Licenciatura em Engenharia Informática

Sistemas Operativos

ProbSched: Simulador de Escalonamento Probabilístico

Autores:
Fábio Horta (nº 51817)
Francisco Pereira (nº 52129)
João Freire (nº 52216)
Daniel Cardoso (nº 52218)

Orientador:
Professor Doutor Paul Crocker

Abril 2025

Resumo

Este documento descreve o desenvolvimento do projeto **ProbSched**, um simulador de algoritmos de escalonamento probabilístico de processos em sistemas operativos. O projeto foi desenvolvido em linguagem **C**, utilizando distribuições probabilísticas para gerar tempos de chegada e de burst dos processos. O objetivo é comparar o desempenho de diferentes algoritmos de escalonamento, avaliando métricas como o tempo de espera, turnaround time, utilização de CPU e misses de deadlines.

Conteúdo

Capítulo 1

Modelos Probabilísticos Usados

1.1 Tempos de Chegada

Utilizámos distribuições **Exponencial** e **Poisson** para modelar os tempos entre chegadas de processos.

1.2 Burst Time (tempo de CPU)

Modelado com:

- **Distribuição Exponencial** para simular bursts curtos e alguns longos.
- **Distribuição Normal** para gerar bursts próximos de um valor médio.

1.3 Prioridades

As prioridades dos processos foram atribuídas através de:

- **Distribuição Uniforme**
- **Distribuição Ponderada** (com mais probabilidade para prioridades mais altas)

Capítulo 2

Algoritmos Implementados

Foram implementados os seguintes algoritmos de escalonamento:

- FCFS (First Come First Served)
- SJF (Shortest Job First)
- Round Robin (com quantum variável)
- Priority Scheduling (Preemptivo e Não-preemptivo)
- EDF (Earliest Deadline First)
- Rate Monotonic Scheduling

Capítulo 3

Execução do Programa

3.1 Compilação

O projeto é compilado utilizando o comando:

```
make
```

O `Makefile` incluído compila automaticamente todos os módulos em C e gera o executável `simulator`.

3.2 Execução

Após a compilação, o simulador é executado com:

```
./simulator
```

Durante a execução, o utilizador seleciona:

- O modo de execução: Monocore ou Multicore (2 a 14 cores).
- O algoritmo de escalonamento.
- O número de processos ou o tempo de simulação.
- O método de geração dos processos: manual, estático ou aleatório.

3.3 Parâmetros de Simulação

Os parâmetros de entrada são pedidos interativamente:

- Algoritmo a utilizar (FCFS, SJF, RR, Priority, EDF, RM, MLQ).
- Quantum (se aplicável, no caso de RR ou MLQ).
- Número de processos a gerar ou tempo máximo de simulação.
- Lambda das distribuições Exponencial/Poisson para processos aleatórios.

Capítulo 4

Implementação e Organização do Trabalho

4.1 Estrutura do Projeto

O projeto **ProbSched** foi desenvolvido em linguagem C, organizado de forma modular para facilitar manutenção e extensões futuras.

- `main.c` — Controlo do fluxo principal da aplicação.
- `algorithms.c` / `algorithms.h` — Definição das estruturas, funções utilitárias e alguns algoritmos.
- `process.c` — Geração de processos (manual, estática e aleatória).
- `scheduler_FCFS.c`, `scheduler_RR.c`, `scheduler_PS.c`, `scheduler_EDF.c`, `scheduler_RT.c`, `scheduler_MQS.c` — Implementações dos vários algoritmos de escalonamento.
- `scheduler_multicore.c` — Execução paralela utilizando threads para ambientes multicore.
- `Makefile` — Automatização da compilação.
- `README.md` — Breve descrição do projeto.

4.2 Distribuição de Tarefas

As tarefas foram divididas de forma colaborativa entre os membros do grupo:

- Implementação dos algoritmos de escalonamento.
- Desenvolvimento do modo multicore.
- Implementação das distribuições probabilísticas.
- Organização do projeto e preparação do relatório.

Capítulo 5

Resultados e Comparação de Performance

Foram realizados testes para comparar os algoritmos implementados com distribuições de tempos de burst (Normal e Exponencial). Obtendo assim estes resultados:

- Tempo médio de espera.
- Tempo médio de turnaround.
- Utilização da CPU (%).
- Throughput (processos completados por unidade de tempo).
- Número de deadlines perdidas (relevante em EDF e RM).

5.1 Resultados com Distribuição Exponencial (lambda = 1)

Algoritmo	Espera	Turnaround	CPU (%)	Throughput	DL Perdidas
SJF	1.5	3.1	100%	0.62	0
FCFS	2.2	3,8	100%	0.62	0
Prio NP	2.3	3.9	100%	0.62	0
RM	0.62	1.77	100%	0.65	21
Prio P	2.3	3.9	100%	0.62	2
MLQ	7	16.2	90.3%	1.1	0
EDF	8.12	9.25	100%	0.8	32
RR	2.2	3.8	100%	0.62	0

Tabela 5.1: Resultados da simulação com distribuição Exponencial ($\lambda = 1$).

5.2 Resultados com Distribuição Poisson (média = 12, desvio padrão = 2)

Algoritmo	Espera	Turnaround	CPU (%)	Throughput	DL Perdidas
SJF	0.9	2	100%	0.83	0
FCFS	2.1	3.54	100%	0.71	0
Prio NP	1.7	2.9	100%	0.83	0
RM	0.95	1.95	100%	1.0	5
Prio P	4.1	5.4	86.67%	0.67	0
MLQ	7.1	15.0	92.1%	1.1	0
EDF	9.17	10.22	100%	0.9	16
RR	0	1.0	100%	10	0

Tabela 5.2: Resultados da simulação com distribuição Poisson (média = 12, desvio padrão = 2).

5.3 Conclusão

Com base nos resultados obtidos nas simulações com as distribuições Exponencial e Poisson, é possível tirar algumas conclusões sobre o desempenho dos algoritmos de escalonamento implementados.

Os algoritmos com foco no tempo de burst, como o **SJF**, mostraram-se mais eficientes em termos de tempo de espera e turnaround. Já os algoritmos sensíveis a deadlines, como **RM** e **EDF**, tiveram dificuldades em manter a eficiência quando os prazos não foram cumpridos, destacando a importância de escolher o algoritmo adequado conforme os processos a serem escalonados.

Isto deve-se aos erros propositados implementados durante o desenvolvimento. Foram introduzidos 4 erros: no arquivo `algorithms.c`, onde configurámos a utilização da CPU como 100%; no `scheduler_EDF`, onde aumentámos o número de deadlines perdidas; no `scheduler_PS`, onde invertemos a lógica de prioridades, considerando que a prioridade maior é melhor em vez de a menor ser a mais alta; e, finalmente, no `scheduler_RR`, onde comentámos grande parte do código, o que pode resultar em acessos indevidos à memória e possíveis falhas (segfaults).

URL do projeto

- https://github.com/Fegue3/ProbSched