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 três (03) 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 Recursos** O gerenciador de processos deve ser capaz de agrupar os processos em quatro níveis de prioridades. O gerenciador de memória deve garantir que um processo não acesse as regiões de memória de um outro processo. E o gerenciador de recursos deve ser responsável por administrar a alocação e a liberação de todos os recursos disponíveis, garantindo uso exclusivo dos mesmos. Os detalhes para a implementação desse pseudo-SO são descritos nas próximas seções.

1.1 Estrutura das Filas

O programa deve ter duas filas de prioridades distintas: a fila de processos de tempo real e a fila de processos de usuários. Processos de tempo real entram para a fila de maior prioridade, sendo gerenciados pela política de escalonamento FIFO (*First In First Out*), sem preempção.

Processos de usuário devem utilizar múltiplas filas de prioridades com realimentação. Para isso, deve-se manter **três** filas com prioridades distintas. Para evitar *starvation*, o sistema operacional deve modificar a prioridade dos processos executados mais frequentemente e/ou utilizar uma técnica de envelhecimento (*aging*). **Quanto menor** for o valor da prioridade atribuída a um processo, **maior** será a sua prioridade no escalonamento. Dessa forma, os processos de tempo real, que são os mais prioritários, terão

prioridade definida como **0 (zero)**. Além disso, é importante destacar que processos de usuário podem ser preemptados e o *quantum* deve ser definido de **1 segundo**.

As filas devem suportar no máximo **1000 processos**. Portanto, recomenda-se utilizar uma fila "global", que permita avaliar os recursos disponíveis antes da execução e que facilite classificar o tipo de processo. A Figura 1 ilustra o esquema indicado.

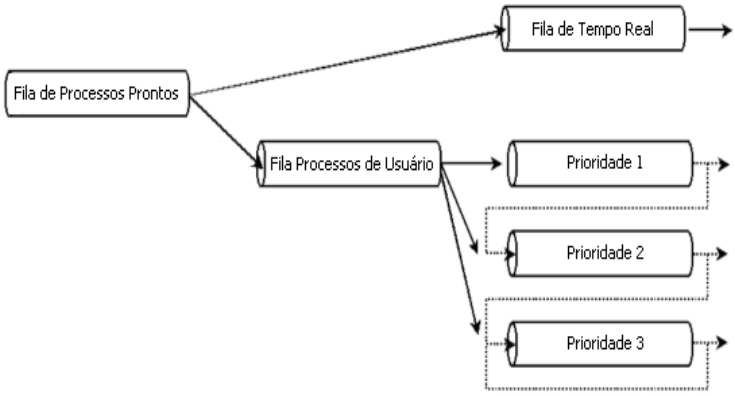

Figura 1 – Estrutura de Filas para o Pseudo-SO.

1.2 Estrutura de Memória

A alocação de memória deve ser implementada como um conjunto de blocos contíguos, onde cada bloco equivale uma palavra da memória real.

Cada processo deve alocar um segmento contíguo de memória, o qual permanecerá alocado durante toda a execução do processo. Deve-se notar que não é necessário a implementação de memória virtual, *swap*, nem sistema de paginação. Portanto, não é necessário gerenciar a memória, apenas verificar a disponibilidade de recursos antes de iniciar um processo.

Deve ser utilizado um tamanho fixo de memória de 1024 blocos. Dessa quantidade, 64 blocos devem ser reservados para processos de tempo-real e os 960 blocos restantes devem ser compartilhados entre os processos de usuário. A Figura 2 ilustra o caso onde cada bloco de memória possui 1 MB.

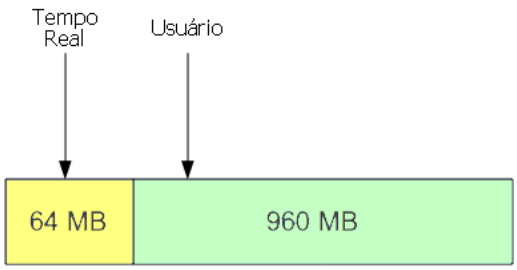

Figura 2 – Modelo de memória com região reservada aos processos de tempo real e aos processos de usuário.

1.3 Estrutura dos Recursos Disponíveis

O pseudo-SO deve, além de escalonar o processo no compartilhamento da CPU, e gerenciar o espaço de memória de acordo com as áreas reservadas, ele deve gerenciar os seguintes recursos:

- 1 *scanner*
- 2 impressoras
- 1 modem
- 2 dispositivos SATA

Todos os processos, com exceção daqueles de tempo-real podem obter qualquer um desses recursos. O pseudo-SO deve garantir que cada recurso seja alocado para um proceso por vez. Portanto, não há preempção na alocação dos dispositivos de E/S.

Processos de tempo-real não precisam de recursos de I/O e podem ter tamanho fixo, ficando a cargo do programador.

2 Estruturamento do Programa

Espera-se que o programa seja dividido em, pelo menos, quatro grandes módulos: processo, memória, filas e recursos. Esses modelos devem ser:

- **Módulo de Processos** – classes e estruturas de dados relativas ao processo. Basicamente, mantém informações específicas do processo.
- **Módulo de Filas** – mantém as interfaces e funções que operam sobre os processos.
- **Módulo de Memória** - provê uma interface de abstração de memória RAM.
- **Módulo de Recurso** – trata a alocação e liberação dos recursos de E/S para os processos.

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

2.1 Interface de Utilização

2.1.1 Saída

O processo principal é o “despachante”. Esse é o primeiro processo criado na execução do problema e deve ser o responsável pela criação de qualquer processo. Na criação do processo, o **despachante** deve exibir uma mensagem de identificação contendo as seguintes informações:

- PID (int);
- Prioridade do processo (int);
- *Offset* da memória (int);
- Quantidade de blocos alocados (int);
- Utilização de impressora (bool);
- Utilização de *scanner* (bool);
- Utilização de *drivers* (bool).

O processo deve exibir alguma mensagem que indique sua execução. Contudo, isso não é obrigatório.

2.1.2 Entrada

O programa deve ler um arquivo contendo as informações dos processos. Neste arquivo, cada linha contém as informações de um único processo. Portanto, um arquivo com 2 linhas disparará 2 processos, outro arquivo com 5 linhas fará com que o programa dispare 5 processos, e assim por diante. Cada processo, portanto, cada linha, deve conter os seguintes dados:

<tempo de inicialização>, <prioridade>, <tempo de processador>, <blocos em memória>, <número-código da impressora requisitada>, <requisição do *scanner*>, <requisição do modem>, <número-código do disco>

2.1.3 Exemplo de Execução do Pseudo-SO

Por exemplo, um arquivo *processes.txt* que contém as linhas:

2, 0, 3, 64, 0, 0, 0, 0

8, 0, 2, 64, 0, 0, 0, 0

Ao ser usado como entrada para o programa desenvolvido deve mostrar na tela, após ser inicializado, algo como:

$ ./dispatcher processes.txt

```
dispatcher =>
    PID: 1
    offset: 0
    blocks: 64
    priority: 0
    time: 3
    printers: 0
    scanners: 0
    modems: 0
    drives: 0

process 1 =>
P1 STARTED
P1 instruction 1
P1 instruction 2
P1 instruction 3
P1 return SIGINT

dispatcher =>
  PID: 2
  offset: 65
  blocks: 64
  priority: 0
  time: 2
  printers: 0
```

```
  scanners: 0
  modems: 0
  drives: 0

process 2 =>
P2 STARTED
P2 instruction 1
P2 instruction 2
P2 return SIGINT
```

Onde **dispatcher =>** indica o momento em que o processo despachante (SO) leu as informações do processo a ser executado, imprimiu e que cedeu tempo de CPU para o processo. **P1** e **P2** são processos que executam, exibindo as informações do que cada um estará fazendo.

3 Estudo Teórico para a Solução

Cada grupo 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, mas 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 grupo 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 *todo o grupo* esteja presente no dia/horário definido para apresentar e explicar o código-fonte.

6 Material a ser entregue

O grupo deverá, além de apresentar o código-fonte executando, entregar todos os arquivos do código desenvolvido e 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
- Bibliográfia

7 Cronograma

O trabalho deverá ser entregue/apresentado nos dias **30/11/2016** e **02/12/2016**, sendo que os grupos de número 01 até 07 devem apresentar no primeiro dia, e os demais grupos no segundo dia. A ordem de apresentação em cada dia será por ordem de chegada na minha sala no prédio do CiC. A apresentação será no horário da aula, ou seja, das 8:00 às 10:00. Nenhum trabalho será recebido fora do prazo, por qualquer que seja o motivo. Dessa forma, os grupos devem apresentar exatamente no dia escalonamento para isso. É obrigatória a presença de **todo o grupo** para a explicação do código-fonte desenvolvido e das decisões tomadas.

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.