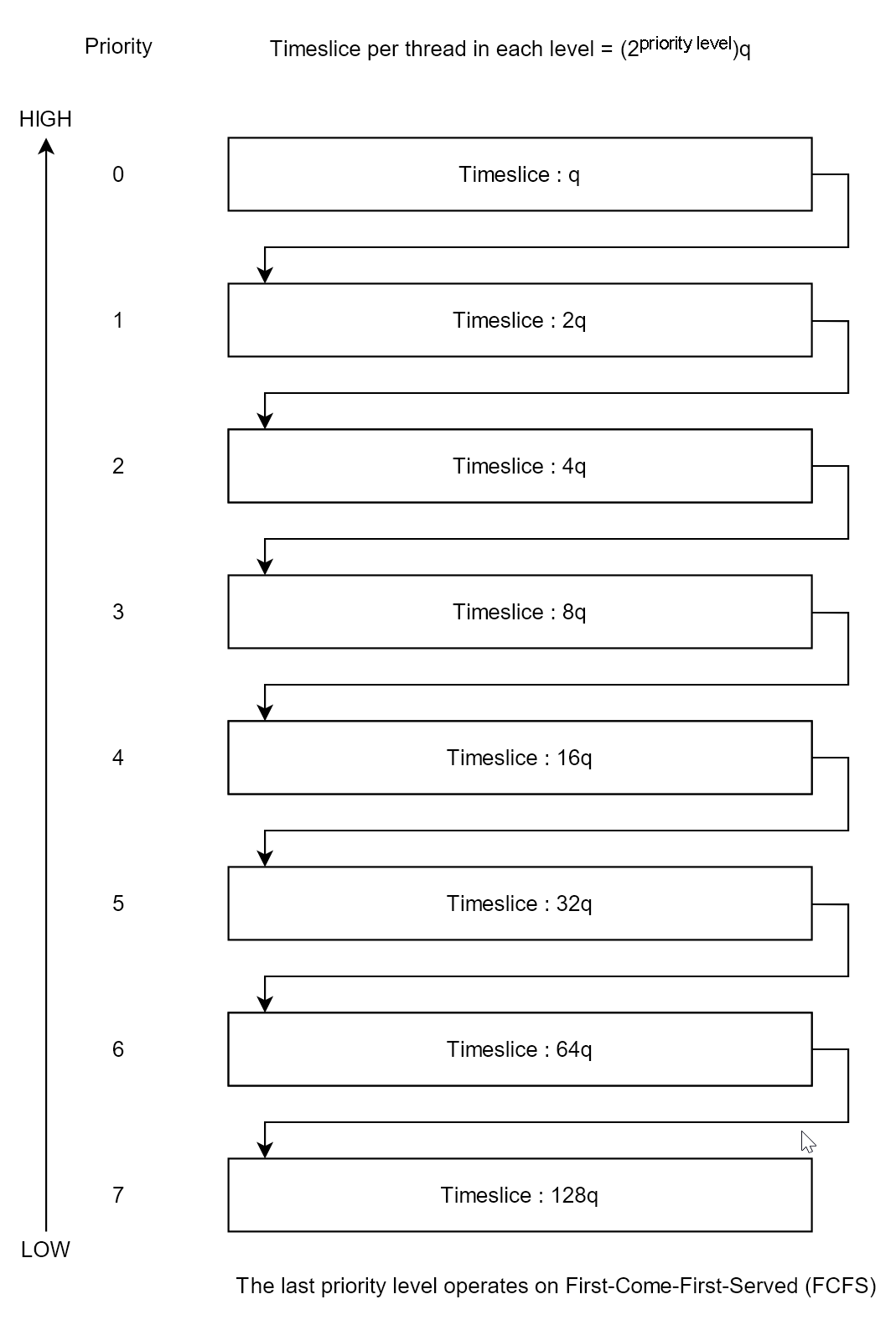
Lab 1 - CS2506 – Alexander Stradnic – 119377263



**Task 1**

If q = 1 for 1GHz, then q = 1.25 when the frequency is reduced to 800MHz.

When there are no user processes left to execute the system runs the Idle Process which is always the lowest priority process.

**Task 2**

while system.running:

|

while queues is empty and blocked\_queue is empty:

execute(idle\_process, idle\_process.time\_slice) *// if the queues are empty, run the idle process. it has the lowest priority of any process*

for queue in queues:

for process in queue:

if process.state == blocked:

move(process, queues[process.priority-1]) *// if the process is blocked, move it up a level*

else if process.state == ready:

execute(process, queue.time\_slice) *// execute the process for (2process.priority)q*

if process.state == ready:

move(process, queues[process.priority+1]) *// if the process is still not finished after being executed for the duration of its time slice, move the process down one level*

for process in blocked\_queue:  
 if not process.blocked:

move(process, queues[process.priority-1])

function execute(process, time\_slice):

while time\_slice > 0:

exec(process)

**Task 3 – CODE**

import time

import math

import random

*# System Class definition*

class System:

running = True

queues = []

blocked\_queue = []

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

for i in range(8):

self.queues.append([])

*# Definition of the Thread Class*

class Thread:

exec\_time = 0

pid = 0

io\_processes = {}

status = "Ready"

priority = 0

def \_\_init\_\_(self, pid, exec\_time, io\_processes, priority):

self.pid = pid

self.exec\_time = exec\_time

self.io\_processes = io\_processes

self.priority = priority

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

return "PID: " + str(self.pid) + ", priority: " + str(self.priority) + ", exec\_time: " + str(self.exec\_time) + ", io: " + str(self.io\_processes)

*# io\_process := {Time in code at which to fetch io : Cycles to fetch io}*

*# Function to execute threads*

def execute(thread):

duration = 2\*\*thread.priority

print("Attempting to run thread ", str(thread))

if thread.status == "Blocked": *# If the thread is blocked, check if it is done fetching the data. If it is done, change its status to Ready*

if thread.io\_processes[max(thread.io\_processes.keys())] == 0:

thread.status = "Ready"

thread.io\_processes.pop(max(thread.io\_processes.keys()))

else: # If the data hasn't been fetched yet, reduce the cycles to fetch by 1

thread.io\_processes[max(thread.io\_processes.keys())] -= 1

return thread.status

final\_time = 0

if thread.exec\_time > duration:

final\_time = thread.exec\_time - duration

actual\_time = final\_time

for io in thread.io\_processes.keys():

if thread.exec\_time == io:

thread.status = "Blocked"

return thread.status

if final\_time == io:

if io > actual\_time:

actual\_time = io

elif final\_time < io:

if io > actual\_time:

actual\_time = io

thread.status = "Blocked"

time.sleep(0.1\*(thread.exec\_time - actual\_time)) *# Sleep for time \* 0.1 to speed up*

thread.exec\_time = actual\_time

if thread.exec\_time == 0:

thread.status = "Complete"

return thread.status

def idle\_process():

time.sleep(5)

return "Idling..."

system = System()

for index, queue in enumerate(system.queues):

for i in range(random.randint(1, 3)): *# Generate between 1 and N threads per queue*

thread = Thread(random.randint(1, 999999), random.randint(1, 100), {}, index)

if random.randint(0,2) != 0: # Generate io requests for threads

thread.io\_processes[random.randint(0, thread.exec\_time)] = random.randint(1, 5)

queue.append(thread)

while system.running:

for index, queue in enumerate(system.queues): *# print all queues at the start of every cycle*

print("Queue", index, ":", [str(thread) for thread in queue])

queuesEmpty = True

for queue in system.queues: *# check if the queues are empty*

if len(queue) != 0:

queuesEmpty = False

break

while queuesEmpty and len(system.blocked\_queue) == 0:

print(idle\_process()) *# if the queues are empty, run the idle process. It has the lowest priority of any process*

a = 0

while a < len(system.queues): *# loop through queues*

b = 0

while b < len(system.queues[a]): *# loop and execute threads in a queue*

status = execute(system.queues[a][b])

if status == "Blocked":

if system.queues[a][b].priority != 0: *# NEW : WAS ADDED AFTER SCREENSHOTS WERE TAKEN. This boosts the thread's priority instead of just remaining at the same level*

system.queues[a][b].priority -= 1

system.blocked\_queue.append(system.queues[a].pop(b))

elif status == "Ready":

if a < len(system.queues)-1: *# move an unfinished thread down to the front of the next highest priority list*

system.queues[a][b].priority += 1

\_temp = [system.queues[a].pop(b)]

for i in system.queues[a+1]:

\_temp.append(i)

system.queues[a+1] = \_temp

else:

system.queues[a].pop(b)

a += 1

*# Go through the blocked queue and reduce each fetch time by 1*

print("Blocked queue: ", [str(thread) for thread in system.blocked\_queue])

for p\_index, process in enumerate(system.blocked\_queue):

execute(process)

if process.status != "Blocked": *# If the thread is not blocked anymore, then move it back to the priority queue it belongs to*

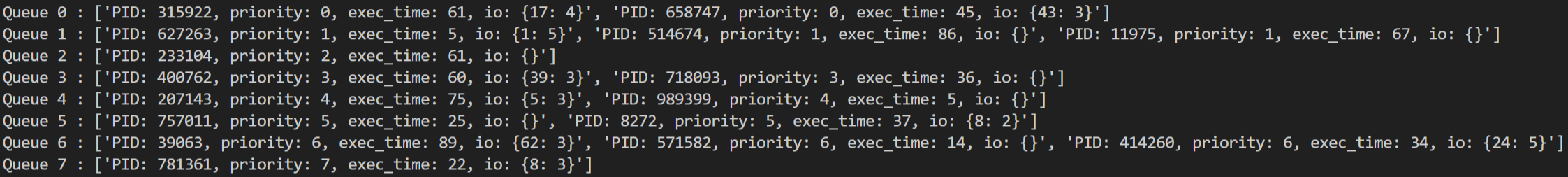
system.queues[process.priority].append(system.blocked\_queue.pop(p\_index))

**Task 4 – Report**

The program starts by creating a System Object. It runs an infinite while loop that checks for its running parameter to be true, which could be stopped in theory. Then a list of lists populated with Thread Objects is created, using Python’s *random* library. It then loops through the threads like a real scheduler simulating their execution using *time.sleep*.

Example

First, the queues are created.



We’ll follow Thread 315922.

It is run for the first time.



It is then moved in lower queues and run.



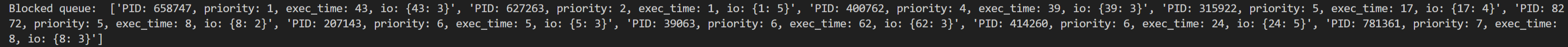




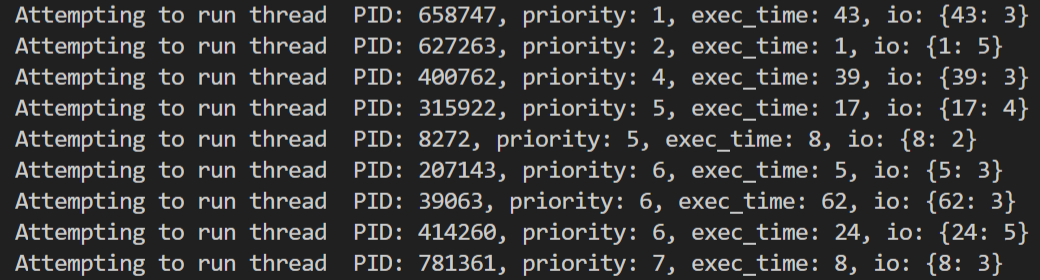




It is then moved to the Blocked Queue as it needs to fetch data for 4 cycles when it has 17q units of time left in its execution. Instead of moving to priority level 6, it stays at level 5. \* This was changed in the code after screenshots were taken to boost the process’s priority instead.



Every cycle reduces its count by 1, eventually reaching 0, when it is moved back to a priority queue.





It is finally completed.

