**Name: Chen Yu Lin**

**Task 1**

Draw 8-level multilevel feedback queues

Table, letter

Description automatically generated

Explain what happens when all user processes terminate and there is no other user process ready to execute.

When there are no other processes to execute, the CPU runs the idle process, which always has the lowest priority access. The idle process does some cleaning and switches the system into sleep state.

**Task 2**

Outline the multilevel feedback queues scheduling and power saving algorithm in pseudo-code.

def run\_system:

for queue in queues:

# Describe algorithm carried out by the idle process #

if queue == empty and blocked\_queue == empty:

# calls the idle process is ran since there are no processes to execute #

idle\_process()

for process in queue:

# process might get blocked for I/O operations #

if process == blocking for I/O:

pop process from current queue and append to block queue

process.priority =- 1 # boost priority level by 1 #

elif process == ready:

# execute the process #

execute(process)

# if process still isn’t finished after being executed for the duration of its time slice, move the process’s priority down one level #

if process == finished:

terminate(process)

else:

process.priority =+ 1

for process in blocked\_queue:

execute(process)

def execute (process):

while process.\_quantum not finished:

run process

def idle\_process():

print “idling”

def terminate(process):

queue.pop(process)

**Task 3**

Code for implementing the algorithm (I also submitted the py file with the code)

import time # using time for delays - simulating execution time of processes

import random

# Queue class

class Queue(object):

queues = []

blocked\_queue = []

def \_\_init\_\_(self):

for i in range(8):

self.queues.append([])

# Process class

class Process(object):

def \_\_init\_\_(self, pid, quantum, priority, io\_block):

self.\_pid = pid

self.\_quantum = quantum

self.\_priority = priority

self.\_io\_block = io\_block

self.\_status = "Ready"

def \_\_str\_\_(self):

return "Process id: " + str(self.\_pid) + ", quantum: " + str(self.\_quantum) + ", priority: " + str(self.\_priority) + ", i/o: " + str(self.\_io\_block) + ", status: " + self.\_status

def execute(process):

'''

Executing processes

Check for I/O interruption: If there's an I/O interrupt, update process status to 'blocked'

If the process quantum is 0, then process is finished and status is updated to 'complete'

Otherwise, process status remains at 'ready'

'''

timeslice = 2\*\*process.\_priority \* 10

print("Executing process: ", str(process), ", time slice: ", timeslice)

time.sleep(timeslice)

new\_quantum = 0

if process.\_quantum > timeslice:

new\_quantum = process.\_quantum - timeslice

for block in process.\_io\_block: # Checking for interruptions

if new\_quantum <= block or process.\_quantum == block:

process.\_quantum = block

process.\_status = "Blocked"

return process.\_status

process.\_quantum = new\_quantum

if process.\_quantum == 0:

process.\_status = "Complete"

return process.\_status

def unblock\_process(process):

'''

Process becomes ready once the I/O is executed

'''

time.sleep(list(process.\_io\_block.values())[0])

process.\_io\_block.pop(max(process.\_io\_block.keys()))

process.\_status = "Ready"

return process.\_status

def terminate(level, process):

'''

Terminating finished process from queue

'''

print("\n!! Process", q.queues[level][process].\_pid, "terminated !!\n")

q.queues[level].pop(process)

def idle\_process():

''' If there are no processes in queue or blocked queue, OS goes into idle state '''

print("There are no more processes in the queue to execute. Idle process will start soon")

time.sleep(10)

print("\n\nIdling...")

def run\_system():

''' OS system runs if not in idle process

For each queue level, each process within those queues are executed.

If not terminated and there are no I/O interrupts, process priority +1 (priority decreases)

If interrupted by I/O, process get moved into block queue and priority -1 (priority increases)

'''

queue\_index = 0

while queue\_index < len(q.queues):

level = q.queues[queue\_index]

process\_index = 0

while process\_index < len(level):

process = q.queues[queue\_index][process\_index]

status = execute(process)

if status == "Ready":

if queue\_index < len(q.queues)-1:

process.\_priority += 1 # unfinished process is moved down a level

q.queues[queue\_index+1][:0] = [level.pop(process\_index)] # unfinished process is moved to the front of the next queue

elif status == "Blocked":

print("\nProcess has been blocked! Process ID:", process.\_pid, "will be added to the blocked queue.\n")

if process.\_priority != 0:

process.\_priority -= 1 # blocked process is moved up a level

q.blocked\_queue.append(level.pop(process\_index))

else:

terminate(queue\_index, process\_index)

queue\_index += 1

if not any(q.queues) and not any(q.blocked\_queue):

idle\_process()

def blocked\_processes():

'''

Execute blocked processes

When the process is ready again, it is moved back into where it belongs in the multilevel queue

'''

print("Blocked queue contains: ")

for process in q.blocked\_queue:

print(str(process))

unblock\_process(process)

while len(q.blocked\_queue) != 0:

for index, process in enumerate(q.blocked\_queue):

q.queues[process.\_priority].append(q.blocked\_queue.pop(index))

run\_system()

**Task 4**

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

# Hardcoding the processes within the queues

q = Queue()

for level, queue in enumerate(q.queues):

process = Process(random.randint(1000, 9999), random.randint(100,999), level, {})

if level == 2 or level == 6:

process.\_io\_block[random.randint(1, process.\_quantum)] = random.randint(1, 3) # when the interruption will happen : its' duration

queue.append(process)

for level, queue in enumerate(q.queues):

for process in queue:

print("Queue", level, ":", str(process))

run\_system()

print("-" \* 80)

blocked\_processes()

**Creation of processes**

A black screen with white text

Description automatically generated with low confidence

**Execution of processes**

Text

Description automatically generated

**Checking blocked queue and executing blocked processes if any. If none, idle process will run.**

**Text

Description automatically generated**