

Sistemas operacionais - Trabalho prático 3: Simulador de Processos Mk. II (com simulador de memória)

Leandro Lazaro Araújo Vieira (3513), Mateus Pinto da Silva (3489), Daniel Fernandes Pinho (2634)

Ciência da Computação – Universidade Federal de Viçosa - Campus Florestal (UFV-CAF) – Florestal – MG – Brasil

{leandro.lazaro, mateus.p.silva, daniel.pinho}@ufv.br

Resumo. Este trabalho foi desenvolvido seguindo as duas especificações para a disciplina de Sistemas Operacionais, sendo elas: “simulação de gerenciamento de processos” e “gerenciamento de memória sobre o gerenciador de processos”. O trabalho apresenta, de forma paralela, a simulação de um sistema gerenciador de processos junto ao seu gerenciamento de memória, utilizando linguagem Python para elaboração do arcabouço dos trabalhos práticos em questão. O sistema gerenciador de processos apresenta cinco funções fundamentais, como criar processos, substituir a imagem atual do processo com uma imagem nova de processo, transição de estado de processo, escalonamento de processos e troca de contexto da CPU. Integrando ao gerenciamento de memória, é possível armazenar as variáveis de cada processo simulado e também é descrito uma implementação de memória virtual proposta pelo próprio grupo.

1. Como executar o simulador

Utilizando o interpretador Python 3, basta executar o comando “make” para instalar as dependências, e depois “make run” no terminal na raiz do projeto. Em seguida o menu principal aparecerá.

```
*** Simple process simulator Mk II ***
By Daniel, Leandro and Mateus

>>> Main Menu

Using:
- 3-Core Processor
- 3 Queues Scheduler
- Infinite Memory

> Interactive Mode
Run file
Config
Quit
```

2. Componentes extras implementados

Fizemos uma implementação interativa que funciona de forma semelhante ao Htop. Por isso, foi impossível utilizar Forks e Pipes. Além disso, fizemos TODOS os escalonadores presentes no livro, e nosso simulador funciona para N processadores. Fizemos também todos os algoritmos de alocação de memória física, e fizemos uma memória virtual usando dicionário do Python, que chamamos de *InfiniteMemory*.

[Process Table]							[3-Core Processor]				[Infinite Memory]						
PID	F PID	PC	V A0	V AS	PRI	INT	CPT	PID	TIM	PID = 000 >>>	012	051	000	000	000		
000	-001	015	000	005	000	000	013	001	005	PID = 002 >>>	052	007					
001	0000	015	005	004	000	007	016										
002	0000	009	009	002	000	013	010										

[Physical Memory (size: 30) (Worst fit)]																	
XXX	XXX	XXX	XXX	XXX	[004]	[014]	[016]	[021]	XXX								
XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	XXX	

[Round Robin Scheduler (quantum: 3)]			[Blocked by IO]			[Blocked by Memory Heap]			[Done list]							
[]									0							
N: 3	D: 11	V: 8	A: 12	S: 0	B: 1	T: 2	F: 2	R: 2	[Final Report]	Context switches: 5	Processes added: 0	Response time: 20.0	Priority changes: 0	Instructions runned: 41	Memory info]	Time spent: 42

3. A escolha da linguagem de programação

Como linguagem de programação, escolhemos Python, pois além de sermos muito familiarizados com ela, é uma linguagem prática, orientada a objetos, com muitos recursos e acima de tudo que oferece um dos melhores (senão os melhores) suportes para lidar com manipulação de strings, listas e arquivos.

4. A estrutura do simulador

Todo o código necessário para o funcionamento do simulador está na pasta src a partir da raiz do projeto. Dentro dessa pasta, existem vários arquivos, cada um deles com suas respectivas classes que têm alguma relação com o nome do arquivo onde estão hospedadas.

Como de costume, o arquivo main.py é o responsável pela execução de todo o projeto. O pacote (e pasta) InterpretedLang contém classes que gerenciam a linguagem interpretada pelo simulador. O Memories contém as memórias suportadas, e o Schedulers os escalonadores implementados.

5. Ajustes finos na especificação

Preferimos por fazer um simulador interativo, ou seja, a cada comando executado os valores na tela automaticamente mudam, para facilitar a nossa visualização. Assim, o comando “I”, que deveria exibir tais valores simplesmente não faz nada.

Além disso, optamos por definir melhor a linguagem interpretada da seguinte forma:

Opcode	Inteiro sem sinal	Inteiro com sinal
V	Variável a ser modificada	Valor a ser definido
A	Variável a ser modificada	Valor a ser somado
S	Variável a ser modificada	Valor a ser subtraído
N	Número de variáveis	
D	Variável a ser definida	
F	Número de linhas a sofrerem Fork	
R	Arquivo para ser substituído	
B		
T		

Note, portanto, que todos os arquivos precisam ter nomes de números inteiros sem sinal. O primeiro programa a ser executado é o 0.txt. O mesmo acontece com os arquivos de entrada da pasta input/.

6. Funcionamento

O funcionamento do código pode ser resumido analisando o arquivo main.py. Após instanciar todas as classes necessárias para o funcionamento do processador, utilizando o método *processTable.appendProcess(...)* e *processor.appendProcess(...)* é possível inserir o primeiro processo que dará início a execução do simulador.

O método *processor.runInstruction(...)* executa a primeira instrução na lista de processos executando independente do método de escalonamento desejado. Em seguida, o método *scheduler.run(...)* escalona os processos de acordo com o método de escalonamento desejado previamente de acordo como explicado na seção, e o *memoryManager.run(...)* verifica se há algum processo bloqueado por memória e o coloca em uma posição privilegiada do processador. Todas as classes contém métodos *toString()* (chamadas no Python de *__str__()*) para exibição dos valores no terminal.

O modo de arquivo simplesmente lê instruções de um arquivo na pasta inputs/. Os programas precisam ficar na pasta programs/, e o primeiro programa se chama 0.txt. Foi desenvolvido um menu super simples que permite executar o modo interativo, a leitura de arquivos ou mudar as configurações.

6. Classes principais

6.1. ProcessTable

Essa classe é responsável por armazenar e manipular todos os processos. Quando é necessário manipular o dado de algum processo, sempre é utilizado os métodos oferecidos por essa entidade. Para funcionar como desejado, esta classe utiliza a classe *ProcessTableItem* que funciona como uma estrutura de dados para armazenar todos os dados pertinentes de um processo, como FPID, número de variáveis, tamanho da memória alocada, o texto do processo, sua prioridade, quando o processo iniciou sua execução, o tempo gasto para executar suas instruções na CPU e o contador de programa.

6.2. Processor

Esta classe é responsável por definir as ações que o processador fará sobre as instruções e o gerenciador de processo como geral. Ela contém dois atributos principais para cumprir com seu objetivo: o número de núcleos, e uma lista de threads que estará naquele momento executando sobre a CPU. Esta classe possui os métodos de retornar o número de núcleos, número de threads, se a lista de threads em execução está cheia ou se está vazia (retorno padrão 0).

Também é responsável por adicionar threads ao método Entrada de Processos, removê-lo quando necessário (através do parâmetro PID da thread em questão), setar (adicionar através de *increaseQuantum(quantum)*) a quantidade de Quantum do processo com o PID como parâmetro; ler instruções da lista de threads da CPU (quando isto é feito, é incrementado a unidade de um Quantum para o Thread em questão); alocar memória para um processo recebendo como parâmetro o seu PID, número de variáveis, a tabela de processos, a referência para o gerenciador de memórias, e uma referência para a classe que imprime os valores finais das variáveis de processo utilizadas durante o programa.

Incrementar valores da tabela de processos como o valor do contador de programa (PC), o tempo de CPU através de suas respectivas chamadas de método são responsabilidades dessa classe. Subtrair valores, quando necessário, também. Há um método de bloquear o processo por requisição de entrada/saída do programa. Isto é feito removendo o processo da lista de processos em execução pelo seu PID, acrescenta-lo na lista de bloqueados por entrada/saída e, em seguida, mudar a sua prioridade na tabela de processos bloqueados por requisições de entrada/saída, passando seu PID e a tabela de processos como referências.

A classe CPU também implementa os métodos de terminar processo, criar um novo processo (fork process), mudar a imagem do processo e executar instruções de

uma fonte de arquivo, decodificando-as. O primeiro método remove o processo da lista de threads da CPU tendo seu PID como parâmetro, acrescenta-o à lista de processos concluídos e o move para a tabela de memória virtual implementada na classe *InfiniteMemory*. Já no segundo método, iniciar um novo processo, é incrementado o valor do contador de Programa da tabela de processos, incrementado o contador de tempo de CPU para futuras verificações de unidades de Quantum, e por fim, adicionado na lista de processos prontos.

```
Processor.py x
class Processor(object):
    """
    A simple processor for the simulated language.

    """

    def __init__(self, numberOFCores: int = 1):
        self.numberOFCores = numberOFCores
        self.threads = []

    def getNumberOfCores(self):
        return self.numberOFCores

    def getNumberOfProcess(self):
        return len(self.threads)

    def __str__(self):
        display = "[Processor]\n"
        display += "Core count = " + str(self.numberOFCores) + "\n"

        display += "PID | TIM"
        for i in self.threads:
            display += "\n" + str(i)

        return display

    def isFull(self):
        return self.numberOFCores == len(self.threads)

    def isEmpty(self):
        return len(self.threads) == 0

    def appendProcess(self, pid: int):
        self.threads.append(ProcessorEntry(pid))
```

```
Processor.py x
def appendPreferencialProcess(self, pid: int):
    self.threads.insert(0,ProcessorEntry(pid))

def removeProcess(self, pid: int):
    self.threads.remove(pid)

def getQuantum(self, pid: int):
    return self.threads[self.threads.index(pid)].getQuantum()

def increaseQuantum(self, pid: int, quantum: int = 1):
    return self.threads[self.threads.index(pid)].increaseQuantum(quantum)

def runInstructions(self, time: int, memory, infiniteMemory, processTable, scheduler, blockedIOList, memoryManager, doneList, diagnostics):
    for thread in self.threads:
        self.increaseQuantum(thread.getPID())
        self.runInstruction(thread.getPID(), time, memory, infiniteMemory,
                            processable, scheduler, blockedIOList, memoryManager, doneList, diagnostics)

def getEmptyThreads(self):
    return self.getNumberOFCores() - self.getNumberOfProcess()

def allocMemory(self, pid: int, numberOfVariables: int, memory, processTable, scheduler, memoryManager, diagnostics):
    """
    Alloc some memory for the process.
    """

    memory_index = memory.appendProcess(pid, numberOfVariables, processTable)
    if memory_index < 0: # Index -1 means the memory allocation failed
        self.blockProcessByMemory(
            pid, numberOfVariables, processTable, scheduler, memoryManager)
        diagnostics.mmAllocFailed += 1
    else:
        processTable.increasePC(pid)
        processTable.increaseCPUTime(pid)
        processTable.setVariablesOffset(pid, memory_index)
        processTable.setMemorySize(pid, numberOfVariables)
        diagnostics.mmAllocSuccess += 1
```

```
Processor.py x

@staticmethod
def declare(pid: int, variableNumber: int, memory, processTable):
    """
    Declare a variable of the process, to be used later.
    """

    processTable.increasePC(pid)
    processTable.increaseCPUTime(pid)

    memory.declare(pid, variableNumber, processTable)

@staticmethod
def setValue(pid: int, variableNumber: int, x: int, memory, processTable):
    """
    Set the value from a variable
    """

    processTable.increasePC(pid)
    processTable.increaseCPUTime(pid)

    memory.setValue(pid, variableNumber, x, processTable)

@staticmethod
def addValue(pid: int, variableNumber: int, x: int, memory, processTable):
    """
    processTable.increasePC(pid)
    processTable.increaseCPUTime(pid)

    memory.setValue(pid, variableNumber, memory.getValue(pid, variableNumber, processTable) + x, processTable)
    """

@staticmethod
def subValue(pid: int, variableNumber: int, x: int, memory, processTable):
    """
    processTable.increasePC(pid)
    processTable.increaseCPUTime(pid)

    memory.setValue(pid, variableNumber, memory.getValue(pid, variableNumber, processTable) - x, processTable)
    """

def blockProcessByIO(self, pid: int, memory, processTable, scheduler, blockedIOList):
    """
    processTable.increasePC(pid)
    processTable.increaseCPUTime(pid)
    """


```

```
Processor.py x

def blockProcessByIO(self, pid: int, memory, processTable, scheduler, blockedIOList):
    """
    processTable.increasePC(pid)
    processTable.increaseCPUTime(pid)

    self.removeProcess(pid)
    blockedIOList.appendProcess(pid)
    scheduler.changePriorityBlockedProcess(pid, processTable)
    """

def blockProcessByMemory(self, pid: int, number_of_variables: int, processTable, scheduler, memoryManager):
    """
    Exception handler for non allocated memory
    """

    self.removeProcess(pid)
    memoryManager.addBlockedProcess(pid, number_of_variables)
    scheduler.changePriorityBlockedProcess(pid, processTable)

def terminateProcess(self, pid: int, memory, infiniteMemory, processTable, doneList):
    """
    self.removeProcess(pid)
    doneList.appendProcess(pid)

    memory.moveToInfiniteMemory(pid, processTable, infiniteMemory)
    """

@staticmethod
def forkProcess(pid: int, howManyLines: int, initialTime: int, memory, processTable, scheduler):
    """
    processTable.increasePC(pid, howManyLines+1)
    processTable.increaseCPUTime(pid)

    son_PID = processTable.fork(pid, howManyLines, initialTime)
    scheduler.addReadyProcess(son_PID, processTable)
    """

@staticmethod
def replaceProcessImage(pid: int, newFileNumber: int, memory, processTable):
    """
    processTable.increaseCPUTime(pid)

    processTable.replaceTextSection(pid, newFileNumber)
    processTable.resetPC(pid)
    """


```

```

def runSpecificInstruction(self, pid: int, line: int, time: int, memory, infiniteMemory, processTable, scheduler, blockedIOList, memoryManager, doneList, diagnostics):
    instruction = processTable.getInstruction(pid, line)
    opcode: str = instruction.opcode
    n: int = instruction.n
    x: int = instruction.x

    diagnostics.instructions += 1

    if opcode == "A":
        self.allocMemory(
            pid, n, memory, processTable, scheduler, memoryManager, diagnostics)
        diagnostics.N += 1
    elif opcode == "B":
        Processor.declare(pid, n, memory, processTable)
        diagnostics.B += 1
    elif opcode == "C":
        Processor.setValue(pid, n, x, memory, processTable)
        diagnostics.C += 1
    elif opcode == "D":
        Processor.addValue(pid, n, x, memory, processTable)
        diagnostics.D += 1
    elif opcode == "E":
        Processor.addValue(pid, n, x, memory, processTable)
        diagnostics.E += 1
    elif opcode == "F":
        Processor.subValue(pid, n, x, memory, processTable)
        diagnostics.F += 1
    elif opcode == "G":
        self.blockProcessByIO(
            pid, memory, processTable, scheduler, blockedIOList)
        diagnostics.G += 1
    elif opcode == "H":
        self.terminateProcess(
            pid, memory, infiniteMemory, processTable, doneList)
        diagnostics.H += 1
        diagnostics.rawResponseTime += time
    elif opcode == "I":
        Processor.forkProcess(pid, n, time, memory,
                             processTable, scheduler)
        diagnostics.I += 1
    elif opcode == "J":
        Processor.replaceProcessImage(pid, n, memory, processTable)
        diagnostics.J += 1

```

6.2.1 Classe Processor Entry

Esta classe contém um PID e o tempo gasto no processo em questão. Ela define métodos como incrementar o valor do Quantum, retornar o número do PID, de Quantum, e inicializar o valor do Quantum.

```

def runInstruction(self, pid: int, time: int, memory, infiniteMemory, processTable, scheduler, blockedIOList, memoryManager, doneList, diagnostics):
    self.runSpecificInstruction(pid, processTable.getPC(
        pid), time, memory, infiniteMemory, processTable, scheduler, blockedIOList, memoryManager, doneList, diagnostics)

class ProcessorEntry(object):
    """
    A processor entry contains one PID and the time spent in that process
    """

    def __init__(self, pid: int):
        self.pid = pid
        self.quantum = 0

    def __str__(self):
        return str(self.pid).zfill(3) + " | " + str(self.quantum).zfill(3)

    def increaseQuantum(self, time: int = 1):
        self.quantum += time

    def getPID(self):
        return self.pid

    def getQuantum(self):
        return self.quantum

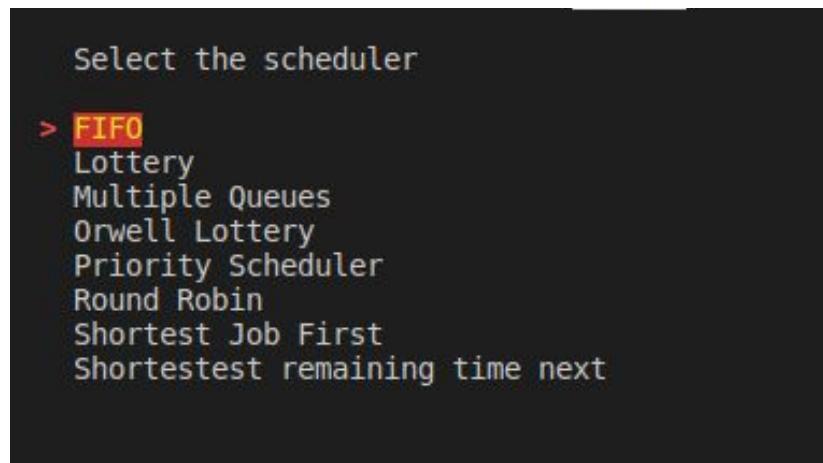
    def __eq__(self, pid: int):
        return self.pid == pid

```

7. Escalonadores

Foram implementadas os seguintes escalonadores: First-in First-Out (*FirstInFirstOutScheduler*), Escalonamento por Loteria (*LotteryScheduler*), (*OrwellLotteryScheduler*), Filas Múltiplas (*MultipleQueuesScheduler*), Escalonamento por Prioridades (*PriorityScheduler*), Escalonamento por Chaveamento Circular (*RoundRobinScheduler*), Tarefa Mais Curta Primeiro (*ShortestJobFirstScheduler*) e Próximo de Menor Tempo Restante (*ShortestRemainingTimeNextScheduler*), todos em conformidade com o texto base da disciplina.

Para tais escalonadores, existe uma classe Lista de Processos (*ProcessList*) que armazena em uma fila todos os PIDs dos processos que serão usados para o escalonamento. Esta classe contém os métodos para adicionar, remover, mostrar o primeiro processo da lista, retornar se a lista vazia e desempilhar um processo, tudo através do PID do processo. Todos os escalonadores, a princípio, leem os a fila de processos que estão atualmente na estrutura de dados da *ProcessList*.



No menu config, é possível escolher a opção “set process scheduler” e selecionar o escalonador desejado.

7.1. First-In First-Out Scheduler

O funcionamento é uma pilha implementada. O escalonador possui uma fila de prontos. Para manipular esta lista tem-se os métodos de adicionar o processo pelo PID e sua tabela por referência; remover um processo pelo seu PID; mudar a prioridade de processos bloqueados; obter o processo mais antigo na lista de prontos através da consulta na tabela de processos, por referência; a classe mantém ativa em um bloco while, enquanto há threads que podem ser executadas (processador não está totalmente ocupado e há pelo menos um processo na fila de processos da classe), o processo em questão é removido da lista de processos prontos (o mais antigo da tabela de processos), e é acrescentado a classe *Processor* com o PID como parâmetro.

```
FIRSTINFIRSTOUTSCHEDULER.PY X
from Schedulers.ProcessList import ProcessList

class FirstInFirstOutScheduler(object):
    """
    A simple First-in First-out Scheduler for multicore CPUS
    """

    def __init__(self):
        self.readyList = ProcessList()

    def __str__(self):
        return "[FIFO Scheduler]\n" + str(self.readyList)

    def isEmpty(self):
        return self.readyList.isEmpty()

    def addReadyProcess(self, pid: int, processTable):
        self.readyList.appendProcess(pid)

    def removeReadyProcess(self, pid: int):
        self.readyList.removeProcess(pid)

    def changePriorityBlockedProcess(self, pid: int, processTable):
        pass

    def getOldestPID(self, processTable) -> int:
        ready_pid_time = []

        # Creating tuples on the form (PID, Init Time)
        for elem in self.readyList.queue:
            ready_pid_time.append((elem, processTable.getInitTime(elem)))

        # Return the PID (first element of the tuple) from the tuple that has the minimal
        # value of Init Time
        return min(ready_pid_time, key=lambda tup: tup[1])[0]

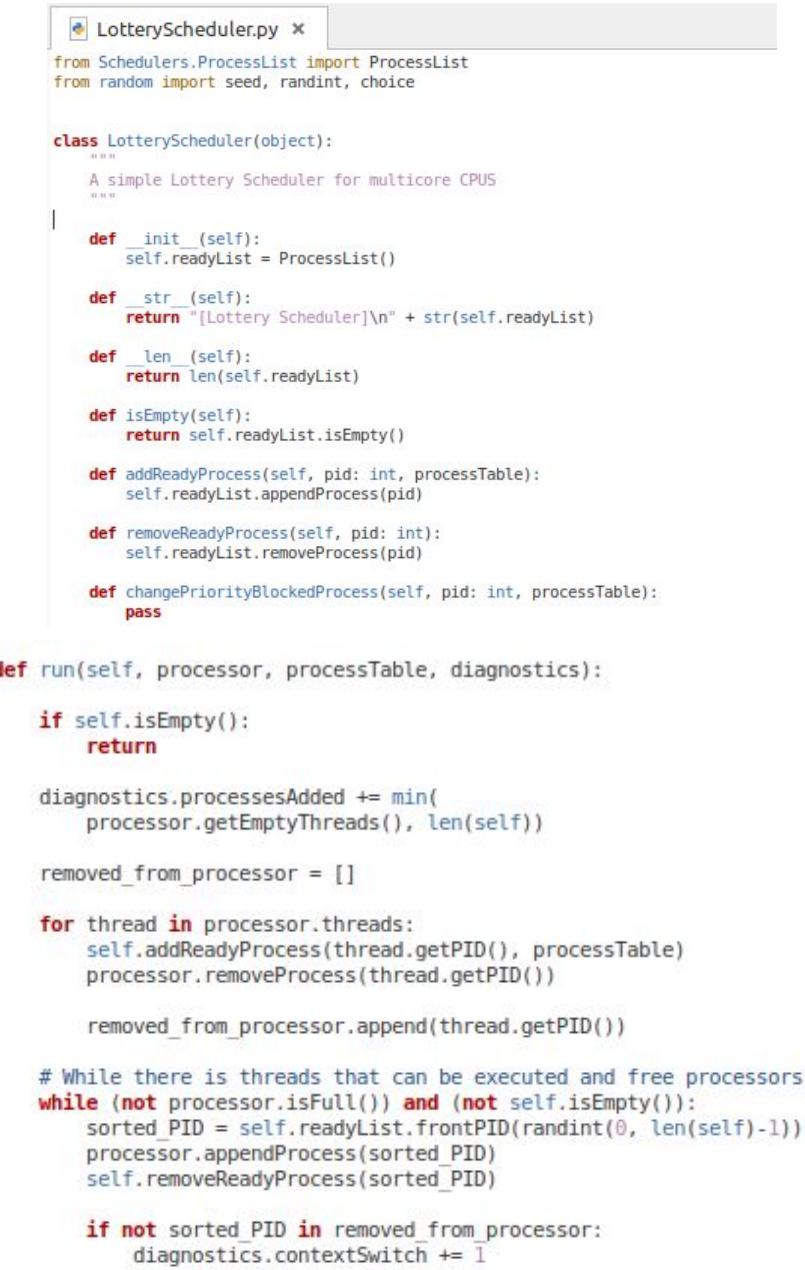
    def run(self, processor, processTable, diagnostics):
        # While there is threads that can be executed and free processors
        while (not processor.isFull()) and (not self.isEmpty()):
            pid_to_be_scheduled = self.getOldestPID(processTable)

            self.removeReadyProcess(pid_to_be_scheduled)
            processor.appendProcess(pid_to_be_scheduled)

            diagnostics.processesAdded += 1
```

7.2. Lottery Scheduler

Este algoritmo sorteia um número de zero até a quantidade de processos prontos da lista de prontos, através da função *randint* da biblioteca *randint* do Python. Este número será o PID sorteado da lista de processos prontos. Este processo então sorteado conseguirá o recurso de uso de CPU, e assim será adicionado à execução da classe CPU e removido da lista de processos prontos. Os processos não sorteados da lista de processos removidos da CPU, são válidos para incrementar o contador de mudança de contexto (ou seja, haverá processos que ainda não foram executados e precisarão de um tempo da CPU para realizar o que for necessário).



```
LotteryScheduler.py x
from Schedulers.ProcessList import ProcessList
from random import seed, randint, choice

class LotteryScheduler(object):
    """
    A simple Lottery Scheduler for multicore CPUS
    """

    def __init__(self):
        self.readyList = ProcessList()

    def __str__(self):
        return "[Lottery Scheduler]\n" + str(self.readyList)

    def __len__(self):
        return len(self.readyList)

    def isEmpty(self):
        return self.readyList.isEmpty()

    def addReadyProcess(self, pid: int, processTable):
        self.readyList.appendProcess(pid)

    def removeReadyProcess(self, pid: int):
        self.readyList.removeProcess(pid)

    def changePriorityBlockedProcess(self, pid: int, processTable):
        pass

    def run(self, processor, processTable, diagnostics):
        if self.isEmpty():
            return

        diagnostics.processesAdded += min(
            processor.getEmptyThreads(), len(self))

        removed_from_processor = []
        for thread in processor.threads:
            self.addReadyProcess(thread.getPID(), processTable)
            processor.removeProcess(thread.getPID())

            removed_from_processor.append(thread.getPID())

        # While there is threads that can be executed and free processors
        while (not processor.isFull()) and (not self.isEmpty()):
            sorted_PID = self.readyList.frontPID(randint(0, len(self)-1))
            processor.appendProcess(sorted_PID)
            self.removeReadyProcess(sorted_PID)

            if not sorted_PID in removed_from_processor:
                diagnostics.contextSwitch += 1
```

7.3. Multiple Queues Scheduler

O escalonador de múltiplas filas implementa prioridades para o escalonamento através da consulta de prioridade por PID na tabela de processos. É possível então, adicionar um processo pronto e mudar a prioridade de um processo bloqueado (quando necessário). Também é possível escolher o número de filas uma vez que esta é variável. Para verificar quanto tempo o processo em questão está sendo usado pela CPU, foi criado uma condição que limita o valor do quantum (2 elevado ao índice do processo na fila). Caso o processo já não tem mais tempo na CPU ele é removido desta, e colocado em uma lista de processos que foram executados pela CPU.

```
MultipleQueuesScheduler.py x
from Schedulers.ProcessList import ProcessList

class MultipleQueuesScheduler(object):
    """
    A simple Priority| Scheduler for multicore CPUS
    """

    def __init__(self, NQueues: int = 3):
        self.NQueues = NQueues
        self.queues = []

        for _i in range(self.NQueues):
            self.queues.append([])

    def __str__(self):
        display = "[Multiple Queues Scheduler]"
        for queue in range(self.NQueues):
            display += "\nQueue " + str(queue) + " :"
            for pid in self.queues[queue]:
                display += " " + str(pid)

        return display

    def __len__(self):
        lenght = 0

        for i in range(self.NQueues):
            lenght += len(self.queues[i])

        return lenght

    def isEmpty(self):
        return len(self) == 0

    def addReadyProcess(self, pid: int, processTable):
        self.queues[processTable.getPriority(pid)].append(pid)

    def changePriorityBlockedProcess(self, pid: int, processTable):
        if processTable.getPriority(pid) > 0:
            processTable.setPriority(pid, processTable.getPriority(pid) - 1)

    def run(self, processor, processTable, diagnostics):
        was_in_processor = []
        diagnostics.processesAdded += min(
            processor.getEmptyThreads(), len(self))
```

```

if self.isEmpty():
    return

for thread in processor.threads:
    if thread.getQuantum() >= pow(2, processTable.getPriority(thread.getPID())):
        self.addReadyProcess(thread.getPID(), processTable)
        processor.removeProcess(thread.getPID())
        was_in_processor.append(thread.getPID())

    if processTable.getPriority(thread.getPID()) < self.NQueues:
        processTable.setPriority(
            thread.getPID(), processTable.getPriority(thread.getPID()) + 1)

i = 0
while not processor.isFull() and not self.isEmpty() and i < self.NQueues:
    try:
        pid_sched = self.queues[i].pop()
        processor.appendProcess(pid_sched)

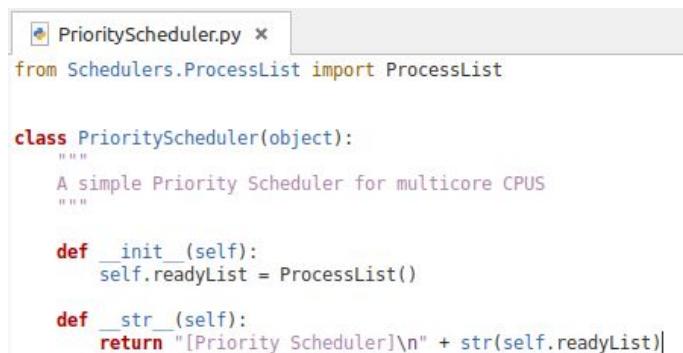
        if not pid_sched in was_in_processor:
            diagnostics.contextSwitch += 1

    except IndexError:
        pass
    finally:
        i += 1

```

7.4. Priority Scheduler

Inicialmente é montada a lista de processos que estão prontos para serem executados. É possível adicionar e remover processos da lista de prontos e mudar a prioridade de um processo bloqueado. No escalonador implementado, o principal método que define exatamente sua funcionalidade real dentro da CPU é a obtenção do PID com maior prioridade da tabela de processos. Isto é feito criando tuplas da forma (PID, init Time). Para cada elemento criado (dentre todos os processos que estão na lista de prontos da classe), este é verificado na tabela de processos e obtido sua prioridade atual. Em seguida, é retornado o PID (primeiro elemento da tupla) da tupla que tem o menor valor de tempo de início de execução. Assim, se define as prioridades. O que tem o valor maior de prioridade tem preferência na execução, e assim por diante.



```

PriorityScheduler.py x
from Schedulers.ProcessList import ProcessList

class PriorityScheduler(object):
    """
    A simple Priority Scheduler for multicore CPUS
    """

    def __init__(self):
        self.readyList = ProcessList()

    def __str__(self):
        return "[Priority Scheduler]\n" + str(self.readyList)

```

```

def isEmpty(self):
    return self.readyList.isEmpty()

def addReadyProcess(self, pid: int, processTable):
    self.readyList.appendProcess(pid)

def removeReadyProcess(self, pid: int):
    self.readyList.removeProcess(pid)

def changePriorityBlockedProcess(self, pid: int, processTable):
    processTable.setPriority(pid, processTable.getPriority(pid) + 1)

def getHighestPriorityPID(self, processTable) -> int:
    ready_pid_time = []

    # Creating tuples on the form (PID, Init Time)
    for elem in self.readyList.queue:
        ready_pid_time.append((elem, processTable.getPriority(elem)))

    # Return the PID (first element of the tuple) from the tuple that has the
    # value of Init Time
    return min(ready_pid_time, key=lambda tup: tup[1])[0]

def run(self, processor, processTable, diagnostics):

    # While there is threads that can be executed and free processors
    while (not processor.isFull()) and (not self.isEmpty()):
        pid_to_be_scheduled = self.getHighestPriorityPID(processTable)

        self.removeReadyProcess(pid_to_be_scheduled)
        processor.appendProcess(pid_to_be_scheduled)

        diagnostics.processesAdded += 1

```

7.5. Round-Robin Scheduler

Para cada processo foi atribuído seu Quantum (a quantidade de tempo permitido para o processo executar). Isso é passado em forma de parâmetro iniciando com o valor 3 (flexível). Nesta classe é possível adicionar e remover processos da fila de prontos; desempilhar um processo pelo seu PID, e alterar a prioridade de um processo bloqueado. É inicializado o tempo atual para controle do próprio escalonador e os Quanta. Se o tempo atual for menor que o valor do Quantum do processo (ou seja, no final do Quantum atribuído o processo ainda está executando, o que não pode acontecer) ou a lista de processos prontos estiver vazia, é incrementado uma unidade de tempo ao tempo atual do processamento. Se o processo foi bloqueado ou terminou antes que o Quantum tenha decorrido, a CPU (dentro de um bloco while) é chaveada para um outro processo, sendo este adicionado novamente à lista de prontos e removido da CPU. Quando o processo usa todo seu quantum, ele é colocado no final da lista.

```

    RoundRobinScheduler.py x
from Schedulers.ProcessList import ProcessList

class RoundRobinScheduler(object):
    """
        Girando girando girando pra um lado, girando pro outro (brazilian music)
    """

    def __init__(self, quantum: int = 3):
        self.readyList = ProcessList()
        self.quantum = quantum
        self.currentTime = 0

    def __str__(self):
        return "[Round Robin Scheduler]\n" + str(self.readyList)

    def __len__(self):
        return len(self.readyList)

    def isEmpty(self):
        return self.readyList.isEmpty()

    def addReadyProcess(self, pid: int, processTable):
        self.readyList.appendProcess(pid)

    def removeReadyProcess(self, pid: int = 0):
        self.readyList.removeProcess(pid)

    def unqueueProcess(self):
        return self.readyList.unqueue()

    def changePriorityBlockedProcess(self, pid: int, processTable):
        pass

    def run(self, processor, processTable, diagnostics):
        if (self.currentTime < self.quantum) or self.isEmpty():
            self.currentTime += 1
            return

        for thread in processor.threads:
            self.addReadyProcess(thread.getPID(), processTable)
            processor.removeProcess(thread.getPID())

        while not processor.isFull() and not self.isEmpty():
            processor.appendProcess(self.unqueueProcess())
            diagnostics.contextSwitch += 1

```

7.6. - Shortest Job First Scheduler

Nesse escalonamento é selecionado para executar o processo com o menor tempo de execução. A classe implementa dois métodos auxiliares para realização do algoritmo de escalonamento: o método para obter o processo com maior tempo da tabela de processos, e o método para obter o maior PID da tabela de processos. A classe, então, realiza no método *run* a invocação de um outro método que obtém o menor tempo da tabela de processos (a tabela de processos é passada como referência).

Enquanto existir processos com tempos menores do que o processo com maior tempo que está na tabela de processos, o método adicionará o processo em questão à lista de prontos e também adicionará à real execução da CPU.



```
ShortestJobFirstScheduler.py x
from Schedulers.ProcessList import ProcessList

class ShortestJobFirstScheduler(object):
    """
    A simple Shortest Job First Scheduler for multicore CPUS
    """

    def __init__(self):
        self.readyList = ProcessList()

    def __str__(self):
        return "[Shortest Job First Scheduler]\n" + str(self.readyList)

    def isEmpty(self):
        return self.readyList.isEmpty()

    def addReadyProcess(self, pid: int, processTable):
        self.readyList.appendProcess(pid)

    def removeReadyProcess(self, pid: int):
        self.readyList.removeProcess(pid)

    def changePriorityBlockedProcess(self, pid: int, processTable):
        pass

    def getShortestPIDFromScheduler(self, processTable) -> int:
        ready_pid_time = []

        for elem in self.readyList.queue:
            ready_pid_time.append(
                (elem, processTable.predictTotalJobTime(elem)))

        return min(ready_pid_time, key=lambda tup: tup[1])[0]

    def getShortestTimeFromScheduler(self, processTable) -> int:
        ready_pid_time = []

        for elem in self.readyList.queue:
            ready_pid_time.append(
                (elem, processTable.predictTotalJobTime(elem)))

        return min(ready_pid_time, key=lambda tup: tup[1])[1]
```

```

ShortestJobFirstScheduler.py x
    (elem, processTable.predictTotalJobTime(elem)))

    return min(ready_pid_time, key=lambda tup: tup[1])[1]

@staticmethod
def getBiggestTimeFromProcessor(processor, processTable) -> int:
    ready_pid_time = []

    for thread in processor.threads:
        ready_pid_time.append(
            (thread.getPID(), processTable.predictTotalJobTime(thread.getPID())))

    return max(ready_pid_time, key=lambda tup: tup[1])[1]

@staticmethod
def getBiggestPIDFromProcessor(processor, processTable) -> int:
    ready_pid_time = []

    for thread in processor.threads:
        ready_pid_time.append(
            (thread.getPID(), processTable.predictTotalJobTime(thread.getPID())))

    return max(ready_pid_time, key=lambda tup: tup[1])[0]

def run(self, processor, processTable, diagnostics):
    if self.isEmpty():
        return

    while not processor.isFull() and not self.isEmpty():
        processor.appendProcess(
            self.getShortestPIDFromScheduler(processTable))
        self.removeReadyProcess(
            self.getShortestPIDFromScheduler(processTable))

        diagnostics.processesAdded += 1

    if not processor.isEmpty() and not self.isEmpty():
        while self.getShortestTimeFromScheduler(processTable) < self.getBiggestTimeFromProcessor(processor, processTable):
            self.addReadyProcess(
                self.getBiggestPIDFromProcessor(processor, processTable), processTable)
            processor.removeProcess(
                self.getBiggestPIDFromProcessor(processor, processTable))

            processor.appendProcess(
                self.getShortestPIDFromScheduler(processTable))
            self.removeReadyProcess(
                self.getShortestPIDFromScheduler(processTable))

```

7.7 - Shortest Remaining Time Next Scheduler

Esta classe é responsável por implementar a fila de processos a serem executados pelo algoritmo e é organizada conforme o tempo estimado de execução, ou seja, de forma semelhante ao Shortest Job First Scheduler, sendo processados primeiros os menores jobs. Na entrada de um novo processo, o algoritmo de escalonamento avalia seu tempo de execução incluindo o job em execução, caso a estimativa de seu tempo de execução seja menor que o do processo concorrentemente em execução, ocorre a substituição do processo em execução pelo recém chegado, de duração mais curta, ou seja, ocorre a preempção do processo em execução. A seguir, o código em figuras:

```

  ShorestRemainingTimeNextScheduler.py ×
from Schedulers.ProcessList import ProcessList

class ShortestRemainingTimeNextScheduler(object):
    """
    A simple Shortest Remaining Time Next Scheduler for multicore CPUS
    """

    def __init__(self):
        self.readyList = ProcessList()

    def __str__(self):
        return "[Shortest Remaining Time Next Scheduler]\n" + str(self.readyList)

    def isEmpty(self):
        return self.readyList.isEmpty()

    def addReadyProcess(self, pid: int, processTable):
        self.readyList.appendProcess(pid)

    def removeReadyProcess(self, pid: int):
        self.readyList.removeProcess(pid)

    def changePriorityBlockedProcess(self, pid: int, processTable):
        pass

    def getShortestPIDFromScheduler(self, processTable) -> int:
        ready_pid_time = []

        for elem in self.readyList.queue:
            ready_pid_time.append(
                (elem, processTable.predictRemainingJobTime(elem)))

        return min(ready_pid_time, key=lambda tup: tup[1])[0]

    def getShortestTimeFromScheduler(self, processTable) -> int:
        ready_pid_time = []

        for elem in self.readyList.queue:
            ready_pid_time.append(
                (elem, processTable.predictRemainingJobTime(elem)))

        return min(ready_pid_time, key=lambda tup: tup[1])[1]

    @staticmethod
    def getBiggestTimeFromProcessor(processor, processTable) -> int:
        ready_pid_time = []

        for thread in processor.threads:
            ready_pid_time.append(
                (thread.getPID(), processTable.predictRemainingJobTime(thread.getPID())))

        return max(ready_pid_time, key=lambda tup: tup[1])[1]

    @staticmethod
    def getBiggestPIDFromProcessor(processor, processTable) -> int:
        ready_pid_time = []

        for thread in processor.threads:
            ready_pid_time.append(
                (thread.getPID(), processTable.predictRemainingJobTime(thread.getPID())))

        return max(ready_pid_time, key=lambda tup: tup[1])[0]

    def run(self, processor, processTable, diagnostics):
        if self.isEmpty():
            return

        while not processor.isFull() and not self.isEmpty():
            processor.appendProcess(
                self.getShortestPIDFromScheduler(processTable))
            self.removeReadyProcess(
                self.getShortestPIDFromScheduler(processTable))

            diagnostics.processesAdded += 1

        if not processor.isEmpty() and not self.isEmpty():
            while self.getShortestTimeFromScheduler(processTable) < self.getBiggestTimeFromProcessor(processor, processTable):
                self.addReadyProcess(
                    self.getBiggestPIDFromProcessor(processor, processTable))
                processor.removeProcess(
                    self.getBiggestPIDFromProcessor(processor, processTable))

            processor.appendProcess(
                self.getShortestPIDFromScheduler(processTable))
            self.removeReadyProcess(
                self.getShortestPIDFromScheduler(processTable))

            diagnostics.contextSwitch += 1

```

7. Memórias

7.1. Memória Virtual (Infinita)

A memória virtual foi implementada utilizando um dicionário de listas. Cada chave do dicionário equivale a um PID, e os elementos das listas são as variáveis.

7.2. Memória Física

Foi implementado uma memória simples através do uso de um vetor. Foram criadas duas funções especiais para a memória que formaram a base para os algoritmos de alocação de memória. A primeira função é “*getFreeSegments*” a qual percorre a memória criando uma lista de tuplas, sendo o primeiro elemento a posição inicial do espaço vazio, e o segundo elemento o tamanho do espaço vazio. Ela retorna uma lista de posições vazias (ou livres). Já a segunda função criada “*getFits*” é responsável por comparar o segundo elemento de cada tupla: se for menor ou igual o solicitado pelos programas, a função mantém a tupla do jeito que está, senão ela a remove. Segue os dois screenshots das duas funções:

```
def getFreeSegments(self):
    fits_list = []
    free_contiguous = 0

    for i in range(len(self.memory)):
        if not self.memory[i].isAllocated():
            free_contiguous += 1
        else:
            if free_contiguous != 0:
                fits_list.append((i-free_contiguous+1, free_contiguous))
            free_contiguous = 0

    if free_contiguous != 0:
        fits_list.append((i-free_contiguous+1, free_contiguous))

    return fits_list

def haveMemoryAvailable(self, numberOfVariables: int) -> bool:
    if len(self.getFits(numberOfVariables)) > 0:
        return True
    else:
        return False

def getFits(self, numberOfVariables: int):
    fits_list = self.getFreeSegments()

    for i in range(len(fits_list)):
        if fits_list[i][1] < numberOfVariables:
            fits_list.pop(i)

    return fits_list
```

Em seguida, foram realizados testes alocando-se as variáveis dos processos simulados. Nos testes que serão apresentados a seguir, definimos tamanho n = 5 para cada uma das memórias.

7.2.1. First fit

```
[Process Table]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 000 | -01 | 000 | 000 | 000 | 000 | PID | TIM |
| | | | | | | | | 000 | 000 |
-----[Physical Memory (size: 5) (First fit)] |
| XXX XXX XXX XXX XXX |
-----[FIFO Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list] |
| | | | |
-----[Instructions runned] | [Final Report] | [Time]
N: 0 Context switches: 0 | Time spent: 0
D: 0 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 0
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 100%
F: 0 Memory allocation sucess: 0
R: 0 Memory allocation failed: 0
External memory fragments: 0
-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.
Type an option: █
```

```
[Process Table]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 002 | 000 | 005 | 000 | 000 | 002 | PID | TIM |
| | | | | | | | | 000 | 002 |
-----[Physical Memory (size: 5) (First fit)] |
| 000 | 000 | 000 | 000 | 000 |
-----[FIFO Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list] |
| | | | |
-----[Instructions runned] | [Final Report] | [Time]
N: 1 Context switches: 0 | Time spent: 2
D: 1 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 2
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1
-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.
Type an option: █
```

[Process Table]								[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT	PID	TIM		
000	-001	003	000	005	000	000	003	000	003		

[Physical Memory (size: 5) (First fit)]			
000	000	000	000

[FIFO Scheduler]		[Blocked by IO]		[Blocked by Memory Heap]		[Done list]	
[]							

[Instructions runned]								[Final Report]		[Time]		
N: 1	D: 2	V: 0	A: 0	S: 0	B: 0	T: 0	F: 0	R: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 3 [Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	Time spent: 3		

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]								[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT	PID	TIM		
000	-001	004	000	005	000	000	004	000	004		

[Physical Memory (size: 5) (First fit)]			
000	000	000	000

[FIFO Scheduler]		[Blocked by IO]		[Blocked by Memory Heap]		[Done list]	
[]							

[Instructions runned]								[Final Report]		[Time]		
N: 1	D: 3	V: 0	A: 0	S: 0	B: 0	T: 0	F: 0	R: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 4 [Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	Time spent: 4		

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]								[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT	PID	TIM		
000	-001	005	000	005	000	000	005	000	005		

[Physical Memory (size: 5) (First fit)]			
000	000	000	000

[FIFO Scheduler]		[Blocked by IO]		[Blocked by Memory Heap]		[Done list]	
[]							

[Instructions runned]								[Final Report]		[Time]		
N: 1	D: 4	V: 0	A: 0	S: 0	B: 0	T: 0	F: 0	R: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 5 [Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	Time spent: 5		

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]						[3-Core Processor]			[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM	
000	-001	006	000	005	000	000	006	000	006	

[Physical Memory (size: 5) (First fit)]				
000	000	000	000	000

[FIFO Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
[]					

[Instructions runned]			[Final Report]			[Time]		
N: 1	D: 5	V: 0	A: 0	S: 0	B: 0	T: 0	F: 0	R: 0
Processes added: 0	Response time: inf	Priority changes: 0	Instructions runned: 6			Time spent: 6		

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]						[3-Core Processor]			[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM	
000	-001	008	000	005	000	000	007	000	007	
001	0000	000	-01	000	000	007	000	001	000	

[Physical Memory (size: 5) (First fit)]				
000	000	000	000	000

[FIFO Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
[]					

[Instructions runned]			[Final Report]			[Time]		
N: 1	D: 5	V: 0	A: 0	S: 0	B: 0	T: 0	F: 1	R: 0
Processes added: 1	Response time: inf	Priority changes: 0	Instructions runned: 7			Time spent: 7		

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]						[3-Core Processor]			[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM	
000	-001	009	000	005	000	000	008	000	008	
001	0000	000	-01	000	000	007	001	001	001	

[Physical Memory (size: 5) (First fit)]				
010	000	000	000	000

[FIFO Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
[]					

[Instructions runned]			[Final Report]			[Time]		
N: 1	D: 5	V: 1	A: 0	S: 0	B: 0	T: 0	F: 1	R: 1
Processes added: 1	Response time: inf	Priority changes: 0	Instructions runned: 9			Time spent: 8		

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

Usando a técnica de first-fit para alocação da memória com tamanho 5, foi possível visualizar que houve uma média de 1 fragmento externo, um tempo médio de 8 e nenhuma requisição de alocação negada, com a lista de bloqueados vazia.

7.2.2. Best fit

```
[Process Table]
PID | FPID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 000 | -01 | 000 | 000 | 000 | 000 | PID | TIM | 000 | 000

[Physical Memory (size: 5) (Best fit)]
XXX XXX XXX XXX XXX

[3 Queues Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list]
Queue 0 : | | | |
Queue 1 : | | | |
Queue 2 : | | | |

[Instructions runned] | [Final Report] | [Time]
N: 0 | Context switches: 0 | Time spent: 0
D: 0 | Processes added: 0 |
V: 0 | Response time: inf |
A: 0 | Priority changes: 0 |
S: 0 | Instructions runned: 0 |
B: 0 | [Memory info] |
T: 0 | Memory allocation failed ratio: 100% |
F: 0 | Memory allocation sucess: 0 |
R: 0 | Memory allocation failed: 0 |
| External memory fragments: 0 |

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■
```

[Process Table]							[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT	PID	TIM	
000	-001	001	000	005	000	000	001	000	001	

[Physical Memory (size: 5) (Best fit)]			
000	000	000	000

[3 Queues Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 :					
Queue 1 :					
Queue 2 :					

[Instructions runned]			[Final Report]			[Time]		
N: 1	Context switches: 0		D: 0	Processes added: 0		V: 0	Response time: inf	
D: 0			V: 0			A: 0	Priority changes: 0	
V: 0			A: 0	Instructions runned: 1		S: 0		
A: 0			S: 0			B: 0	[Memory info]	
S: 0			B: 0			T: 0	Memory allocation failed ratio: 0.0%	
B: 0			T: 0			F: 0	Memory allocation sucess: 1	
T: 0			F: 0			R: 0	Memory allocation failed: 0	
F: 0			R: 0				External memory fragments: 1	

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]							[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT	PID	TIM	
000	-001	002	000	005	000	000	002	000	002	

[Physical Memory (size: 5) (Best fit)]			
000	000	000	000

[3 Queues Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 :					
Queue 1 :					
Queue 2 :					

[Instructions runned]			[Final Report]			[Time]		
N: 1	Context switches: 0		D: 1	Processes added: 0		V: 0	Response time: inf	
D: 1			V: 0			A: 0	Priority changes: 0	
V: 0			A: 0	Instructions runned: 2		S: 0		
A: 0			S: 0			B: 0	[Memory info]	
S: 0			B: 0			T: 0	Memory allocation failed ratio: 0.0%	
B: 0			T: 0			F: 0	Memory allocation sucess: 1	
T: 0			F: 0			R: 0	Memory allocation failed: 0	
F: 0			R: 0				External memory fragments: 1	

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]							[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT	PID	TIM	
000	-001	003	000	005	000	000	003	000	003	

[Physical Memory (size: 5) (Best fit)]			
000	000	000	000

[3 Queues Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 :					
Queue 1 :					
Queue 2 :					

[Instructions runned]			[Final Report]			[Time]		
N: 1	Context switches: 0		D: 2	Processes added: 0		V: 0	Response time: inf	
D: 2			V: 0			A: 0	Priority changes: 0	
V: 0			A: 0	Instructions runned: 3		S: 0		
A: 0			S: 0			B: 0	[Memory info]	
S: 0			B: 0			T: 0	Memory allocation failed ratio: 0.0%	
B: 0			T: 0			F: 0	Memory allocation sucess: 1	
T: 0			F: 0			R: 0	Memory allocation failed: 0	
F: 0			R: 0				External memory fragments: 1	

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]					[3-Core Processor]			[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM
000	-001	004	000	005	000	000	004	000	004
[Physical Memory (size: 5) (Best fit)]									
000	000	000	000	000					

[3 Queues Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 :					
Queue 1 :					
Queue 2 :					

[Instructions runned]				[Final Report]	[Time]
N: 1	D: 3	V: 0	A: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 4	Time spent: 4
S: 0	B: 0	T: 0	F: 0	[Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	
R: 0					

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]					[3-Core Processor]			[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM
000	-001	005	000	005	000	000	005	000	005
[Physical Memory (size: 5) (Best fit)]									
000	000	000	000	000					

[3 Queues Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 :					
Queue 1 :					
Queue 2 :					

[Instructions runned]				[Final Report]	[Time]
N: 1	D: 4	V: 0	A: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 5	Time spent: 5
S: 0	B: 0	T: 0	F: 0	[Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	
R: 0					

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]					[3-Core Processor]			[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM
000	-001	006	000	005	000	000	006	000	006
[Physical Memory (size: 5) (Best fit)]									
000	000	000	000	000					

[3 Queues Scheduler]			[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 :					
Queue 1 :					
Queue 2 :					

[Instructions runned]				[Final Report]	[Time]
N: 1	D: 5	V: 0	A: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 6	Time spent: 6
S: 0	B: 0	T: 0	F: 0	[Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	
R: 0					

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

Usando a técnica de best-fit para alocação da memória com tamanho 5, foi possível visualizar que houve uma média de 1 fragmento externo, um tempo médio de 6 e nenhuma requisição de alocação negada, com a lista de bloqueados vazia.

7.2.3. Worst fit

```

[Process Table]
PID | FPID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 000 | -01 | 000 | 000 | 000 | 000 | PID | TIM
000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000

[Physical Memory (size: 5) (Worst fit)]
XXX XXX XXX XXX XXX

-----[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]-----
Queue 0 : | | | |
Queue 1 : | | | |
Queue 2 : | | | |

-----[Instructions runned] [Final Report] [Time]
N: 0 Context switches: 0 Time spent: 0
D: 0 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 0
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 100%
F: 0 Memory allocation sucess: 0
R: 0 Memory allocation failed: 0
External memory fragments: 0

-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■

-----[Process Table]
PID | FPID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 001 | 000 | 005 | 000 | 000 | 001 | PID | TIM
000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 001

[Physical Memory (size: 5) (Worst fit)]
000 000 000 000 000

-----[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]-----
Queue 0 : | | | |
Queue 1 : | | | |
Queue 2 : | | | |

-----[Instructions runned] [Final Report] [Time]
N: 1 Context switches: 0 Time spent: 1
D: 0 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 1
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1

-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■

```

[Process Table]							[3-Core Processor]		[Infinite Memory]		
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM		
000	-001	002	000	005	000	000	002	000	002		

[Physical Memory (size: 5) (Worst fit)]									
000	000	000	000	000					

[3 Queues Scheduler]			[Blocked by IO]		[Blocked by Memory Heap]			[Done list]			
Queue 0 :											
Queue 1 :											
Queue 2 :											

[Instructions runned]				[Final Report]				[Time]			
N: 1	Context switches: 0	D: 1	Processes added: 0	V: 0	Response time: inf	A: 0	Priority changes: 0	S: 0	Instructions runned: 2	B: 0	Memory info
T: 0	Memory allocation failed ratio: 0.0%	F: 0	Memory allocation sucess: 1	R: 0	Memory allocation failed: 0		External memory fragments: 1				

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]							[3-Core Processor]		[Infinite Memory]		
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM		
000	-001	003	000	005	000	000	003	000	003		

[Physical Memory (size: 5) (Worst fit)]									
000	000	000	000	000					

[3 Queues Scheduler]			[Blocked by IO]		[Blocked by Memory Heap]			[Done list]			
Queue 0 :											
Queue 1 :											
Queue 2 :											

[Instructions runned]				[Final Report]				[Time]			
N: 1	Context switches: 0	D: 2	Processes added: 0	V: 0	Response time: inf	A: 0	Priority changes: 0	S: 0	Instructions runned: 3	B: 0	Memory info
T: 0	Memory allocation failed ratio: 0.0%	F: 0	Memory allocation sucess: 1	R: 0	Memory allocation failed: 0		External memory fragments: 1				

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

[Process Table]							[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VAD	VAS	PRI	INT	CPT	PID	TIM	
000	-001	004	000	005	000	000	004	000	004	

[Physical Memory (size: 5) (Worst fit)]		
000	000	000

[3 Queues Scheduler]	[Blocked by IO]	[Blocked by Memory Heap]	[Done list]
Queue 0 : Queue 1 : Queue 2 :			

[Instructions runned]	[Final Report]	[Time]
N: 1	Context switches: 0	Time spent: 4
D: 3	Processes added: 0	
V: 0	Response time: inf	
A: 0	Priority changes: 0	
S: 0	Instructions runned: 4	
B: 0		
I: 0	[Memory info]	
F: 0	Memory allocation failed ratio: 0.0%	
R: 0	Memory allocation sucess: 1	
	Memory allocation failed: 0	
	External memory fragments: 1	

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

```

[Process Table]          [3-Core Processor]          [Infinite Memory]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | PID | TIM
000 | -001 | 005 | 000 | 005 | 000 | 000 | 005 | 000 | 005

[Physical Memory (size: 5) (Worst fit)]
|000| 000| 000| 000| 000| 

-----[3 Queues Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list]
Queue 0 :               |               |               | 
Queue 1 :               |               |               | 
Queue 2 :               |               |               | 

-----[Instructions runned] | [Final Report] | [Time]
N: 1                   | Context switches: 0 | Time spent: 5
D: 4                   | Processes added: 0
V: 0                   | Response time: inf
A: 0                   | Priority changes: 0
S: 0                   | Instructions runned: 5
B: 0                   | [Memory info]
T: 0                   | Memory allocation failed ratio: 0.0%
F: 0                   | Memory allocation sucess: 1
R: 0                   | Memory allocation failed: 0
                           | External memory fragments: 1

-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■

```



```

[Process Table]          [3-Core Processor]          [Infinite Memory]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | PID | TIM
000 | -001 | 006 | 000 | 005 | 000 | 000 | 006 | 000 | 006

[Physical Memory (size: 5) (Worst fit)]
|000| 000| 000| 000| 000| 

-----[3 Queues Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list]
Queue 0 :               |               |               | 
Queue 1 :               |               |               | 
Queue 2 :               |               |               | 

-----[Instructions runned] | [Final Report] | [Time]
N: 1                   | Context switches: 0 | Time spent: 6
D: 5                   | Processes added: 0
V: 0                   | Response time: inf
A: 0                   | Priority changes: 0
S: 0                   | Instructions runned: 6
B: 0                   | [Memory info]
T: 0                   | Memory allocation failed ratio: 0.0%
F: 0                   | Memory allocation sucess: 1
R: 0                   | Memory allocation failed: 0
                           | External memory fragments: 1

-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■

```

Usando a técnica de worst-fit para alocação da memória com tamanho 5, foi possível visualizar que houve uma média de 1 fragmento externo, um tempo médio de 6 e nenhuma requisição de alocação negada, com a lista de bloqueados vazia.

7.2.4. Next fit

```
[Process Table]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 000 | -01 | 000 | 000 | 000 | 000 | PID | TIM
000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000

[Physical Memory (size: 5) (Next fit)]
XXX XXX XXX XXX XXX | |

[3 Queues Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list]
Queue 0 :
Queue 1 :
Queue 2 :

[Instructions runned] | [Final Report] | [Time]
N: 0 Context switches: 0 Time spent: 0
D: 0 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 0
B: 0
T: 0 [Memory info] Memory allocation failed ratio: 100%
F: 0 Memory allocation sucess: 0
R: 0 Memory allocation failed: 0
External memory fragments: 0

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■
```

```
[Process Table]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | [3-Core Processor] | [Infinite Memory]
000 | -001 | 001 | 000 | 005 | 000 | 000 | 001 | PID | TIM
000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000

[Physical Memory (size: 5) (Next fit)]
000 000 000 000 000 | |

[3 Queues Scheduler] | [Blocked by IO] | [Blocked by Memory Heap] | [Done list]
Queue 0 :
Queue 1 :
Queue 2 :

[Instructions runned] | [Final Report] | [Time]
N: 1 Context switches: 0 Time spent: 1
D: 0 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 1
B: 0
T: 0 [Memory info] Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1

U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■
```

```

[Process Table]   [3-Core Processor]   [Infinite Memory]
PID | FPID | PC | VA0 | VAS | PRI | INT | CPT | PID | TIM
000 | -001 | 002 | 000 | 005 | 000 | 000 | 002 | 000 | 002
-----
[Physical Memory (size: 5) (Next fit)]
|000| 000 000 000 000 |

[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]
Queue 0 :
Queue 1 :
Queue 2 :

-----
[Instructions runned] [Final Report] [Time]
N: 1 Context switches: 0 Time spent: 2
D: 1 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 2
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1

-----
U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

```

```

[Process Table]   [3-Core Processor]   [Infinite Memory]
PID | FPID | PC | VA0 | VAS | PRI | INT | CPT | PID | TIM
000 | -001 | 003 | 000 | 005 | 000 | 000 | 003 | 000 | 003
-----
[Physical Memory (size: 5) (Next fit)]
|000| 000 000 000 000 |

[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]
Queue 0 :
Queue 1 :
Queue 2 :

-----
[Instructions runned] [Final Report] [Time]
N: 1 Context switches: 0 Time spent: 3
D: 2 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 3
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1

-----
U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: █

```

[Process Table]							[3-Core Processor]		[Infinite Memory]	
PID	FPID	PC	VA0	VAS	PRI	INT	CPT		PID	TIM
000	-001	004	000	005	000	000	004		000	004
[Physical Memory (size: 5) (Next fit)]										
000 000 000 000 000										
[3 Queues Scheduler]			[Blocked by IO]		[Blocked by Memory Heap]		[Done list]			
Queue 0 :										
Queue 1 :										
Queue 2 :										
[Instructions runned]			[Final Report]				[Time]			
N: 1	D: 3	V: 0	A: 0	S: 0	B: 0	T: 0	F: 0	R: 0	Context switches: 0 Processes added: 0 Response time: inf Priority changes: 0 Instructions runned: 4 [Memory info] Memory allocation failed ratio: 0.0% Memory allocation sucess: 1 Memory allocation failed: 0 External memory fragments: 1	Time spent: 4
U or Enter - Next time quantum; L - Unlock next process from blocked by IO list; M - Quit.										
Type an option: ■										

```
[Process Table] [3-Core Processor] [Infinite Memory]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | PID | TIM |
000 | -001 | 005 | 000 | 005 | 000 | 000 | 005 | 000 | 005 | 000 | 005 |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
[Physical Memory (size: 5) (Next fit)] | |
|000| |000| |000| |000| |000| | |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]
Queue 0 : | | | |
Queue 1 : | | | |
Queue 2 : | | | |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
[Instructions runned] [Final Report] [Time]
N: 1 Context switches: 0 Time spent: 5
D: 4 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 5
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■
```

```
[Process Table] [3-Core Processor] [Infinite Memory]
PID | F PID | PC | VA0 | VAS | PRI | INT | CPT | PID | TIM |
000 | -001 | 006 | 000 | 005 | 000 | 000 | 006 | 000 | 006 | 000 | 006 |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
[Physical Memory (size: 5) (Next fit)] | |
|000| |000| |000| |000| |000| | |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]
Queue 0 : | | | |
Queue 1 : | | | |
Queue 2 : | | | |
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
[Instructions runned] [Final Report] [Time]
N: 1 Context switches: 0 Time spent: 6
D: 5 Processes added: 0
V: 0 Response time: inf
A: 0 Priority changes: 0
S: 0 Instructions runned: 6
B: 0 [Memory info]
T: 0 Memory allocation failed ratio: 0.0%
F: 0 Memory allocation sucess: 1
R: 0 Memory allocation failed: 0
External memory fragments: 1
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: ■
```

Usando a técnica de next-fit para alocação da memória com tamanho 5, foi possível visualizar que houve uma média de 1 fragmento externo, um tempo médio de 6 e nenhuma requisição de alocação negada, com a lista de bloqueados vazios.

7.2.5 - First fit com memória de tamanho 50:

```
[Process Table]
PID | F PID | PC    | VA0 | VAS | PRI  | INT  | CPT
000 | -001 | 000 | -01 | 000 | 000 | 000 | 000 |
-----[Physical Memory (size: 50) (First fit)]
      XXX  XXX  XXX  XXX  XXX  XXX  XXX | XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX
      XXX  XXX  XXX  XXX  XXX  XXX  XXX | XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX
      XXX  XXX  XXX  XXX  XXX  XXX  XXX | XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX  XXX
-----[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]
Queue 0 : | | | | |
Queue 1 : | | | | |
Queue 2 : | | | | |

-----[Instructions runned] [Final Report] [Time]
N: 0   Context switches: 0           Time spent: 0
D: 0   Processes added: 0
V: 0   Response time: inf
A: 0   Priority changes: 0
S: 0   Instructions runned: 0
B: 0   [Memory info]
T: 0   Memory allocation failed ratio: 100%
F: 0   Memory allocation sucess: 0
R: 0   Memory allocation failed: 0
        External memory fragments: 0

-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: 1

-----[Physical Memory (size: 50) (First fit)]
      000 | -001 | 015 | 000 | 005 | 002 | 000 | 013 | | PID = 002 >>> | 052 | 007 |
      001 | 0000 | 010 | 005 | 004 | 009 | 007 | 011 | |
      002 | 0000 | 009 | 001 | 002 | 001 | 013 | 010 | |
-----[3 Queues Scheduler] [Blocked by IO] [Blocked by Memory Heap] [Done list]
Queue 0 : | 1 | | |
Queue 1 : | | 0 | |
Queue 2 : | | 2 | |

-----[Instructions runned] [Final Report] [Time]
N: 3   Context switches: 2           Time spent: 50
D: 11  Processes added: 4
V: 8   Response time: 29.0
A: 7   Priority changes: 0
S: 0   Instructions runned: 36
B: 1   [Memory info]
T: 2   Memory allocation failed ratio: 0.0%
F: 2   Memory allocation sucess: 3
R: 2   Memory allocation failed: 0
        External memory fragments: 2

-----U or Enter - Next time quantum;
L - Unlock next process from blocked by IO list;
M - Quit.

Type an option: 
```

Como uma memória de tamanho maior foi possível notar duas mudanças de contextos no processador, um processo bloqueado por entrada/saída, uma alocação contígua na memória e duas fragmentações externas.

9. Conclusão

Com este trabalho foi possível aplicar e visualizar em tempo real o gerenciamento de processos simulados em um sistema. Mesmo sendo processos simulados, foi possível ver que para o funcionamento adequado das requisições de processos da CPU são necessárias regras e padrões claros e bem definidos (até suas exceções) sobre quando escalar, quando manter processos bloqueados e qual fração de tempo cada processo tem direito no uso da CPU. Nosso trabalho foi implementado atendendo todas as exigências da especificação, acrescido de algumas funções extras, contribuindo para o aprendizado do grupo no tema de Processos e Threads que é de fundamental importância para compreensão de máquinas e sistemas computacionais.

Também foi possível perceber como a escolha dos algoritmos de gerenciamento de memória são de suma importância para que o sistema operacional funcione bem. Uma escolha ruim pode ser muito custosa para alocar memória, ou pode criar segmentos de memória com fatias tão pequenas de memória livre que nunca serão usados.

Foi muito feliz a escolha da linguagem de programação Python, pois resultou em um código enxuto e modular. Caso escolhêssemos uma linguagem de mais baixo nível, como C ou Rust (que eram nossas alternativas), o programa executaria mais rápido, entretanto o código ficaria maior e, possivelmente, com mais bugs.

Mesmo sendo um trabalho escolar, o código ficou grande perto do que esperávamos. Fomos muito felizes em já fazer o código esperando a implementação da memória física, e re-utilizamos a memória infinita do antigo trabalho como memória virtual. Isso fez com que não tivéssemos problema algum em adaptar o código do trabalho anterior nesse.