Projeto 1 - Sistemas Operacionais

Elton Mauricio Da Silva - RA 11201810955

link de arquivos: <https://github.com/eltonmds/ufabc_operational_systems>

1 Problema 1

Código

Para melhor organização do código, foram criadas as seguintes classes:

MQP - Uma implementação que representa a Multilevel Queue Priority

```
from process import Process
from typing import List
import operator

class MQP:
    def __init__(self) -> None:
        self.priority_0: List[Process] = []
        self.priority_1: List[Process] = []
        self.priority_2: List[Process] = []
        self.priority_3: List[Process] = []

    def is_empty(self) -> bool:
        for i in range(4):
            if eval(f"len(self.priority_{i}) > 0"):
                return False

        return True


    def insert_process(self, process: Process) -> None:
        eval(f"self.priority_{process.priority}.append(process)")


    def remove_process(self, process: Process) -> None:
        eval(f"self.priority_{process.priority}.remove(process)")


    def increase_priority(self, process: Process) -> None:
        process.priority -= 1
        eval(f"self.priority_{process.priority}.append(process)")
        eval(f"self.priority_{process.priority + 1}.remove(process)"
                                            )


    def find_biggest_cpu_burst(self, priority_list: List[Process]) -
                                        > None:
        priority_list.sort(key=operator.attrgetter("CPU_burst"))
        biggest_process: Process = priority_list[-1]
```

```
        return biggest_process

    def find_smallest_cpu_burst(self, priority_list: List[Process])
                                        -> None:
        priority_list.sort(key=operator.attrgetter("CPU_burst"))
        biggest_process: Process = priority_list[0]
        return biggest_process


    def change_smallest_priority_process(self) -> None:
        for i in range(3, 0, -1):
            current_level = eval(f"self.priority_{i}")
            if len(current_level) > 0:
                biggest_process = self.find_biggest_cpu_burst(
                                                    current_level)
                self.increase_priority(biggest_process)
                break


    def increase_wait_time(self, current_process: Process) -> None:
        for priority in range(4):
            for process in eval(f"self.priority_{priority}"):
                if process != current_process:
                    process.wait_time += 10


    def print_processes(self) -> None:
        print("  +" + "-" * 15 + "+" + "-" * 8 + "+" + "-" * 9 + "+"
                                            )
        print(f"  |{'process_name':15}|{'priority':8}|{'CPU_burst':9
                                            }|")
        print("  +" + "-" * 15 + "+" + "-" * 8 + "+" + "-" * 9 + "+"
                                            )

        for i in range(4):
            for process in eval(f"self.priority_{i}"):
                process.print_table()

        print("  +" + "-" * 15 + "+" + "-" * 8 + "+" + "-" * 9 + "+\
                                            n")
```

Process - implementação de uma estrutura de dados para representar os processos

```
    class Process:
    def __init__(self, name: str, priority: int, CPU_burst: int) ->
                                        None:
        self.name = name
        self.priority = priority
```

```
            self.CPU_burst = CPU_burst
            self.wait_time = 0

        def print_table(self) -> None:
            print(f"  |{self.name:15}|{self.priority:8}|{self.CPU_burst:
                                                    9}|")


        def print_info(self) -> None:
            print("  " + "+" + "-" * 18 + "+" + "-" * 13 + "+")
            print(f"  |{'Process name':18}: {str(self.name):12}|")
            print(f"  |{'Process priority':18}: {str(self.priority):12}|
                                                    ")
            print(f"  |{'Process CPU_burst':18}: {str(self.CPU_burst):12
                                                    }|")
            print(f"  |{'Process wait_time':18}: {str(self.wait_time):12
                                                    }|")
            print("  " + "+" + "-" * 18 + "+" + "-" * 13 + "+")
```

E então, utilizadas as classes no código principal

```
from time import sleep
from typing import Tuple, List
import logging

from process import Process
from mqp import MQP
from scheduler import Scheduler

logging.basicConfig()
logger = logging.getLogger("")
logger.setLevel(logging.INFO)


def create_process(input: Tuple[str]) -> Process:
        process = Process(input[0], int(input[1]), int(input[2]))
        if process.priority not in range(4):
            Exception("Priority must be between 0 and 3")
            pass

        return process


def receive_user_input() -> List[tuple]:
    processes_attributes: List[str] = []

    with open("processes.txt", "r") as processes:
        lines = processes.readlines()
        for line in lines:
```

```
            line = line.split(", ")
            processes_attributes.append(tuple(line))

        processes.close()

    return processes_attributes


def receive_processes(processes_attributes: List[tuple]) -> MQP:
    mqp: MQP = MQP()

    for process_attribute in processes_attributes:
        new_process: Process = create_process(process_attribute)
        logger.info(f"New process created: {new_process.name}")

        mqp.insert_process(new_process)

    return mqp



def select_process(mqp: MQP) -> Process:
    for i in range(4):
        current_priority_level = eval(f"mqp.priority_{i}")
        if current_priority_level:
            chosen_process = mqp.find_smallest_cpu_burst(
                                                current_priority_level
                                                )

            return chosen_process
    else:
        logger.info("No processes to select")
        return None


def main():
    processes_input = receive_user_input()
    mqp = receive_processes(processes_input)

    print("This are the processes received: ")
    mqp.print_processes()

    while not mqp.is_empty():
        current_process = select_process(mqp)
        logger.info("Executing the following process:")
        current_process.print_info()
        while current_process.CPU_burst > 0:
```

```
                current_process.CPU_burst -= 10
                sleep(10)
                mqp.increase_wait_time(current_process)
                mqp.change_smallest_priority_process()

            logger.info(f"Process finished: {current_process.name}\n")
            mqp.remove_process(current_process)

        logger.info("All processes terminated")

if __name__ == "__main__":
    main()
```

O programa recebe os processos através do arquivo 'processes.txt'. A partir do texto recebido, cria instâncias da classe Process para representar os processos.

Após isso, instancia a classe MQP para trabalhar com a Multilevel Queue Priority. E então printa uma tabela contendo os processos recebidos.

A partir disso entra em um loop principal, que só será interrompido quando não há mais processos para rodar, isto é, quando todas as listas de prioridades estiverem vazias.

A cada iteração, é feito o processo de seleção do processo atual, que segue as seguintes regras: - Maior prioridades * Maior CPU-Burst

Escolhido o processo atual, são printadas as informações a respeito do mesmo. E então entra em um segundo loop, onde é simulada a execução do processo. Onde a cada 10 unidades de tempo (determinadas pela função sleep): - É aumentado o $\text{wait}_t imedosdemaisprocessosem10 - Éaumentadaem1aprioridadedoprocessocomomaior CPU_B urstd$

Resultados

```
INFO:root:New process created: Paint
INFO:root:New process created: Meu joguinho
INFO:root:New process created: Firefox
These are the processes received:
  +---------------+--------+---------+
  |process_name   |priority|CPU_burst|
  +---------------+--------+---------+
  |Meu joguinho   |       0|      190|
  |Firefox        |       0|       20|
  |Paint          |       2|        1|
  +---------------+--------+---------+

INFO:root:Starting the scheduling process...
INFO:root:Executing the following process:
  +------------------+-------------+
  |Process name      : Firefox     |
  |Process priority  : 0           |
  |Process CPU_burst : 20          |
  |Process wait_time : 0           |
  +------------------+-------------+
INFO:root:Process finished: Firefox

INFO:root:Executing the following process:
  +------------------+-------------+
  |Process name      : Paint       |
  |Process priority  : 0           |
  |Process CPU_burst : 1           |
  |Process wait_time : 20          |
  +------------------+-------------+
INFO:root:Process finished: Paint

INFO:root:Executing the following process:
  +------------------+-------------+
  |Process name      : Meu joguinho|
  |Process priority  : 0           |
```

Os primeiros outputs indicam a criação dos processos que vieram do arquivo ‘processes.txt‘. Mais informações a respeito deles é mostrada na primeira tabela.

Então é printado "Starting the scheduling process" que indica que entrou na fase de escolar os processos. E então, para cada processo executado, é printada uma tabela com as seguintes informaçoes: * Nome do processo * Prioridade do processo * $\mathrm{CPU}_burstdoprocesso * tempodeesperadoprocesso$

Após cada processo ser finalizado, é printado "Process finished" indicando que aquele processo foi finalizado.

2 Problema 2

Código