Capítulo 13 - Impasses

P. 10, questão 4.

De acordo com a leitura, não faz-se necessário nenhuma medida preventiva para prevenir ou evitar os impasses produzidos. As tarefas, por sua vez, seguem o fluxo normalmente de suas respectivas atividades, alocando-se e liberando-se os recursos necessários. Ao ocorrer um impasse, o S.O o detecta e determina quais são as tarefas e recursos envolvidos, o mesmo se responsabiliza para executar um 'undo' (desfazer).

P. 11, questão 8.

Figura 1. Impasse registrado, uma vez que **t1** necessita do recurso de **r2**, entretanto, **t2** retem este recurso. Por sua vez, **t2** necessita do recurso **r1**, mas o **t1** o possuí.

Figura 2. Impasse registrado, **t** necessita de **r2**, entretanto, **t2** está consumindo. Por outro lado, **t2** necessita de **r3**, mas, ele está sendo utilizado por **t3** em conjunto com **t4**. **t3** necessita de **r1**, mas **t1** está consumindo.

P. 11, questão 9.

Exclusão mútua: Os automóveis em sentidos divergentes - linhas verticais e horizontais - não devem acessar o mesmo cruzamento ao mesmo tempo;

Posse e espera: Os automóveis que estão em um cruzamento, solicitam, acesso ao outro cruzamento sem liberar o cruzamento que ele está ocupando;

Não-preempção: Os automóveis que ocupam um cruzamento só libera, quando ocorrer uma decisão;

Espera circular: Os automóveis vermelhos dependente da liberação do cruzamento dos automóveis azuis, que depende dos automóveis verdes, que depende dos automóveis amarelos e os automóveis amarelos dependem dos automóveis vermelhos, entrando em um ciclo (loop) de esperas contínuas;

A regra para evitar o impasse: Torna-se notório o uso de um recurso compartilhado nos cruzamento, facilitando assim, uma organização entre as vias (horizontais e verticais), por fim, cada cruzamento que possa chegar a garantir um conflito, deve ter um semáforo para informar se está livre/fechado;

Capítulo 14 - Conceitos básicos

P. 10, questão 1.

Codificação: O software escolhe a posição de cada variavél e do código-fonte do software;
Compilação: Compilador escolhe a posição das variavéis na memória, código-fonte é conhecido no momento que o processo de compilação ocorre, evitando possíveis conflitos de endereços na memória;
Ligação: Compilar produz símbolos que representam ás variáveis;

Carga: Responsável por definir os objetos de variáveis e funções de carga do código-fonte em memória para iniciar um novo processo;
Execução: Após todo o processo, estes, são analisados e consequentemente convertidos pelo processador para a memória final;

P. 10, questão 2.

O processo é organizado em: **text, data, heap, stack**
Text: Responsável por conter o código-fonte a ser executado durante o processo, gerado durante a compilação e a inclusão de bibliotecas no código-fonte;
* Data: Responsável por conter as estáticas geradas pelo uso de softwares;
* Heap: Responsável por armazenar os dados para alocação dinâmica: calloc, malloc, free;
* Stack: Responsável por mater a pilha de execução do processo;

Capítulo 15 - Hardware de memória

P. 21, questão 1.

Endereço lógico é o endereço obtido quando o software está em execução, como endereços lógicos iguais podem existir endereços físicos divergentes entre sí, uma vez que, softwares podem está contidos em espaços de endereçamento distintos. Logo, o endereço lógico é o endereço a nível de software, gerado na fase de compilação, o mesmo visualiza a memória como parte única para o software. A realocação dinâmica faz-se necessário utilização de um endereço base, em outras palavras, o endereço fisico e os endereços lógicos como offset, obtendo-se o endereço físico para cada endereço lógico. Portanto, o endereço físico acaba se tornando um endereço que representa uma localização válida e real na memória.

P. 21, questão 7.

Segmento Lógico	**0**	**1**	**2**	**3**	**4**
Offset Lógico	45	100	90	1.900	200
Endereço Físico	89	300	90	Violação de segmentação	1.400

P. 21, questão 8.

Página	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Quadro	3 - 4.000	12	6	-	9	741	2 - 1.995 - 6.633	-	0	5	-	1.995	-	7	414	1

Capítulo 16 - Alocação de memória

P. 9, questão 5.

Alternativa correta: B)

Capítulo 17 - Paginação em disco

P. 19, questão 1.

Uma falta de página é uma exceção disparada **(throw)** pelo hardware quando um programa acessa uma página mapeada no espaço de memória virtual, mas que não foi carregada na memória física do computador.

Possíveis causas:

1. A página correspondente ao endereço requisitado não está carregada na memória;
1. A página correspondente ao endereço de memória acessado está carregada, mas o seu estado corrente não foi atualizado no hardware;
1. A página não é parte do programa, e portanto não é mapeada na memória do programa;
1. O programa não tem privilégios suficientes para ler ou escrever a página;
1. O acesso à página é legal, mas está mapeada como página sob demanda;

Geralmente, o programa que recebe a exceção deve tratá-la, caso contrário, o S.O realiza um padrão já implementado para tratar, deste modo, executa o término do processo que causou e mostra ao usuário que o programa apresenta um mal funcionamento, dependendo do S.O essa mensagem pode ser mostrando de diferentes maneiras, entretanto, todas com a mesma finalidade.

Capítulo 19 - Hardware de entrada/saída

Para obter rapidez, a maioria dos sistemas operacionais implementa o tratamento de interrupções em dois níveis: um tratador primário (FLIH - First-Level Interrupt Handler) e um tratador secundário (SLIH - Second-Level Interrupt Handler). O tratador primário, também chamado hard/fast interrupt handler ou ainda top-half handler, é ativado a cada interrupção recebida e executa rapidamente, com as demais interrupções desabilitadas. Sua tarefa consiste em reconhecer a ocorrência da interrupção junto ao controlador de interrupções, criar um descritor de evento contendo os dados da interrupção ocorrida, inserir esse descritor em uma fila de eventos pendentes junto ao driver do respectivo dispositivo e notificar o driver. O tratador secundário, também conhecido como soft/slow interrupt handler ou ainda bottom-half handler, tem por objetivo tratar os eventos pendentes registrados pelo tratador primário. Ele geralmente é escalonado como uma thread de núcleo com alta prioridade, executando quando um processador estiver disponível. Embora mais complexa, esta estrutura em dois nível traz vantagens: ao permitir um tratamento mais rápido de cada interrupção, ela minimiza o risco de perder interrupções simultâneas; além disso, a fila de eventos pendentes pode ser analisada para remover eventos redundantes (como atualizações consecutivas de posição do mouse).