UNIVERSIDADE DO VALE DO ITAJAÍ
CIÊNCIA DA COMPUTAÇÃO

GABRIEL BELTRÃO LAUS
RAFAEL DA CUNHA

TRABALHO DE SISTEMAS OPERACIONAIS

Simulador de escalonamento Round Robin

ITAJAÍ
(2025)

RESUMO

Esse trabalho tem como principal intuito o aprofundamento e consolidação com relação aos estudos de escalonamento e simulação de sistemas operacionais. Para isso, será mostrado o desenvolvimento de um simulador completo de um processo de escalonamento Round Robin feito em Python. Nele, haverá suporte a múltiplos núcleos de CPU, operações de E/S e inserções dinâmicas de processos.

Palavras-chaves: escalonamento, round-robin, processos, tempo de espera, turnaround, ociosidade.

SUMÁRIO

1. Introdução.

2. Desenvolvimento.

    2.1 - O que é Round-Robin?

    2.2 - A criação do código.

    2.3 - Testes.

    2.4 - Análise de resultados.

3. Conclusão.

4. Referências bibliográficas

1. Introdução.

O gerenciamento de processos está entre as funções fundamentais de um sistema operacional, é nele que são executadas diferentes tarefas de maneira mais rápida, confiável e eficiente. Dentro desse contexto, os escalonadores de processos estão presentes para otimizar diferentes aspectos de desempenho interno, fazendo com que a escolha do escalonamento dependa da necessidade específica do ambiente e dos objetivos de desempenho do sistema.

Dentre os principais tipos de escalonamento de sistemas se encontra o Round-Robin, em que por mais que seja um dos mais antigos, é um dos escalonamentos mais bem quistos entre usuários e em outras aplicações. Conhecido por sua simplicidade e fácil uso, esse método é amplamente utilizado em sistemas operacionais tradicionais com múltiplos usuários e aplicações interativas, sendo por diversas vezes uma das principais fontes de estudos quando o assunto é gerenciamento de processos.

Para poder compreender melhor o seu uso, será visto o resultado da implementação do algoritmo de escalonamento Round-Robin com quantum fixo e dinâmico. Sendo desenvolvido totalmente na linguagem de programação Python, esse algoritmo terá execuções em múltiplos núcleos com uma entrada sendo feita através de um arquivo txt, mostrando em sua saída uma tabela com as informações de sua execução e uma linha do tempo textual no estilo Gantt, marcando os núcleos e execuções. Após a simulação, serão analisados os resultados da mesma, visando compreender o comportamento e desempenho do algoritmo frente a diferentes cenários de carga e concorrência.

Com a implementação desse algoritmo, espera-se entender como funciona o uso desse tipo de escalonamento e obter os seus resultados referentes a tipos diferentes de implementação, para que dessa forma possa-se determinar se o escalonamento Round-Robin é realmente um bom método de gerenciamento de processos.

2. Desenvolvimento.

2.1 - O que é Round-Robin?

Se trata de um tipo de escalonamento preemptivo, que consiste no repartimento uniforme do tempo da CPU entre todos os processos prontos para serem executados, sendo projetado principalmente para sistemas operacionais de tempo compartilhado.

Os processos são então organizados em fila, alocando a cada um uma quantidade de tempo fixo chamado de quantum. Caso o processo não seja terminado no tempo de quantum, ele é colocado no fim da fila e então, outro processo é colocado no início da fila e é executado durante o tempo de quantum determinado.

2.2 - A criação do código.

Para poder executar o algoritmo de escalonamento por Round-Robin, foi necessário a criação de um código capaz de realizar todas as funções e especificações necessárias para seu funcionamento. O código foi implementado através da linguagem de programação Python, e nele há certos segmentos que realizam funções específicas para o seu funcionamento.

Sendo tudo implementado em um único arquivo “.py”, a primeira parte que foi feita foi com relação às importações, colocadas justamente para permitir utilizar várias threads, o uso de um queue para poder organizar os processos em execução e para então controlar o tempo de simulação do código.

```
import threading
import time
from queue import Queue
```

A próxima parte criada foi a classe do processo, se tratando de cada processo da simulação. Em que, o construtor recebe os parâmetros para que a simulação ocorra da melhor maneira, com o controle de cada estado de cada fase do processo, os tempos de execução e outras informações.

```
# ====================== CLASSE DO PROCESSO ======================

class Processo:
    def __init__(self, id, chegada, exec1, bloqueio, espera, exec2):
        self.id = id
        self.chegada = chegada
        self.exec1 = exec1
        self.exec2 = exec2
        self.bloqueio = bloqueio
        self.espera = espera

        self.estado = 'novo'
        self.tempo_restante = exec1
        self.em_execucao2 = False
        self.tempo_bloqueado = 0
        self.tempo_espera = 0
        self.tempo_inicio = None
        self.tempo_fim = None
        self.quantum_restante = 0
        self.turnaround = 0
        self.contextos = 0

    def __repr__(self):
        return f"Processo({self.id})"
```

A terceira parte criada foi a função para carregar os processos, para que seja possível ler dados que são inseridos através de um outro arquivo de entrada “txt”, tendo dois tipos de processos definidos (iniciais e dinâmicos). E ao finalizar, a função irá retornar duas listas para que o escalonador possa iniciar os processos em seu devido tempo.

```
# ====================== FUNÇÃO PARA CARREGAR PROCESSOS ======================

def carregar_processos(caminho):
    processos_iniciais = []
    processos_dinamicos = []
    with open(caminho, 'r') as f:
        for linha in f:
            if linha.strip() == "":
                continue
            partes = linha.strip().split('|')
            id = partes[0].strip()
            chegada = int(partes[1].strip())
            exec1 = int(partes[2].strip())
            bloqueio = partes[3].strip().lower() == 's'
            espera = int(partes[4].strip())
            exec2 = int(partes[5].strip())

            proc = Processo(id, chegada, exec1, bloqueio, espera, exec2)
            if chegada == 0:
                processos_iniciais.append(proc)
            else:
                processos_dinamicos.append(proc)

    return processos_iniciais, processos_dinamicos
```

A quarta parte trata-se da criação do escalonador em Round-Robin, que irá coordenar a criação dos processos em múltiplos núcleos, desde a definição de seus parâmetros até a iniciação e verificações durante a simulação.

Durante todo o processo, o Round-Robin também gerencia os processos bloqueados e os novos que chegam. E ao finalizar todo o processo, o método “gerar_relatorio” irá criar um novo arquivo com o nome de ‘relatorio.txt’, em que nele terá as informações de todos os dados e trocas de processos, além de criar a linha do tempo Gantt, mostrando cada etapa do escalonador durante toda a simulação.

```python
# ====================== ESCALONADOR ROUND ROBIN ======================

class Escalonador:
    def __init__(self, num_nucleos, quantum, processos_iniciais, processos_dinamicos):
        self.num_nucleos = num_nucleos
        self.quantum = quantum
        self.prontos = Queue()
        self.bloqueados = []
        self.execucao = [None] * num_nucleos
        self.nucleos = []
        self.lock = threading.Lock()
        self.tempo = 0
        self.dinamicos = processos_dinamicos
        self.finalizados = []
        self.timeline = [[] for _ in range(num_nucleos)]

        for p in processos_iniciais:
            self.prontos.put(p)

    def iniciar(self):
        for i in range(self.num_nucleos):
            t = threading.Thread(target=self.executar_nucleo, args=(i,))
            t.start()
            self.nucleos.append(t)

        threading.Thread(target=self.monitorar_dinamicos).start()
        threading.Thread(target=self.verificar_bloqueios).start()

    def atualizar_tempo(self, tempo):
        self.tempo = tempo
```

```
    def executar_nucleo(self, idx):
        while True:
            with self.lock:
                if self.prontos.empty() and all(p is None for p in self.execucao) and not
self.bloqueados and not self.dinamicos:
                    break
                if not self.prontos.empty():
                    processo = self.prontos.get()
                    processo.contextos += 1
                    if processo.tempo_inicio is None:
                        processo.tempo_inicio = self.tempo
                    processo.estado = 'executando'
                    processo.quantum_restante = self.quantum
                    self.execucao[idx] = processo
                else:
                    self.timeline[idx].append(" ocioso ")
                    time.sleep(1)
                    continue

            while processo.quantum_restante > 0 and processo.tempo_restante > 0:
                time.sleep(1)
                processo.tempo_restante -= 1
                processo.quantum_restante -= 1
                self.timeline[idx].append(processo.id)

                if processo.tempo_restante == 0:
                    if processo.bloqueio and not processo.em_execucao2:
                        processo.estado = 'bloqueado'
                        processo.tempo_bloqueado = processo.espera
                        processo.tempo_restante = processo.exec2
                        processo.em_execucao2 = True
                        with self.lock:
                            self.bloqueados.append(processo)
                        break
                    else:
                        processo.estado = 'finalizado'
                        processo.tempo_fim = self.tempo + 1
                        processo.turnaround = processo.tempo_fim - processo.chegada
                        with self.lock:
                            self.finalizados.append(processo)
                        break
```

```
            if processo.estado == 'executando':  # Quantum acabou antes de terminar
                processo.estado = 'pronto'
                with self.lock:
                    self.prontos.put(processo)

            with self.lock:
                self.execucao[idx] = None

    def verificar_bloqueios(self):
        while True:
            time.sleep(1)
            with self.lock:
                for processo in self.bloqueados[:]:
                    processo.tempo_bloqueado -= 1
                    if processo.tempo_bloqueado <= 0:
                        processo.estado = 'pronto'
                        self.bloqueados.remove(processo)
                        self.prontos.put(processo)
            if not self.bloqueados and self.prontos.empty() and all(p is None for p in
self.execucao) and not self.dinamicos:
                break

    def monitorar_dinamicos(self):
        while self.dinamicos:
            with self.lock:
                for p in self.dinamicos[:]:
                    if p.chegada <= self.tempo:
                        self.prontos.put(p)
                        self.dinamicos.remove(p)
            time.sleep(1)

    def aguardar_fim(self):
        for t in self.nucleos:
            t.join()

    def finalizado(self):
        return not self.dinamicos and self.prontos.empty() and all(p is None for p in
self.execucao) and not self.bloqueados
```

```
    def gerar_relatorio(self):
        with open('relatorio.txt', 'w') as f:
            f.write("ID | Espera | Turnaround | Trocas de contexto\n")
            for p in sorted(self.finalizados, key=lambda x: x.id):
                espera = p.turnaround - (p.exec1 + (p.exec2 if p.bloqueio else 0) + (p.espera
if p.bloqueio else 0))
                f.write(f"{p.id} | {espera} | {p.turnaround} | {p.contextos - 1}\n")

            f.write("\nLinha do tempo (Gantt):\n")
            for idx, linha in enumerate(self.timeline):
                f.write(f"Núcleo {idx}: {' | '.join(linha)}\n")
```

A quinta e última parte criada para o código se trata do arquivo principal, em que basicamente se trata da inicialização de toda a simulação. É nele que é definido o quantum, o número de núcleos e qual será o arquivo de entrada (nesse caso em específico sendo o ‘entrega.txt’).

O escalonamento é iniciado após ser carregado todos os processos, em que então é esperado até o fim de todas as threads. Após todas elas serem finalizadas, o método “gerar_relatorio” é chamado e é criado um novo arquivo com todos os resultados da simulação.

```
# ====================== MAIN.PY ======================

QUANTUM = 3
NUM_NUCLEOS = 2
ARQUIVO_ENTRADA = 'entrada.txt'
tempo_global = 0
tempo_lock = threading.Lock()

def atualizar_tempo():
    global tempo_global
    while not escalonador.finalizado():
        time.sleep(1)
        with tempo_lock:
            tempo_global += 1
        escalonador.atualizar_tempo(tempo_global)

if __name__ == "__main__":
    print("Iniciando simulação Round Robin com múltiplos núcleos...\n")

    processos_iniciais, processos_dinamicos = carregar_processos(ARQUIVO_ENTRADA)
    escalonador = Escalonador(NUM_NUCLEOS, QUANTUM, processos_iniciais, processos_dinamicos)

    tempo_thread = threading.Thread(target=atualizar_tempo)
    tempo_thread.start()

    escalonador.iniciar()
    escalonador.aguardar_fim()
    tempo_thread.join()

    escalonador.gerar_relatorio()
    print("\nSimulação concluída. Relatório gerado em 'relatorio.txt'")
```

2.3 - Testes.

Para que o algoritmo do escalonador seja observado de uma maneira mais eficiente, será realizada uma sessão com 3 testes. Para que dessa forma possa ser observado o desempenho de cada quando posto para trabalhar com quantums, núcleos e outras informações diferentes.

O primeiro teste foi realizado com o seguinte arquivo de entrada e com as seguintes informações:

```
P1 | 0 | 4 | S | 3 | 2
P2 | 0 | 5 | N | 0 | 0
P3 | 2 | 3 | S | 2 | 1
P4 | 4 | 2 | N | 0 | 0
P5 | 6 | 6 | S | 2 | 3
```

```
QUANTUM = 3
NUM_NUCLEOS = 2
```

O segundo teste foi realizado com o seguinte arquivo de entrada e com as seguintes informações:

```
P1 | 12 | 5 | S | 3 | 1
P2 | 7 | 5 | N | 0 | 9
P3 | 8 | 6 | S | 8 | 5
```

```
QUANTUM = 4
NUM_NUCLEOS = 3
```

O terceiro e último teste foi realizado com o seguinte arquivo de entrada e com as seguintes informações:

```
P1 | 5 | 2 | S | 3 | 4
P2 | 7 | 5 | N | 5 | 9
P3 | 9 | 2 | S | 8 | 4
P4 | 8 | 3 | S | 6 | 1
```

```
QUANTUM = 3
NUM_NUCLEOS = 2
```

2.4 - Análise de resultados.

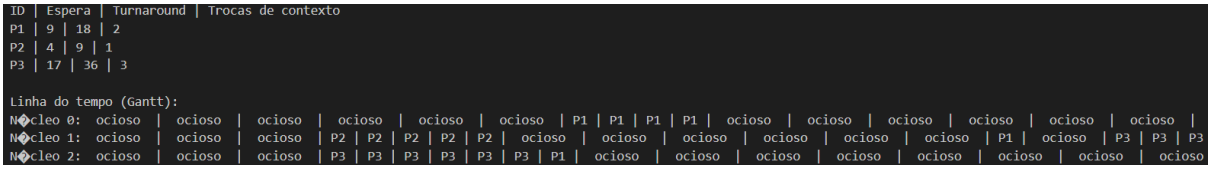
Os resultados obtidos a partir do primeiro teste:

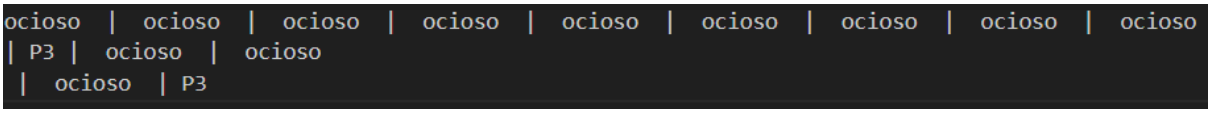
```
ID | Espera | Turnaround | Trocas de contexto
P1 | 4 | 13 | 1
P2 | 1 | 6 | 1
P3 | 5 | 11 | 1
P4 | 1 | 3 | 0
P5 | 7 | 18 | 2

Linha do tempo (Gantt):
N◆cleo 0: P1 | P1 | P1 | P1 |  ocioso  |  ocioso  |  ocioso  |  ocioso  |  ocioso  | P3 |  ocioso  |  ocioso  |  ocioso  |  ocioso  |  ocioso  |  ocioso  |  ocioso
N◆cleo 1: P2 | P2 | P2 | P2 | P2 |  ocioso  | P5 | P5 | P5 | P5 | P5 | P5 |  ocioso  |  ocioso  | P5 | P5 | P5
N◆cleo 2:  ocioso  |  ocioso  | P3 | P3 | P3 | P4 | P4 |  ocioso  | P1 | P1 |  ocioso  |  ocioso  |  ocioso  |  ocioso  |  ocioso
```

É possível ver através da imagem que o P5 foi o processo que mais teve tempo de espera, turnaround e trocas de contexto. Enquanto P1 e P3 obtiveram resultados quase semelhantes, e P2 e P4 tiveram o melhor resultado dentre os processos.

A linha do tempo mostrou que, por mais que no início houvesse um bom uso de cada núcleo, logo depois mostrou um longo período de ociosidade por causa dos tempos de espera.

Os resultados obtidos a partir do segundo teste:

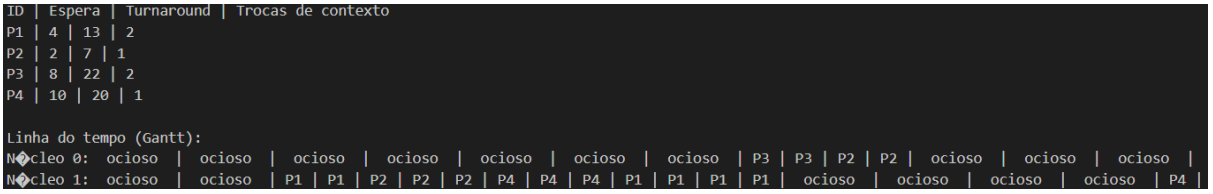


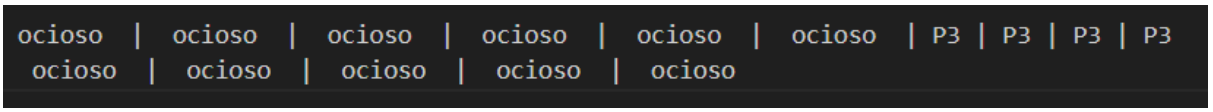


Nesse novo teste, quem obteve o maior tempo de espera, turnaround e trocas de contexto foi o P3. Isso pode acontecer devido ao fato de que ele retornou para a fila após muito tempo, acumulando mais tempo do que deveria. Em comparação, o P2 teve o melhor desempenho e o P1 teve um desempenho mediano.

A linha do tempo desse teste conseguiu mostrar como que tiveram diversos momentos de ociosidade em cada um dos núcleos. Isso indica que cada uma dessas ociosidades causaram um grande comprometimento com relação ao uso da CPU.

Os resultados obtidos a partir do terceiro e último teste:





No último teste o maior tempo de espera foi P1, quem obteve o maior turnaround foi o P3 e os mesmos empatam em número de trocas de contexto. Igual ao teste anterior, o P2, por falta de um bloqueio, foi o que teve os menores tempos de espera, turnaround e troca de contexto.

Com relação a linha do tempo, foi possível perceber que teve muitos períodos de ociosidade nos núcleos, podendo resultar em um menor aproveitamento da CPU.

3. Conclusão.

Foi possível observar, através desse trabalho, o funcionamento do escalonador de Round-Robin através da criação e simulação do mesmo através de 3 testes criados com quantums, núcleos e outras informações diferentes. Com isso gerando resultados que nos permite compreender o funcionamento do escalonador de Round-Robin.

Com os resultados obtidos, o Round-Robin se mostrou ser capaz em ambientes que demandam uma maior quantidade de alternância de processos. No entanto, há limitações que precisam ser observadas, principalmente com relação ao desempenho, em que dependendo das características do processo pode gerar um maior tempo de espera ou um maior número de trocas de contexto, podendo causar mais ociosidade e com isso, comprometer ainda mais como a CPU trabalha na organização desses processos.

Esse método é apenas um em que, igual a diversos outros tipos de escalonamento, vai obter resultados e que irão afetar de diferentes maneiras como os processos são escalonados. Porém, através de testes, como os que foram vistos, é possível estudar e determinar quando que um escalonamento de processo será melhor para um melhor desempenho de um sistema operacional, sendo o Round-Robin apenas o início de um longo caminho de testes para serem feitos e ser aprofundado ainda mais o conhecimento nesse aspecto tão importante para o funcionamento das máquinas virtuais.

4. Referências Bibliográficas.

ALUNOSO. Simulador de escalonamento Round Robin. 2016. Disponível em: https://deinfo.uepg.br/~alunoso/2016/ROUNDROBIN/. Acesso em: 1 jun. 2025.

ALVES, Jósis. Sistemas Operacionais – Gerenciamento de processos. Blog Gran Cursos Online, 21 fev. 2024. Disponível em: https://blog.grancursosonline.com.br/sistemas-operacionais-gerenciamento-de-processos/. Acesso em: 1 jun. 2025.

PANDEY, Nikita. Round Robin Scheduling Algorithm. Studytonight. Disponível em: https://www.studytonight.com/operating-system/round-robin-scheduling. Acesso em: 1 jun. 2025.