



Sistemas Operacionais

Especificação do Trabalho Prático (Implementação)

Profa.: Aletéia Patrícia Favacho de Araújo

Orientações Gerais

O trabalho de implementação da disciplina de SO, a ser desenvolvido em grupo com quatro (04) componentes, compreenderá as seguintes fases:

- Estudo teórico relacionado ao assunto do trabalho;
- Apresentação da solução teórica dada ao problema;
- Implementação da solução proposta;
- Apresentação e explicação detalhada do código-fonte implementado;
- Relatório explicando o processo de construção e o uso da aplicação.

Orientações Específicas

1 Problema

Implementação de um **pseudo-SO** multiprogramado, composto por um **Gerenciador de Processos**, por um **Gerenciador de Memória** e por um **Gerenciador de Entrada/Saída**. O gerenciador de processos deve ser capaz de aplicar o algoritmo de escalonamento definido por meio de parâmetro pelo usuário do SO. O gerenciador de memória deve garantir que um processo não acesse as regiões de memória de um outro processo, e que o algoritmo de substituição de página seja adequadamente usado. E o gerenciador de entrada/saída deve ser responsável por administrar o algoritmo especificado para a busca em disco. Cada módulo será testado de acordo com as especificações determinadas abaixo. Além disso, o pseudo-SO deve receber como parâmetro um inteiro e um arquivo texto, por exemplo \$ 1 processes.txt. O inteiro determina qual módulo deve ser ativado (no exemplo dado significa que será ativado o módulo de processos, pois foi o inteiro 1), e o arquivo texto (com extensão .txt) repassa os dados de entrada necessários para a execução do módulo escolhido. Os detalhes para a implementação desse pseudo-SO são descritos nas próximas seções.

1.1. Módulo de Gerência de Processos

Neste módulo a equipe deve implementar um conjunto de algoritmos de escalonamento de CPU e escrever um programa que calcula uma série de estatísticas baseada nestes algoritmos. Os algoritmos de escalonamento a serem implementados são os seguintes:



- FIFO: *First-In, First-Out*
- SJF: *Shortest Job First*
- RR: *Round Robin* (com quantum = 2)

O módulo de gerência de processos deverá ler da entrada padrão uma lista de processos com seus respectivos tempos de chegada e de duração, e deverá imprimir na saída padrão uma tabela contendo os valores para as seguintes métricas:

- Tempo médio de execução total do processo - *turnaround*;
- Tempo médio de resposta;
- Tempo médio de espera.

Tempo de execução total do processo é a quantidade de tempo necessária para executar totalmente um processo, ou seja, é o tempo total entre a criação de um processo e seu término. Tempo de resposta é a quantidade de tempo entre a requisição de execução de um programa (quando ele é colocado na fila de pronto) e o seu tempo de ir para a execução (sistema de compartilhamento de tempo), isso significa que é o tempo que o processo demora para produzir cada resposta a uma requisição (assim, vamos considerar que é o tempo decorrido entre uma ida e outra para a CPU). Essa métrica é importante para processos interativos). E tempo de espera é a quantidade total de tempo que um processo aguardou na fila de prontos esperando para ser escalonado, ou seja, é o tempo que processo ficou esperando na fila de prontos.

1.1.1 Descrição da Entrada do Módulo de Gerência de Processos

O arquivo texto é composto por uma série de pares de números inteiros separados por um espaço em branco indicando o tempo de chegada e a duração de cada processo. A entrada termina com o fim do arquivo. A ativação deste módulo deve ser com os parâmetros \$ <executável> 1 <nomeArquivoTexto.txt>

Exemplo de entrada:

```
0 20
0 10
4 6
4 8
```

1.1.2 Descrição da Saída do Módulo de Gerência de Processos

A saída é composta por linhas contendo a sigla de cada um dos três algoritmos e os valores das três métricas solicitadas.



Cada linha apresenta a sigla do algoritmo e os valores médios (com uma casa decimal) para tempo total de execução, tempo de resposta e tempo de espera, exatamente nesta ordem, separados por um espaço em branco.

Exemplo de saída:

```
FIFO 30,5 19,5 19,5  
SJF 21,5 10,5 10,5  
RR 31,5 2,0 20,5
```

Para facilitar a depuração, a equipe deve gerar em um arquivo de saída a ordem de execução dos processos. Como apresentado a seguir, para cada algoritmo, do exemplo descrito anteriormente.

FIFO:

```
Rodar processo [0] de [0] ate [20]  
Rodar processo [1] de [20] ate [30]  
Rodar processo [2] de [30] ate [36]  
Rodar processo [3] de [36] ate [44]
```

SJF:

```
Rodar processo [1] de [0] ate [10]  
Rodar processo [2] de [10] ate [16]  
Rodar processo [3] de [16] ate [24]  
Rodar processo [0] de [24] ate [44]
```

RR:

```
Rodar processo [0] de [0] ate [2]  
Rodar processo [1] de [2] ate [4]  
Rodar processo [0] de [4] ate [6]  
Rodar processo [2] de [6] ate [8]  
Rodar processo [3] de [8] ate [10]  
Rodar processo [1] de [10] ate [12]  
Rodar processo [0] de [12] ate [14]  
Rodar processo [2] de [14] ate [16]  
Rodar processo [3] de [16] ate [18]  
Rodar processo [1] de [18] ate [20]  
Rodar processo [0] de [20] ate [22]  
Rodar processo [2] de [22] ate [24]  
Rodar processo [3] de [24] ate [26]  
Rodar processo [1] de [26] ate [28]  
Rodar processo [0] de [28] ate [30]  
Rodar processo [3] de [30] ate [32]  
Rodar processo [1] de [32] ate [34]  
Rodar processo [0] de [34] ate [36]  
Rodar processo [0] de [36] ate [38]  
Rodar processo [0] de [38] ate [40]  
Rodar processo [0] de [40] ate [42]  
Rodar processo [0] de [42] ate [44]
```



1.2. Módulo de Gerência de Memória

Neste módulo a equipe deve escrever um programa para simular o funcionamento dos principais algoritmos de substituição de páginas estudados na disciplina. Os algoritmos de substituição de páginas a serem implementados são os seguintes:

- FIFO (*First In, First Out*);
- Segunda Chance (com o bit R sendo zerado a cada 3 referências feitas à memória);
- LRU: (*Least Recently Used* ou Menos Recentemente Utilizado).

O programa deverá ler da entrada padrão um conjunto de números inteiros, dos quais o primeiro número representa a quantidade de quadros de memória disponíveis na RAM e os demais representam a sequência de referências às páginas, sempre um número por linha.

Além disso, o programa deverá imprimir na saída o número de faltas de páginas obtido com a utilização de cada um dos algoritmos.

1.2.1 Descrição da Entrada para o Módulo de Gerência de Memória

A entrada é composta por uma série de números inteiros, um por linha, indicando, primeiro a quantidade de quadros (frames) disponíveis na memória RAM e, em seguida, a sequência de referências à memória. A ativação deste módulo deve ser com os parâmetros \$ <executável> 2 <nomeArquivoTexto.txt>

Exemplo de entrada:

```
4  
1  
2  
3  
4  
1  
2  
5  
1  
2  
3  
4  
5
```

1.2.2 Descrição da Saída para o Módulo de Gerência de Memória

A saída é composta por linhas contendo a sigla de cada um dos três algoritmos e a quantidade de faltas de página obtidas com a utilização de cada um deles.



FIFO 10
SC 10
LRU 8

1.3. Módulo de Gerência de Entrada/Saída

Neste módulo a equipe deve escrever um programa para simular o funcionamento dos principais algoritmos de escalonamento de disco estudados na disciplina. Os algoritmos de escalonamento de disco a serem implementados são os seguintes:

- FCFS (*First Come, First Serve*);
- SSF (ou SSTF – braço inicialmente para baixo);
- SCAN.

O programa deverá ler da entrada padrão um conjunto de número inteiros, no qual o primeiro número representa a quantidade de cilindros no disco, o segundo número representa o cilindro sobre o qual a cabeça de leitura do disco está inicialmente posicionada, e os demais representam uma sequência de requisições de acesso a serem atendidas, sempre um número por linha.

O programa deverá imprimir na saída o número total de cilindros percorridos pela cabeça de leitura para atender todas as requisições solicitadas utilizando cada um dos algoritmos.

1.3.1 Descrição da Entrada do Módulo de Entrada/Saída

A entrada é composta por uma série de números inteiros, um por linha, indicando, primeiro o número do último cilindro no disco (os cilindros variam de 0 até este número), o cilindro sobre o qual a cabeça de leitura está inicialmente posicionada e a sequência de requisições de acesso. A ativação deste módulo deve ser com os parâmetros \$ <executável> 3 <nomeArquivoTexto.txt>

Exemplo de entrada:

199
53
98
183
37
122
14
124
65
67



1.3.2 Descrição da Saída do Módulo de Entrada/Saída

A saída é composta por linhas contendo a sigla de cada um dos três algoritmos e a quantidade total de cilindros percorridos pela cabeça de leitura para atender todas as requisições de acesso ao disco.

Exemplo de saída:

```
FCFS 640  
SSTF 236  
SCAN 236
```

2 Estrutura do Programa

Espera-se que o programa seja estruturado em, pelo menos, quatro grandes módulos: kernel, processo, memória e entrada/saída. Esses modelos devem ser:

- **Módulo Kernel** – contém as chamadas para os demais módulos.
- **Módulo de Processos** – classes e estruturas de dados relativas ao processo. Basicamente, mantém informações específicas do processo.
- **Módulo de Memória** – provê uma interface de abstração de memória RAM.
- **Módulo de Entrada/Saída** – trata a alocação do braço do disco para realização de escrita/leitura nos blocos do disco para os processos.

É importante ressaltar também que outros módulos podem ser utilizados, caso sejam necessários.

3 Estudo Teórico para a Solução

Cada equipe deverá buscar a solução para o compartilhamento de recursos e a sincronização de processos (ou *threads*), quando se fizer necessário, de acordo com o problema proposto. É responsabilidade de cada grupo estudar a maneira mais eficiente para implementar o pseudo-SO. Contudo, é importante ressaltar que para o problema de compartilhamento de recursos não há soluções mágicas, as soluções possíveis são exatamente as mesmas vistas em sala de aula: semáforos (baixo nível), monitores (alto nível, mas devem ser implementados pela linguagem de programação, pois o compilador deve reconhecer o tipo monitor) e troca de mensagens (usadas preferencialmente para garantir a sincronização entre processos em máquinas diferentes, também podem ser usadas para processos na mesma máquina. Nesse caso, as mensagens trocadas passam por toda a pilha de protocolos como se estivessem sendo enviadas para outra máquina).



4 Implementação

O trabalho valerá 10 pontos, sendo 9 pontos dados à corretura dos algoritmos e 1 ponto dado à qualidade do código. Portanto, o código-fonte deve seguir as boas práticas de programação: nomes de variáveis auto-explicativos, código comentado e identado. As métricas de medida a serem usadas para avaliar a qualidade do código serão baseadas naquelas descritas em [2]. Dessa forma, funções grandes, classes com baixa coesão, código mal-identado e demais práticas erradas acarretarão no prejuízo da nota.

Contudo, é fundamental que apenas os padrões das linguagens, sem bibliotecas de terceiros, sejam utilizados. Ou seja, se você for executar um programa escrito em Java no Linux, que eu precise apenas do SDK padrão para recompilá-lo, se o programa for escrito em C, que eu possa compilá-lo com o padrão -ansi ou c99, etc. As especificações de padrão devem ser explicitadas. Isto é para evitar problemas de executar na máquina de vocês e não executar na minha. Padronização é fundamental!

A implementação poderá ser feita em qualquer linguagem e utilizando qualquer biblioteca, desde que o código seja compatível com ambientes UNIX. Programas que utilizarem bibliotecas ou recursos adicionais deverão conter no relatório informações detalhadas das condições para a reprodução correta do programa. Além disso, bibliotecas proprietárias não são permitidas.

5 Responsabilidades

Cada equipe é responsável por realizar, de maneira exclusiva, toda a implementação do seu trabalho e entregá-lo funcionando corretamente. Além disso, o grupo deverá explicar detalhadamente todo o código-fonte desenvolvido. É fundamental que *toda a equipe* esteja presente no dia/horário definido para apresentar e explicar o código-fonte.

6 Material a ser entregue

A equipe deverá, além de entregar todo o código fonte zipado em um arquivo .ZIP no Ambiente Aprender 3, apresentar o código-fonte executando no dia e horário definido para cada equipe. Além disso, a equipe deve entregar um relatório impresso (mínimo de 2 e máximo de 5 páginas) contendo, obrigatoriamente, **no mínimo**, os seguintes itens:

- Descrição das ferramentas/linguagens usadas;
- Descrição teórica e prática da solução dada;
- Descrição das principais dificuldades encontradas durante a implementação;
- Para todas as dificuldades encontradas, explicar as soluções usadas;
- Descrever o papel/função de cada aluno na realização do trabalho;
- Bibliografia.



7 Cronograma

O trabalho deverá ser entregue/apresentado no dia especificado para cada equipe, no horário da aula (das 8:00 às 9:50). Nenhum trabalho será recebido fora do prazo, por qualquer que seja o motivo. É obrigatória a presença de todos os membros da equipe para a explicação do código-fonte desenvolvido e das decisões tomadas. No dia da apresentação todos os alunos da equipe devem estar com o microfone e a câmera ligados para que seja comprovada a sua real participação no trabalho. Para cada dia, a ordem de apresentação seguirá a ordem de numeração dos grupos, a qual ocorrerá em *link* específico, criado antecipadamente para cada equipe, na plataforma ZOOM. Assim, a data de apresentação/entrega dos trabalhos deve seguir as datas abaixo:

- **Dia 10/05/2021 – Grupos 01, 02 e 03**
- **Dia 12/05/2021 – Grupos 04, 05 e 06**
- **Dia 14/05/2021 – Grupos 07, 08 e 09**
- **Dia 17/05/2021 – Grupos 10, 11 e 12**

8 Referências

- [1] Stallings, W. *Operating Systems Internals and Design Principles*, Pearson Education, 2009.
- [2] Martin, R. C. *Clean Code. A Handbook of Agile Software Craftsmanship*, Prentice Hall, 2008.