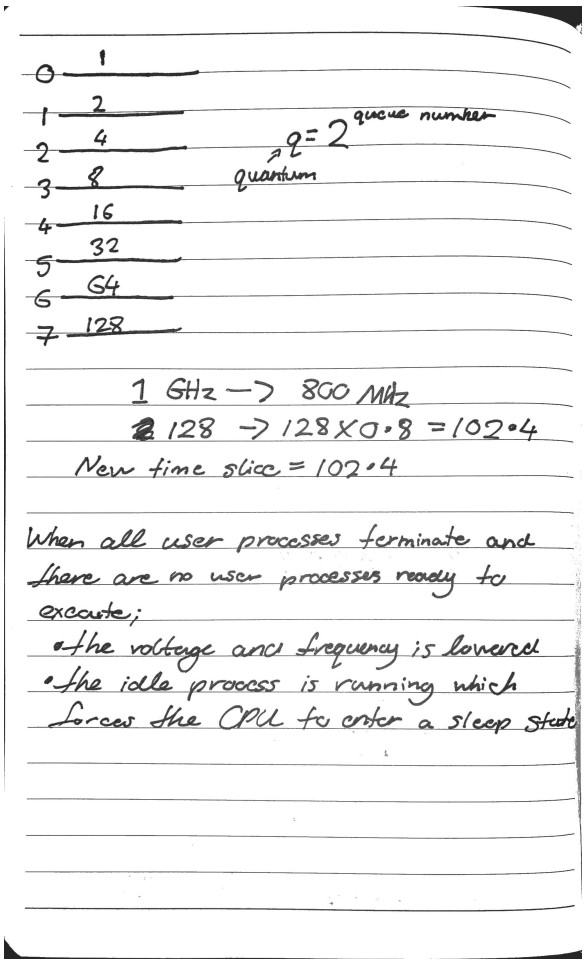
Assignment 1 - Operating Systems II

Name: Eimantas Pusinskas

Student ID: 120312336

Task 1



Task 2

while True (this will be an infinite loop)

if there are processes enqueued and there are no blocked processes

get the next ready process by order of priority

determine the amount of CPU time it gets by its priority but if there are any IO operations meant to start or finish in the time period during this time, then the time slice will be shortened to accommodate the IO

once the time slice is determined, the process gets this exact amount of CPU time

if the process finishes execution within its time slice

it is terminated

otherwise

its priority is downgraded, it is dequeued from its current queue and enqueued at the queue of its new priority

if a process is about to start IO

its state is now blocked

it is removed from the multilevel feedback queues and enqueued at the blocked queue

if a process has finished IO

its state is now ready

it is removed from the blocked queue

its priority is increased by one level if possible

it is enqueued back in the multilevel feedback queue at the queue of its new priority

else

the scheduler enters power saving mode

voltage and capacity is decreased

sleep state is entered

Task 3

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

# this code for the class QueueV0 has been taken from the module Data Structures & Algorithms I
# I do not take any credit for writing this code
# all credit goes to Ken Brown and the UCC Department of Computer Science

import time
import sys

class QueueV0:
    """ A queue using a python list, with the head at the front.

    """
    def __init__(self):
        self.body = []

    def __str__(self):
        if len(self.body) == 0:
            return '<--<'
        stringlist = ['<']
        for item in self.body:
            stringlist.append('-' + str(item))
        stringlist.append('-<')
        return ''.join(stringlist)
        # output = '<-'
        # i = 0
        # for item in self.body:
        #     output = output + str(item) + '-'
        # output = output +'<'
        # return output

    def summary(self):
        """ Return a string summary of the queue. """
        return ('length:' + str(len(self.body)))

    def get_size(self):
        """ Return the internal size of the queue. """
        return sys.getsizeof(self.body)

    def enqueue(self, item):
        """ Add an item to the queue.

        Args:
            item - (any type) to be added to the queue.
        """
        self.body.append(item)

    def dequeue(self):
        """ Return (and remove) the item in the queue for longest. """
        return self.body.pop(0)

    def length(self):
        """ Return the number of items in the queue. """
        return len(self.body)
```

```
    def first(self):
        """ Return the first item in the queue. """
        return self.body[0]
```

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

class Process(object):
    def __init__(self, pid, time, priority, state, io, io_time):
        self._pid = pid
        self._time = time
        self._priority = priority
        self._state = state
        self._io = io
        self._io_time_left = io_time
        self._io_time_required = io_time

    def get_time(self):
        return self._time

    def set_time(self, time):
        self._time = time

    def get_priority(self):
        return self._priority

    def set_priority(self, priority):
        self._priority = priority

    def get_state(self):
        return self._state

    def set_state(self, state):
        self._state = state

    def get_pid(self):
        return self._pid

    def get_io(self):
        return self._io

    def set_io(self, io):
        self._io = io

    def get_io_time_left(self):
        return self._io_time_left

    def set_io_time_left(self, io_time):
        self._io_time_left = io_time

    time = property(get_time, set_time)
    priority = property(get_priority, set_priority)
    state = property(get_state, set_state)
    pid = property(get_pid)
    io = property(get_io, set_io)
    io_time_left = property(get_io_time_left, set_io_time_left)

    def __str__(self):
        return f"[PID:{self._pid}-TIME:{self._time}-{self._state}]"
```

```
# Name: Eimantas Pusinskas
# Student ID: 120312336

from queues import QueueV0
from process import Process
import time

class scheduler(object):

    def __init__(self):

        # creates 8 queues and adds them to a list to create a matrix of queues
        # which will represent the multilevel feedback queues
        self._queues = []
        for i in range(8):
            q = QueueV0()
            self._queues.append(q)
```

```
        # captures the total time elapsed since the schduler has started running/
        # the total CPU time every process has had
        self._time = 0
        self._blocked = []
        self._load = "high"
        self._voltage = "high"
        self._capacity = "high"
        self._upcoming_io = []

    def __str__(self):
        queueStr = ""
        queueStr += f"Time Elapsed: {self._time}\nBlocked Queue: {[self._blocked[i].pid for i in range(len(self._blocked))]}\n"
        for queue in self._queues:
            queueStr += f"{self._queues.index(queue)} : {queue}"
            if self._queues[-1] != queue:
                queueStr += "\n"
        return queueStr


    """
        adds a process to a queue depending on its priority
    """
    def add_process(self, process):
        self._queues[process.priority].enqueue(process)

        # if a process has any IO operations scheduled it is also added to an upcoming_io list which will
        # help in the execute() function to see if there is any upcoming IO for any process
        # my reasoning for using this list is to increase efficiency by not having to search through every queue
        # to find a process with upcoming IO and instead having it all in one list
        if process.io != None:
            self._upcoming_io.append(process)
            self._upcoming_io.sort(key = lambda x:x._io_time_left)

    """
        process is donwgraded in priority
    """
    def downgrade_process_priority(self, queue_num):
        proc = self._queues[queue_num].dequeue()
        if proc.priority < 7:
            proc.priority += 1
            self._queues[proc.priority].enqueue(proc)


    def terminate_process(self, queue_num):
        self._queues[queue_num].dequeue()


    """
        this function determines the time slice for a process and simulates giving a process CPU control for that time
        the time slice is determined by three factors;
        1. if there are any upcoming IO operations meant to be starting.
            It uses the upcoming_io list to look this up. Since the list is ordered by the IO time each process is starting
            from lowest to highest, it gets the first process in the list if the list is not empty and sets the time left till
            IO starts as the second time slice. Suppose there are 4 time units till the next IO operation starts, then time slice A =
        2. if there are any IO operations finishing
            it uses the blocked list to look up when the next IO operation finishes and sets the time slice as the time left till this
            suppose time slice B = 6
        3. if the process will finish its execution in less than the time it will be given CPU control. e.g. process priority = 3, and
            will be 2**3 = 8 time units, but it only need 2 time units and so only 6 will be given. Thus time slice C = 6
        initially all 3 time slices will have the same time slice that is determined by a processes priority. e.g process priority = 3
        but if any of the three conditions above are satisfied then that will be its new time slice
        And so we have three different potential time slices in my example above (time slice A = 4, time slice B = 6, time slice C = 6
        the final time slice that is used is the minimum of these three different time slices and so the time slice = 4
    """
    def execute(self, queue_num):
        # time slice for a queue depending on its priority
        time_slice = 2 ** queue_num

        # process that will be given CPU control
        proc = self._queues[queue_num].first()

        # check every incoming io operation, if the time slice will collide with the
        # expected io, it adjusts time slice accordingly
        time_sliceA = time_slice
        for io_upcoming in self._upcoming_io:
            if self._time + time_slice >= io_upcoming.io:
                time_sliceA = io_upcoming.io - self._time
                break

        # checks when every io finishes, adjusts time slice accordingly
        time_sliceB = time_slice
        for blocked in self._blocked:
            if time_slice >= blocked._io_time_left:
                time_sliceB = blocked._io_time_left
                break
```

```
        # if the time slice allocated is larger than the
        # time required for full execution of the process then
        # only the required amount of time is allocated for execution
        time_sliceC = time_slice
        if time_slice >= proc.time:
            time_sliceC = proc.time

        # the minimum time of all three slices is used as the final time slice
        time_slice = min(time_sliceA, time_sliceB, time_sliceC)

        # updates io time left for any process in the blocked queue
        for blocked in self._blocked:
            blocked._io_time_left -= time_slice

        # updates total time passed since the very start
        self._time += time_slice

        # updates a processes time required left for execution
        proc.time -= time_slice

        # mimicks the waiting process of a process by
        # having control of the CPU for the time slice
        if time_slice != 0:
            time.sleep(time_slice/1000)

        # if a process doesn't finish execution within its time slice, it's priority is downgraded if possible
        # if it has the lowest priority possible, it is dequeued and enqueued at the same queue to maintain the round-robin policy for
        # otherwise a process has finished and it is terminated
        if proc.time > 0 and proc.priority != 7:
            self.downgrade_process_priority(queue_num)
        elif proc.time > 0 and proc.priority == 7:
            self._queues[7].dequeue()
            self.add_process(proc)
        else:
            self.terminate_process(queue_num)

        # prints queues for testing
        print(sch)
        print("-------------")

    """
        this function gets the next ready process
        the next ready process in my implementation will always be at the front of the queue
        therefore only the queue number is returned from the function
        if a process at the front of a queue is ready to start IO, it is blocked, dequeued, and enqueue at the blocked list
        this ensures that a process at the front of a queue is always ready for execution

    """
    def get_next_process(self):
        i = 0
        found = False
        queue = None
        while (found == False) and (i < 8):
            # checks that the queue i is not empty
            if (self._queues[i].length() != 0) and (self._queues[i].first().state == "ready"):
                proc = self._queues[i].first()
                # blocks process if it is time for it to execute io operation
                if proc.io != None and self._time >= proc.io:
                    proc.state = "blocked"
                    self._blocked.append(proc)
                    self._queues[i].dequeue()

                    print(f"PID:{proc.pid} Expected block time {proc.io}. Actual time blocked {self._time}. ")
                    print("-------------")

                    # once a process is bloked and added to the blocked list, it is removed from the upcoming IO list,
                    #  since its has IO is no longer upcoming and instead is about to start
                    for io_proc in self._upcoming_io:
                        if io_proc.pid == proc.pid:
                            del self._upcoming_io[self._upcoming_io.index(io_proc)]
                            break
                else:
                    queue = i
                    found = True
            else:
                i += 1
        return queue


    """
        this function iterates over every process in the blocked list,
        checks if its IO has finished and if true, removes process from blocked list,
        increases its priority by 1 and enqueues it back in the multilevel feedback queue

    """
    def unblock(self):
```

```
        for proc in self._blocked:
            if proc.io_time_left <= 0:
                print(f"PID:{proc.pid} Expected unblock time {proc._io + proc._io_time_required}. Actual unblock time {self._time}")
                print("------------")

                proc.state = "ready"
                proc.io = None
                proc.io_time_left = 0
                if proc.priority != 0:
                    proc.priority -= 1

                self._queues[proc.priority].enqueue(proc)
                del self._blocked[self._blocked.index(proc)]

    """
        checks if the system load is low
        the load is considered low if the only processes are in the idle queue
        and there are less than two processes
    """
    def if_load_low(self):
        low = True
        for i in range(6):
            if self._queues[i].length() != 0:
                low = False

        if low == True and self._queues[7].length() < 2:
            return True
        else:
            return False


    def decrease_power(self):
        self._load = "low"
        self._capacity = "low"
        self._voltage = "low"

    def increase_power(self):
        self._load = "high"
        self._capacity = "high"
        self._voltage = "high"


    """
        this simulates the scheduler sleeping
        the sleep function is used to prevent from exceeding maximum recursion
    """
    def sleep(self):
        print("zzz")
        time.sleep(60)


    """
        this simulates the CPU running with an infinite loop
    """
    def run(self):
        while True:
            if self.get_next_process() != None or len(self._blocked) != 0:

                if len(self._blocked) != 0:
                    self.unblock()

                next_proc = self.get_next_process()
                self.execute(next_proc)

                if self.if_load_low() == True:
                    self.decrease_power()
                else:
                    self.increase_power()

            else:
                self.sleep()


if __name__ == "__main__":
    sch = scheduler()
    proc1 = Process(1, 100, 1, "ready", 20, 10)
    proc2 = Process(2, 200, 2, "ready", None, None)
    proc3 = Process(3, 100, 5, "ready", None, None)
    proc4 = Process(4, 150, 3, "ready", 200, 50)
    proc5 = Process(5, 400, 7, "ready", None, None)
    proc6 = Process(6, 130, 2, "ready", None, None)
    sch.add_process(proc1)
    sch.add_process(proc2)
    sch.add_process(proc3)
    sch.add_process(proc4)
    sch.add_process(proc5)
    sch.add_process(proc6)
```

```
print(sch)
print("-------------")

sch.run()
```

Task 4

In my simulation of my algorithm I have 6 processes, with 2 processes having I/O operations. For clarity, I will explain the parameters of process 1 (proc1) by the order they appear in the image below

- 1 = the process id
- 100 = the total time the process needs CPU control for its complete execution
- 1 = the process priority
- “ready” = the state of the process
- 20 = the time when the process is starting its I/O operation
- 10 = the time duration for which its I/O will last

Image 1.1

As we can see below, all processes are initially (time = 0) enqueued in their queue depending on their priority. Each process shows its process ID, its total time left required for execution and its state

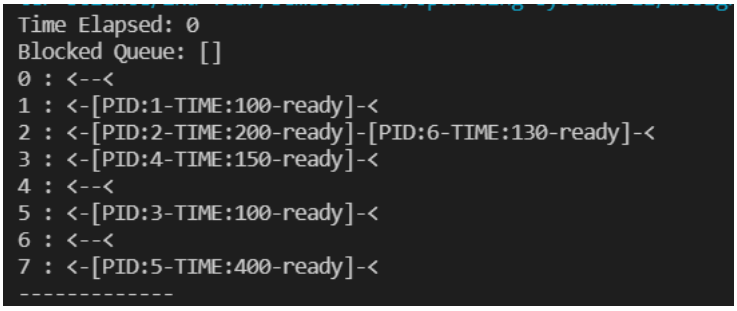


Image 1.2

As we saw in picture 1.1, process 1 is meant to be starting IO at 20 time units for a duration of 10 time units

In 1.3 below we see that at 20 time units, process 1 with PID 1 is blocked and added to the blocked queue. 10 time units later when time = 30, it is unblocked, upgraded in priority to queue 2, and gets a time slice of 2**2 = 4 time units at 30 time units. At 34 time units we see that process 1 has has its 4 time units of CPU time as its total time left for execution has decreased form 94 to 90 time units

```
-------------
Time Elapsed: 20
Blocked Queue: []
0 : <--<
1 : <--<
2 : <--<
3 : <-[PID:1-TIME:94-ready]-<
4 : <-[PID:4-TIME:144-ready]-[PID:2-TIME:196-ready]-[PID:6-TIME:126-ready]-<
5 : <-[PID:3-TIME:100-ready]-<
6 : <--<
7 : <-[PID:5-TIME:400-ready]-<
-------------
PID:1 Expected block time 20. Actual time blocked 20.
-------------
Time Elapsed: 30
Blocked Queue: [1]
0 : <--<
1 : <--<
2 : <--<
3 : <--<
4 : <-[PID:2-TIME:196-ready]-[PID:6-TIME:126-ready]-<
5 : <-[PID:3-TIME:100-ready]-[PID:4-TIME:134-ready]-<
6 : <--<
7 : <-[PID:5-TIME:400-ready]-<
-------------
PID:1 Expected unblock time 30. Actual unblock time 30
------------
Time Elapsed: 34
Blocked Queue: []
0 : <--<
1 : <--<
2 : <--<
3 : <-[PID:1-TIME:90-ready]-<
4 : <-[PID:2-TIME:196-ready]-[PID:6-TIME:126-ready]-<
5 : <-[PID:3-TIME:100-ready]-[PID:4-TIME:134-ready]-<
6 : <--<
7 : <-[PID:5-TIME:400-ready]-<
-------------
```

Image 1.3

Process 4 also starts IO at its expected time of 200 and finished IO at its expected time of 250

All of the IO operations in my code finish start and finish exactly at their scheduled times as a result of using three different factors when determining the time slice a process gets. These three factors are explained in more detail in “scheduler.py” as a comment that explains the execute() function

```
-------------
PID:4 Expected block time 200. Actual time blocked 200.
-------------
Time Elapsed: 250
Blocked Queue: [4]
0 : <--<
1 : <--<
2 : <--<
3 : <--<
4 : <--<
5 : <--<
6 : <-[PID:6-TIME:96-ready]-[PID:1-TIME:66-ready]-<
7 : <-[PID:5-TIME:400-ready]-[PID:3-TIME:68-ready]-[PID:2-TIME:98-ready]-<
-------------
PID:4 Expected unblock time 250. Actual unblock time 250
------------
```

Once all processes are executed the queues look as follows in image 1.4 below. The time elapsed is 1080 which is the sum of all processes total time required for execution (100 + 200 + 100 + 150 + 400 + 130). Since there is no point in my scheduled processes when all queues are empty but there are blocked processes, IO does not affect the total time elapsed.

Once all queues are empty we see that the scheduler enters sleep mode as it prints “zzz”

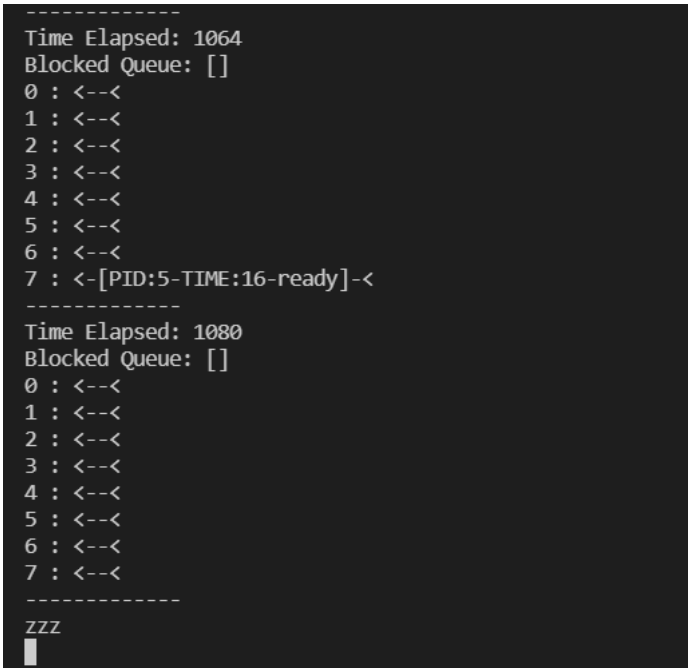

Time Elapsed: 1064
Blocked Queue: []
0 : <--<
1 : <--<
2 : <--<
3 : <--<
4 : <--<
5 : <--<
6 : <--<
7 : <-[PID:5-TIME:16-ready]-<
Time Elapsed: 1080
Blocked Queue: []
0 : <--<
1 : <--<
2 : <--<
3 : <--<
4 : <--<
5 : <--<
6 : <--<
7 : <--<
zzz

Image 1.4