages)):

if pages[i] in pages[current\_index + 1:]:

return i

return float('inf') # If the page is not used anymore, set next use to infinity

if \_\_name\_\_ == "\_\_main\_\_":

# Example pages reference string

pages\_reference = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1]

# Set the number of page frames (capacity)

page\_frame\_capacity = 3

optimal\_page\_replacement(pages\_reference, page\_frame\_capacity)

PRACTILAL 8

class Process:

def \_\_init\_\_(self, process\_id, arrival\_time, burst\_time):

self.process\_id = process\_id

self.arrival\_time = arrival\_time

self.burst\_time = burst\_time

def sjf\_scheduler(processes):

processes.sort(key=lambda x: (x.arrival\_time, x.burst\_time)) # Sort processes based on arrival time and burst time

completion\_time = [0] \* len(processes)

turnaround\_time = [0] \* len(processes)

waiting\_time = [0] \* len(processes)

completion\_time[0] = processes[0].burst\_time

turnaround\_time[0] = completion\_time[0] - processes[0].arrival\_time

waiting\_time[0] = turnaround\_time[0] - processes[0].burst\_time

for i in range(1, len(processes)):

completion\_time[i] = completion\_time[i - 1] + processes[i].burst\_time

turnaround\_time[i] = completion\_time[i] - processes[i].arrival\_time

waiting\_time[i] = turnaround\_time[i] - processes[i].burst\_time

print("Process\t Arrival Time\t Burst Time\t Completion Time\t Turnaround Time\t Waiting Time")

for i in range(len(processes)):

print(f"{processes[i].process\_id}\t\t{processes[i].arrival\_time}\t\t{processes[i].burst\_time}\t\t"

f"{completion\_time[i]}\t\t\t{turnaround\_time[i]}\t\t\t{waiting\_time[i]}")

if \_\_name\_\_ == "\_\_main\_\_":

# Example processes

processes = [

Process(1, 0, 6),

Process(2, 2, 8),

Process(3, 3, 4),

Process(4, 5, 3),

]

sjf\_scheduler(processes)