LIVRO

UNIDADE 2

Fundamentos de Sistemas Operacionais

Processos e *threads*

Juliana Schiavetto Dauricio

2015
Editora e Distribuidora Educacional S. A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041 -100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

INTRODUÇÃO AOS SISTEMAS OPERACIONAIS

Convite ao estudo

Olá, aluno! Vamos começar a trabalhar com os conceitos e práticas relacionados a processos e *threads*. Você já notou que uma das principais funções dos sistemas operacionais é controlar o processamento de informações de modo que cada uma das etapas, desde a criação de um processo até o seu encerramento, possa ser devidamente registrada e processada, além de garantir a sua continuidade em casos de interrupção e exceções. E, então, já percebeu o quanto isso é importante para o bom funcionamento de sua máquina ou estação de trabalho? A partir de agora, você é convidado a compartilhar desse momento de estudos! Para que você possa desenvolver algumas competências básicas para o uso e trabalho com os sistemas operacionais, vamos relembrar quais são elas:

• **Competência geral**: o aluno deverá ser capaz de identificar quais são as principais funções de um sistema operacional, bem como ter conhecimento sobre como se dá o compartilhamento de recursos e a sua gerência.

• **Competências técnicas**: conhecer a evolução dos sistemas operacionais e suas respectivas especificidades; conhecer e saber identificar os principais processos e como ocorre o compartilhamento de recursos; conhecer como se dá a gerência de processos e de armazenamento de arquivos; conhecer e saber gerenciar os dispositivos de entrada e saída.

Além dessas, você também precisa relembrar os objetivos específicos desta disciplina: saber manipular as informações do sistema operacional, ter conhecimento sobre as principais funcionalidades e como gerenciá-las, além de saber analisar a utilização de recursos e promover a sua otimização. Desse

modo, fica a recomendação de estudos e leituras frequentes de seu livro didático, resolução dos exercícios propostos e o acompanhamento da webaula.

O seu desafio nesta unidade de ensino é gerenciar rotinas e processos, criar, excluir e executar comandos para o processamento de dados de forma a otimizar o sistema operacional que interage com o ERP. Se você tivesse que realizar esse serviço em uma clínica médica que acabou de adquirir um módulo de sistema integrado de gestão, como realizaria essa tarefa sem afetar as múltiplas operações em andamento? Desde já bons estudos e práticas a você!

Seção 2.1

Introdução a processos: modelo, criação, término, hierarquia, estados, implementação e *threads*

Diálogo aberto

Vamos iniciar esta seção de autoestudos conhecendo a estrutura de um sistema operacional. Ele é carregado todas as vezes em que o computador for ligado ou reiniciado. Esse procedimento se chama ativação de sistema ou *boot* (MACHADO; MAIA, 2013).

O sistema operacional é composto basicamente por um conjunto de rotinas que conhecemos como núcleo do sistema, também chamado de kernel, que tem por função realizar o controle e tratamento de interrupções e exceções, criar e eliminar processos e *threads*, sincronizar a comunicação entre eles, bem como escalonar e controlá-los. Desse modo, também é de responsabilidade desse conjunto de rotinas gerenciar memória, sistemas de arquivos, dispositivos de E/S, permitir suporte a redes locais de distribuídas, realizar a contabilização das ações do sistema e também a sua auditoria e segurança (MACHADO; MAIA, 2013).

Para cada uma das rotinas que o sistema executará, há um mecanismo de controle de chamadas de sistema, o *system call*, que pode ser explícito ou implícito. No explícito, há uma instrução de qual chamada deverá ser executada no próprio programa, através da implementação de uma função que carrega os seus respectivos parâmetros. Já na implícita, há a inserção de um comando da linguagem de programação. O *system call* é responsável por verificar os parâmetros da solicitação e enviar a sua respectiva resposta com o estado do processo, no caso, concluído, ou se houve algum erro e precise retornar à pilha de processos. Além disso, também é preciso conhecer as linguagens de comando, pois essas são importantes ferramentas para a criação de arquivos de comandos, também chamados de *batch* ou *shell scripts*. Esses arquivos têm por função viabilizar a automatização de algumas tarefas do sistema operacional que fazem a gerência do sistema. O *shell* é responsável por interpretar esses comandos.

A arquitetura do kernel pode ser monolítica, em camadas, máquina virtual ou, ainda, ser do tipo microkernel. Confira no Quadro 2.1 cada uma das respectivas descrições:

Quadro 2.1 | Tipos de arquitetura de núcleo

Tipo de arquitetura do núcleo	Descrição
Monolítica (MS- DOS e Unix)	Módulos executados separadamente mas que compõem um único executável.
Camadas (MULTICS e OpenVMS)	Devido à complexidade do sistema, ele é dividido em níveis e suas funções só podem ser utilizadas por camadas superiores (usuário, supervisor,executivo e kernel).
Máquina Virtual (VM)	Faz o intermédio entre o hardware e o sistema operacional. Oferece todos os serviços do SO. Pode haver várias máquinas virtuais em uma única máquina.
Microkernel	Menor e mais simples. Trata serviços por processos que oferece funções específicas.

Fonte: Adaptado de Machado e Maia (2013, p. 53- 58).

Tendo em mente que você precisa definir quais são as configurações necessárias para se implementar um sistema de gestão integrado em uma clínica médica, identifique tais características e descreva como acontece essa gestão de processos no sistema operacional indicado. Dessa forma, considere as configurações técnicas do servidor e das estações de trabalho. Boas práticas!

Não pode faltar

Quando estudamos o comportamento dos processos em um sistema operacional, temos de ter em mente que este, mesmo que a máquina contenha uma única unidade central de processamento, poderá criar várias CPUs virtuais. Isso se dá porque um único processo pode gerar outros processos filhos e dividir recursos de processamento. Nesse sentido, é necessário separar o modo como tais informações serão tratadas sob o contexto do *hardware*, do *software* e também do armazenamento de informações (MACHADO; MAIA, 2013).

Figura 2.1 | Modos de tratamento de dados em processos

Hardware	*Software*	Armazenamento
Os dados do processo ficam armazenados nos registradores (status, PC e SP).	Há a especificação de recursos e suas limitações para que possam ser alocados os processos. Nome, PID, Owner (usr), prioridade, data/ hora de criação, tempo de processador, quotas e privilégios.	Refere-se à área de memória que será alocado o processo para que possa ser executado.

Fonte: Adaptado de Machado e Maia (2013).

Como descrito, há a separação de contexto para que seja realizado o devido processamento do sinal enviado. Desse modo, o sistema operacional se torna responsável por realizar esse controle. A possibilidade de recuperar o processo a partir do ponto de interrupção é essencial para que exista a concorrência. No momento em que cessa um deles, a UCP é imediatamente ocupada por outro que será executado, substituindo o contexto de *hardware* de um processo pelo de outro.

Em contexto de *hardware*, os dados são tratados de acordo com o seu estado de processamento e armazenado no respectivo registro responsável por armazenar aquela determinada informação.

No contexto de *software*, as informações que o sistema operacional deve controlar referem-se à quantidade de arquivos que poderão ser abertos concomitantemente e quais processos detêm prioridade de execução, tamanho do *buffer* de E/S, por exemplo. Considera a identificação do processo e de quem o criou (PID – *Process Identification e Owner*), quotas (limites de recursos alocados) e privilégios (pode alterar desde o *status* do processo como o de outros, se este tiver um privilégio de controle de processos de administrador de sistema, por exemplo). Então, durante a criação do processo, há a especificação dos recursos que serão necessários.

No contexto do armazenamento, cada processo possui um endereço específico na memória (MACHADO; MAIA, 2013).

Reflita

A ideia principal é que um processo constitui uma atividade. Ele possui programa, entrada, saída e um estado. Um único processador pode ser compartilhado entre os vários processos, com algum algoritmo de escalonamento usado para determinar quando parar o trabalho sobre um processo e servir outro (TANEMBAUM, 2009, p. 51).

Nos sistemas operacionais multiprogramáveis, os processos não devem receber de forma dedicada todos os recursos da máquina. Com isso, os processos são divididos em estados: execução (*running*), pronto (*ready*) e espera (*wait*). O processo está em execução enquanto é processado pela UCP, sendo que os processos revezam o tempo de processamento controlado pelo sistema operacional. Quando o processo se encontra no estado de pronto, quer dizer que o processo está aguardando para ser processado, enquanto o estado de espera acontece quando o processo aguarda um recurso para continuar o processamento ou, ainda, aguarda o tratamento de um evento para que possa prosseguir. Os processos em espera são organizados no sistema em listas encadeadas e de acordo com o tipo de evento ocorrido. Quando recebem os recursos necessários, mudam para o estado de pronto (MACHADO; MAIA, 2013).

Quando se fala em mudança de estado, é preciso saber que essa só acontecerá se tiver algum evento que interfira na execução do processo. Então, um processo pode sair do estado de pronto e entrar em execução, bem como sair do estado de execução e entrar em espera, ou, ainda, do estado de espera passar para o estado de pronto novamente, e de execução para o estado de pronto (MACHADO; MAIA, 2013).

Além das características estudadas até o momento, um processo também pode representar apenas ações do sistema operacional e por esse motivo chama-se "processo de sistema operacional". Os serviços que o sistema operacional implementa através dos processos são: auditorias e segurança, serviços de rede, contabilização do uso de recursos e de erros, gerência de impressão, de processos em lote tipo *bacth*, a temporização de processos, a comunicação entre eventos e, ainda, a interface de comandos (*shell*).

Os processos do sistema operacional que administram a comunicação entre os eventos e a sua sincronização ocorrem através do envio de sinais. Esses sinais são bits que compõem o bloco de controle de processos também conhecido pela sigla PCB (*Process Control Block*) e que podem ficar em modo de espera até que o processo seja escalonado. Se o processo for eliminado, será acionado o bit correspondente ao evento e ele será excluído apenas quando for entrar em execução, com isso conclui-se que os sinais respondem diretamente aos processos (MACHADO; MAIA, 2013).

Além desses aspectos, quando se cria (new) um processo, é preciso também informar o seu término (exit) ou encerramento. Desse modo, a partir do momento de sua criação, o sistema operacional começa a gerenciá-lo.

Assimile

Há quatro eventos principais que fazem com que processos sejam criados:

1. Início do sistema.

2. Execução de uma chamada de sistema de criação de processo por um processo em execução.

3. Uma requisição do usuário para criar um novo processo.

4. Início de uma tarefa em lote (*batch/job*) (TANEMBAUM, 2009, p. 52).

Para que o sistema operacional possa controlar todas essas informações, há uma estrutura de dado chamada bloco de controle de processo ou PCB (*Process Control Block*), que trabalha da seguinte forma: faz a leitura dos ponteiros, que justamente têm por função apontar para o endereço de memória em que se encontram os registros,

lê o estado do processo, identifica, verifica a prioridade, verifica a informação dos registradores, limites de memória e cria uma lista de arquivos que ainda estão abertos para serem executados.

Mas quando um processo pode ser encerrado? Veja, no exemplo a seguir, o modo como acontece o término de um processo:

Exemplificando

Um processo pode ser encerrado quando:

a) há a saída normal ou voluntária do processo;

b) há a saída por erro, que também é voluntária;

c) ocorreu algum erro considerado fatal para a continuidade de execução do processo, neste caso involuntário;

d) quando ocorre o cancelamento de um processo por uma solicitação de outro processo, que também é involuntário (MACHADO; MAIA, 2013).

Faça você mesmo

Assista à videoaula sobre processos e *threads* e compreenda como ocorre esse passo a passo. Disponível em: https://www.youtube.com/watch?v=BBoo8frWvrc.

Os processos são classificados em dois tipos:

• CPU-bound: ocupa mais recursos da unidade central de processamento (UCP), ou seja, passa mais tempo em execução e pronto. Facilmente encontrado em aplicações com maior quantidade de operações de cálculo;

• I/O-bound: este processo passa a maior parte do tempo em estado de espera. Encontrado em aplicações comerciais em que é necessário realizar muitas tarefas de leitura, gravação e processamento.

Pesquise mais

Assista ao vídeo de criação de processos no SOSim. Disponível em: https://www.youtube.com/watch?v=_bMRr_oPBWg.

Além disso, há dois canais para realizar a comunicação entre os processos. Eles são chamados de *foreground* e *background*. Processos *foreground* (primeiro plano)

são aqueles que permitem que o usuário interaja com ele, por exemplo, os de entrada e saída. Já os *background* (segundo plano) são os processos que não permitem a comunicação com o usuário durante o seu processamento (MACHADO; MAIA, 2013).

Pesquise mais

Leia o artigo e saiba mais sobre processos e *threads*. Disponível em: http://www.tecmundo.com.br/9669-o-que-sao-threads-em-um-processador-.htm.

Os processos podem ser independentes (não há vínculo com outros processos), subprocessos ou *threads*. Isso significa que há modos diferentes de implementar a concorrência, subdividindo o código em partes. Um processo (pai) pode gerar outros, que serão chamados de subprocessos (filhos), e esses compartilham quotas de recursos com o processo gerador.

Já o conceito de *thread* foi desenvolvido com o intuito de reduzir o tempo que se leva para criar um novo processo em aplicações concorrentes, bem como o uso de recursos. Isso é possível em função de um processo permitir que sejam criados ao menos um *thread*, o que o torna um processo *monothread* (processo suporta apenas um *thread*) ou *multithread* (um processo suporta a criação de vários *threads*).

Quando falamos em *thread*, quer dizer que um processo, ou os seus subprocessos, estão ocupando o mesmo endereço em memória, reduzindo o tempo de comunicação entre processos. Compartilham os contextos de *software* e de armazenamento, o que configura o contexto de hardware aplicado de maneira independente para cada processo que, por sua vez, é criado de forma que haja um *thread* correspondente a cada um. Com isso, há a otimização do processamento da informação quando controlado por *threads* (MACHADO; MAIA, 2013).

Assimile

A partir do conceito de múltiplos *threads* (*multithread*), é possível projetar e implementar aplicações concorrentes de forma eficiente, pois um processo pode ter partes diferentes do seu código, sendo executadas concorrentemente com um *overhead* menor do que utilizando múltiplos processos. Como os *threads* de um processo compartilham o mesmo espaço de endereçamento, a comunicação entre *threads* não envolve mecanismos lentos de intercomunicação entre processos, aumentando, consequentemente, o desempenho da aplicação (MACHADO; MAIA, 2013, p. 81).

Além disso, quando um processo é criado, é preciso alocar os recursos, o que consome muito tempo de processamento e, se o *thread* permite alocar no mesmo endereço de memória um processo e os seus subprocessos, isso faz com que haja a otimização desse tempo que seria gasto com a criação de outros processos e alocação de recursos para eles, que trabalharão de forma concorrente.

Há outros fatores que podem ser mencionados para se distinguir um processo de um *thread* e evidenciar a importância desses para os sistemas operacionais e o gerenciamento de processos. Dentre eles está o fato de que, para cada processo criado, além da alocação de recursos, a comunicação entre os processos é essencial para a otimização do tempo. Logo, há os seguintes mecanismos que servem para realizar o envio de sinais:

a) pipe: permite o tráfego unidirecional de informações entre processos e utiliza a estrutura de dados *array* com apenas duas posições, que indicam 0 para leitura e 1 para gravação;

b) semáforos: servem para testar e incrementar a sincronização de processos;

c) troca de mensagens: este também pode ser descrito como uma forma de sincronização e comunicação entre processos, uma vez que esses podem estar localizados em outro endereço na memória e, por esse motivo, será mais demorada a comunicação. Com isso, é possível dizer que um *thread* é uma subrotina de um programa que é executado de forma concorrente e assíncrona, portanto, aos processos em execução.

Essas definições de ações de *threads* devem ser estabelecidas no momento do planejamento da arquitetura do sistema operacional e são implementadas por desenvolvedores que associarão cada ação de um *thread* ao respectivo tipo de processo ou comportamento que o sistema deve ter.

Em função dessas características, *threads* também passam pelas mesmas mudanças de estados que os processos. Da mesma forma que há o bloco de controle de processos, há um bloco de controle de *threads* conhecido como TCB (*Thread Control Block*). O TCB é responsável por controlar a prioridade e o estado de execução, além de conter os bits de estado do *thread*.

Reflita

A independência entre os conceitos de processo e *thread* permite separar a unidade de alocação de recursos da unidade de escalonamento, que em ambientes *monothread* estão fortemente relacionadas. Em um ambiente *multithread*, a unidade de alocação de recursos é o processo onde todos os seus *threads* compartilham o espaço de endereçamento, descritores de arquivos e dispositivos de E/S (MACHADO; MAIA, 2013, p. 86).

Quando se implementa um *thread*, é preciso saber para qual arquitetura está sendo desenvolvido e como se dará a sua implementação. Nesse sentido, deve ser considerado que um *thread* influencia em desempenho de máquina, processos e concorrência. Desse modo, *threads* podem ser implementados em bibliotecas externas ao kernel, ou seja, no modo usuário, ou ainda, pelo próprio núcleo do sistema (modo kernel) e, também, por ambos os modos, chamado de modo híbrido.

Reflita

Talvez um dos maiores problemas na implementação de TMU (*threads* em modo usuário) seja o tratamento individual de sinais. Como o sistema reconhece apenas processos e não *threads*, os sinais enviados para um processo devem ser reconhecidos e encaminhados a cada *thread* para tratamento. No caso de recebimento de interrupções de *clock*, fundamental para a implementação do tempo compartilhado, esta limitação é crítica (MACHADO; MAIA, 2013, p. 90).

Lembre-se de que o sistema operacional, justamente por ser um *software*, precisa que todas as rotinas que executará sejam especificadas detalhadamente, pois interfere no funcionamento da máquina e até mesmo nos sistemas que interagem com ele. Nesse sentido, quando se fala de bibliotecas de rotinas externas ao núcleo, estamos falando de rotinas que deverão ser executadas quando um *thread* for criado ou mesmo um processo. Logo, tamanha a sua complexidade, a proporção de sistemas operacionais disponíveis no mercado, com relação à quantidade de aplicações e programas, sempre será muito distinta, pois um programa gerencia funções específicas de um processo de negócios. Um aplicativo também oferece funcionalidades para facilitar tarefas do cotidiano profissional. Já os sistemas operacionais precisam gerenciar os recursos e as chamadas de sistema necessárias ao bom funcionamento da máquina, *softwares* e dispositivos de entrada e saída.

Vocabulário

Overhead: excesso em tempo de processamento ou armazenamento.

Escalonador: é um serviço do sistema operacional que tem a função de determinar qual processo será liberado para execução de acordo com o seu nível de prioridade.

Sem medo de errar

Para que você possa levantar os dados de configurações que a clínica médica deve ter em seu servidor e estações de trabalho para instalar e implementar o ERP em sua

rotina de trabalho, a sugestão fica em:

a) identificar o *software* de gestão integrada e quais são as suas configurações básicas ou padrão;

b) verificar quais são as especificações e se são compatíveis com as estações de trabalho;

c) levantar dados de cache, *clock* e núcleos/*threads*;

d) identificar quais são as chamadas de sistema para o gerenciamento de um dos sistemas operacionais identificados.

Tabela 2.1 | Configurações servidor e estações de trabalho

Recursos	Servidor	Estações de trabalho
Processador	Intel® Xeon® Processor E7-8830	Intel® Core™ M vPro™
Sistema Operacional	Linux CentOS 6.5	Windows XP SP3, Seven, 8 e 8.1.
Cache	24 MB	4MB
Velocidade do Clock	2.13 GHz	900 MHz
Núcleos/ Threads	8/16	2/4

Fonte: Elaborada pelo autor (2015).

Tendo em vista que você agora precisa listar as chamadas de sistema, escolhemos o Linux para a sua apresentação. Observe no quadro a seguir como acontece o gerenciamento de processos neste sistema.

Quadro 2.2 | Tipos de chamadas de sistema em Linux

Chamada de sistema	Descrição
pid = fork( )	Cria um processo filho idêntico ao pai
pid = waitpid(pid, &statloc, opts)	Espera o processo filho terminar
s = execve(name, argv, envp)	Substitui a imagem da memória de um processo
exit(status)	Termina a execução de um processo e retorna o status
s = sigaction(sig, &act, &oldact)	Define a ação a ser tomada nos sinais
s = sigreturn(&context)	Retorna de um sinal
s = sigprocmask(how, &set, &old)	Examina ou modifica a máscara do sinal
s = sigpending(set)	Obtém o conjunto de sinais bloqueados
s = sigsuspend(sigmask)	Substitui a máscara de sinal e suspende o processo
s = kill(pid, sig)	Envia um sinal para um processo
residual = alarm(seconds)	Ajusta o relógio do alarme
s = pause( )	Suspende o chamador até o próximo sinal

Fonte: Tanembaum (2009).

Atenção!

Acesse o *site* e veja mais especificações de processadores: http://ark.intel.com/pt-br/products/53677/Intel-Xeon-Processor-E7-8830-24M-Cache-2_13-GHz-6_40-GTs-Intel-QPI.

Lembre-se

Em um ambiente *multithread*, ou seja, com múltiplos *threads*, não existe a ideia de programas associados a processos, mas, sim, a *threads*. O processo neste modo tem pelo menos um *thread* de execução, mas pode compartilhar o seu espaço de endereçamento com inúmeros outros *threads* (MACHADO; MAIA, 2013, p. 84).

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com as de seus colegas.

Introdução a processos: o modelo, criação, término, hierarquia, estados, implementação e threads	
1. Competência de fundamentos de área	O aluno deverá ser capaz de identificar quais são as principais funções de um sistema operacional, bem como ter conhecimento sobre a sua gerência e como se dá o compartilhamento de recursos.
2. Objetivos de aprendizagem	Conhecer a evolução dos sistemas operacionais e suas respectivas especificidades; conhecer e saber identificar os principais processos e como ocorre o compartilhamento de recursos; conhecer como se dá a gerência de processos e de armazenamento de arquivos; conhecer e saber gerenciar os dispositivos de entrada e saída.
3. Conteúdos relacionados	Introdução a processos: o modelo, criação, término, hierarquia, estados, implementação e *threads*.
4. Descrição da SP	Considere que você precise implementar *threads* em um processo de carga automática no sistema de lançamento de notas, a fim de otimizá-lo. No entanto, além do procedimento implementado, você precisa desenvolver um programa que exibe uma mensagem de confirmação de criação do *threads*. Nesse caso, a recomendação é que você consulte o material de referência bibliográfica básica da disciplina e compreenda o processo relacionado, pois esse programa que cria *threads* será desenvolvido em uma linguagem de programação.

	Veja qual foi a solução apresentada pelos autores Machado e Maia para resolver o problema. Com isso, entenda e explique o procedimento de implantação de um *thread*.

5. Resolução da SP	O programa a seguir cria três *threads* a partir de uma classe denominada Loop. Veja no exemplo em Java como seria essa implementação.
	import java.util.*;
	public class CriaThreads{
	public static void main (String [] args) {
	int i, n = 3;
	for (i =1; i <= n; i++){
	Loop loop = new Loop (i);
	Thread t = new Thread (loop) ;
	t.start ();
	System.out.println("Thread " + i + "criado");
	}
	}
	}
	Class Loop implemente Runnable {
	int j;
	public Loop (int i) {
	j = i;
	}
	public void run (){
	Random random = new Random();
	while (true) {
	System.out.printl("Thread " + j + "executado");
	try {
	Thread.sleep (random.nextInt (5000));
	}
	catch (InterruptedException iex){}
	}
	}
	}
	(MACHADO; MAIA, 2013, p. 88)

Lembre-se

A utilização de processos independentes e subprocessos permite dividir uma aplicação em partes que podem trabalhar de forma concorrente. Um exemplo do uso de concorrência pode ser encontrado nas aplicações com interface gráfica, como em um *software* de gerenciamento de *e-mails*. Neste ambiente um usuário pode estar lendo suas mensagens antigas, ao mesmo tempo que pode enviar e receber novas mensagens (MACHADO; MAIA, 2013, p. 83).

Faça você mesmo

Elabore um relatório com o que compreendeu do mecanismo de um

thread. Diferencie *Job*, Processo e *Thread*. Em uma atividade em equipes, discutam sobre o objeto implantado, o programa CriaThreads e tente compreender esse processo que considera tantos aspectos fundamentais de eficiência de sua máquina.

Faça valer a pena!

1. Leias as afirmações e assinale a alternativa correspondente:

I – O *system call*, que pode ser explícito ou implícito.

II – No explícito, há uma instrução de qual chamada deverá ser executada no próprio programa.

III – No implícito, a instrução da chamada que deverá ser executada está apenas no kernel.

a) V-V-V.

b) F-F-F.

c) V-V-F.

d) V-F-V.

e) F-V-V.

2. Assinale a alternativa que contém a definição de arquitetura de Kernel Monolítica (MS- DOS e Unix):

a) São processos independentes e subprocessos.

b) Serve para ler mensagens antigas dos processos.

c) Realiza carga automática no sistema.

d) Refere-se a módulos executados separadamente, mas que compõem um único executável.

e) É um serviço de comunicação do sistema operacional com o usuário diretamente.

3. Complete as lacunas da frase com as palavras disponíveis na alternativa correspondente:

"No _____, as informações que o sistema operacional deve

controlar referem-se à quantidade de arquivos que poderão ser abertos concomitantemente, quais processos detêm prioridade de execução, tamanho do buffer de E/S, por exemplo".

a) Contexto de *hardware*.

b) Contexto de armazenamento.

c) Contexto de *software*.

d) Modo kernel.

e) Modo híbrido.

4. Cite e descreva os estados de processos:

5. Descreva ambiente *monothread* e *multithread*:

6. De acordo com as afirmações a seguir, assinale a alternativa que representa os respectivos conceitos de *threads*:

I – *Threads* podem ser implementados em bibliotecas externas ao kernel.

II – *Threads* podem ser implementados pelo próprio núcleo do sistema.

III – *Threads* podem ser implementados por ambos modos.

a) Modo usuário, modo kernel, modo híbrido.

b) Modo kernel, modo usuário, modo híbrido.

c) Modo kernel, modo híbrido, modo usuário.

d) Modo híbrido, modo kernel, modo usuário.

e) Modo híbrido, modo usuário, modo kernel.

7. Dada a tabela a seguir de arquitetura de *threads*, assinale a alternativa que contém os seus respectivos modelos (modos de *threads*), de forma a completar a lacuna da tabela:

Ambientes	Arquitetura
Distributed Computing Environment (DCE)	
Microsoft Windows 2000	
Sun Solaris versão 2	

a) Modo usuário, modo kernel, modo híbrido.
b) Modo kernel, modo híbrido, modo usuário.
c) Modo híbrido, modo usuário, modo kernel.
d) Modo usuário, modo híbrido, modo kernel.
e) Modo híbrido, modo kernel, modo usuário.

Seção 2.2

Comunicação entre processos e problemas clássicos de comunicação entre processos

Diálogo aberto

O sistema operacional se comunica com o usuário de três formas: através de procedimentos próprios do sistema, por meio da interação com os aplicativos ou, ainda, através das linguagens de comando. Cada um deles tem o seu respectivo acesso e armazenamento de dados reservado em memória e, se um arquivo for compartilhado, por exemplo, será preciso garantir a veracidade e precisão dessas informações.

Por esse motivo, o acesso às informações deve estabelecer qual é o tipo de comunicação que está acontecendo e se é em modo usuário ou em modo kernel. Para identificar qual deles deverá ser acionado, o SO recebe o *status* daquela situação que é definido por uma sequência de bits de identificação (ID) no registrador responsável por essa operação. Quando falamos que um processo está acontecendo em modo usuário, isso quer dizer que apenas instruções chamadas não privilegiadas poderão ser executadas e, por isso, uma quantidade menor de instruções a executar. Já quando se trata de um processo que será executado no modo kernel, o sistema operacional tem acesso irrestrito às instruções do processador. Entenda que informações não privilegiadas são aquelas que não oferecem risco ao sistema e, privilegiadas, refere-se às instruções que podem interferir no funcionamento do kernel (MACHADO; MAIA, 2013).

Uma das funcionalidades do sistema de gestão integrada que será implementado para a clínica médica mencionada na seção de autoestudos 2.1 deverá trabalhar com um princípio muito comum em sincronização de processos, que é fundamentado no algoritmo de Dijkstra, que estudaremos a seguir.

Desse modo, a comunicação que será estabelecida com o sistema operacional será por meio da aplicação. Nesse caso, você precisa considerar que a sincronização dos processos é baseada no princípio proposto por Dijkstra, e explicá-lo, passo a passo. Com isso, o seu cliente poderá identificar de que forma o sistema poderá alternar entre as mais variadas tarefas que deverá gerenciar. Para facilitar a compreensão, é usado o problema do filósofo, e faremos a respectiva associação aos processos e à sua sincronização.

Pensando nessa possibilidade, uma das funcionalidades do *software* de gestão integrada será a de identificar, a partir da localização do cliente final, um consultório mais próximo de acordo com a especialidade que ele deseja e o horário mais próximo para que realize uma ligação direta com o consultório, facilitando a busca e otimizando o tempo gasto pelo cliente, que, nesse caso, é o paciente. Agora, vamos compreender como esse processo é realizado. Então, bons estudos, pesquisas e práticas!

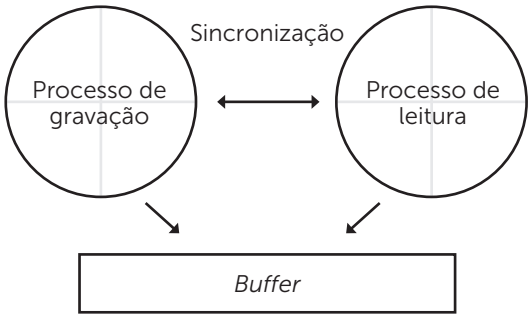
Não pode faltar

Em ambientes computacionais, os sistemas operacionais classificados como multiprogramáveis trouxeram a possibilidade de se estabelecer a concorrência durante o processamento de dados. Por esse motivo, você precisa compreender como acontece a comunicação e sincronização entre os processos que precisam ser executados. De que forma, afinal, pode ser feita? Nesse sentido, você precisa, primeiramente, compreender o que é se entende por sincronização de processos e por que isso pode configurar a comunicação entre eles. O exemplo a seguir ilustra essas afirmações. Confira a seguir na Figura 2.2.

Assimile

A Figura 2.2 indica como acontece a troca de informações para operações de gravação e leitura entre processos concorrentes, em que há o compartilhamento do *buffer*, que armazenará temporariamente as informações para que sejam acessadas de forma mais rápida para processamento. A gravação ocorre apenas se o *buffer* estiver vazio, e, assim também, a leitura dos dados acontece apenas se houver dados para leitura. Nesse sentido, observe como ocorre a sincronização de leitura e gravação:

Figura 2.2 | Sincronização de leitura e gravação de processos



Fonte: Adaptado de Machado e Maia (2013, p. 94).

Para realizar a sincronização entre os processos, são acionados o que chamamos de mecanismos de sincronização. Esses visam garantir a integridade e confiabilidade das ações de sistema.

As primeiras especificações de concorrência, ou seja, as precursoras desse modelo de controle de comunicação entre processos foram desenvolvidas por Conway (1963), Dennis e Van Horn (1966). Trazem a notação dos comandos FORK e JOIN. O comando FORK tem por função realizar uma chamada do processo que está no *buffer* para ser executado e, a partir da sua identificação, o associa ao seu subprocesso, ou seja, ao processo filho. FORK também assume a função de acompanhamento de execução desse processo (MACHADO; MAIA, 2013).

Assim como FORK tem a finalidade de criar processos, o comando JOIN tem o objetivo de sincronizar os processos criados pelo FORK. Isso significa que, enquanto há um processo.

"X" em execução, por exemplo, e o seu respectivo subprocesso, é possível que pelo comando FORK tenha sido criado ainda um novo processo "Z" e a alocação de recursos precisa ser gerenciada e os respectivos *status* dos processos também. Porém, quando isso acontece, o comando JOIN permitirá a execução de "X" apenas após o encerramento da execução do processo "Z" (MACHADO; MAIA, 2013).

Assim como FORK e JOIN, outro exemplo de notação de controle de concorrência e comunicação entre processos são os comandos PARBEGIN (antes chamado de COBEGIN, que tem por função criar um processo de forma aleatória) e PAREND (antes chamado de COEND, cria um novo processo apenas após o encerramento das execuções dos processos anteriores), que seguem o modelo introduzido pelo algoritmo de Dijkstra. Podem ser implementados por comandos simples de chamada a procedimentos ou ainda de atribuição.

Exemplificando

Machado e Maia (2013) trazem o exemplo de implementação dos comandos COBEGIN E COEND para calcular uma expressão aritmética. Então, observe que o exemplo traz a execução de todas as prioridades matemáticas da expressão e em seguida, faz o cálculo da expressão completa com os respectivos resultados obtidos na execução de cada uma das instruções. Suponha que a expressão aritmética seja "X:= SQRT (1024) + (35.4 * 0.23) – (302/7). O exemplo deixa claro como ocorre a sincronização de processos com uso de especificação de concorrência com PARBEGIN e PAREND.

De acordo com as regras matemáticas e de execução, consideradas pelos sistemas operacionais, as prioridades matemáticas são preservadas,

então a lógica é a seguinte: acompanhe no algoritmo os comandos mencionados:

```
PROGRAM Expressao;
    VAR X, Temp1, Tem2, Temp3: REAL;
BEGIN
     PARBEGIN
          Temp1:= SQRT (1024);
          Temp2:= 35.4 * 0.23;
          Temp3:= 302 / 7;
      PAREND
      X:= SQRT (1024) + (35.4 * 0.23) – (302/7)
      WRITELN ('x = ', X);
END.
```

Fonte: Machado e Maia (2013, p. 95).

Com os sistemas operacionais multiprogramáveis, trabalhar com processos concorrentes pode apresentar alguns problemas quando se trata do compartilhamento de recursos. Os principais mencionados por Machado e Maia (2013) estão correlacionados com o compartilhamento de um arquivo em disco e o compartilhamento de uma variável na memória principal entre dois processos. Para corrigir esses erros, são inseridos mecanismos de controle que possibilitam minimizar problemas de execução de processos concorrentes, estabelecendo condições de corrida, também conhecidas como *race conditions*.

Para tratar os erros, são propostos alguns algoritmos que reduzem a sua probabilidade de ocorrência. Dentre eles, podem ser citados:

a) Exclusão mútua (*mutual exclusion*): esse mecanismo impede que dois ou mais processos sejam executados compartilhando o mesmo recurso simultaneamente. Sendo assim, os processos precisam esperar o encerramento da execução para que possam utilizá- lo. Esse método evita que outro processo acesse a região crítica do programa; com isso, protocolos de entrada e saída são implementados para garantir que essa verificação seja realizada. A comunicação deve acontecer de forma sincronizada para que envie o *status* de encerramento de um processo e informe quando o outro se inicia, no entanto, isso ocorrerá apenas após a confirmação de encerramento do processo anterior.

Quando se trata de exclusão mútua, essa pode ser implementada com mecanismos de comunicação com o *hardware* ou com o *software*. Em soluções de *hardware*, podemos citar as de desabilitação de interrupções, em que todas são impedidas de entrar em execução ao solicitar a entrada na região crítica do programa e o *test-and-ser*. No entanto, um problema que pode ser elencado quanto à desabilitação de interrupções é o risco de não acontecer a habilitação da instrução de interrupção após a mudança de *status* do processo (MACHADO; MAIA, 2013) e ele permanecer em espera ou não sair do modo de execução.

Reflita

Em sistemas com múltiplos processadores, essa solução torna-se ineficiente devido ao tempo de programação quando um processador sinaliza aos demais que as interrupções devem ser habilitadas ou desabilitadas. Outra consideração é que o mecanismo de clock do sistema é implementado através de interrupções, devendo esta solução ser utilizada com bastante critério (MACHADO; MAIA, 2013, p. 99).

Além dessa, a outra solução para o erro oriundo da exclusão mútua é a implantação da instrução *test-and-set*, que é basicamente uma instrução de máquina que trata uma exceção. Ela faz a leitura da variável, armazena o seu conteúdo em outra área e atribui um novo valor à variável então vazia. Não há interrupção durante a execução.

Sob o ponto de vista de comunicação com o *software*, são apresentados como solução quatro algoritmos de controle para os problemas relacionados aos processos de exclusão mútua (MACHADO; MAIA, 2013). São eles:

a) primeiro algoritmo: os processos alternam a execução na região crítica do programa através de uma repetição infinita, ou seja, acessa o recurso diversas vezes a fim de verificar e, se houver solicitação de execução, altera o *status* e termina o processo alternado de forma independente. Os dois processos utilizam a mesma variável global que indicam se este poderá acessar a região crítica ou não (MACHADO; MAIA, 2013);

b) segundo algoritmo: vem corrigir o problema do primeiro algoritmo que utiliza a mesma variável global e insere uma variável para cada processo criado. Cada variável é responsável por informar se o processo está ou não na região crítica. Um ponto de atenção nesse algoritmo é que ele não trabalha exclusivamente com a alternação dos processos, mas pode bloqueá-lo por tempo indeterminado, o que não garante a exclusão mútua e acarreta outro erro;

c) terceiro algoritmo: vem corrigir o problema apresentado no segundo algoritmo inserindo as instruções de atribuição de valores às variáveis antes do bloco de repetição

de verificação de disponibilidade de alocação de recurso, o que, então, permite garantir a exclusão mútua. No entanto, se dois processos em execução alteram o conteúdo das variáveis que foram criadas (segundo algoritmo), antes de iniciar execução, corre-se o risco de não acontecer o acesso à região crítica do programa, pois o kernel pode interpretar que o recurso já foi alocado e não há a execução dos processos. Os processos são executados de forma independente (MACHADO; MAIA, 2013);

d) quarto algoritmo: vem sanar o problema de comunicação de estados entre os processos, alterando o *status* da variável antes de acessar a região crítica, então existe também a possibilidade de o *status* das variáveis alterar para falso e os processos não entrarem em execução.

Para que você possa visualizar exemplos desses algoritmos, acesse o material disponível em: http://www.inf.ufrgs.br/~johann/sisop2/aula04.algorithms.2.pdf.

Observe que, mesmo com os algoritmos de correção que foram apresentados, ainda existe a possibilidade de ocorrer um erro de comunicação, ou seja, de identificação do *status*, seja em função do tipo de variável envolvida ou pela desabilitação das interrupções. Então, alguns outros cientistas e pesquisadores desenvolveram algoritmos para sanar essas lacunas e garantir que aconteça a exclusão mútua de processos sem gerar os erros mencionados. Dentre eles, podem ser citados o algoritmo de Dekker, em 1990, e Peterson G. L. (que definiram um algoritmo para tratar da exclusão mútua entre dois processos), que apresenta a possibilidade de exclusão mútua entre N processos. O algoritmo de Peterson resolve o problema da quantidade de vezes que o comando terá de repetir (indefinidamente ou *busy wait*, espera ocupada), pois insere a verificação antes de iniciar efetivamente a alternância dos processos de forma a garantir que não aconteça o bloqueio indefinido do processo (MACHADO; MAIA, 2013).

Mas, além da exclusão mútua, existem outros mecanismos. Confira a seguir os demais e suas respectivas descrições:

a) sincronização condicional: como o próprio nome diz, implementa a sincronização de execução dos processos associada a uma verificação condicional de acesso à região crítica. O exemplo mencionado de leitura e gravação de dados no *buffer* ilustra bem esse processo, pois um somente será executado quando o outro estiver com *status* de encerrado, para, então, mesmo que compartilhem recursos de forma concorrente, sincronizá-los, evitando perdas de dados e atribuição de *status* de alocação indevido, como já mencionado em exclusão mútua;

b) semáforos: este mecanismo foi implementado por Dijkstra em 1965, permite a prática da exclusão mútua com a inserção de condição para acesso à região crítica e execução dos processos. Utiliza as instruções DOWN, originalmente P de proeberen ou teste; UP, originalmente V de verhogen, que significa incremento. DOWN e UP são instruções que não permitem interrupção. UP realiza o incremento ao valor do

semáforo e DOWN tem por função deixar o processo em estado de espera quando o semáforo estiver com o valor 0. Há dois tipos de semáforos: contadores que recebem valor positivo inclusive zero (0) e exclusão mútua com semáforos, que permitem apenas valores binários, ou seja, 0 ou 1 para indicar o *status* do processo (MACHADO; MAIA, 2013).

c) monitores: propostos por Brinch Hansen em 1972. São implementados pelo compilador e, por esse motivo, são considerados estruturados. Realiza um procedimento com variáveis encapsuladas e aplica a exclusão mútua, porém, apenas um processo pode executar as instruções de um monitor por vez. Os monitores trabalham com a estrutura de fila para gerenciar as solicitações de acesso ao recurso. Pode ser chamada a execução apenas a partir do programa em que foi declarado (comportamento próprio do encapsulamento). Há um algoritmo de exclusão mútua com o uso de monitores.

d) troca de mensagens: não necessita de variáveis compartilhadas, mas estabelece um canal de comunicação em que seja possível enviar (SEND) e receber (RECEIVE) mensagens para a sincronização de execução dos processos. Há a direta e a indireta. A primeira estabelece um endereço explícito para o processo receptor ou emissor. Na indireta, as mensagens podem ser alocadas pelo processo transmissor e retiradas pelo receptor; vários processos podem estar associados.

e) *deadlock*: ocorre essa situação quando um processo está aguardando por tempo indeterminado a alocação de um recurso ou um evento que não ocorrerá em função da alocação dinâmica de recursos que trabalham com concorrência. Mas como identificar essa situação e não prever? Para que não haja *deadlock*, é preciso que aconteça simultaneamente a exclusão mútua, a espera por recursos, a não liberação de um processo que aguarda um recurso de forma concorrente com outros processos, ou seja, não preempção e, ainda, precisa ocorrer o que se chama espera circular, em que um processo deve esperar a execução de um processo terminar para que utilize aquele determinado recurso (MACHADO; MAIA, 2013). Deve haver a prevenção, a detecção e a correção do *deadlock*.

Faça você mesmo

a) investigue como são os algoritmos das soluções de *hardware*: desabilitação de interrupções e instrução *test-and-set*;

b) teste os quatro algoritmos de exclusão mútua;

c) em grupos de pesquisa, organizem-se para simular os algoritmos de sincronização condicional;

d) dê preferência para as referências bibliográficas da disciplina, conforme indicado no material.

Pesquise mais

O vídeo é uma animação sobre o funcionamento do computador e de como ocorre a comunicação entre dispositivos e, também, ilustra um pouco a importância dessa, e, ainda, a necessidade de eficiência de processamento e da organização de arquivos nesse processo. Disponível em: https://www.youtube.com/watch?v=jH5gOJvvCSQ.

Veja também material explicativo dos processos FORK e JOIN sobre concorrência de processos. Disponível em: http://slideplayer.com.br/slide/47099/.

Sem medo de errar

Agora, considerando que a aplicação (*software* de gestão integrada) trabalha de forma concorrente e esse é o tipo de comunicação que se estabelece, explique de que forma o algoritmo de Dijkstra pode auxiliar na alocação de recursos para a comunicação e sincronização de processos, oriundos da aplicação.

Dessa forma, lembre-se de que a tarefa agora é localizar o endereço de solicitação de informações *on-line* do sistema de gestão da clínica médica e informar, a partir disso, os consultórios e clínicas mais próximos do usuário, de acordo com o seu plano de saúde, além de oferecer o serviço de discagem direta para a realização do agendamento. Pensando nisso, explique o mecanismo de funcionamento do algoritmo proposto. Apresente o mecanismo e materiais de apoio com as suas possibilidades de aplicação.

Sendo assim, é possível associar a necessidade de processamento das solicitações dos usuários de forma concorrente. Com isso, vamos estudar o processo de Dijkstra que implanta o uso do conceito de semáforo para a sincronização dos processos. Nesse sentido, o exemplo que se enquadra nessas condições é o do problema dos filósofos. Nós consideraremos cada ação que eles precisam tomar como sendo os processos que o sistema deverá executar, implementando uma solução que previna a ocorrência de *deadlocks*. O problema: há uma mesa com cinco pratos e cinco garfos (os recursos), em que os filósofos podem sentar, comer e pensar (processos). Quando um filósofo para de pensar e deseja comer (mudança de estado do processo), ele precisa usar mais recursos: dois garfos à direita e à esquerda. Associando às ações do programa, vamos considerar que os recursos são aqueles que poderão ser compartilhados no servidor da aplicação e gerenciados pelo sistema operacional. As solicitações de clínicas e consultórios próximos ao cliente (paciente ou usuário) representam os processos. Para cada solicitação, será preciso alocar os recursos e com isso verificar, de acordo com o algoritmo proposto, a sua disponibilidade. Vamos ver como fica a solução proposta por Dijkstra em Machado e Maia (2013, p. 111):

Considere as seguintes regras: apenas quatro filósofos podem sentar-se à mesa simultaneamente (haverá uma determinação de quantidade de usuários e o sistema deve prever esse cenário e tratar de acordo com a demanda, ser desenvolvido o procedimento considerando este critério). Um filósofo só pode usar um garfo se ele estiver disponível. Ou seja, a solicitação do usuário será executada apenas se houver recurso para alocar que esteja de fato com o *status* de disponível. Um filósofo só pode usar primeiro o garfo da direita e depois o da esquerda (é estabelecida uma regra para priorizar o acesso à região crítica do programa para que aconteça a execução do processo). Veja como é o seu algoritmo:

```
PROGRAM Filosofo_2;
    VAR
        Garfos: ARRAY [0..4] of Semaforo:= 1;
        Lugares : Semaforo := 4;
        I :  INTEGER;
         PROCEDURE Filosofo (I: INTEGER);
         BEGIN
                REPEAT
                        Pensando;
                        DOWN (Lugares);
                        DOWN (Garfos [I]);
                DOWN (Garfos [(I +1) MOD 5]);
```

```
Comendo;
                UP (Garfos [I]);
                UP (Garfos [(I +1) MOD 5]);
                UP (Lugares);
          UNTIL false;
END;
BEGIN
        PARBEGIN
                FOR I:= 0 TO 4 DO
                        Filosofo (I);
        PAREND;
END.
```

Atenção!

Assista ao vídeo que explica a lógica do algoritmo de Dijkstra e compreenda o mecanismo envolvido nesse tipo de sincronização de processos. Disponível em: https://www.youtube.com/watch?v=J4TZgD1As0Q.

Lembre-se

O conceito de semáforos foi proposto por E. W. Dijkstra em 1965, sendo apresentado como um mecanismo de sincronização que permitia implementar, de forma simples, a exclusão mútua e a sincronização condicional entre processos. De fato, o uso de semáforos tornou-se um dos principais mecanismos utilizados em projetos de sistemas operacionais e em aplicações concorrentews (MACHADO; MAIA, 2013, p. 107).

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com as de seus colegas.

Comunicação entre processos e problemas clássicos de comunicação entre processos	
1. Competência de fundamentos de área	O aluno deverá ser capaz de identificar quais são as principais funções de um sistema operacional e ter conhecimento sobre como se dá o compartilhamento de recursos e a sua gerência.
2. Objetivos de aprendizagem	Conhecer a evolução dos sistemas operacionais e suas respectivas especificidades; conhecer e saber identificar os principais processos e como ocorre o compartilhamento de recursos; conhecer como se dá a gerência de processos e de armazenamento de arquivos; conhecer e saber gerenciar os dispositivos de entrada e saída.
3. Conteúdos relacionados	Comunicação entre processos e problemas clássicos de comunicação entre processos.
4. Descrição da SP	De acordo com as necessidades de execução desse mecanismo de comunicação, explique o seu funcionamento e considere este procedimento como uma opção que reduz a possibilidade de ocorrência de erros para o bom funcionamento do software de gestão que pode ser implementado na clínica.

5. Resolução da SP	Há três formas de aplicar a comunicação de processos por troca de mensagens:
	1ª) sincronização de envio, recepção e leitura de processos. O mesmo deve acontecer para o caso do recebimento de mensagens; o processo deve aguardar até que a mensagem esteja pronta para envio (rendevouz);
	2ª) utiliza o tempo de espera do processo de transmissão e permite enviar mensagens para outros destinatários, enquanto o processo de envio não finaliza o tratamento da mensagem;
	3ª) trabalha de forma assíncrona, ou seja, considera buffers para armazenar as informações e também outras formas de controle de envio e recebimento de mensagens.
	Considere o exemplo trazido por Machado e Maia (2013) para realizar a transmissão de mensagens:
	PROGRAM Produtor_consumidor_4;
	PROCEDURE Produtor;
	VAR Msg: Tipo_Msg;
	BEGIN
	REPEAT
	Produz_ Mensagem (Msg);
	SEND (Msg);
	UNTIL false;
	END;
	PROCEDURE Consumidor;
	VAR Msg: Tipo_Msg;
	BEGIN

	REPEAT
	RECEIVE (Msg);
	Consome_Mensagem (Msg);
	UNTIL false;
	END;
	BEGIN
	PARBEGIN
	Produtor;
	Consumidor;
	PAREND;
	END.

Fonte: Machado e Maia (2013, p. 119).

Lembre-se

Troca de mensagens é um mecanismo de comunicação e sincronização entre processos. O sistema operacional possui um subsistema de mensagem que suporta esse mecanismo sem que haja necessidade do uso de variáveis compartilhadas. Para que haja a comunicação entre os processos deve existir um canal de comunicação, podendo esse meio ser um *buffer* ou um *link* de uma rede de computadores (MACHADO; MAIA, 2013, p. 117).

Faça você mesmo

É recomendada, além da leitura, um resumo do material indicado que trata da comunicação entre processos por troca de mensagens e exibe as formas de implementação com *sockets* com e sem conexão. Disponível em: <http://www.oocities.org/walterchagas/process.html>.

Faça valer a pena!

1. Descreva como é o processo de comunicação de processos a notação FORK e JOIN.

2. Assinale a alternativa que contém uma característica da comunicação e sincronização através do procedimento de exclusão mútua:

a) Implementa um procedimento com variáveis encapsuladas e aplica a exclusão mútua.

b) Deve haver a prevenção, a detecção e a correção do *deadlock*.

c) DOWN e UP são instruções que não permitem interrupção.

d) Impede que dois ou mais processos sejam executados compartilhando o mesmo recurso.

e) Implementa a sincronização de execução dos processos associada a uma verificação condicional de acesso à região crítica.

3. Complete: "[...] No mecanismo de comunicação entre processos por _____, não se necessita de variáveis compartilhadas, mas se estabelece um canal de comunicação em que seja possível enviar (_____) e receber (_____) mensagens para a sincronização de execução dos processos. Há a comunicação direta e a indireta".

a) parbegin/ PAREND/ RECEIVE.

b) troca de mensagens/ SEND/ RECEIVE.

c) RECEIVE/ troca de mensagens/ SEND.

d) END/ BEGIN/ deadlock.

e) deadlock/ PAREND/ BEGIN.

4. Assinale V ou F:

I – Monitores: são considerados estruturados. Trabalha com a estrutura de fila para gerenciar as solicitações de acesso ao recurso.

II – *Deadlock*: ocorre essa situação quando um processo está aguardando por tempo indeterminado a alocação de um recurso ou um evento que não ocorrerá em função da alocação dinâmica de recursos.

III – Semáforo: UP realiza o incremento ao valor do semáforo e DOWN tem por função deixar o processo em estado de espera quando o semáforo estiver com o valor 0.

a) V-F-F.

b) F-V-F.

c) F-F-F.

d) V-V-V.

e) V-F-F.

5. Avalie o procedimento a seguir e assinale a alternativa correta:

```
REPEAT
                    Pensando;
                    DOWN (Lugares);
                    DOWN (Garfos [I]);
              DOWN (Garfos [(I +1) MOD 5]);
               Comendo;
                UP (Garfos [I]);
                UP (Garfos [(I +1) MOD 5]);
                UP (Lugares);
        UNTIL false;
```

6. Refere-se a um teste do procedimento de comunicação através de:

a) Monitores.

b) Semáforos.

c) *Deadlock*.

d) Exclusão mútua.

e) *Send*.

7. Analise as afirmações e assinale a alternativa que apresenta a sequência correta no que tange ao mecanismo de exclusão mútua.

I – Os processos alternam a execução na região crítica do programa através de uma repetição infinita.

II – Visa sanar o problema de comunicação de estados entre os processos, alterando o *status* da variável antes de acessar a região crítica.

III – Insere uma variável global para cada processo associado.

a) Segundo algoritmo/primeiro algoritmo/terceiro algoritmo.

b) Quarto algoritmo/quinto algoritmo/terceiro algoritmo.

c) Primeiro algoritmo/quarto algoritmo/segundo algoritmo.

d) Primeiro algoritmo/terceiro algoritmo/segundo algoritmo.

e) Primeiro algoritmo/terceiro algoritmo/quarto algoritmo.

Seção 2.3

Introdução ao escalonamento: conceitos, tipos e escalonamento de *threads*

Diálogo aberto

Olá, aluno! Vamos iniciar os estudos em escalonamento de processos. Afinal, estamos estudando o gerenciamento de processos e threads e aprendemos, na seção anterior, quais são os mecanismos de comunicação e sincronização de processos que os sistemas operacionais utilizam. Agora, precisamos compreender de que forma o sistema operacional seleciona esses processos e a partir de quais critérios.

Então, nesse sentido é que vamos aprender quais são as funções básicas quando o assunto é escalonamento, ou seja, a seleção do processo que será executado a partir do momento em que ele entra em estado de pronto e precisa efetivamente ser executado.

Com isso, algumas características podem ser elencadas, pois são consideradas para realizar a gerência do processador de forma a mantê-lo em exercício, ou seja, trabalhando, a maior parte do tempo possível, enquanto a máquina está em uso.

Sendo assim, sua tarefa é comprovar a eficiência do sistema operacional para realizar o gerenciamento dos processos do sistema de gestão integrado da aplicação da clínica médica de forma a otimizar, através das configurações de escalonamento, a execução dos processos solicitados pelo sistema. Fica a recomendação de que você utilize o simulador SOsim para que consiga realizar essa atividade.

Além dessas especificações, você também poderá verificar como acontece a definição da política de escalonamento em sistemas de tempo compartilhado, por exemplo, além de como acontecem os escalonamentos preemptivos e não preemptivos (MACHADO; MAIA, 2013).

Outras formas de escalonar também serão evidenciadas aqui em seu livro didático, tais como: escalonamento por filas, ou seja, em que o primeiro processo que entra na fila será também o primeiro alocado para processamento, o conhecido FIFO (*first in first out*).

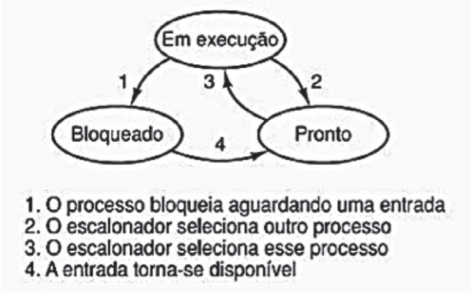
Para tal, fica a sugestão de leitura e estudos de seu material didático, que lhe proporcionará o contato necessário para que desenvolver as competências e habilidades necessárias e aplicar em sua rotina profissional quando necessário! Estude, investigue e pratique!

Desde já desejamos a você bons estudos e práticas!

Não pode faltar

Vamos iniciar retomando quais são os estados que um processo pode assumir. Você se recorda? Pois bem, temos basicamente três tipos de estados de processos: de execução, de pronto e de espera. Um processo fica em espera a partir de sua criação e permanece nesse estado até que esteja com todos os recursos dimensionados e devidamente alocados. Com isso, passa, então, ao estado de pronto. Com isso, ele poderá ser chamado para processamento e, até o término dessa operação, permanecerá no estado de execução. Observe a Figura 2.3, que traz os três estados que um processo pode assumir. Considere o estado "bloqueado" com o mesmo mecanismo e função do estado de "espera", pois precisa que aconteça um evento que determine a mudança de estado para que possa entrar em execução. Essa definição de bloqueado foi mencionada por Andrew Tanembaum (2010):

Figura 2.3 | Estados de um processo



Fonte: Tanembaum (2010, p. 54).

O processo, quando entra em execução, permanece com esse estado até o término do processamento. Quando encerra, é liberado o acesso para outro processo, que deixará o estado de pronto e entrará em execução, e assim por diante. Mas como é possível controlar quais os processos que devem ter prioridade, ou, ainda, quais processos estão na fila para processamento?

Para que isso aconteça, os mecanismos de escalonamento de processos foram desenvolvidos. Alguns critérios de escalonamento são necessários e determinados de acordo com as características do sistema operacional. Dentre os critérios, podemos

elencar a análise de eficiência e utilização do processador. Nesse caso, o recomendado é que o nível de capacidade esteja ocupando, em média, 90% para ser considerado alto, ou seja, com bom potencial de aproveitamento do recurso. Outro elemento que estabelece critério para a definição do escalonamento é o *throughput*. Esse é um indicador que mostra quantos processos foram executados dentro de um intervalo de tempo. É uma medida diretamente proporcional, pois, quanto maior o *throughput*, maior será também a quantidade de tarefas realizadas naquele determinado período (MACHADO; MAIA, 2013).

Tempo de processador é outro critério relevante para a escolha ou determinação do tipo de escalonamento aplicado. Também descrito como Tempo de UCP, esse evidencia justamente o tempo que um processo leva para ser executado e finalizado. Além desse, há o tempo de espera que define o tempo em que um processo fica na fila dos processos em estado de pronto. Outro tempo importante a considerar é o de *turnaround*, que tem a função de apresentar o tempo total que um processo ocupa, desde a sua criação até o seu encerramento, e, por fim, o tempo de resposta (MACHADO; MAIA, 2013). Esse último apresenta o tempo que leva a partir da criação do processo para que este seja atendido pelo sistema e depende muito da velocidade atrelada aos dispositivos de entrada e saída, principalmente quando se trata de aplicações *web*. Imagine se o tempo de resposta for longo? O acesso com certeza ficará comprometido. O tempo de resposta também é importante e deve ser considerado ao estabelecer a política de escalonamento.

Quanto aos tipos de escalonamento vamos estudar:

a) não preemptivos e preemptivos;	b) *first in first out* (FIFO);
c) *shortest job first* (SJF);	d) cooperativo;
e) circular;	f) por prioridades;
g) circular por prioridades;	h) por múltiplas filas;
i) por múltiplas filas com realimentação;	j) políticas de escalonamento em sistemas de tempo compartilhado e de tempo real.

Então, vamos iniciar as respectivas definições. Siga em frente!

A classificação adotada para definir a política de escalonamento em sistemas operacionais deve considerar se trabalhará com o modo não preemptivo ou preemptivo. O escalonamento não preemptivo foi implementado inicialmente para os sistemas tipo *batch* e foi um dos primeiros a ser utilizados em sistemas multiprogramáveis. Todos os recursos do processador ficam dedicados até que todo o processo seja finalizado ou por erro de execução, ou por tempo de processamento. Nesse caso, as instruções do processo já são desenvolvidas de forma a alterar para o estado de espera. Nesse modo não preemptivo, o sistema operacional não gera interrupções (MACHADO; MAIA, 2013).

Já o escalonamento preemptivo permite que interrupções do sistema operacional alterem o estado de execução de um processo para o de pronto, pois pode, em função de uma determinação por prioridade, alocar outro processo para execução. Isso também pode acontecer em sistemas de tempo real e em sistemas em que há o compartilhamento de processadores, balanceando os recursos da CPU entre os processos existentes.

O escalonamento do tipo FIFO (*first in first out*) ou fila, considera o primeiro processo que entra em estado de pronto para ser o primeiro a entrar em estado de execução. Também conhecido como FIFO *scheduling* ou FCFS *scheduling*, agrupa os processos por ordem de chegada em estado de pronto e faz o escalonamento assim que chamados à execução, no caso, quando chegam a ser o primeiro processo da fila, e, com isso, o estado que estava em espera passa ao estado de pronto, e assim por diante. O tempo médio de execução não importa muito neste tipo de escalonamento em função da organização por fila (MACHADO; MAIA, 2013).

Mas, para melhorar as opções e utilizar um tipo de escalonamento que considera o tempo de execução de um processo e não a ordem de chegada na fila, temos a opção do SJF (escalonamento por *shortest job first*). Esse algoritmo seleciona o processo com o menor tempo de execução e este tem a prioridade, saindo do estado de pronto e passando ao de execução, e, então, o escalonamento realiza tal conferência novamente e ordena os processos de acordo com o seu tempo de processamento. Foi muito utilizado para os sistemas do tipo *batch*, pois o tempo de processamento era mensurado com base em análises estatísticas e daí, então, aplicado o algoritmo de escalonamento SJF. Mas ele também é aplicado a sistemas interativos, com a parada de processamento para a realização de algumas operações de entrada e saída e, com isso, repete essas operações até que o processo em execução se encerre efetivamente.

No entanto, quando isso acontece, o sistema operacional já não calcula mais o tempo que o processo levará para finalizar, ou seja, quanto tempo ainda de utilização da CPU precisará. Trabalha apenas com uma previsão de tempo de processamento com base no tempo da última operação de execução (MACHADO; MAIA, 2013). É considerado não preemptivo e uma das vantagens com relação a FIFO é que reduz o tempo total de processamento, o turnaround.

Pesquise mais

Assista ao vídeo que faz a demonstração do mecanismo de escalonamento SJF (*Shortwst-Job-Firts*). Disponível em: https://www.youtube.com/watch?v=OVWc4wDX1u4.

Baixe o simulador SOsim no *link*: http://www.training.com.br/sosim/.

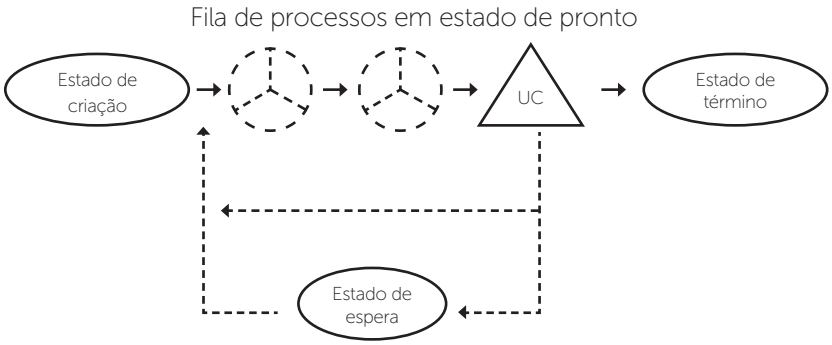
Além desses escalonamentos, temos o conhecido como cooperativo. Esse escalonamento basicamente considera que um processo em execução, de forma voluntária, pode ceder o recurso para outro processo de forma cooperativa, através da verificação de uma fila de *status* dos processos em estado de pronto.

Outro importante que precisamos estudar é o escalonamento circular. Ele trabalha com fila (FIFO), e isso significa que o primeiro processo em estado de pronto será também o primeiro a ser enviado à execução, de forma a permanecer neste estado até que finde o seu processamento.

Assimile

Vamos, agora, compreender como é o mecanismo de escalonamento circular. Observe a Figura 2.4 e como acontece a alocação do processo para execução a partir do momento em que entra em estado de pronto.

Figura 2.4 | Escalonamento circular



Fonte: Adaptado de Machado e Maia (2013, p. 133).

Além dessa forma de escalonar e identificar a ocorrência, podem ser elencados alguns fatores que influenciam na mudança de estado. Por exemplo, um processo pode passar do estado de execução para o estado de espera quando excede o tempo de entrada em processamento e necessita de uma entrada preemptiva para retornar a fila. Outro fator que pode acarretar a mudança de estado no escalonamento circular é a entrada em estado de espera de forma voluntária, ou por razão não identificada (MACHADO; MAIA, 2013).

Confira, a seguir, a apresentação dos demais tipos de escalonamento existentes.

Até aqui você estudou alguns tipos de escalonamento; conheça agora o escalonamento por prioridades. Esse tipo de escalonamento precisa que seja determinada qual é a prioridade de execução de um processo e, então, o que for

escolhido para ser executado será aquele que tem maior prioridade enquanto estiver no estado de pronto. Conheça alguns dos critérios utilizados para essa verificação:

a) se os valores das prioridades dos processos são iguais, então esses serão ordenados em fila (FIFO). Esse tipo de processo não pode ser preemptivo, ocorrendo por uma interrupção de tempo, ou de clock, de forma que o sistema identifique novamente os estados dos processos e quais estão em "pronto";

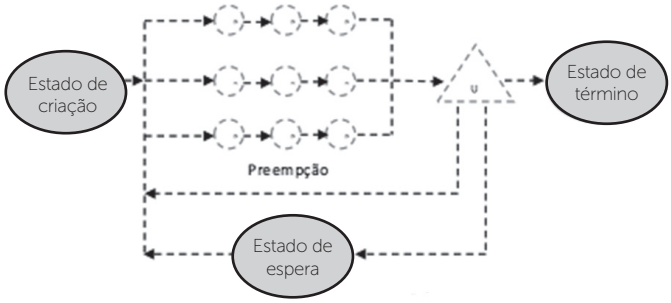
b) se nessa verificação for identificado um processo que contenha um nível de prioridade maior, o processo em execução entra voluntariamente em estado de espera até que o sistema encontre outro processo com maior prioridade;

c) pode ocorrer a mudança voluntária do estado execução para o de espera, caso haja perda em eficiência do processador.

Exemplificando

Observe qual é a lógica do mecanismo de escalonamento por prioridades ilustrado na Figura 2.5:

Figura 2.5 | Escalonamento por prioridade



Fonte: Adaptado de Machado e Maia (2013, p. 135).

Na figura, há a preempção de processos, caso esse passe do estado de execução para o de pronto, em função do nível de prioridade estabelecido para o tipo de processo. Com isso, ele deverá retornar à fila em estado de espera e seguir o fluxo para, então, ser enviado novamente à execução.

Outro tipo de escalonamento que considera a priorização de processos é o circular com prioridades. O algoritmo desse escalonamento associa a cada processo uma fatia de tempo e a sua ordem de prioridade com relação aos demais. Com isso, ou o processo será iniciado e encerrado sem interrupções, ou passará voluntariamente para o estado de espera. Além disso, o escalonamento circular permite que o administrador do sistema configure as prioridades dos processos I/O- bound, ou seja, que vêm de

operações de entrada e saída, com valores maiores do que os processos provenientes de operações de CPU (MACHADO; MAIA, 2013).

Nesse sentido, existem dois tipos de escalonamento circular: o dinâmico e o estático. As prioridades de processos no escalonamento estático são definidas no contexto de *software* e não são alteradas enquanto existir esse procedimento estabelecido no sistema. Já no escalonamento circular com prioridades dinâmicas, o administrador do sistema pode interferir manualmente nas configurações das prioridades e alterá-las de acordo com a política de escalonamento que foi estabelecida. No escalonamento dinâmico, o próprio sistema operacional pode intervir no nível de prioridade do processo para que este não demore muito tempo em execução, no caso, quando se tratam dos processos de E/S e, com isso, há um ganho em escalonamento que pode compensar a espera sem prejudicar os processos do tipo CPU-bound.

Mas existem outros escalonamentos que visam otimizar ainda mais a organização dos processos e o seu tempo de execução. Podemos citar o escalonamento por múltiplas filas, que, como o próprio nome diz, trabalha com a formação de várias filas que são tratadas de acordo com a importância da aplicação para o sistema operacional ou mesmo a quantidade e a área da memória que será alocada, ou seja, a prioridade não está associada ao processo e sim à fila de processos.

Reflita

Como processos possuem características de processamento distintas, é difícil que um único mecanismo de escalonamento seja adequado a todos. A principal vantagem de múltiplas filas é a possibilidade da convivência de mecanismos de escalonamento distintos em um mesmo sistema operacional. Cada fila possui um mecanismo próprio, permitindo que alguns processos sejam escalonados pelo mecanismo FIFO, enquanto outros pelo circular (MACHADO; MAIA, 2013, p. 137).

Além desse, ainda há um com o mesmo princípio, mas que implementa o escalonamento por múltiplas filas com realimentação, o que infere em um processo ser alternado de fila de acordo com a prioridade da fila. Esse é um controle que o próprio sistema operacional realiza para conseguir dinamizar o processamento, com base no comportamento do processo. Para que isso aconteça de forma ordenada, esse escalonamento utiliza o mecanismo de FIFO com a característica de controle por fatia de tempo, sendo assim, quanto maior a prioridade da fila, menor a fatia de tempo que o processo levará para ser encerrado, é inversamente proporcional.

Além dos tipos de escalonamento existentes nos sistemas operacionais, é preciso que se estabeleça uma política de escalonamento que pode ser para sistemas de tempo compartilhado ou de tempo real. Vamos falar agora da política de escalonamento

em sistemas de tempo compartilhado, que trabalham de modo mais interativo. Isso ocorre porque os usuários podem, através das aplicações, manipular informações do sistema, o que pode alterar o comportamento do sistema de modo geral, em função do nível de compartilhamento de recursos exigido no ambiente. Em políticas de escalonamento de sistemas de tempo real, o controle de tempo de execução do processo é mais rigoroso, pois não é permitido comprometer o processamento de modo geral, pois a atualização precisa ser constante. Um bom exemplo são os sistemas industriais voltados à produção e também de controle de tráfego aéreo (MACHADO; MAIA, 2013).

Faça você mesmo

Elabore uma tabela com uma sequência de processos; recomendo três. Defina o tempo de processamento para cada um e o seu respectivo nível de prioridade.

Desenvolva o diagrama de escalonamento dos processos por:

- prioridades;
- múltiplas filas com realimentação;
- circular.
- cooperativo.

Compreenda o mecanismo de escalonamento de todos eles e siga em frente!

Sem medo de errar

Considere que você precisa realizar um teste de eficiência do sistema operacional quando o sistema de gestão integrada está em execução, e de forma a identificar o comportamento da gerência de processos e como está acontecendo o escalonamento circular por prioridades. No SOsim você pode fazer as verificações a seguir, considerando que o tipo de escalonamento aplicado será o circular:

a) configure no console do SOsim os parâmetros que o sistema deverá seguir;

b) a princípio, crie quatro processos que tenham o mesmo nível de prioridade. Defina quais serão CPU-bound e quais serão I/O- bound;

c) anote os tempos que os processos levam desde a criação até o encerramento, *turnaround*, bem como as mudanças de estados;

d) altere, também, a fatia de tempo que um processo pode levar, ou seja, configure esse parâmetro;

e) compare os tempos antes da alteração e depois, para poder dimensionar as respectivas mudanças de estados;

f) observe quais foram as variações em processamento, estados e tempo.

Tome nota dos parâmetros utilizados e também dos resultados obtidos nesse processo. Associe essa atividade às tarefas que o sistema operacional deverá controlar, com a implantação do sistema de gestão integrada de monitoramento e controle de agendamentos de consultas e exames para a rede da clínica médica.

O SOsim é um simulador de comportamento de um sistema operacional, no que tange ao gerenciamento e alocação de recursos necessários para a realização das tarefas necessárias. Desenvolvido pelo professor Luiz Paulo Maia, no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, com esse simulador é possível visualizar como ocorre e quais são os mecanismos de funcionamento de um sistema operacional multiprogramável. Aproveite e conheça mais sobre essa importante ferramenta em: http://www.training.com.br/sosim/.

Atenção!

Faça um relatório com as observações da simulação e compare os tempos de processamento quando você altera os parâmetros no SOsim. Veja como funcionam os mecanismos de escalonamento e aprenda mais. Pratique!

Assista, também, a um um vídeo que mostra de uma forma bastante simples de como você pode criar um minissistema operacional. É uma oportunidade de conhecer e ter contato com uma linguagem de programação (Visual Basic) e também de compreender como é o contexto de *software* de sistemas. Disponível em: https://www.youtube.com/watch?v=f6arCgdUDwc.

Lembre-se

A principal vantagem deste escalonamento é permitir o melhor balanceamento no uso do processador em sistema de tempo compartilhado. Processos com o perfil I/O-bound devem receber do administrador do sistema prioridades com valores maiores que as dos processos CPU-bound. Isso permite ao sistema operacional praticar uma política compensatória entre processos de perfis distintos, compartilhando o processador de forma mais igualitária. Esse tipo de escalonamento é

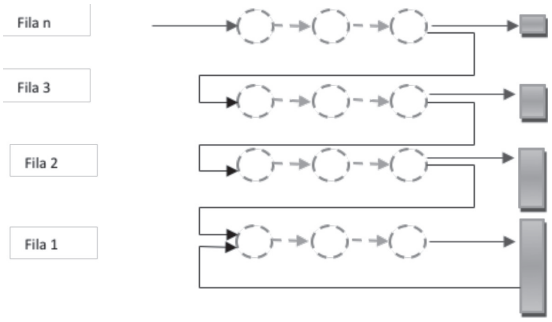
amplamente utilizado em sistemas de tempo compartilhado, como o Windows e o Unix (MACHADO; MAIA, 2013, p. 136).

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com as de seus colegas.

Introdução ao escalonamento: conceitos, tipos, e escalonamento de threads	
1. Competência de fundamentos de área	O aluno deverá ser capaz de identificar quais são as principais funções de um sistema operacional, bem como ter conhecimento sobre como se dá o compartilhamento de recursos e a sua gerência.
2. Objetivos de aprendizagem	Conhecer a evolução dos sistemas operacionais e suas respectivas especificidades; conhecer e saber identificar os principais processos e como ocorre o compartilhamento de recursos; conhecer como se dá a gerência de processos e de armazenamento de arquivos; conhecer e saber gerenciar os dispositivos de entrada e saída.
3. Conteúdos relacionados	Introdução ao escalonamento: conceitos, tipos e escalonamento de *threads*.
4. Descrição da SP	Suponha que os parâmetros a utilizar são os do mecanismo de escalonamento por múltiplas filas com realimentação. Explique como é aplicado e a lógica deste procedimento realizado pelo sistema operacional.
5. Resolução da SP	Para compreender o escalonamento proposto, considere: a) no caso do escalonamento por múltiplas filas com realimentação, apenas a fila com menor prioridade utiliza o mecanismo de escalonamento circular. Para as demais, é o FIFO; b) o escalonamento só acontecerá em uma fila a partir do momento em que as de maior prioridade estiverem vazias; c) saiba também que, quanto maior a prioridade da fila, menor será a sua fatia de tempo; d) logo que o processo é criado, ele é direcionado para a fila de maior prioridade, em último lugar e, de acordo com a preempção por fatia de tempo, pode ser redirecionado a uma fila que tenha uma prioridade menor. Observe o esquema a seguir que representa o escalonamento por múltiplas filas com realimentação:

Figura 2.6 – Escalonamento por múltiplas filas com realimentação



Fonte: Adaptado de Machado e Maia (2013, p. 139).

Faça você mesmo

Simule esse tipo de escalonamento no SOsim. Crie ao menos três processos e defina prioridades e fatias de tempo diferentes. Aloque em filas com prioridades distintas e verifique o comportamento desse procedimento, as mudanças de estado e o tempo de processamento.

Lembre-se

O escalonamento por múltiplas filas com realimentação é um algoritmo de escalonamento generalista, podendo ser implementado em qualquer tipo de sistema operacional. Um dos problemas encontrados nesta política é que a mudança de comportamento de um processo CPU-bound para I/O-bound pode comprometer seu tempo de resposta. Outro aspecto a ser considerado é sua complexidade de implementação, ocasionando um grande *overhead* ao sistema (MACHADO; MAIA, 2013, p. 139).

Faça valer a pena!

1. Complete as lacunas da frase com as palavras da alternativa correta: "Um processo fica em _____ a partir de sua criação e permanece neste até que estejam os recursos dimensionados e devidamente alocados e ele passa, então, ao estado de_____. Com isso, ele poderá ser chamado para processamento e, até o término desta operação, permanecerá no estado de_____".

a) espera/pronto/execução.

b) estado/execução/pronto.

c) espera/tempo/pronto.

d) pronto/estado/execução.

e) pronto/espera/criação.

2. Descreva o escalonamento por prioridade.

3. Explique qual é a diferença dos escalonamentos por múltiplas filas e por múltiplas filas com realimentação.

4. Associe na tabela e assinale a alternativa que contém a sequência obtida:

Conceito	Descrição
I – Escalonamento preemptivo	( ) Permite que o sistema operacional interrompa o processamento de acordo com a prioridade.
II – Escalonamento não preemptivo	( ) O processo, de forma voluntária, interrompe a execução, entra em espera e libera recursos para o processamento de outro processo que esteja no estado de pronto, e o escalonamento faz a respectiva verificação para a redistribuição dos processos.
III – Escalonamento colaborativo	( ) O sistema operacional não gera interrupções.

A sequência que representa os conceitos e suas respectivas definições é:

a) II, I, e III.

b) I, II e III.

c) I, III e II.

d) III, II e I.

e) III, I e II.

5. Considere as afirmações:

I – Nesse tipo de escalonamento, é determinada qual é a prioridade de execução de um processo.

II – Se os valores das prioridades dos processos são iguais, então esses

serão ordenados em fila (FIFO).

III – Pode ocorrer a mudança voluntária do estado execução para o de espera, caso haja perda em eficiência do processador.

São características do escalonamento:

a) Cooperativo.

b) Preemptivo.

c) Por prioridades.

d) Por múltiplas filas.

e) FIFO.

6. Assinale a alternativa que apresenta a sequência correta:

I – O escalonamento circular por prioridades só acontecerá em uma fila a partir do momento em que as de maior prioridade estiverem vazias.

II – Em um escalonamento por múltiplas filas com realimentação, quanto maior a prioridade da fila, menor será a fatia de tempo utilizada para executar os processos.

III – Em escalonamento por múltiplas filas com realimentação, logo que o processo é criado. ele é direcionado para a fila de maior prioridade, em último lugar e, de acordo com a preempção por fatia de tempo, ele pode ser redirecionado a uma fila que tenha uma prioridade menor.

a) F-F-F.

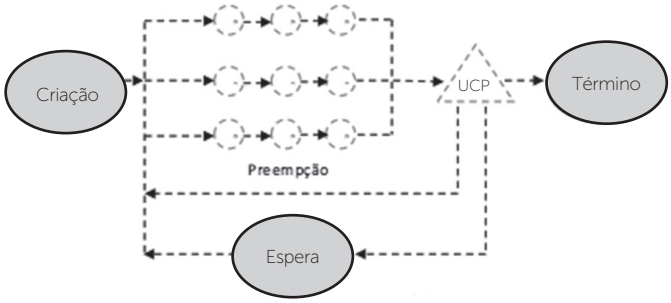
b) F-V-V.

c) V-V-V.

d) F-F-F.

e) V-F-V.

7. Analise a figura e assinale a alternativa correta quanto ao tipo de escalonamento ilustrado:



a) Circular.

b) Cooperativo.

c) Múltiplas filas.

d) Por prioridade.

e) FIFO.

Seção 2.4

Algoritmos de escalonamento: características, políticas, tipos e exemplos

Diálogo aberto

Olá, aluno! Bem-vindo a mais uma seção de autoestudos! Vamos conhecer outros algoritmos de escalonamento, identificar as suas principais características, bem como sintetizar a ideia de política de níveis de escalonamento, objetivos e critérios que precisam ser considerados quando se trata da otimização do processamento. Além disso, vamos conhecer também como é tratado pelo sistema operacional o escalonamento por threads.

Estudamos até aqui algumas especificidades quanto à definição e conceitos envolvidos em processos e *threads*. Nesse contexto, conseguimos conhecer como acontece a comunicação entre processos e de que forma o sistema operacional controla esse tipo de operação. Além desses, vimos também as definições de escalonamentos e quais deles são mais utilizados em sistemas multiprogramáveis.

No caso desta seção, como estudaremos outros algoritmos de escalonamento, temos de apresentar uma forma de selecionar ou avaliar um algoritmo de escalonamento que atenda às necessidades de desempenho de *software* e *hardware* que a empresa precisa, para que sejam otimizadas as rotinas do ERP e aconteça a efetiva implantação do sistema.

Mas você se recorda da realidade profissional que estamos trabalhando?

Então, vamos relembrar: devemos implantar um sistema de gestão integrada em uma clínica médica de forma a conseguir utilizar todo o potencial de seu parque tecnológico, incluindo servidores e os sistemas já instalados. Dessa forma, saber gerenciar as rotinas e entender as atividades controladas pelo sistema operacional e como ocorrem as ações que ele é responsável é de extrema importância.

Quanto mais associar os conceitos às variadas práticas e necessidades do mercado de trabalho, maiores serão as possibilidades de você aproveitar esse contato para aprender, analisar, construir e desenvolver novas formas de solucionar os possíveis problemas do cotidiano organizacional, quando se trata de ambientes computacionais.

Nesse sentido, você agora é convidado a participar de mais uma etapa desse processo de ensino-aprendizagem, associando sempre o conteúdo a uma possível situação de mercado.

Aproveite todo o potencial de estudos advindos das leituras, pesquisas, aulas e interações que o seu material didático e ambiente educacional oferecem! Desde já, bons estudos e práticas a você!

Não pode faltar

Já estudamos os algoritmos de escalonamento FIFO (*First in first out*), como sugerido por Machado e Maia (2013), ou ainda FCFS (*first come first served*) como mencionado por Silberschatz et al. (2004). O algoritmo de escalonamento FCFS considera que o primeiro processo a chegar na fila será o primeiro a entrar em execução, no entanto este procedimento pode fazer com que processos pequenos e de custo baixo em termos de processamento esperem por processos mais longos. Ambos são de baixa complexidade.

Vimos também o escalonamento por prioridade em que o processo que tiver o maior nível será executado primeiro. Então, nesse sentido, visando reduzir o tempo de espera mesmo com níveis distintos de prioridade, foi implementado o algoritmo SJF (shortest job first). Além desses, vimos também o algoritmo de escalonamento circular ou RR (round robin), que considera a divisão do processo por fatias de tempo, também chamadas de quantum (SILBERSCHATZ et al., 2004).

Ao estudar os algoritmos utilizados para realizar o escalonamento de processos, temos de considerar fatores como níveis da política de escalonamento, bem como os objetivos e os critérios que são utilizados para essa ação. Sendo assim, as políticas de definição dos níveis de escalonamento podem ser (DEITEL, 2005):

a) de alto nível: que determina quais grupos de processos poderão trabalhar de forma concorrente, e, ainda, controla o número de processos em um espaço de tempo, ou seja, o seu grau de multiprogramação, procurando evitar o uso de todos os recursos do sistema para processamento. Com isso, alguns processos podem ter de esperar a conclusão de outros para que sejam executados;

b) de nível intermediário: em que são definidos os processos que competirão por recursos do processador. É possível interromper um processo e retomá-lo em seguida com o seu processamento para que possa garantir o bom desempenho da máquina;

c) de baixo nível: nessa política de escalonamento, normalmente há a atribuição de prioridade para o processo, com isso aumenta a probabilidade dele ser escolhido para processamento. Em uma política de escalonamento de baixo nível, há a determinação

de processos que estão ativos para que o sistema possa designar um processador quando estiver disponível.

Mas, afinal, qual é o objetivo de se estabelecer uma política de escalonamento?

Bem, de modo geral, o objetivo é potencializar o uso de recursos da máquina, porém vamos especificar essa informação. Com a realização de escalonamento, é possível maximizar o uso de recursos de forma a atender uma quantidade maior de processos e em menor tempo de execução. Minimizar os tempos de resposta também é um dos objetivos do escalonamento. Outro fator importante é trabalhar com memória e processador de forma a evitar que aconteça a interrupção por tempo indefinido de um determinado processo. Além disso, é importante que, com o escalonador, sejam impostas prioridades na escolha de processos e alocação de recursos. Reduzir a probabilidade de ocorrência de sobrecarga e, ainda, melhorar a previsão de duração do processo, ou seja, a sua previsibilidade. Este último é um conceito que determina o tempo mínimo de resposta, de espera e de execução de um processo.

Assimile

Sobrecarga frequentemente resulta em desperdício de recursos, mas uma certa porção dos recursos do sistema se investida efetivamente como sobrecarga pode melhorar muito o desempenho geral do sistema (DEITEL, 2005, p. 213).

Como mencionado, alguns critérios para se estabelecer a política de escalonamento adotada precisam ser considerados. Dentre eles, podemos elencar os critérios que consideram se o processo é orientado ao processador, orientado a operações de entrada e saída, em lote, ou se é um processo interativo. Veja a seguir as breves descrições de cada um deles segundo os critérios de comportamento de processos (DEITEL, 2005):

a) orientado a processador: critério que determina que um processo tende a utilizar todos os recursos do processador que lhe foi atribuído;

b) operações de E/S: utiliza o processador apenas para atender as requisições das operações de entrada e saída de dados e, em seguida, libera o recurso;

c) processo em lote: há a alocação de recursos de processamento sem a interação com o usuário para executar um determinado processo ou grupo de processos.

d) processo interativo: nesse critério, o usuário precisa participar com as entradas de dados. Os tempos de resposta do processo precisam ser mais rápidos.

Porém, não basta que se estabeleça a política de escalonamento de acordo com os respectivos níveis de priorização de processamento e alocação de recursos, nem que

se saiba os critérios que norteiam as ações do escalonamento se não forem de seu conhecimento os algoritmos que realizam essas tarefas, que são tão importantes para a análise de desempenho de um sistema computacional. A partir de agora, conheça também outros algoritmos de escalonamento. Siga em frente! Veja no Quadro 2.3 uma breve descrição de alguns deles:

Quadro 2.3 | Algoritmos de escalonamento

Algoritmo de escalonamento	Descrição
Escalonamento por próxima taxa de escalonamento mais próxima.	Conhecido como HRRN (highest response ratio next). Visa corrigir um erro relacionado ao escalonamento SJB/SPF mencionado acima, em que há essencialmente a priorização de processos maiores a menores, fazendo com que esses tenham de esperar mais tempo do que o previsto.
Escalonamento por menor tempo de execução- restante.	SRT (shortest remaining time), ao contrário do SJB, determina o processo que terá o menor tempo de execução, por estimativa e dá prioridade a esse.
Escalonamento por fração justa	FSS (fair share scheduling) esse algoritmo permite que, em sistemas multiusuários, os processos relacionados a um usuário sejam agrupados aos de outros usuários para processamento. Também restringe cada grupo ao seu respectivo subconjunto de recursos do sistema.
Escalonamento por prazo	Esse tipo de algoritmo necessita da disponibilização exata de recursos necessários de processamento para que o sistema operacional possa distribuir sem prejudicar as rotinas comuns e desempenho. Dessa forma, o objetivo é que se cumpra o prazo predeterminado para o processamento do conjunto descrito.
Escalonamento de tempo real	Visa utilizar os recursos de processamento de forma potencializada. Isso quer dizer que pode aplicar de acordo com a operação, um algoritmo diferente para cada um dos processos solicitados.
Escalonamento de threads Java	Esse tipo de escalonamento é realizado pelo sistema operacional e considera os níveis de suporte para threads, no entanto, não distingue se é um processo multithread ou se deve trabalhar individualmente com os processos se este for de usuário, e então, necessita de uma biblioteca padrão para escalonar. Se for de núcleo, o sistema escalona um por vez, de forma a considerar a alocação de recursos para cada um deles.

Fonte: Adaptado de Deitel (2005, p. 218- 228).

Vamos descrever melhor cada um deles?

O algoritmo HRRN trabalha de forma a considerar a relação existente entre o tempo que o processo leva para ser concluído e o seu tempo de espera. Com isso, quando

há a alocação do processo para execução, esta ação acontecerá até o encerramento das instruções, de forma ininterrupta e, por esse motivo, ele é não preemptivo. Há uma fórmula que é utilizada para o cálculo da prioridade inerente a cada processo. A fórmula é a seguinte (DEITEL, 2005):

Prioridade = (tempo de espera + tempo de serviço) / tempo de serviço

Nesse algoritmo de escalonamento, os processos menores têm preferência e há o favorecimento aos processos mais longos em função da identificação do seu tempo de espera.

O SRT, através de uma ação preemptiva, pode interromper um processamento em função da alocação temporária do recurso para que um processo menor seja eliminado da fila. Lembre-se que essa determinação e ação do SRT são fundamentadas na estimativa de tempo de execução de um processo e a escolha se dá pelo que apresentar uma estimativa menor.

Reflita

O algoritmo SRT teoricamente oferece tempos de espera mínimos, mas, em certas ocasiões, devido à sobrecarga de preempção, o SPF pode se sair melhor. Por exemplo, considere um sistema no qual um processo em execução esteja quase no final e chegue um novo processo, cujo tempo estimado de serviço seja pequeno. O processo em execução pode se sair melhor? A disciplina do SRT faria a preempção, mas essa pode não ser a alternativa ótima. Uma solução é garantir que um processo em execução não possa mais sofrer preempção quando o tempo de execução restante atingir um determinado nível baixo (DEITEL, 2005, p. 219).

Muito interessante também é o algoritmo de escalonamento por fração justa, o FSS. Esse trabalha da seguinte forma: a partir da quantidade de processos e subprocessos, associados a um determinado indivíduo, que compartilha recursos da máquina com outros indivíduos, o escalonamento procura identificar o grupo de processos associados que mais necessita de recursos em função de suas operações (1), e também, o grupo de processos que pode até ser maior, porém, que o tipo de prioridade associada à atividade é menor em termos de consumo de recursos de processamento (2). Então, o sistema determinará uma quantidade de recursos para o processamento do grupo 1 que não comprometa o processamento do grupo 2, de forma literalmente mais justa, quanto à distribuição dos recursos. A razão dessa ação está entre a quantidade de recursos de processamento utilizado versus o tempo real decorrido para o processamento (DEITEL, 2005).

Outro algoritmo que elencamos é o escalonamento por prazo. Como enfatizado,

esse algoritmo necessita da requisição prévia de recursos do sistema, com a inserção exata que será necessária para que se cumpra o prazo previsto de processamento de um conjunto de processos.

Uma desvantagem desse tipo de escalonamento é que pode haver sobrecarga do uso de recursos, o que pode impactar no desempenho da máquina, principalmente no que tange à dedicação de serviços e recursos para outros processos. No entanto, esse é importante para o escalonamento de processos em tempo real. Uma das formas de trabalho deste algoritmo é alocar os processos que necessitam de menor quantidade de recursos para que tenham prioridade de execução.

Como mencionado, os algoritmos de escalonamento de processos em tempo real permitem a alocação de algoritmos distintos de acordo com a solicitação que foi realizada. Dessa forma, há algumas classificações para eles. Vejamos:

a) escalonamento de tempo real não crítico: visa garantir que processos em tempo real sejam executados, porém sem a garantia de que cumprirá alguma restrição de tempo;

b) escalonamento de tempo real crítico: ao contrário do anterior, garante o processamento antes do prazo estabelecido;

c) escalonamento de tempo real estático: nesse, as prioridades dos processos são determinadas uma única vez;

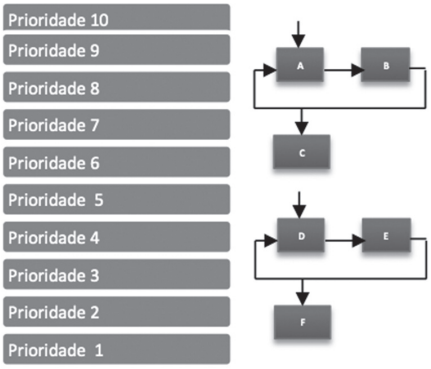
d) escalonamento de tempo real dinâmico: estes realizam o escalonamento por prioridades e isso acontece durante a execução dos processos. Pode haver a interrupção preemptiva, de acordo com as necessidades de alocação identificadas, de forma a preencher, inclusive, os espaços considerados de folga entre a seleção de processos ou de execução. Essa folga refere-se ao tempo restante ao processo para que seja finalizada a sua execução.

Por fim, vamos falar um pouco sobre o escalonamento de *thread* Java. No Quadro 2.3, evidenciamos que o modo de tratamento de um escalonamento de *thread* de usuário (escalonamento por *timeslicing* ou espaço de tempo) e de núcleo são distintos no sistema operacional. Outras considerações, como a quantidade de *threads*, sua ordem, bem como a prioridade estabelecida, devem ser definidas em nível de *software*, ou seja, no desenvolvimento, pelo responsável pela implementação. Nesse caso, pode ser aplicado, por exemplo, o algoritmo de escalonamento por fração justa, ou ainda, de acordo com a quantidade de *threads* alternância circular ou mesmo o algoritmo de escalonamento por intervalo de tempo. Tudo dependerá do tipo de *thread* e a quantidade de recursos necessários de forma a buscar sempre garantir um bom desempenho da máquina.

Exemplificando

Ao mencionar *threads*, precisamos considerar que na linguagem Java todo tipo de aplicação é *multithread*. Os threads Java recebem níveis de prioridade entre 0 e 1 (Thread.MIN_PRIORITY e Thread.MAX_PRIORITY, que são, respectivamente, comandos que definem valores constantes da prioridade, no caso, 1 no mínimo e 10 no máximo.)

Figura 2.7 | Escalonamento de threads Java



Fonte: Adaptado de Deitel (2005, p. 227).

Threads Prontos

1. Um *thread* pode ser executado até o encerramento das instruções.

2. Pode sair de execução e entrar em espera, para que outro processo de maior prioridade seja escalonado.

3. Pode sofrer preempção para que outro *thread* de maior prioridade seja executado.

4. O *thread* de maior prioridade pode executar todo o tempo até o seu encerramento ou, se for *multithread*, poderá realizar a alternância circular.

Faça você mesmo

Compreenda os mecanismos de escalonamento relacionados a *threads* em: https://strongloop.com/strongblog/dica-da-semana-sobre-desempenho-node-js-monitoramento-de-ciclo-de-eventos/.

Leia também matéria de suporte sobre o Windows 8. Disponível em: https://technet.microsoft.com/pt-br/library/Cc771692(v=WS.10).aspx#BKMK_Scen2. .

Pesquise mais

Leia e estude com atenção o material sobre escalonamento que está disponível em: http://www.ece.ufrgs.br/~fetter/eng04008/sched.pdf.

Leia mais sobre escalonamento no artigo Política de Escalonamento de Processos em Linux. Disponível em: http://revista.grupointegrado.br/revista/index.php/campodigital/article/view/344/159.

Sem medo de errar

Para essa atividade, é preciso fazer um levantamento de como avaliar um algoritmo que seja ideal para atender às necessidades tanto do sistema operacional quanto das aplicações que interagem no ambiente organizacional. Para tal, a seguir estão algumas contribuições que podem auxiliar nesse processo de seleção do algoritmo de escalonamento. Vamos lá!

A avaliação de um algoritmo de escalonamento está diretamente relacionada à sua seleção. Ele deve ser adequado às necessidades de processamento e funcionalidades que devem ser aliadas ao sistema computacional.

Desse modo, você precisa identificar quais são as necessidades de processamento do sistema. Para tal, podem ser elencados alguns pontos de atenção, como:

a) selecionar os critérios de escalonamento: orientado a processador, em lote, interativo, E/S;

b) saber qual é o tempo de resposta que a aplicação precisa (*througtput*);

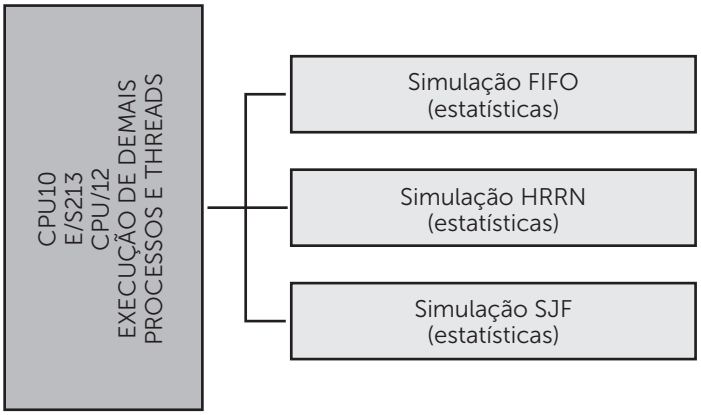
Ex.: será preciso potencializar e aumentar a utilização dos recursos de processamento da CPU com um tempo mínimo de resposta de "x";

c) ou, ainda, equalizar o processamento para que não exceda os limites de tempo de execução total e diminui tanto o turnaround quanto o *througtput*;

d) para tal realiza-se uma avaliação analítica que pode ser por: modelagem determinística, por enfileiramento, simulações e mesmo a implementação em nível de teste.

Veja um exemplo que dimensiona por simulação, as estatísticas de desempenho que podem auxiliar na determinação de um algoritmo de escalonamento:

Figura 2.8 | Simulações para avaliação de algoritmos de escalonamento



Fonte: Adaptado de Deitel (2005, p. 147).

Atenção!

Simulações podem ser caras e exigir muito tempo de uso de recursos do sistema computacional. Para verificar se os dados de uma simulação de fato são coerentes, é aconselhável que se codifique e avalie o comportamento do algoritmo diretamente no sistema operacional.

Lembre-se

O simulador possui uma variável representando um relógio; à medida que o valor dessa variável é aumentado, o simulador modifica o estado do sistema para refletir as atividades dos dispositivos, dos processos e do escalonador. Enquanto a simulação é executada, as estatísticas que indicam o desempenho do algoritmo são colhidas e impressas (DEITEL, 2005, p. 146).

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com as de seus colegas.

<table>
<tr><th colspan="2">Introdução ao escalonamento: conceitos, tipos e escalonamento de threads</th></tr>
<tr><td>1. Competência de fundamentos de área</td><td>O aluno deverá ser capaz de identificar quais são as principais funções de um sistema operacional, bem como ter conhecimento sobre como se dá o compartilhamento de recursos e a sua gerência.</td></tr>
<tr><td>2. Objetivos de aprendizagem</td><td>Conhecer a evolução dos sistemas operacionais e suas respectivas especificidades; conhecer e saber identificar os principais processos e como ocorre o compartilhamento de recursos; conhecer como se dá a gerência de processos e de armazenamento de arquivos; conhecer e saber gerenciar os dispositivos de entrada e saída.</td></tr>
<tr><td>3. Conteúdos relacionados</td><td>Introdução ao escalonamento: conceitos, tipos e escalonamento de threads</td></tr>
<tr><td>4. Descrição da SP</td><td>Para realizar a sua primeira avaliação de algoritmo através da modelagem determinística, estabeleça uma quantidade de processos que deseja testar, por exemplo, 5. Determine qual o tempo de espera de cada um. E teste de acordo com o algoritmo de escalonamento FIFO (FCFS). Represente esse procedimento e calcule o tempo médio de espera em milissegundos (quantum = 10 ms):</td></tr>
<tr><td>5. Resolução da SP</td><td>
<table>
<tr><th>Processos</th><th>Tempo de Burst
(tempo que a CPU leva para inserir outro processo da fila para execução e a espera por processos de E/S)</th><th>Tempo de espera (ms)</th></tr>
<tr><td>P1</td><td>10</td><td>0</td></tr>
<tr><td>P2</td><td>29</td><td>10</td></tr>
<tr><td>P3</td><td>3</td><td>39</td></tr>
<tr><td>P4</td><td>7</td><td>42</td></tr>
<tr><td>P5</td><td>12</td><td>49</td></tr>
</table>
Considere que os tempos são:
Representação:


Tempo médio de espera = (0 + 10 + 39+ 42+ 49)/5 = 28 ms
Fonte: Deitel (2005, p. 145).
</td></tr>
</table>

Faça você mesmo

Estabeleça outros tempos de espera para os cinco processsos e calcule o seu tempo médio de espera a partir da modelagem determinística. Simule no soSim e tome nota dos resultados para comparar os algoritmos de escalonamento (SJF e RR) e o seu respectivo comportamento.

Lembre-se

Esse método apanha uma carga de trabalho específica predeterminada e define o desempenho de cada algoritmo para essa carga de trabalho. [...] A modelagem determinística é simples e rápida. Ela nos dá números exatos, permitindo que os algoritmos sejam comparados (DEITEL, 2005, p. 145).

Faça valer a pena!

1. Analise as afirmações. Com relação ao escalonamento de *threads*, é correto o que se afirma em:

I – Pode sair de execução e entrar em espera, para que outro processo de maior prioridade seja escalonado.

II – O *thread* de maior prioridade pode executar todo o tempo até o seu encerramento ou, se for *multithread*, poderá realizar a alternância circular.

III – Não pode sofrer preempção para que outro *thread* de maior prioridade seja executado.

a) I e II.

b) I, II e III.

c) I e III.

d) II e III.

e) I apenas.

2. Complete as lacunas da frase com as palavras de uma das alternativas abaixo:

"O _____, através de uma ação_____, pode interromper um processamento em função da alocação temporária do recurso para que um processo menor seja eliminado da _____".

a) SRT/não preemptiva/fila.

b) SJF/preemptiva/fila.

c) FIFO/preemptiva/pilha.

d) SRT/não preemptiva/lista.

e) SRT/preemptiva/fila.

3. Associe na tabela os conceitos às suas respectivas descrições:

Critérios	Descrição
a. Baixo nível	( ) Determina quais grupos de processos poderão trabalhar e controla o número de processos criados em um espaço de tempo.
b. Intermediário	( ) É possível interromper um processo e retomar em seguida com o seu processamento para que possa garantir o bom desempenho da máquina.
c. Alto nível	( ) Nessa política de escalonamento, normalmente há a atribuição de prioridade para o processo. Com isso, aumenta a probabilidade de ele ser escolhido para processamento.

Assinale a alternativa que contém a ordem de associação correta:

a) a-b-c.

b) b-c-a.

c) a-c-b.

d) c-b-a.

e) b-a-c.

4. Descreva como é o mecanismo de funcionamento do escalonamento HRRN.

5. Explique como acontece a seleção de processos no algoritmo SRT.

6. As afirmações abaixo são verdadeiras ou falsas? Assinale a alternativa correta:

I – A modelagem determinística é simples e rápida.

II – O modo de tratamento de um escalonamento de *thread* de usuário (escalonamento por *timeslicing* ou espaço de tempo) e de núcleo são distintos no sistema operacional.

III – Escalonamento de tempo real não crítico: visa garantir que processos em tempo real sejam executados, porém sem a garantia de que cumprirá alguma restrição de tempo.

a) F-F-F.

b) V-V-V.

c) V-F-V.

d) F-F-V.

e) F-V-V.

7. Está descrito o algoritmo de escalonamento por tempo real em:

I – Permitem a alocação de algoritmos distintos de acordo com a solicitação que foi realizada.

II – Escalonamento de tempo real dinâmico: esses realizam o escalonamento por prioridades e isso acontece durante a execução dos processos.

III – A partir da quantidade de processos e subprocessos que estão associados a um determinado indivíduo, sendo que este compartilha recursos da máquina com outros colegas, o escalonamento procura identificar o grupo de processos associados aos usuários de forma a identificar aquele que mais necessita de recursos em função de suas operações para que possa realizar o escalonamento.

a) I e II.

b) II e III.

c) I, II e III.

d) I e III.

e) Apenas III.

Referências

DEITEL, H. M.; DEITEL P. J.; CHOFFNES, D. R. **Sistemas operacionais**. 3. ed. São Paulo: Prentice Hall, 2005.

MACHADO, Francis B.; MAIA, Luiz P. **Arquitetura de Sistemas Operacionais**. 5. ed. Rio de Janeiro: LTC, 2013.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGME, Greg. **Sistemas operacionais**: sistemas e aplicações. 6. ed. Rio de Janeiro: Elsevier, 2004.

STUART, Brian L. **Princípios de sistemas operacionais**: projetos e aplicações. São Paulo: Cengage Learning, 2010.

TANEMBAUM, Andrew S. **Sistemas operacionais modernos**. 3. ed. São Paulo: Pearson Prentice Hall, 2009.