Universidade da Beira Interior

Departamento de Informática

Relatório ProbSched: Um Simulador para Algoritmos de Escalonamento Probabilístico para Sistemas Operativos

Elaborado por:

Pedro Gouveia - 50125
Miguel Pires - 49505
Dinis Ramos - 49471
Rafael Nabais - 48736

Orientador:

Doutor Professor Paul Andrew Crocker

27 de abril de 2025

1 Modelos Probabilísticos Utilizados para Geração de Processos

O simulador utiliza diferentes distribuições probabilísticas para criar os atributos dos processos, como tempos de chegada, tempos de burst, prioridades e períodos. Abaixo estão as fórmulas matemáticas utilizadas para cada distribuição:

1.1 Distribuição Uniforme

A distribuição uniforme gera valores aleatórios dentro de um intervalo definido $[a, b]$. A fórmula utilizada é:

$$x \sim \text{Uniforme}(a, b)$$

onde x é o valor criado aleatoriamente, com probabilidade uniforme em $[a, b]$.

1.2 Distribuição Exponencial

A distribuição exponencial é utilizada para modelar o tempo entre eventos, como tempos de chegada de processos. A fórmula utilizada é:

$$t_{\text{chegada}}(n) = t_{\text{chegada}}(n-1) + \text{Exponencial}(\lambda)$$

onde:

- $t_{\text{chegada}}(n)$ é o tempo de chegada do processo n.
- λ é a taxa de eventos por unidade de tempo.
- Exponencial(λ) cria um valor aleatório seguindo a distribuição exponencial com parâmetro λ.

1.3 Distribuição Normal (Gaussiana)

A distribuição normal é utilizada para criar valores com maior concentração em torno de uma média μ, com desvio padrão σ. A fórmula utilizada é:

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

onde:

- μ é a média dos valores criados.
- σ^2 é a variância (o quadrado do desvio padrão).

1.4 Distribuição Poisson

A distribuição de Poisson é utilizada para modelar a frequência de eventos em um intervalo de tempo fixo. A fórmula utilizada para criar o número de eventos k em um intervalo é:

$$P(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

onde:

- k é o número de eventos.
- λ é a taxa média de eventos por unidade de tempo.

1.5 Aplicação das Distribuições

As distribuições são aplicadas para criar os seguintes atributos dos processos:

- **Tempo de chegada**: Utiliza a distribuição poisson para simular a chegada de processos em tempos aleatórios.
- **Tempo de burst**: Utiliza a distribuição normal para criar tempos de execução com maior concentração em torno de uma média.
- **Prioridade**: Utiliza a distribuição uniforme para criar prioridades aleatórias dentro de um intervalo definido.
- **Período**: Utiliza a distribuição normal para criar tempos de execução com maior concentração em torno de uma média.
- **Deadline**: Utiliza a distribuição uniforme para criar prioridades aleatórias dentro de um intervalo definido.

2 Algoritmos de Escalonamento Implementados

O simulador suporta os seguintes algoritmos de escalonamento, cada um com características específicas:

2.1 First-Come, First-Served (FCFS)

O algoritmo FCFS executa os processos na ordem em que chegam ao sistema. É simples de implementar, mas pode levar ao problema de *convoy effect*, onde processos longos atrasam a execução de processos curtos. Este algoritmo não é preemptivo.

2.2 Shortest Job First (SJF)

O algoritmo SJF seleciona o processo com o menor tempo de burst para execução.

2.3 Priority Scheduling

O algoritmo de escalonamento por prioridade seleciona processos com base em sua prioridade. Existem duas variações:

- **Não Preemptivo**: O processo com maior prioridade é executado até a conclusão.
- **Preemptivo**: Um processo em execução pode ser interrompido por outro com maior prioridade.

2.4 Round Robin (RR)

O algoritmo Round Robin utiliza um quantum fixo para alternar entre os processos. Cada processo é executado por uma fatia de tempo (*quantum*) antes de ser colocado de volta na fila, garantindo justiça entre os processos. Este algoritmo é preemptivo e ideal para sistemas interativos.

2.5 Rate Monotonic (RM)

O algoritmo RM é utilizado em sistemas de tempo real e atribui prioridades estáticas aos processos com base em seus períodos. Processos com menor período têm maior prioridade. Este algoritmo assume que todos os processos são periódicos e independentes.

2.6 Earliest Deadline First (EDF)

O algoritmo EDF é utilizado em sistemas de tempo real e atribui prioridades dinâmicas aos processos com base em seus deadlines. Processos com deadlines mais próximos têm maior prioridade. Este algoritmo é mais eficiente que o RM em cenários dinâmicos, mas requer maior complexidade de implementação.

2.7 Resumo dos Algoritmos

A tabela abaixo resume as principais características dos algoritmos implementados:

Algoritmo	**Preemptivo**	**Aplicação**
FCFS	Não	Sistemas simples
SJF	Não	Sistemas otimizados para tempo de burst
Priority	Sim/Não	Sistemas baseados em prioridades
Round Robin	Sim	Sistemas interativos
Rate Monotonic	Não	Sistemas de tempo real periódicos
Earliest Deadline First	Sim	Sistemas de tempo real dinâmicos

Tabela 1: Resumo dos Algoritmos de Escalonamento

3 Como Executar o Programa

O programa pode ser executado com os seguintes passos:

3.1 Compilação

Certifique-se de que o programa foi compilado corretamente. Use o comando:

```
make
```

Se a compilação for bem-sucedida, um executável chamado `probsched` será criado no diretório do projeto.

3.2 Execução

O programa é executado com os seguintes argumentos:

```
./probsched <algoritmo> <num_processos> [quantum]
```

- `<algoritmo>`: Nome do algoritmo de escalonamento a ser utilizado. Os algoritmos disponíveis são:
  - `FCFS` - First-Come, First-Served

  - `SJF` - Shortest Job First
  - `PRIORITY_NP` - Priority Scheduling (não preemptivo)
  - `PRIORITY_P` - Priority Scheduling (preemptivo)
  - `RR` - Round Robin (requer quantum)
  - `RM` - Rate Monotonic (para processos periódicos)
  - `EDF` - Earliest Deadline First

- `<num_processos>`: Número de processos a serem criados. Deve ser um número inteiro positivo.
- `[quantum]`: (Opcional) Quantum de tempo para o algoritmo Round Robin. Deve ser um número inteiro positivo.

3.3 Exemplos de Execução

Abaixo estão alguns exemplos de como executar o programa com diferentes algoritmos:

- **FCFS com 5 processos**:

```
./probsched FCFS 5
```

- **Round Robin com 5 processos e quantum 3**:

```
./probsched RR 5 3
```

- **Rate Monotonic com 5 processos**:

```
./probsched RM 5
```

4 Exemplos de Entrada e Saída

Abaixo estão exemplos de execução do simulador utilizando diferentes algoritmos de escalonamento. Cada exemplo inclui o comando utilizado, os processos criados, a linha do tempo de execução e as estatísticas finais.

4.1 Exemplo 1: FCFS (First-Come, First-Served)

Entrada

O comando utilizado foi:

```
./probsched FCFS 5
```

Saída

```
=== Simulador de Escalonamento de Processos ===

PID   Chegada  Burst  Prioridade Período Deadline
-----------------------------------------------
1     4        1      4          0       0
2     6        7      5          0       0
3     3        1      10         0       0
4     8        3      4          0       0
5     2        1      4          0       0

=== Executando FCFS (First-Come, First-Served) ===

EXECUÇÃO: ##P5[]P3[]P1[]#P2[][][][][][][]P4[][][]

=== Resultados Finais ===

PID   Chegada  Burst  Prioridade Deadline Período  Finalizado Espera
-----------------------------------------------------------------
5     2        1      4          0        0        3          0
3     3        1      10         0        0        4          0
1     4        1      4          0        0        5          0
2     6        7      5          0        0        13         0
4     8        3      4          0        0        16         5

=== Estatísticas da Simulação ===

- Tempo médio de espera: 1.00
- Tempo médio de turnaround: 3.60
- Utilização da CPU: 81.25%
- Throughput: 0.31 processos/unidade de tempo
- Deadlines perdidos: 0
```

4.2 Exemplo 2: RR (Round Robin)

Entrada

O comando utilizado foi:

```
./probsched RR 5 3
```

Saída

```
=== Simulador de Escalonamento de Processos ===

PID   Chegada  Burst  Prioridade Período Deadline
-----------------------------------------------
1     4        5      4          0       0
2     10       4      8          0       0
```

```
3     4        4      9          0         0
4     12       6      4          0         0
5     3        2      9          0         0

=== Executando Round Robin (Quantum=3) ===

EXECUÇÃO: ###|P5[]|P1[][]|P3[][][]|P1[][]|P2[][][]|P3[]|P4[][][]|P2[]|P4[][][]

=== Resultados Finais ===

PID   Chegada  Burst  Prioridade Deadline Período  Finalizado Espera
-----------------------------------------------------------------
1     4        5      4          0        0         10         1
2     10       4      8          0        0         14         0
3     4        4      9          0        0         18         10
4     12       6      4          0        0         24         6
5     3        2      9          0        0         5          0

=== Estatísticas da Simulação ===

- Tempo médio de espera: 3.40
- Tempo médio de turnaround: 7.60
- Utilização da CPU: 87.50%
- Throughput: 0.21 processos/unidade de tempo
- Deadlines perdidos: 0
```

4.3 Exemplo 3: RM (Rate Monotonic)

Entrada

O comando utilizado foi:

```
./probsched RM 5
```

Saída

```
=== Simulador de Escalonamento de Processos ===

PID   Chegada  Burst  Prioridade Período Deadline
-----------------------------------------------
1     6        1      5          7         1
2     3        6      7          4         1
3     4        4      6          9         5
4     7        4      9          15        8
5     7        4      7          13        6

=== Executando Rate Monotonic Scheduling ===
Utilização: 8.97, Limite: 0.74

EXECUÇÃO: ###P2[][][]|P1[]|P2[][][]|P3[][][][]|P5[][][][]|P4[][][][]
```

```
=== Resultados Finais ===

PID   Chegada  Burst  Prioridade Deadline Período  Finalizado Espera
-----------------------------------------------------------------
1     6        1      5          7        1        7          0
2     3        6      7          4        1        10         1
3     4        4      6          9        5        14         6
5     7        4      7          13       6        18         7
4     7        4      9          15       8        22         11

=== Estatísticas da Simulação ===

- Tempo médio de espera: 5.00
- Tempo médio de turnaround: 8.80
- Utilização da CPU: 86.36%
- Throughput: 0.23 processos/unidade de tempo
- Deadlines perdidos: 2
```

5 Análise dos Resultados

Nesta seção, analisamos os resultados obtidos para diferentes algoritmos de escalonamento com base nos exemplos fornecidos. A análise considera métricas como tempo médio de espera, tempo médio de turnaround, utilização da CPU, throughput e deadlines perdidos (quando aplicável).

5.1 FCFS (First-Come, First-Served)

O algoritmo FCFS executa os processos na ordem de chegada. A análise dos resultados mostra:

- **Tempo médio de espera**: 3.40 unidades de tempo, indicando que os processos esperaram pouco antes de serem executados.
- **Tempo médio de turnaround**: 7.60 unidades de tempo, o que é eficiente para um algoritmo simples.
- **Utilização da CPU**: 87.50%, mostrando que a CPU esteve ocupada durante a maior parte do tempo.
- **Throughput**: 0.21 processos/unidade de tempo, o que reflete uma boa taxa de conclusão.
- **Deadlines perdidos**: 0, pois o algoritmo não considera deadlines.

O FCFS é simples e eficiente em cenários com baixa variabilidade, mas pode levar ao problema de *convoy effect* em cenários com processos longos.

5.2 RR (Round Robin)

O algoritmo Round Robin seleciona processos em ordem circular, dando a cada um um quantum de tempo. A análise dos resultados mostra:

- **Tempo médio de espera**: 3.40 unidades de tempo, maior que o FCFS devido à priorização de processos curtos.
- **Tempo médio de turnaround**: 7.60 unidades de tempo, indicando um bom desempenho geral.
- **Utilização da CPU**: 87.50%, mostrando uma alta eficiência no uso da CPU.
- **Throughput**: 0.21 processos/unidade de tempo, ligeiramente inferior ao FCFS.
- **Deadlines perdidos**: 0 pois o algoritmo não considera deadlines.

O RR é eficiente para balancear o tempo de resposta entre processos, mas pode resultar em tempos de turnaround maiores para processos longos devido à alternância constante.

5.3 RM (Rate Monotonic)

O algoritmo RM é utilizado para sistemas de tempo real com tarefas periódicas. A análise dos resultados mostra:

- **Tempo médio de espera**: 5.00 unidades de tempo, significativamente maior devido à priorização de processos com menor período.
- **Tempo médio de turnaround**: 8.80 unidades de tempo, refletindo o impacto de deadlines perdidos.
- **Utilização da CPU**: 86.36%, indicando que a CPU não foi totalmente utilizada devido a processos que perderam deadlines.
- **Throughput**: 0.23 processos/unidade de tempo, muito baixo devido aos deadlines perdidos.
- **Deadlines perdidos**: 2, mostrando que o sistema não conseguiu atender todos os requisitos de tempo real.

O RM é eficiente para sistemas com alta previsibilidade, mas pode falhar em cenários com alta carga ou processos com períodos conflitantes.

5.4 Comparação Geral

A tabela abaixo resume os resultados obtidos para os três algoritmos analisados:

Algoritmo	Espera Média	Turnaround Médio	Utilização CPU	Throughput	Deadlin
FCFS	0.80	2.40	72.73%	0.45	
RR	3.40	7.60	87.50%	0.21	
RM	20.00	25.00	58.06%	0.03	

Tabela 2: Comparação de Desempenho dos Algoritmos

5.5 Conclusão

Os resultados obtidos mostram que cada algoritmo de escalonamento possui características específicas que os tornam mais adequados para diferentes cenários. Abaixo está uma análise geral de todos os algoritmos implementados:

- **FCFS (First-Come, First-Served)**:
  - Simples de implementar e eficiente em cenários com baixa variabilidade.
  - Pode sofrer com o problema de *convoy effect*, onde processos longos atrasam a execução de processos curtos.
- **SJF (Shortest Job First)**:
  - Reduz o tempo médio de turnaround, priorizando processos com menor tempo de burst.
  - A versão preemptiva é mais eficiente, mas pode causar starvation para processos longos.
- **Priority Scheduling**:
  - Adequado para sistemas onde a prioridade dos processos é crítica.
  - A versão preemptiva permite maior flexibilidade, mas pode causar overhead devido à troca de contexto.
  - A versão não preemptiva é mais simples, mas pode levar à ineficiência em cenários dinâmicos.
- **Round Robin (RR)**:
  - Justo para sistemas interativos, garantindo que todos os processos recebam tempo de CPU.
  - O desempenho depende do valor do quantum: valores muito pequenos aumentam o overhead, enquanto valores muito grandes se aproximam do comportamento do FCFS.
- **Rate Monotonic (RM)**:
  - Algoritmo de tempo real baseado em prioridades estáticas, eficiente para sistemas com tarefas periódicas.
  - Pode falhar em cenários com alta carga ou quando os períodos dos processos não são harmônicos.
- **Earliest Deadline First (EDF)**:

  - Algoritmo de tempo real baseado em prioridades dinâmicas, eficiente para sistemas com deadlines variáveis.
  - Apresenta melhor desempenho que o RM em cenários dinâmicos, mas requer maior complexidade de implementação.

Recomendações

A escolha do algoritmo de escalonamento depende dos requisitos do sistema:

- Para sistemas simples e previsíveis, o **FCFS** ou **SJF** são boas escolhas.
- Para sistemas interativos, o **Round Robin** é mais adequado.
- Para sistemas onde a prioridade é importante, o **Priority Scheduling** é recomendado.
- Para sistemas de tempo real, o **Rate Monotonic** é eficiente para tarefas periódicas, enquanto o **Earliest Deadline First** é mais flexível para cenários dinâmicos.

Essa análise destaca a importância de selecionar o algoritmo correto com base nas características e requisitos do sistema, garantindo eficiência e previsibilidade no escalonamento dos processos.

Projeto disponível em: `https://github.com/FoodEat12104/ProbSched`