Sistemas Operacionais

Relatório EP1: Escalonador de Processos

ACH2055 - 10/2025

Integrantes do grupo

Alan Moura	15436668
Arthur Hernandes	15552518
Felipe Ferreira	15494604
Gabriel Luis	15494841
Isabella Morija	14579951

1. Introdução

O objetivo deste relatório é avaliar o impacto do tamanho do *quantum* em um sistema de escalonamento Round Robin para processos de uma máquina simulada de um único processador, encontrando um equilíbrio que minimize a sobrecarga (*overhead)* e, ao mesmo tempo, garanta uma distribuição justa do tempo de processador entre os processos. O *quantum* determina o número máximo de instruções que um processo pode executar antes de ser interrompido e colocado novamente na fila de prontos, influenciando diretamente o desempenho do sistema em termos de trocas de contexto e aproveitamento da CPU.

Os códigos referentes ao programa de simulação desenvolvido podem ser consultados em Adamarac/EP1-SO-Escalonador.

2. Metodologia

Foram executados diversos testes variando o valor do *quantum*, conforme especificado em arquivos de configuração. Para cada execução, o escalonador gerou um arquivo de log (logXX.txt) contendo informações sobre interrupções, trocas de contexto e instruções executadas.

a) Estruturas da Implementação

A tabela abaixo apresenta, resumidamente, o funcionamento do programa, especificando as estruturas.

Estrutura	Local	Lógica de execução
Estratégia Geral	Modules/ Escalonador.java, Interpreter.java, Logger.java	Quantum lido de programas/quantum.txt; a cada ciclo processa desbloqueios, escolhe próximo pronto (FIFO), executa até E/S, SAIDA ou fim do quantum, registra no log e realoca (prontos/bloqueados/término).
BCP	Structures/ BCP.java	PC inicia em 1, registradores X/Y, estado (PRONTO / EXECUTANDO / BLOQUEADO) e “memória” reference (PID = primeira linha do arquivo); contexto salvo/restaurado pelo Interpreter.
Tabela de Processos	Structures/ processTable.java	HashMap<PID,BCP> com add/remove/lista; usada para ordenação inicial e remoção em SAIDA; o laço encerra quando fica vazia.
Lista de Processos Prontos	Structures/ readyList.java	Fila FIFO: inserir marca PRONTO; poll seleciona o próximo; ao fim do quantum (sem E/S / SAIDA) o processo retorna ao final da fila.
Lista de Processos Bloqueados	Structures/ blockedList.java, Instructions/ ES.java	Em E/S entra com temporizador (inicial 2); clock() decrementa e, ao expirar, devolve para prontos; retomada ocorre na próxima instrução.
Logs e Métricas	Modules/ Logger.java	Registra “Carregando / Executando / E/S / Interrompendo / Terminado”; calcula média de trocas por processo e média de instruções por quantum; escreve em logXX.txt (XX = quantum).

3. Resultados

Os resultados foram agrupados e organizados em planilhas (statistics.csv) e posteriormente comparados entre dois conjuntos de logs — o primeiro com menor número de operações de E/S (*CPU bound*) e o segundo com carga de I/O mais significativa (*I/O bound*).

A partir dos resultados, foram gerados gráficos de linha para acompanhar as tendências de cada variável em função do *quantum* (Gráficos 1, 2, 3 e 4). Também são informados os valores exatos.

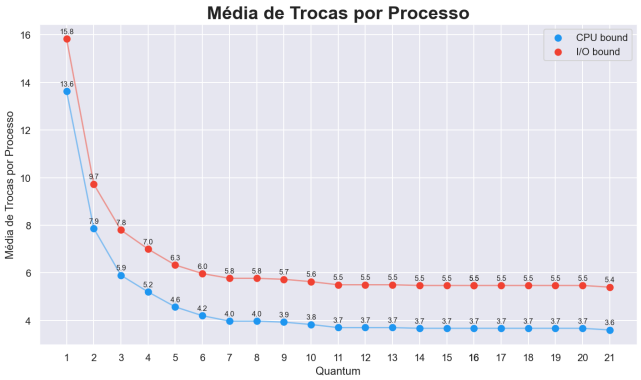

Gráfico 1 - Número Médio de Trocas de Contexto vs Quantum

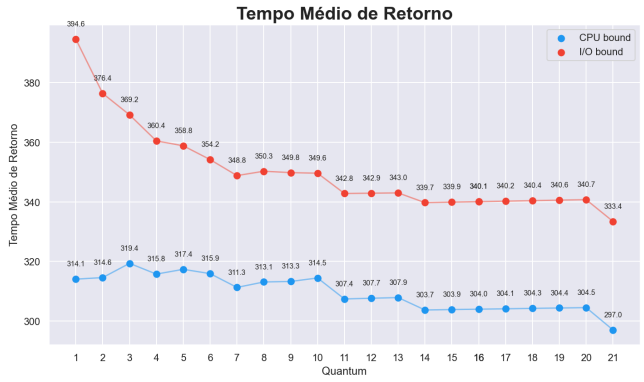

Gráfico 2 - Tempo Médio de Retorno (MTT) vs Quantum

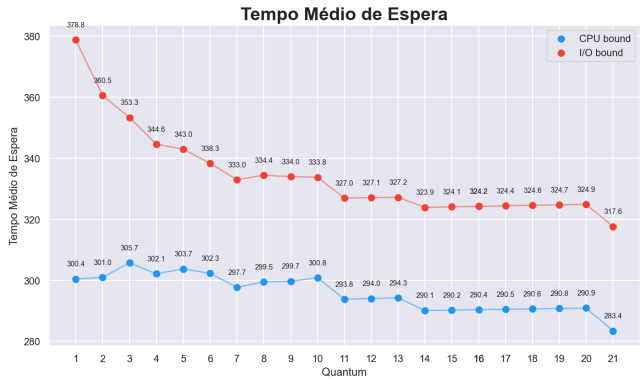

Gráfico 3 - Tempo Médio de Espera vs Quantum

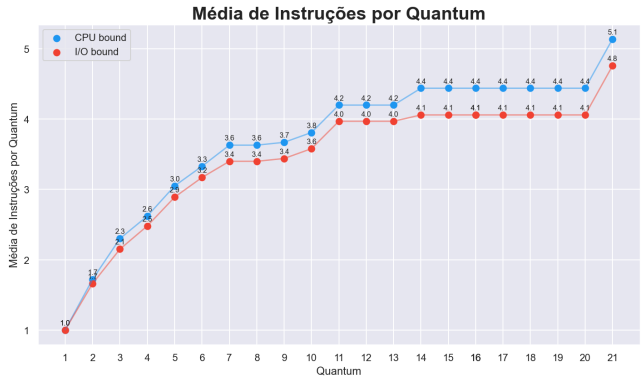

Gráfico 4 - Média de instruções vs Quantum

4. Discussão

A análise dos resultados obtidos permite uma clara compreensão de como o tamanho do *quantum* influencia o desempenho do escalonador Round Robin sob diferentes cargas de trabalho, especificamente em cenários de baixa e alta densidade de E/S. Os dados revelam um *trade-off* fundamental entre a responsividade do sistema e o overhead gerado pelas trocas de contexto.

a) Impacto no Número de Trocas de Contexto

O Gráfico 1 demonstra a relação mais direta e esperada: o aumento do *quantum* leva a uma redução drástica no número de trocas de contexto. Com um *quantum* muito pequeno (ex: 1), o sistema incorre em um número excessivo de trocas (média 15,8 para *CPU bound* e 13,6 para *I/O bound*), pois os processos são constantemente interrompidos e salvos. À medida que o *quantum* aumenta, essa curva decresce rapidamente e tende a uma estabilização (aprox. no *quantum* de 7), indicando que as trocas, em maioria, passam a ser causadas por E/S.

b) Análise do Tempo de Retorno e Tempo de Espera

Os Gráficos 2 e 3 apresentam um comportamento similar. Para *quantums* muito pequenos, ambos os tempos são elevados devido à alta *overhead* gerada pelo excesso de trocas de contexto.

Conforme o *quantum* aumenta, há uma melhora significativa no desempenho, com a redução de ambos os tempos. No entanto, após um certo ponto (aprox. *quantum* de 11), a curva se estabiliza, sugerindo que ganhos adicionais são marginais.

c) Eficiência de Execução: Média de Instruções por Quantum

O Gráfico 4 é um bom objeto de análise para nosso estudo, pois resume as conclusões anteriores sobre a eficiência da CPU simulada:

I) Para *quantums* pequenos, a média de instruções acompanha o tamanho do *quantum*, indicando que os processos usam toda a fatia de tempo concedida.

II) Como já apontado anteriormente, **conforme o *quantum* aumenta**, a curva da média de instruções começa a se achatar por volta de *quantum* de 11 (especialmente em *I/O bound*), por conta das interrupções E/S. Por exemplo, com um *quantum* de 20, a média de instruções executadas no cenário *I/O bound* é de apenas 4.84, mostrando que a maior parte do tempo de CPU alocado não foi utilizada.

III) Nota-se uma **melhora abrupta no *quantum* 21** (notada também nos Gráficos 2 e 3). Neste ponto, a maioria dos processos finalizam ou solicitam E/S antes das interrupções. Embora isso otimize os tempos de retorno para o contexto e condições de simulação (por hipótese), descaracteriza o algoritmo, pois anula o objetivo de garantir um equilíbrio justo no uso da CPU.

5. Conclusão

Em síntese, os resultados confirmam que a escolha de um *quantum* ótimo é um balanceamento. Um *quantum* muito pequeno degrada o desempenho devido à *overhead* decorrida das trocas de contexto. Um *quantum* muito grande pode reduzir a responsividade do sistema e não traz benefícios em cenários com muitas operações de E/S, onde os processos se bloqueiam antes de usar todo o seu tempo de CPU. Para os resultados analisados do programa desenvolvido, **o *quantum* com valor de 11** parece oferecer um bom equilíbrio entre os fenômenos discutidos.