Sistemas Operacionais - PCS 3446 - 2019

Rodrigo Perrucci Macharelli - 9348877

Introdução

O projeto tem como objetivo implementar programaticamente um sistema de simulação orientado a eventos cuja intenção é implementar as funcionalidades básicas de um sistema operacional simples, capaz de receber Jobs criados a partir de uma linha de comando, organizar as suas ordens de execução, tratar eventos (aqui simulando o funcionamento de um sistema de interrupções) gerados pelos processos em execução, simular acesso a diferentes dispositivos de entrada e saída e implementa um sistema simples de segmentação de memória com partições fixas, possibilitando um sistema multiprogramado. A idéia é conseguir simular tal funcionamento de forma a obter estatísticas de utilização e dados de execução dos processos de forma indivídual e também uma visão geral do sistema.

Estrutura e Execução do projeto

O projeto foi estruturado de forma a conter uma interface de linha de comando **CLI**, pela qual é possível interagir com o sistema, adicionando Jobs e obtendo dados da simulação, uma **Máquina de Eventos** responsável pela simulação propriamente dita de um sistema operacional simples, um módulo de **Memória** segmentada e **Devices I/O** para os quais os processos em execução podem efetuar requerimentos.

Máquina de eventos

Baseado no artigo apresentado e nas aulas, foi implementado um motor de eventos que implementa uma máquina de estados com seis possíveis estados:

1. SUBMIT: Estado inicial atribuído aos jobs que são adicionados ao sistema
2. WAIT_RESOURCES: Estado de espera de alocação de recursos para o Job. Mais especificamente, é durante este estágio que o sistema de administração de memória verifica a possibilidade de alocação do Job, para que este possa ser executado.
3. READY: Após o Job ser alocado em memória passa para este estado, no qual aguarda ser escalonado para utilização da CPU, e consequentemente, sua execução.
4. RUNNING: Neste estado o Job passa a ser efetivamente um processo em execução no simulador, podendo gerar eventos de comunicação com os dispositivos de entrada e saída disponíveis.
5. WAIT_IO: Estado que representa os Jobs que estão aguardando a resposta de um pedido a um dispositivo de entrada e saída (descritos adiante), para que possa então seguir a sua execução, voltando ao estado READY e podendo ser escalonado novamente para a utilização da CPU.
6. DONE: Estado final da simulação, representando o fim da execução e finalização de todos os pedidos de comunicação com os dispositivos I/O.

Este motor de eventos é representado pela classe **OS**.

```
class OS:
    def __init__(self, num_threads=4):
        self.clock = 0.0
```

```
        self.memory = Memory()

        self.event_queue = Queue()
        self.jobs_queue = PriorityQueue()
        self.ready_jobs = []
        self.active_jobs = []
        self.completed_jobs = 0

        self.waiting_io_jobs = {
            "disco": [],
            "leitora1": [],
            "leitora2": [],
            "impressora1": [],
            "impressora2": []
        }

        self.running_jobs = 0
        self.jobs_list = []

        self.num_threads = num_threads
        self.current_cycle = 0

        self.schedulers = None
        self.processor = None
        self.started = False
```

Para sua implementação foram considerados alguns dos principais elementos presentes em um sistema operacional, implementados nesta classe e descritos a seguir.

Job Scheduler

Responsável por obter os jobs submetidos, alocados na fila de prioridade "jobs_queue", de forma que possam ser alocados em memória. As prioridades dos Jobs implementadas nesta simulação são as seguintes: LOW, NORMAL, HIGH, CRITICAL, sendo o último de maior prioridade. Neste elemento estão também incluídas funcionalidades do administrador de memória, que tem como finalidade verificar a disponibilidade de alocação de memória para o Job que, em caso positivo deve faze-lo nas partições disponíveis, sejam elas contíguas ou não. Em caso negativo o Job volta para a fila de jobs em estado SUBMIT para que possa aguardar até que um outro job seja concluído e parte da memória seja liberada. O módulo de memória implementado será tratado mais adiante.

```
def _job_scheduler(self):

        try:
            new_job = self.jobs_queue.get(False)
        except Empty:
            return

        allocated_segments = self.memory.allocate(new_job.id, new_job.size)
```

```
        if allocated_segments:
            print(f'[{self.current_cycle:05d}] Job Scheduler: Job {new_job.id}
está no estado READY depois de {new_job.current_cycle} ciclos.')
            print(self.memory)
            new_job.state = JobState.READY
            new_job.start_time = self.current_cycle
            self.ready_jobs.append(new_job)
            self._update_jobs_list(new_job)
        else:
            self.jobs_queue.put(new_job)
            self._update_jobs_list(new_job)
```

Process Scheduler

Responsável por transformar os Jobs prontos para execução (READY) em processos própriamente ditos, ou seja, os escalona para execução na CPU, associando-os ao estado RUNNING. É neste elemento em que tomam-se as medidas para possibilitar a multiprogramação do sistema, isto é, pode haver mais de um job em memória ao mesmo tempo, sendo executados de forma intercalada pelo sistema. Os processos em execução sao alocados em "active_jobs".

```
def _process_scheduler(self):
        for job in self.ready_jobs[:]:
            if self.running_jobs >= self.num_threads:
                return

            print(f'[{self.current_cycle:05d}] Process Scheduler: Iniciando job
{job.id} depois de {job.current_cycle} ciclos')
            self.ready_jobs.remove(job)
            job.state = JobState.RUNNING
            self.running_jobs += 1
            self.active_jobs.append(job)
            self._update_jobs_list(job)
```

Tratador de Eventos

Responsável por tratar os eventos gerados pelo sistema, isto é, gerados pelos dispositivos de I/O ao término de suas execuções ou por eventos externos como o evento de finalização forçada de um processo. Os Jobs atendidos por este elemento são os que estão no estado WAIT_IO, que passam ao estado READY ao final do tratamento de um evento de termino de I/O. Neste caso os Jobs são removidos da lista de "waiting_io_jobs" e voltam a lista de "ready_jobs".

```
    def _event_process(self):
        try:
            event = self.event_queue.get(False)
        except Empty:
            return
```

```
        event_name = type(event).__name__
        event_base_class = type(event).__bases__[0]

        if event_base_class == IOFinishedEvent:
            print(f'SO: Processando evento {event_name} para o Job
{event.job_id}.')
            event.process()

            waiting_io_jobs_cp = copy.deepcopy(self.waiting_io_jobs)

            dev = event.device_name

            for j in waiting_io_jobs_cp[dev]:
                if j.id == event.job_id:
                    self.waiting_io_jobs[dev].remove(j)
                    j.state = JobState.READY
                    j.current_io_req = None
                    j.waiting_current_io_cycles = 0
                    self.ready_jobs.append(j)
                    self._update_jobs_list(j)
                    return

        if type(event) == KillProcessEvent:
            print(f'SO: Processando evento {event_name} para o Job
{event.job_id}.')
            event.process()

            for dev in waiting_io_jobs_cp.keys():
                for j in waiting_io_jobs_cp[dev]:
                    if j.id == event.job_id:
                        self.waiting_io_jobs[dev].remove(j)
                        j.state = JobState.DONE
                        return

            for j in self.active_jobs[:]:
                if j.id == event.job_id:
                    self.running_jobs -= 1
                    self.active_jobs.remove(j)
                    j.state = JobState.DONE
                    return

            for j in self.ready_jobs[:]:
                if j.id == event.job_id:
                    self.ready_jobs.remove(j)
                    j.state = JobState.DONE
                    return

        print(f"SO: Evento desconhecido {event_name}")
```

Processador

O processador foi aqui representado pela função mostrada a seguir.

```
def _run(self):

        if len(self.jobs_list) == self.completed_jobs:
            print(f"[{self.current_cycle}] Todos os jobs foram completados!")
            self.started = False
            return

        for job in self.active_jobs[:]:
            job.cycle()
            self._update_jobs_list(job)

            for dev in job.io.keys():
                if job.io[dev] == None:
                    continue

                request_io = False

                for req in job.io[dev].io_requests:
                    if req[0] == job.cpu_cycles:
                        request_io = req
                        break

                if request_io:
                    print(f'[{self.current_cycle:05d}] SO: Job {job.id} pedindo
acesso ao dispositivo I/O {job.io[dev].name}.')
                    job.state = JobState.WAIT_IO
                    job.current_io_req = (dev, request_io)
                    self.running_jobs -= 1
                    self.active_jobs.remove(job)
                    self.waiting_io_jobs[dev].append(job)
                    self._update_jobs_list(job)



            if job.state == JobState.DONE:
                print(f'[{self.current_cycle:05d}] SO: Job {job.id} finalizado
depois de {self.current_cycle - job.start_time} ciclos.')

                self.memory.deallocate(job.id)
                print(f"[{self.current_cycle:05d}] SO: Estado atual da memõria:")
                print(self.memory)
                self.running_jobs -= 1
                self.active_jobs.remove(job)
                self.completed_jobs += 1
                self._update_jobs_list(job)


            self.current_cycle += 1
            time.sleep(self.clock)

        for dev in self.waiting_io_jobs:
            for job in self.waiting_io_jobs[dev]:
```

```
                job.cycle()
                self.current_cycle += 1
                if job.current_io_req[1][1] == job.waiting_current_io_cycles:
                    self.io_finish(job.id,
job.io[job.current_io_req[0]].finish_event)

        time.sleep(0.1*self.clock)
```

Este é responsável por simular a execução dos processos ativos no sistema, atualizando seus estados de ciclos e dando continuidade a simulação. Caso existam requerimentos de dispositivos de entrada e saída associados ao job sendo processado no tempo de simulação corrente, serão tratados e inseridos os respectivos eventos a lista de eventos a serem processados, alterando o estado do job para WAIT_IO. Caso o Job tenha sido finalizado, este é removido da memória, liberando espaço para outros jobs que estejam aguardando a disponibilidade de recursos. Caso o número de jobs completos seja igual ao número de Jobs submetidos ao sistema, a simulação é finalizada.

Execução da simulação

Sendo expostos os principais componentes da simulação, pode-se mostrar como foi feita a elaboração de sua execução, representada por um laço que executa os componentes citados acima de forma a representar o processo real de funcionamento de um sistema operacional.

```
def start(self):
    self.schedulers = self._schedulers
    self.processor = self._run
    self.started = True
    while self.started:
        self.schedulers()
        self.processor()
```

Sendo que "_schedulers" representa os três elementos de manipulação dos jobs:

```
def _schedulers(self):
    self._job_scheduler()
    self._process_scheduler()
    self._event_process()
```

Representação do Job

Os Jobs foram representados pela classe de mesmo nome descrita a seguir:

```
JobState = enum.Enum('JobState', 'SUBMIT WAIT_RESOURCES READY RUNNING WAIT_IO
DONE')
JobPriority = enum.IntEnum('JobPriority', 'LOW NORMAL HIGH CRITICAL')

class Job:
```

```
    def __init__(self, _id, execution_time,
            priority=JobPriority.NORMAL,
            io={ "disco": None,"leitora1": None,"leitora2": None,"impressora1":
None,"impressora2": None },
            size=10):

        self.id = _id
        self.total_cycles = execution_time
        self.priority = priority
        self.size = size

        self.arrive_time = 0
        self.start_time = 0
        self.current_cycle = 0

        self._state = JobState.SUBMIT

        self.waiting_current_io_cycles = 0
        self.current_io_req = None
        self.io = io

        self.cpu_cycles = 0
        self.io_cycles = 0
```

Nas primeiras duas linhas são definidos os estados e prioridades já discutidos anteriormente. A classe Job apresenta os seguintes atributos:

- **id**: Identificador único do Job.
- **total_cycles**: Quantidade de ciclos de execução de CPU, representa, a grosso modo, a quantidade de instruções a serem executadas na CPU (esta representação torna-se mais realista se considerarmos um processador com arquitetura de pipeline considerando-o cheio, de forma que apresentaria um throughput de 1 instrução por ciclo em funcionamento normal). É com base neste valor que sabemos o fim do ciclo de vida do Job.
- **priority**: Prioridade associada ao Job.
- **size**: Representa o tamanho médio que o programa ocupará na memória em tempo de execução, tanto com dados quanto com instruções.
- **arrive_time, start_time, current_cycle**: Marcadores dos ciclos de chegada do job ao sistema, ciclo de inicio de execução do processo e ciclo atual do Job.
- **_state**: Estado atual do Job (SUBMIT, WAIT_RESOURCES, READY, RUNNING, WAIT_IO, DONE).
- **waiting_current_io_cycles**: Contador de ciclos de espera de resposta de dispositivo, reinicializado para cada pedido distinto.
- **current_io_req**: Requerimento de dispositivo I/O. Apresenta o formato seguinte: *(device, (start_cycle, io_cycles))*. No qual *device* representa o dispositivo sendo requerido e os respectivos ciclos de início e duração da espera por resposta (para efeitos de simulação).
- **io**: Representação de todos os eventos de requerimento de dispositivos de I/O associados ao Job. É constituído por um dicionário contendo todos os 5 dispositívos simulados, cada um com um vetor de requerimentos de dispositivo.
- **cpu_cycles, io_cycles**: Representam a quantidade de ciclos em execução na CPU e em espera por resposta de dispositivo.

Ciclo de Job

Efetua a execução e atualização dos contadores associados ao Job.

```
def cycle(self):
    self.current_cycle += 1

    if self._state == JobState.RUNNING:
        self.cpu_cycles += 1

    if self._state == JobState.WAIT_IO:
        self.waiting_current_io_cycles += 1
        self.io_cycles += 1

    if self.cpu_cycles == self.total_cycles:
        self.state = JobState.DONE
```

Dispositivos

Foram simulados 5 dispositivos de Entrada/Saída neste projeto, cada um com suas respectivas características de tempos de acesso mínimo e máximo. Todos os valores de tempo neste projeto são representados em ciclos do motor de eventos e considera-se que uma instrução seria executada por ciclo em estado normal. Os ciclos de espera de I/O representam o tempo de espera para o processamento do pedido pelos dispositivos, que normalmente duram mais que apenas um ciclo, uma vez que este tipo de operação é muito mais custosa do que qualquer instrução executada diretamente pelo processador, sem necessidade de comunicação com os outros componentes do sistema. Os valores de ciclos de espera são aleatórios para cada requerimento de dispositivo, se baseando em um valor mímino e máximo, que por sua vez foram escolhidos de maneira a representar as relações de velocidade entre os dispositivos simulados.

```
io_config = {
        "disco": (10, 50),
        "leitora1": (30, 70),
        "leitora2": (50, 90),
        "impressora1": (70,100),
        "impressora2": (70, 120)
}

class Device:
    def __init__(self, name, io_requests, finish_event: Event):
        self.name = name
        self.io_requests = io_requests
        self.finish_event = finish_event
```

Em "io_config" pode-se observar os dispositivos simulados, assim como seus valores de máximo e mínimo de espera. Cada device apresenta também um evento de termino associado, que será emitido ao fim do tempo de espera, sendo tratado pelo tratador de eventos.

Eventos

São responsáveis por sinalizar a ocorrência de interrupções no sistema, tanto internas (vindas dos dispositivos) quanto externas (vindas de fora do sistema).

```
class Event:
    def process(self):
        print(f'Processamento do evento não implementado!')

class IOFinishedEvent(Event):
    def __init__(self, job_id):
        super().__init__()
        self.job_id = job_id

    def process(self):
        print(f'Job {self.job_id}: evento de dispositivo I/O: Finalizado.')

class DiskFinishedEvent(IOFinishedEvent):
    def __init__(self, job_id):
        super().__init__(job_id)
        self.device_name = "disco"

class LeitoraUmFinishedEvent(IOFinishedEvent):
    def __init__(self, job_id):
        super().__init__(job_id)
        self.device_name = "leitora1"

class LeitoraDoisFinishedEvent(IOFinishedEvent):
    def __init__(self, job_id):
        super().__init__(job_id)
        self.device_name = "leitora2"

class ImpressoraUmFinishedEvent(IOFinishedEvent):
    def __init__(self, job_id):
        super().__init__(job_id)
        self.device_name = "impressora1"

class ImpressoraDoisFinishedEvent(IOFinishedEvent):
    def __init__(self, job_id):
        super().__init__(job_id)
        self.device_name = "impressora2"

class KillProcessEvent(Event):
    def __init__(self, job_id):
        super(KillProcessEvent, self).__init__()
        self.job_id = job_id

    def process(self):
        print(f'Job {self.job_id} finalizado.')
```

Temos um tipo de evento para sinalizar o fim do tempo de espera para cada um dos dispositivos simulados, além do evento de finalização forçada do Job.

Memória

Foi implementado um esquema de memória segmentada com partições estáticas, de modo que os segmentos tem todos os mesmos tamanhos nos quais os Jobs são alocados conforme a disponibilidade atual, podendo ser feito de maneira contígua ou não. A classe que representa a memória é a seguinte:

```
class Memory:
    def __init__(self):
        self.size = 1000 # bytes
        self.segment_size = 10
        self.number_segments = self.size // self.segment_size
        self.segments = [None for _ in range(self.number_segments)]
        self.free_segments = self.number_segments
```

A memória é iniciada com um valor de 1000 bytes (valor escolhido para facilidade de representação gráfica) e é segmentada em partições de 10 bytes. Os segmentos são representados por um array, que na realidade corresponde ao segment map, uma vez que os dados não são utilizados de fato, mas sim apenas representados.

É possível efetuar duas operações em memória: alocação e desalocação:

```
def allocate(self, job_id, job_size):
        requested_number_segments = job_size//self.segment_size # partições cheias
        requested_number_segments += 1 if job_size % self.segment_size else 0 #
resto

        # não há espaço disponível
        if requested_number_segments > self.free_segments:
            return False

        allocated_segments = []
        for i in range(len(self.segments)):
            if self.segments[i] == None:
                self.segments[i] = job_id
                allocated_segments.append(i)
                self.free_segments -= 1

                if len(allocated_segments) == requested_number_segments:
                    break

        return allocated_segments

def deallocate(self, job_id):
        has_job = False
        for i, seg in enumerate(self.segments[:]):
            if seg == job_id:
```

```
                if not has_job:
                    has_job == True
                self.segments[i] = None
                self.free_segments += 1
        return has_job
```

A primeira verifica quantos segmentos serão necessários para alocar o Job. Caso tenha espaço livre suficiente, aloca o Job e retorna os segmentos em que foi alocado. Caso não tenha espaço disponível a função retorna False. A segunda efetua a desalocação dos segmentos de memória associados ao Job, liberando espaço na memória para outros processos.

Interface de Linha de Comando

Promove a interação com o sistema de simulação, apresenta os seguintes comandos:

- **add**: Adiciona um ou mais jobs ao sistema. add <cpu_cycles> <io_wait_cycles>
- **start**: Inicia a simulação do sistema operacional.
- **ls**: Lista os comandos disponíveis.
- **jobs-list**: Lista os Jobs presentes no sistema, assim como estatisticas de execução.
- **file**: Redireciona as saídas do programa para o arquivo out.txt.
- **exit**: Termina a execução do simulador.

Vale a pena ressaltar que é este o módulo responsável pela geração aleatória dos Jobs, sorteando quantos e quais serão as operações de I/O que serão requisitadas por cada Job individualmente.

Testes e Demonstrações do Projeto

A utilização do simulador é bastante simples e pode ser obtido com poucos comandos.

Funcionamento

Inserção de Jobs

Ao utilizar o comando

```
add 100 5
```

São criados 5 Jobs com pedidos aleatórios de operaçoes de entrada e saída

```
SO: Recebeu Job (id 0) com prioridade HIGH e acessos I/O: disco 2 | leitora1 1.
Adicionando a lista.
SO: Recebeu Job (id 1) com prioridade NORMAL e acessos I/O: leitora1 1.
Adicionando a lista.
SO: Recebeu Job (id 2) com prioridade NORMAL e acessos I/O: disco 1. Adicionando a
lista.
SO: Recebeu Job (id 3) com prioridade HIGH e acessos I/O: disco 2. Adicionando a
lista.
SO: Recebeu Job (id 4) com prioridade NORMAL e acessos I/O: leitora1 1.
Adicionando a lista.
```

Note que há Jobs com diferentes prioridades, com ou sem requerimentos de entrada e saída para diferentes dispositivos.

Detalhamento dos Jobs

Ao utilizar o comando

> jobs-list

É possível ter mais dados dos Jobs inseridos no sistema:

```
JOB ID: 0 | STATE: JobState.WAIT_RESOURCES | SIZE: 69
        IO:
         disco: 10[25] | 26[38]
         leitora1: 36[32]

        Total Cycles: 1
        CPU Cycles: 0 (0.00%)
        IO Cycles: 0 (0.00%)

===============
Ciclos totais de simulação: 1
Ciclos de utilização de CPU: 0 (0.00%)
Ciclos de espera de I/O: 0 (0.00%)
JOB ID: 1 | STATE: JobState.WAIT_RESOURCES | SIZE: 50
        IO:
         leitora1: 18[32]

        Total Cycles: 1
        CPU Cycles: 0 (0.00%)
        IO Cycles: 0 (0.00%)

===============
Ciclos totais de simulação: 1
Ciclos de utilização de CPU: 0 (0.00%)
Ciclos de espera de I/O: 0 (0.00%)
JOB ID: 2 | STATE: JobState.WAIT_RESOURCES | SIZE: 66
        IO:
         disco: 4[27]

        Total Cycles: 1
        CPU Cycles: 0 (0.00%)
        IO Cycles: 0 (0.00%)

===============
Ciclos totais de simulação: 1
Ciclos de utilização de CPU: 0 (0.00%)
Ciclos de espera de I/O: 0 (0.00%)
JOB ID: 3 | STATE: JobState.WAIT_RESOURCES | SIZE: 22
        IO:
         disco: 8[24] | 12[14]
```

```
        Total Cycles: 1
        CPU Cycles: 0 (0.00%)
        IO Cycles: 0 (0.00%)

===============
Ciclos totais de simulação: 1
Ciclos de utilização de CPU: 0 (0.00%)
Ciclos de espera de I/O: 0 (0.00%)
JOB ID: 4 | STATE: JobState.WAIT_RESOURCES | SIZE: 65
        IO:
         leitora1: 2[46]

        Total Cycles: 1
        CPU Cycles: 0 (0.00%)
        IO Cycles: 0 (0.00%)

===============
Ciclos totais de simulação: 1
Ciclos de utilização de CPU: 0 (0.00%)
Ciclos de espera de I/O: 0 (0.00%)
```

Pode-se ver o estado de cada Job, seus tamanhos e seus acessos a dispositivos, representados por *start_cycle[io_wait_cycles]*. Além disso pode-se ver o total de ciclos de utilização de cpu e de espera de I/O de cada job individualmente, assim como os dados para a simulação como um todo.

Iniciando a Simulação

Após adicionar os Jobs no sistema pode-se iniciar a simulação dos eventos utilizando o comando

> start

Este comando inicia o processo de tratamento dos jobs e eventos, sendo terminado quando todos os jobs submetidos estiverem finalizados.

Resultado de Testes

Simulação de 5 Jobs com 100 ciclos de CPU

```
> SO: Recebeu Job (id 0) com prioridade LOW e acessos I/O: disco 2. Adicionando a
lista.
SO: Recebeu Job (id 1) com prioridade LOW e acessos I/O: leitora1 1. Adicionando a
lista.
SO: Recebeu Job (id 2) com prioridade CRITICAL sem acessos I/O. Adicionando a
lista.
SO: Recebeu Job (id 3) com prioridade CRITICAL e acessos I/O: disco 1. Adicionando
a lista.
SO: Recebeu Job (id 4) com prioridade HIGH e acessos I/O: disco 2. Adicionando a
lista.
> [00000] Job Scheduler: Job 2 está no estado READY depois de 0 ciclos.
2 2 2 2 2 2 x x x x
x x x x x x x x x x
```

```
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00000] Process Scheduler: Iniciando job 2 depois de 0 ciclos
[00001] Job Scheduler: Job 3 está no estado READY depois de 0 ciclos.
2 2 2 2 2 2 3 3 3 3
3 x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00001] Process Scheduler: Iniciando job 3 depois de 0 ciclos
[00003] Job Scheduler: Job 4 está no estado READY depois de 0 ciclos.
2 2 2 2 2 2 3 3 3 3
3 4 4 x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00003] Process Scheduler: Iniciando job 4 depois de 0 ciclos
[00006] Job Scheduler: Job 1 está no estado READY depois de 0 ciclos.
2 2 2 2 2 2 3 3 3 3
3 4 4 1 1 1 x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00006] Process Scheduler: Iniciando job 1 depois de 0 ciclos
[00010] Job Scheduler: Job 0 está no estado READY depois de 0 ciclos.
2 2 2 2 2 2 3 3 3 3
3 4 4 1 1 1 0 0 0 0
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

```
x x x x x x x x x x
x x x x x x x x x x
[00017] SO: Job 1 pedindo acesso ao dispositivo I/O leitora1.
[00019] Process Scheduler: Iniciando job 0 depois de 0 ciclos
[00021] SO: Job 4 pedindo acesso ao dispositivo I/O disco.
[00046] SO: Job 3 pedindo acesso ao dispositivo I/O disco.
[00062] SO: Job 0 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento DiskFinishedEvent para o Job 4.
Job 4: evento de dispositivo I/O: Finalizado.
[00091] Process Scheduler: Iniciando job 4 depois de 18 ciclos
SO: Processando evento DiskFinishedEvent para o Job 3.
Job 3: evento de dispositivo I/O: Finalizado.
[00125] Process Scheduler: Iniciando job 3 depois de 26 ciclos
SO: Processando evento DiskFinishedEvent para o Job 0.
Job 0: evento de dispositivo I/O: Finalizado.
[00169] Process Scheduler: Iniciando job 0 depois de 30 ciclos
[00180] SO: Job 4 pedindo acesso ao dispositivo I/O disco.
[00257] SO: Job 0 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento LeitoraUmFinishedEvent para o Job 1.
Job 1: evento de dispositivo I/O: Finalizado.
[00325] Process Scheduler: Iniciando job 1 depois de 64 ciclos
SO: Processando evento DiskFinishedEvent para o Job 4.
Job 4: evento de dispositivo I/O: Finalizado.
[00404] Process Scheduler: Iniciando job 4 depois de 81 ciclos
SO: Processando evento DiskFinishedEvent para o Job 0.
Job 0: evento de dispositivo I/O: Finalizado.
[00474] SO: Job 2 finalizado depois de 474 ciclos.
[00474] SO: Estado atual da memória:
x x x x x 3 3 3 3
3 4 4 1 1 1 0 0 0 0
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00478] Process Scheduler: Iniciando job 0 depois de 84 ciclos
[00538] SO: Job 3 finalizado depois de 537 ciclos.
[00538] SO: Estado atual da memória:
x x x x x x x x x x
x 4 4 1 1 1 0 0 0 0
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00669] SO: Job 4 finalizado depois de 666 ciclos.
[00669] SO: Estado atual da memória:
x x x x x x x x x x
```

```
x x x 1 1 1 0 0 0 0
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00679] SO: Job 1 finalizado depois de 673 ciclos.
[00679] SO: Estado atual da memória:
x x x x x x x x x x
x x x x x 0 0 0 0 0
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[00689] SO: Job 0 finalizado depois de 679 ciclos.
[00689] SO: Estado atual da memória:
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
[690] Todos os jobs foram completados!
> JOB ID: 0 | STATE: JobState.DONE | SIZE: 33
        IO:
         disco: 9[21] | 27[36]

        Total Cycles: 157
        CPU Cycles: 100 (63.69%)
        IO Cycles: 57 (36.31%)

===============
Ciclos totais de simulação: 690
Ciclos de utilização de CPU: 100 (14.49%)
Ciclos de espera de I/O: 57 (8.26%)
JOB ID: 1 | STATE: JobState.DONE | SIZE: 27
        IO:
         leitora1: 3[61]

        Total Cycles: 161
        CPU Cycles: 100 (62.11%)
        IO Cycles: 61 (37.89%)
```

```
===============
Ciclos totais de simulação: 690
Ciclos de utilização de CPU: 200 (28.99%)
Ciclos de espera de I/O: 118 (17.10%)
JOB ID: 2 | STATE: JobState.DONE | SIZE: 51
        Total Cycles: 100
        CPU Cycles: 100 (100.00%)
        IO Cycles: 0 (0.00%)

===============
Ciclos totais de simulação: 690
Ciclos de utilização de CPU: 300 (43.48%)
Ciclos de espera de I/O: 118 (17.10%)
JOB ID: 3 | STATE: JobState.DONE | SIZE: 42
        IO:
         disco: 11[15]

        Total Cycles: 115
        CPU Cycles: 100 (86.96%)
        IO Cycles: 15 (13.04%)

===============
Ciclos totais de simulação: 690
Ciclos de utilização de CPU: 400 (57.97%)
Ciclos de espera de I/O: 133 (19.28%)
JOB ID: 4 | STATE: JobState.DONE | SIZE: 20
        IO:
         disco: 5[13] | 24[44]

        Total Cycles: 157
        CPU Cycles: 100 (63.69%)
        IO Cycles: 57 (36.31%)

===============
Ciclos totais de simulação: 690
Ciclos de utilização de CPU: 500 (72.46%)
Ciclos de espera de I/O: 190 (27.54%)
```

Este exemplo mostra o funcionamento completo do sistema de simulação. Na primeira parte mostra-se os Jobs criados e adicionados ao sistema. Em seguida o JobScheduler e o ProcessScheduler tratam os Jobs recebidos os deixando prontos para execução em cpu e alocados na memória.

Note que a representação da memória é mostrada sempre que há uma alocação ou desalocação de um job. Observando no exemplo vemos que o JobScheduler consegue alocar todos os Jobs em memória, pois a memória é maior que a somatória de seus tamanhos. A alocação final fica:

```
2 2 2 2 2 2 3 3 3 3
3 4 4 1 1 1 0 0 0 0
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

```
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
```

A ordem de alocação seguiu a fila de prioridades utilizada pelo JobScheduler, de modo que os Jobs CRITICAL foram alocados primeiro (2 e 3), HIGH (4) e por fim LOW (1 e 0).

Em seguida pode-se observar como ocorrem os pedidos de operações de entrada e saída, além do tratamento dos eventos de termino.

Os dados obtidos com o comando "jobs-list" formam a última parte da saída do programa, com estatísticas de utilização de CPU e espera de dispositivos.

Simulação com Memória Reduzida

Para testar a administração da memória segmentada diminuiremos o tamanho da memória em 10 vezes para forçar que Jobs tenham que aguardar liberação de recursos.

```
> SO: Recebeu Job (id 0) com prioridade NORMAL e acessos I/O: disco 2. Adicionando
a lista.
SO: Recebeu Job (id 1) com prioridade NORMAL e acessos I/O: disco 3. Adicionando a
lista.
SO: Recebeu Job (id 2) com prioridade CRITICAL e acessos I/O: disco 3. Adicionando
a lista.
SO: Recebeu Job (id 3) com prioridade CRITICAL e acessos I/O: disco 1. Adicionando
a lista.
SO: Recebeu Job (id 4) com prioridade CRITICAL e acessos I/O: impressora1 1.
Adicionando a lista.
> [00000] Job Scheduler: Job 2 está no estado READY depois de 0 ciclos.
2 2 x x x x x x x x
[00000] Process Scheduler: Iniciando job 2 depois de 0 ciclos
[00001] Job Scheduler: Job 3 está no estado READY depois de 0 ciclos.
2 2 3 3 x x x x x x
[00001] Process Scheduler: Iniciando job 3 depois de 0 ciclos
[00001] SO: Job 2 pedindo acesso ao dispositivo I/O disco.
[00004] Job Scheduler: Job 4 está no estado READY depois de 0 ciclos.
2 2 3 3 4 4 4 4 4 x
[00004] Process Scheduler: Iniciando job 4 depois de 0 ciclos
[00008] SO: Job 4 pedindo acesso ao dispositivo I/O impressora1.
[00026] SO: Job 3 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento DiskFinishedEvent para o Job 2.
Job 2: evento de dispositivo I/O: Finalizado.
[00119] Process Scheduler: Iniciando job 2 depois de 40 ciclos
SO: Processando evento DiskFinishedEvent para o Job 3.
Job 3: evento de dispositivo I/O: Finalizado.
[00142] Process Scheduler: Iniciando job 3 depois de 47 ciclos
[00163] SO: Job 2 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento ImpressoraUmFinishedEvent para o Job 4.
Job 4: evento de dispositivo I/O: Finalizado.
```

```
[00244] Process Scheduler: Iniciando job 4 depois de 80 ciclos
SO: Processando evento DiskFinishedEvent para o Job 2.
Job 2: evento de dispositivo I/O: Finalizado.
[00252] Process Scheduler: Iniciando job 2 depois de 85 ciclos
[00326] SO: Job 2 pedindo acesso ao dispositivo I/O disco.
[00412] SO: Job 3 finalizado depois de 411 ciclos.
[00412] SO: Estado atual da memória:
2 2 x x 4 4 4 4 4 x
SO: Processando evento DiskFinishedEvent para o Job 2.
Job 2: evento de dispositivo I/O: Finalizado.
[00426] Process Scheduler: Iniciando job 2 depois de 145 ciclos
[00494] SO: Job 4 finalizado depois de 490 ciclos.
[00494] SO: Estado atual da memória:
2 2 x x x x x x x
[00496] Job Scheduler: Job 0 está no estado READY depois de 0 ciclos.
2 2 0 0 0 0 0 x x x
[00496] Process Scheduler: Iniciando job 0 depois de 0 ciclos
[00513] SO: Job 0 pedindo acesso ao dispositivo I/O disco.
[00539] SO: Job 2 finalizado depois de 539 ciclos.
[00539] SO: Estado atual da memória:
x x 0 0 0 0 0 x x x
[00541] Job Scheduler: Job 1 está no estado READY depois de 0 ciclos.
1 1 0 0 0 0 0 1 1 x
[00541] Process Scheduler: Iniciando job 1 depois de 0 ciclos
SO: Processando evento DiskFinishedEvent para o Job 0.
Job 0: evento de dispositivo I/O: Finalizado.
[00550] Process Scheduler: Iniciando job 0 depois de 27 ciclos
[00550] SO: Job 1 pedindo acesso ao dispositivo I/O disco.
[00571] SO: Job 0 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento DiskFinishedEvent para o Job 1.
Job 1: evento de dispositivo I/O: Finalizado.
[00579] Process Scheduler: Iniciando job 1 depois de 19 ciclos
SO: Processando evento DiskFinishedEvent para o Job 0.
Job 0: evento de dispositivo I/O: Finalizado.
[00592] Process Scheduler: Iniciando job 0 depois de 48 ciclos
[00602] SO: Job 1 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento DiskFinishedEvent para o Job 1.
Job 1: evento de dispositivo I/O: Finalizado.
[00634] Process Scheduler: Iniciando job 1 depois de 47 ciclos
[00723] SO: Job 1 pedindo acesso ao dispositivo I/O disco.
SO: Processando evento DiskFinishedEvent para o Job 1.
Job 1: evento de dispositivo I/O: Finalizado.
[00750] Process Scheduler: Iniciando job 1 depois de 105 ciclos
[00750] SO: Job 0 finalizado depois de 254 ciclos.
[00750] SO: Estado atual da memória:
1 1 x x x x x 1 1 x
[00786] SO: Job 1 finalizado depois de 245 ciclos.
[00786] SO: Estado atual da memória:
x x x x x x x x x x
[787] Todos os jobs foram completados!
> JOB ID: 0 | STATE: JobState.DONE | SIZE: 41
        IO:
         disco: 9[18] | 20[10]
```

```
        Total Cycles: 128
        CPU Cycles: 100 (78.12%)
        IO Cycles: 28 (21.88%)

===============
Ciclos totais de simulação: 787
Ciclos de utilização de CPU: 100 (12.71%)
Ciclos de espera de I/O: 28 (3.56%)
JOB ID: 1 | STATE: JobState.DONE | SIZE: 39
        IO:
         disco: 6[13] | 19[15] | 64[13]

        Total Cycles: 141
        CPU Cycles: 100 (70.92%)
        IO Cycles: 41 (29.08%)

===============
Ciclos totais de simulação: 787
Ciclos de utilização de CPU: 200 (25.41%)
Ciclos de espera de I/O: 69 (8.77%)
JOB ID: 2 | STATE: JobState.DONE | SIZE: 16
        IO:
         disco: 2[38] | 18[29] | 43[35]

        Total Cycles: 202
        CPU Cycles: 100 (49.50%)
        IO Cycles: 102 (50.50%)

===============
Ciclos totais de simulação: 787
Ciclos de utilização de CPU: 300 (38.12%)
Ciclos de espera de I/O: 171 (21.73%)
JOB ID: 3 | STATE: JobState.DONE | SIZE: 19
        IO:
         disco: 9[38]

        Total Cycles: 138
        CPU Cycles: 100 (72.46%)
        IO Cycles: 38 (27.54%)

===============
Ciclos totais de simulação: 787
Ciclos de utilização de CPU: 400 (50.83%)
Ciclos de espera de I/O: 209 (26.56%)
JOB ID: 4 | STATE: JobState.DONE | SIZE: 46
        IO:
         impressora1: 2[78]

        Total Cycles: 178
        CPU Cycles: 100 (56.18%)
        IO Cycles: 78 (43.82%)

===============
Ciclos totais de simulação: 787
```

```
Ciclos de utilização de CPU: 500 (63.53%)
Ciclos de espera de I/O: 287 (36.47%)
```

Repare como a dinamica do sistema muda com uma memória reduzida, de modo que inicialmente apenas três Jobs conseguem ser alocados em memória em um primeiro momento, sendo que os Jobs 0 e 1 têm de esperar o fim da execução dos outros Jobs para que possam ser executados. Os dados da simulação nos mostram que, para o mesmo número de Jobs com o mesmo tempo máximo de uso de CPU, a parcela de utilização efetiva do processador diminui com a memória reduzida (de 72.46% para 63.53%), justamente pelo fato de que os Jobs têm de esperar a liberação de recursos do sistema.

Conclusão e Considerações Finais

Trata-se de um trabalho bastante ilustrativo das funcionalidades de um sistema operacional e como são manejados os processos e Jobs dentro deste contexto, possibilitando um melhor entendimento do sistema como um todo. Não foi aqui implementado um sistema de Arquivos e referencias pelos Jobs, mas de maneira geral, a sua implementação se assemelha àquela utilizada na memória segmentada com as representações de segmentos referenciados, porém no contexto do disco.