Sistemas Operacionais - 2° Bimestre
Luca Gouveia – 2019

Capítulo 13
Impasses– Respostas

4. Uma vez detectado um impasse, quais as abordagens possíveis para resolvê-lo? Explique-as e comente sua viabilidade.

4R - Quando ocorrer um impasse, o sistema deve detectá-lo, determinar quais as tarefas e recursos envolvidos e tomar medidas para desfazê-lo.

Uma vez detectado um impasse e identificadas as tarefas e recursos envolvidos, o sistema deve proceder à resolução do impasse:

Ignorar o problema: *Caso a frequência de DeadLocks (Impasses) seja baixa o o custa de detecção alto, o melhor a ser feito é ignorar o impasse.*

Eliminar tarefas: *uma ou mais tarefas envolvidas no impasse são eliminadas, liberando seus recursos para que as demais tarefas possam prosseguir. A escolha das tarefas a eliminar deve levar em conta vários fatores, como o tempo de vida de cada uma, a quantidade de recursos que cada tarefa detém, o prejuízo para os usuários que dependem dessas tarefas, etc.*

Retroceder tarefas (rollback): *uma ou mais tarefas envolvidas no impasse têm sua execução parcialmente desfeita (uma técnica chamada rollback), de forma a fazer o sistema retornar a um estado seguro anterior ao impasse. Para retroceder a execução de uma tarefa, é necessário salvar periodicamente seu estado, de forma a poder recuperar um estado anterior quando necessário . Além disso, operações envolvendo a rede ou interações com o usuário podem ser muito difíceis ou mesmo impossíveis de retroceder.*

A detecção de impasses é pouco usada, pois possui um custo elevado de detecção e as alternativas de resolução sempre implicam perder tarefas ou parte das execuções.Essa técnica é aplicada, por exemplo, no gerenciamento de transações em sistemas de bancos de dados.

8. Uma vez detectado um impasse, quais as abordagens possíveis para resolvê-lo? Explique-as e comente sua viabilidade.

8R -

Grafo 1 :

Ciclo: *T1 ---> R2 → T2 ---> R1 → T1*

Grafo 2 :

Ciclo: *T1 ---> R2 → T2 ---> R3 → T3 ---> R1 → T1*

9. A figura a seguir representa uma situação de impasse em um cruzamento de trânsito. Todas as ruas têm largura para um carro e sentido único. Mostre que as quatro condições necessárias para a ocorrência de impasses estão presentes nessa situação.
Em seguida, defina uma regra simples a ser seguida por cada carro para evitar essa situação; regras envolvendo algum tipo de informação centralizada não devem ser usadas.

9R -

Exclusão Mútua - *O recurso (via) deve ser temporariamente exclusiva para um veículo (Um recurso só pode ser alocado para um processo), ou seja, apenas um carro por vez pode ser acesso àquela via.*

Hold and Wait - *Cada carro já possui sua via mas requer acesso à outra via .*

Espera Circular - *Um carro solicita a via alocada para outro veículo. (carro1 ---> viaX → carro2).*

Não Preempção - *Uma via (recurso) alocado para um veículo não pode ser removido, só o próprio veículo pode liberar o acesso a via novamente.*

Uma estratégia que pode ser usada para evitar novamente essa situação é usar o ***rollBack,*** *retornando à um estado anterior onde isso não estava ocorrendo e a partir disso ordenar os recursos que serão usados para cada veículo.*

Capítulo 14
Gestão de memória – Respostas

1. Explique em que consiste a resolução de endereços nos seguintes momentos: codificação, compilação, ligação, carga e execução.

1R -

Programação*: o programador escolhe o endereço de cada uma das variáveis e do código do programa na memória. Esta abordagem normalmente só é usada na programação de sistemas embarcados simples, programados diretamente em Assembly.*

Compilação*: ao traduzir o código-fonte, o compilador escolhe as posições das variáveis na memória. Para isso, todos os códigos-fontes necessários ao programa devem ser conhecidos no momento da compilação, para evitar conflitos de endereços entre variáveis em diferentes arquivos ou bibliotecas. Essa restrição impede o uso de bibliotecas pré compiladas. Esta abordagem era usada em programas executáveis com extensão .COM do MS-DOS e Windows.*
Ligação*: na fase de compilação, o compilador traduz o código fonte em código binário, mas não define os endereços das variáveis e funções, gerando como saída um arquivo objeto (object file) 2 , que contém o código binário e uma tabela de símbolos descrevendo as variáveis e funções usadas, seus tipos, onde estão definidas e onde são usadas. A seguir, o ligador (linker) pega os arquivos objetos com suas tabelas de símbolos, define os endereços de memória dos símbolos e gera o programa executável.*

Carga*: também é possível definir os endereços de variáveis e de funções durante a carga do código em memória para o lançamento de um novo processo. Nesse caso, um carregador (loader) é responsável por carregar o código do processo na memória e definir os endereços de memória que devem ser utilizados. O carregador pode ser parte do núcleo do sistema operacional ou uma biblioteca ligada ao executável, ou ambos. Esse mecanismo normalmente é usado na carga de bibliotecas dinâmicas (DLL).*

Execução*: os endereços emitidos pelo processador durante a execução do processo são analisados e convertidos nos endereços efetivos a serem acessados na memória real. Por exigir a análise e a conversão de cada endereço gerado pelo processador, este método só é viável com o auxílio do hardware.*

2. Como é organizado o espaço de memória de um processo?

2R -

TEXT*: contém o código binário a ser executado pelo processo, gerado durante a compilação e a ligação com as bibliotecas e armazenado no arquivo executável. Esta seção se situa no início do espaço de endereçamento do processo e tem tamanho fixo, calculado durante a compilação.*

DATA*: esta seção contém as variáveis estáticas inicializadas, ou seja, variáveis que estão definidas do início ao fim da execução do processo e cujo valor inicial é declarado no código-fonte do programa. Esses valores iniciais são armazenados no arquivo do programa executável, sendo então carregados para esta seção de memória quando o processo inicia. Nesta seção são armazenadas tanto variáveis globais quanto variáveis locais estáticas (por exemplo, declaradas como static em C).*

BSS*: historicamente chamada de Block Started by Symbol, esta seção contém as variáveis estáticas não-inicializadas. Esta seção é separada da seção DATA porque as variáveis inicializadas precisam ter seu valor inicial armazenado no arquivo executável, o que não é necessário para as variáveis não inicializadas. Com essa separação de variáveis, o arquivo executável fica menor.*

HEAP*: esta seção é usada para armazenar variáveis alocadas dinamicamente, usando operadores como malloc(), new() e similares. O final desta seção é definido por um ponteiro chamado Program Break, ou simplesmente break, que pode ser ajustado através de chamadas de sistema para aumentar ou diminuir o tamanho da mesma.*

STACK*: esta seção é usada para manter a pilha de execução do processo, ou seja, a estrutura responsável por gerenciar o fluxo de execução nas chamadas de função e também para armazenar os parâmetros, variáveis locais e o valor de retorno das funções. Geralmente a pilha cresce "para baixo", ou seja, inicia em endereços maiores e cresce em direção aos endereços menores da memória. O tamanho total desta seção pode ser fixo ou variável, dependendo do sistema operacional. Em programas com múltiplas threads, esta seção contém somente a pilha do programa principal. Como threads podem ser criadas e destruídas dinamicamente, a pilha de cada thread é mantida em uma seção de memória própria, alocada dinamicamente no heap ou em blocos de memória alocados na área livre para esse fim.*

Capítulo 15
Hardware de memória – Respostas

1. Explique a diferença entre endereços lógicos e endereços físicos e as razões que justificam o uso de endereços lógicos.

1R -

Endereços físicos: *Endereços dos bytes de memória física do computador. Estes endereços são definidos pela quantidade de memória disponível na máquina.*

Endereços lógicos: *Endereços de memória usados pelos processos e pelo sistema operacional e, portanto, usados pelo processador durante a execução. Estes endereços são definidos de acordo com o espaço de endereçamento do processador.*

7. Considerando a tabela de segmentos a seguir (com valores em decimal), calcule os endereços físicos correspondentes aos endereços lógicos 0:45, 1:100, 2:90, 3:1.900 e 4:200.

7R -

Physical Address = Base Ad + Offset
OffSet < Limit

- ***0:45 = 89***
- ***1:100 = 300***
- ***2:90 = 90***
- ***3:1900 = 3900 INVÁLIDO (1900 > 1000 limit)***
- ***4:200 = 1400***

8. Considerando a tabela de páginas a seguir, com páginas de 500 bytes , informe os endereços físicos correspondentes aos endereços lógicos 414, 741, 1.995, 4.000 e 6.633, indicados em decimal.

8R-

(Physical Frame * Page Size) + Offset
physalF = *logicalA//pageS* ***(divisão inteira)***
Offset = *locicalA%PageS*

- ***414 = 1914***
- ***741 = 741***
- ***1995 = <u>495</u> (NULL - não tem frame referente)***
- ***4000 = 0***
- ***6633 = 3633***

Capítulo 16
Alocação de memória – Respostas

5. Considere um banco de memória com os seguintes "buracos" não-contíguos: Nesse banco de memória devem ser alocadas áreas de 5MB, 10MB e 2MB, nesta ordem, usando os algoritmos de alocação First-fit, Best-fit ou Worst-fit. Indique a alternativa correta:

5R -

(b) Se usarmos Worst-fit, o tamanho final do buraco B4 será de 15 Mbytes.

Capítulo 17
Paginação em disco – Respostas

1. O que é uma falta de página? Quais são suas causas possíveis e como o sistema operacional deve tratá-las?

1R -

É a ausência de página na memória RAM verificada pela MMU, gerando assim, uma interrupção que desvia a execução para o sistema operacional. O sistema operacional verifica se a página é válida, usando os flags de controle da tabela de páginas. Caso a página seja inválida, o processo tentou acessar um endereço inválido e deve ser abortado. Por outro lado, caso a página solicitada seja válida, o processo deve ser suspenso enquanto o sistema transfere a página de volta para a memória RAM e faz os ajustes necessários na tabela de páginas. Uma vez a página carregada em memória, o processo pode continuar sua execução.