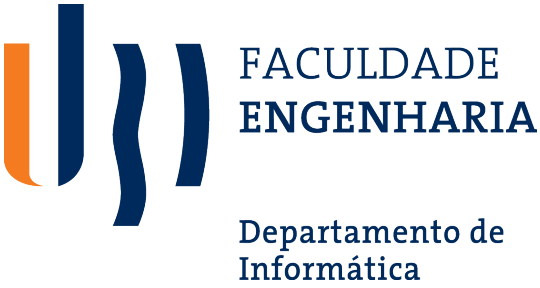
Universidade da Beira Interior

Departamento de Informática



Relatório Tecnico

Elaborado por: Juliana Ferreira (51886), Sara Bastos (49499)

Orientador: Professor Doutor Paul Crocker

25 de abril de 2025

Introdução

Este relatório descreve o desenvolvimento do simulador modular de algoritmos de escalonamento para processos em sistemas operativos, intitulado **ProbSched**. O objetivo principal foi criar uma ferramenta capaz de gerar processos de forma probabilística e simular diferentes algoritmos de escalonamento de CPU, avaliando seu desempenho através de métricas como tempo médio de espera, tempo de resposta, utilização da CPU e throughput.

Funcionamento do Programa

O simulador foi desenvolvido em C, com uma estrutura modular e compilado através de um `Makefile`. O programa permite a execução via linha de comando, onde o usuário pode especificar o algoritmo de escalonamento a ser utilizado, o quantum para o algoritmo `Round Robin` e uma seed para a geração pseudoaleatória de processos.

Atributos dos Processos Gerados:

- Tempo de chegada
- Tempo de execução (*burst*)
- Prioridade
- Tempo restante de execução

Componentes Principais:

- **Gerador de Processos**: Geração estática ou aleatória com distribuições uniformes e exponenciais.
- **Escalonadores**: Implementações de *FCFS*, Prioridade e *Round Robin*.
- **Módulo de Métricas**: Cálculo do tempo médio de espera, tempo de resposta, utilização da CPU e throughput.

Algoritmos Implementados

- ***FCFS* (*First Come First Served*)**: Executa os processos pela ordem de chegada.
- **Prioridade**: Escolhe o processo com menor valor de prioridade.

- **PrioridadeP**: Semelhante ao Prioridade, mas com preempção. O processo com a menor prioridade pode ser interrompido por outro com maior prioridade.
- ***Round Robin* (RR)**: Usa *quantum* fixo para execução circular.
- **SJF (*Shortest Job First*)**: Executa os processos com o menor tempo de execução (burst) primeiro.
- **EDF (*Earliest Deadline First*)**: Executa os processos com o prazo mais próximo.
- **RM (*Rate Monotonic*)**: Prioriza processos com menor intervalo de tempo entre suas execuções (mais curto prazo de repetição).

Exemplos de Execução

```
$ ./simulator FCFS 0 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 391.06
Tempo médio de resposta: 391.06
Utilização da CPU: 100.00%
Throughput: 83 processos

$ ./simulator PRIORITY 0 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 374.39
Tempo médio de resposta: 374.39
Utilização da CPU: 100.00%
Throughput: 83 processos

$ ./simulator PRIORITYP 0 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 377.98
Tempo médio de resposta: 377.51
Utilização da CPU: 100.00%
Throughput: 83 processos
```

```
$ ./simulator RR 4 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 462.75
Tempo médio de resposta: 166.66
Utilização da CPU: 100.00%
Throughput: 83 processos

$ ./simulator SJF 0 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 312.35
Tempo médio de resposta: 312.35
Utilização da CPU: 100.00%
Throughput: 83 processos

$ ./simulator EDF 0 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 361.89
Tempo médio de resposta: 361.89
Utilização da CPU: 100.00%
Throughput: 83 processos

$ ./simulator RM 0 123
--- Estatísticas ---
Processos simulados: 83
Tempo médio de espera: 374.96
Tempo médio de resposta: 374.96
Utilização da CPU: 100.00%
Throughput: 83 processos
```

Modelos Probabilísticos

O modelo probabilístico foi utilizado para gerar os dados dos processos de forma a refletir cenários realistas de carga do sistema. As distribuições probabilísticas aplicadas foram:

- **Distribuição Uniforme**: Utilizada para gerar o tempo de chegada e o burst de CPU, de forma que todos os valores dentro de um intervalo definido tenham a mesma probabilidade de ocorrer.

- **Distribuição Exponencial**: Inicialmente planejada para modelar o tempo de chegada dos processos, representando cenários onde os processos tendem a chegar de forma mais concentrada em determinados intervalos de tempo. No entanto, para simplificação, o tempo de chegada é gerado de forma uniforme no código.

- **Distribuição Normal**: Inicialmente planejada para modelar o burst de CPU, representando uma distribuição mais concentrada em torno de uma média. No código, o burst de CPU é gerado de maneira uniforme.

Execução do Programa

O programa foi executado em ambiente Linux, utilizando terminal e compilação via `Makefile`. Os comandos usados para testar o simulador foram os seguintes:

1. **Compilação do Projeto**:

   ```
   make clean && make
   ```

2. **Execução com Geração Aleatória de Processos**:

   ```
   ./simulator FCFS 0 123
   ./simulator PRIORITY 0 123
   ./simulator PRIORITYP 0 123
   ./simulator RR 4 123
   ./simulator SJF 0 123
   ./simulator EDF 0 123
   ./simulator RM 0 123
   ```

3. **Execução com Ficheiro de Processos**:

   ```
   ./simulator FICHEIRO processos.txt FCFS
   ./simulator FICHEIRO processos.txt RR 3
   ./simulator FICHEIRO processos.txt PRIORITYP
   ./simulator FICHEIRO processos.txt EDF
   ```

4. **Testes Automáticos**:

   Foi criado um script chamado `testes.sh` com o seguinte conteúdo:

   ```
   #!/bin/bash
   rm -f resultados.txt
   for algo in FCFS SJF PRIORITY PRIORITYP RR EDF RM; do
     echo "=== $algo ===" >> resultados.txt
     ./simulator $algo 0 123 >> resultados.txt
     echo "" >> resultados.txt
   done
   ```

   Depois, o script foi tornado executável e executado com:

   ```
   chmod +x testes.sh
   ./testes.sh
   cat resultados.txt
   ```

   Cada execução do programa forneceu métricas como:

   - Número de processos simulados
   - Tempo médio de espera
   - Tempo médio de resposta
   - Utilização da CPU
   - *Throughput* (processos completados)

Essas métricas foram fundamentais para a análise do desempenho dos algoritmos de escalonamento.

Comparação de Desempenho com Diferentes Valores de Quantum para o Algoritmo *Round Robin* (RR)

A comparação de desempenho foi realizada ao variar o valor do quantum no algoritmo *Round Robin* (RR). A tabela abaixo resume os resultados obtidos para os diferentes valores de *quantum*, mostrando como eles impactam nas métricas de desempenho.

Quantum	Algoritmo	Tempode Espera	Tempo de Resposta	Utilização da CPU	Throughput
4	RR	462.75	166.66	100.00%	83 processos
5	RR	479.34	204.53	100.00%	83 processos
6	RR	508.04	241.02	100.00%	83 processos
7	RR	508.43	274.19	100.00%	83 processos
8	RR	504.04	303.51	100.00%	83 processos

Tabela 1: Resultados de Algoritmo *Round Robin* com diferentes valores de *Quantum*

Análise dos Resultados

A análise dos resultados revelou os seguintes insights:

1. **FCFS (*First-Come, First-Served*)**:

    - Este algoritmo apresentou tempos de espera mais elevados, especialmente quando a carga do sistema era mais variada.
    - A sua natureza sequencial, onde os processos são executados até o fim, pode levar a um desempenho insatisfatório quando há processos com tempos de execução longos.
    - A utilização da CPU foi de 100% apenas com a distribuição uniforme.

2. ***Round Robin* (RR)**:

    - A comparação de desempenho ao variar o valor do quantum no algoritmo Round Robin mostrou que o aumento do quantum resultou em tempos de espera e tempos de resposta mais elevados. Isso é particularmente visível à medida que o quantum aumenta de 4 para 8.
    - O aumento do quantum no algoritmo Round Robin resultou em tempos de espera e tempos de resposta mais elevados, sugerindo que valores de quantum maiores podem causar um aumento no tempo total de execução dos processos. No entanto, a utilização da CPU manteve-se em 100% independentemente do valor do quantum.

3. **Prioridade**:

    - Mostrou resultados equilibrados, com bons tempos de espera e resposta, e uma utilização adequada da CPU.

- O algoritmo foi eficaz em cenários onde a justiça e a prioridade de execução dos processos são importantes.
- Não houve grande variação entre as distribuições utilizadas.

4. ***Priority Preemptive*** **(PRIORITYP)**:
    - Resultados semelhantes ao algoritmo de Prioridade, com tempos de espera e resposta elevados.
    - O PRIORITYP prioriza processos com maior prioridade, mas pode levar a tempos de espera maiores para processos com menor prioridade.
    - A utilização da CPU foi de 100%, mas o desempenho geral pode ser impactado pela natureza preemptiva do algoritmo.
5. ***Shortest Job First*** **(SJF)**:
    - Este algoritmo foi o mais eficiente em termos de tempo médio de espera e tempo médio de resposta.
    - Ao priorizar os processos com menor burst de CPU, o SJF minimiza o tempo de espera para a maioria dos processos.
    - A utilização da CPU foi de 100%, indicando que o algoritmo foi capaz de maximizar a utilização do processador.
6. ***Earliest Deadline First*** **(EDF)**:
    - Apresentou tempos médios de espera e resposta melhores que o algoritmo Prioridade, mas não tão bons quanto o SJF.
    - O algoritmo prioriza processos com prazos de execução mais próximos, mas ainda assim apresenta uma utilização eficiente da CPU.
    - A sua eficiência é mais visível em cenários com deadlines críticos, mas pode ser menos eficaz em sistemas com variação de burst de CPU.
7. ***Rate Monotonic*** **(RM)**:
    - Resultados semelhantes aos do algoritmo Prioridade, com tempos de espera e resposta elevados.
    - O RM prioriza processos com períodos mais curtos, mas isso pode não ser ideal em sistemas com burst de CPU variado.
    - A utilização da CPU foi de 100%, o que indica que o processador foi totalmente utilizado, mas o desempenho geral pode ser impactado pela escolha da prioridade.

Distribuição do Trabalho

O desenvolvimento deste projeto foi realizado em colaboração entre os elementos do grupo, com uma divisão de tarefas criteriosa e bem definida, garantindo que cada etapa do trabalho estivesse a cargo de um membro responsável e competente para a sua execução. Esta abordagem permitiu uma maior eficiência, paralelização do esforço e especialização por componentes. Abaixo descrevemos em maior detalhe as responsabilidades assumidas por cada membro:

Sara:

- Foi responsável pela lógica principal do simulador, tendo desenvolvido a função principal (`main`) e estruturado a forma como os diferentes módulos interagem.
- Implementou o sistema de leitura de parâmetros da linha de comandos e integrou os algoritmos com os dados dos processos, garantindo modularidade e clareza no código.
- Desenvolveu o módulo de geração de processos, focando-se na aplicação de distribuições probabilísticas (Exponencial, Uniforme e Normal) para simular cenários realistas de carga no sistema.
- Trabalhou também na parametrização da aleatoriedade através de seeds, o que permitiu a reprodutibilidade dos testes.

Juliana:

- Focou-se na implementação dos algoritmos de escalonamento: FCFS e Round Robin. Para o FCFS, foi feita uma ordenação eficiente dos processos com base no tempo de chegada. No caso do *Round Robin*, foi criado um sistema de fila circular com gestão de quantum e controle de execução temporal. Ambos foram validados com casos de teste próprios.
- Criou o módulo de cálculo de métricas, como tempo médio de espera, tempo médio de resposta, utilização da CPU e *throughput*.
- Executou múltiplos cenários de simulação para recolha de resultados, comparando os valores obtidos com expectativas teóricas.
- Encarregou-se da organização e redação final do relatório técnico.

Esta colaboração permitiu uma abordagem multidisciplinar ao projeto, enriquecendo a solução final com diferentes perspetivas técnicas e reforçando o trabalho em equipa.

Conclusão

Em resumo, o desenvolvimento do ProbSched demonstrou que a escolha do algoritmo de escalonamento deve depender das características do sistema e do comportamento dos processos. O *Round Robin* (RR) se mostrou eficiente em sistemas com cargas de trabalho mais imprevisíveis, sendo especialmente útil quando é necessário distribuir o tempo de CPU de forma equitativa entre os processos. O FCFS revelou-se adequado para cenários com pouca variabilidade nos tempos de execução, mas apresentou tempos de espera mais elevados quando a carga do sistema era mais variada. Já o algoritmo Prioridade (com e sem preempção) foi eficaz na gestão de prioridades, mas teve resultados mistos em termos de tempos de espera e resposta. O *Rate Monotonic* (RM) teve resultados semelhantes aos de Prioridade, mas pode não ser ideal em sistemas com burst de CPU mais variado.

O simulador desenvolvido permite uma análise detalhada do comportamento desses algoritmos e pode ser utilizado como uma ferramenta útil para entender o desempenho de diferentes estratégias de escalonamento de CPU em diferentes cenários.

Repositório do Projeto

O código-fonte do projeto encontra-se disponível no seguinte URL:
`https://github.com/srsbastos/Sistemas-Operativos.git`