LISTA1

1. Quais são as três principais finalidades de um sistema operacional?
   - executar os programas de usuário de maneira simples
   - tornar o sistema do computador conveniente para o uso
   - utilizar os recursos de hardware de forma eficiente

2. Por que a abstração de recursos é importante para os desenvolvedores de aplicações? Ela tem alguma utilidade para os desenvolvedores do próprio sistema operacional?

A abstração de recursos no desenvolvimento de uma aplicação é importante para a abrangência de utilização que aquela aplicação poderá ter. Com uma maior abstração de recursos, aquela aplicação poderá ser aplicada para diferentes sistemas operacionais e diferentes disponibilidades de recursos de hardware. A abstração de recursos é particularmente importante para os desenvolvedores do SO, uma vez que, com abstraçõe̊s do tipo interfaces e terminais, a conexão aplicação-desenvolvedores fica mais compreensível, diferentemente do que em uma conexão por linguagem de máquina.

3. O que caracteriza um sistema operacional de tempo real?

Um sistema operacional de tempo real é caracterizado pela estipulação de períodos de tempo para todas as tarefas a serem executadas. Este tipo de sistema operacional se divide em dois, sendo o primeiro o hard-real-time, que não dá flexibilidade quando o tempo estipulado é extrapolado, e o soft-real-time, que dá margens de aceitação quando o período de tempo é extrapolado.

4. O que diferencia o núcleo (kernel) do restante do sistema operacional?

O kernel é o núcleo do SO, ele é o conjunto de rotinas que gerencia os processos do SO, bem como o escalonamento de memória e utilização do hardware.

5. Para permitir diferenciar os privilégios de execução dos diferentes tipos de software, os processadores modernos implementam níveis de privilégio de execução. Quais são eles?

Temos dois níveis de privilégio de execução, o de usuário e o de supervisor. O nível de usuário não consegue executar instruções privilegiadas, apenas o supervisor. Essa medida é criada para assegurar mais segurança ao SO, de forma que aplicações de usuário não causem danos ao sistema.

6. Seria possível construir um sistema operacional seguro usando um processador que não tenha níveis de privilégio? Por quê?

Não, pois sem os níveis de privilégio torna-se muito mais provável que uma aplicação de usuário interfira em tarefas cruciais do sistema. Além disso, não teríamos instruções reservadas para os níveis de gerência. Então tudo dependeria do usuário para que não houvesse sobreposição de execuções.

7. O comando em linguagem C fopen é uma chamada de sistema ou uma função de biblioteca? Por quê?

O comando fopen() é uma chamada de sistema feita por uma biblioteca em C, pois faz o gerenciamento de arquivos e a alteração dos mesmos.

8. Quais vantagens e desvantagens dos sistemas monolíticos? E dos Sistemas em Camadas?

- monolíticos:

As vantagens são que possui maior rapidez e desempenho
As desvantagens são que a manutenção e evolução são custosas.

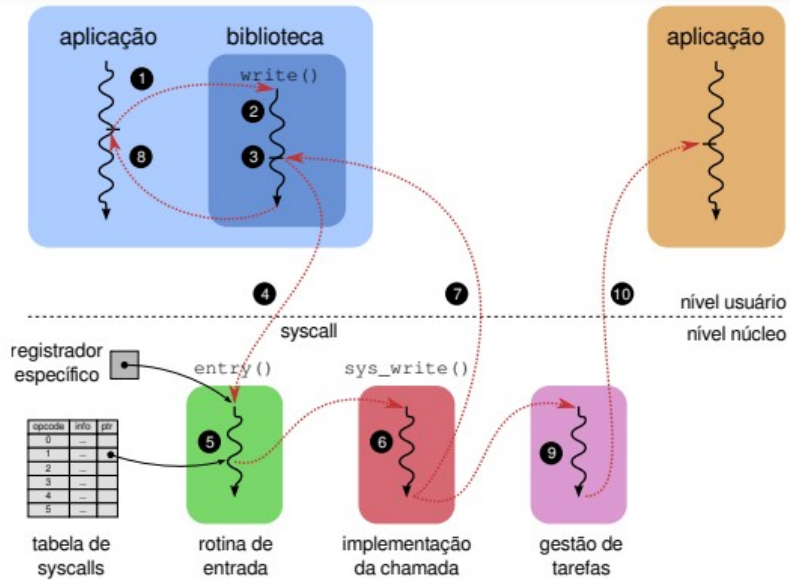
- em camadas:

As vantagens são que a manutenção e evolução do kernel é mais fácil e menos custosa.
As desvantagens são que possui menor desempenho e uma camada apenas se comunica com as camadas adjacentes, então é mais demorada a comunicação usuario-hardware.

9. Quais as diferenças entre interrupções e exceções em SO? Dica: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-02.pdf
As interrupções em SO são feitas por agentes externos à SO, por exemplo os dispositivos de entrada e saída. As interrupções acontecem cotidianamente.
Já as exceções são feitas pela própria SO e acontecem apenas quando houve algum problema, por exemplo divisão por 0. Assim, acontecem mais raramente.

10.A figura abaixo detalha o funcionamento básico da chamada de sistema write, que escreve dados em um arquivo previamente aberto. Descreva brevemente cada etapa.

Dica:
http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-02.pdf



1 - A aplicação utiliza o comando write() da biblioteca em C
2 - A função write preenche os registradores da CPU com os parâmetros recebidos
3 - A função invoca uma chamada de sistema através da instrução syscall
4- O processador troca para o nível kernel/núcleo
5 - A rotina entry() recebe o upcode da operação da função write, invoca a tabela de syscalls, acha a função sys_write na tabela e faz a implementação da chamada.

6- A função sys_write() verifica os parâmetros recebidos e efetua a operação.
7- Ao final da execução o retorno é escrito nos registradores, e o processador retorna para a write() em modo usuário.
8 - Finaliza a write() e retorna para o código.
9- quando a operação sys_write() não pode ser executada imediatamente ela encaminha para o gestor de tarefas. Um exemplo de quando isso ocorre é quando existe uma entrada de teclado.
10 - O gestor de tarefas suspende a execução da operação e finaliza a execução de alguma outra operação em que também estava ocorreu chamada de sistema.

11.Descreva em oito linhas, uma linha para cada uma das oito etapas do processo de uma aplicação ler dados de um arquivo, no sistema Minix 3, em Sistemas Micronúcleo.

1. a aplicação envia msg1 ao servidor de arquivos, pedindo a leitura de dados de um arquivo;
2. o servidor de arquivos verifica se os dados estão na cache; se não estiver envia msg2 ao driver de disco solicitando a leitura dos dados do disco em seu espaço de memória;

3. o driver de disco envia msg3 ao núcleo solicitando operações nas portas de entrada/saída do controlador de disco, para ler dados do mesmo;
4. o núcleo verifica se o driver de disco tem permissão para usar as portas de entrada/saída e agenda a operação solicitada;
5. quando o disco concluir a operação, o núcleo transfere os dados lidos para a memória do driver de disco e responde msg4 a este;
6. o driver de disco solicita msg5 então ao núcleo a cópia dos dados recebidos para a memória do servidor de arquivos e responde msg6 à mensagem anterior deste, informando a conclusão da operação;
7. o servidor de arquivos solicita msg7 ao núcleo a cópia dos dados recebidos para a memória da aplicação e responde msg8 à mensagem anterior desta
8. a aplicação recebe a resposta de sua solicitação e usa os dados lidos.

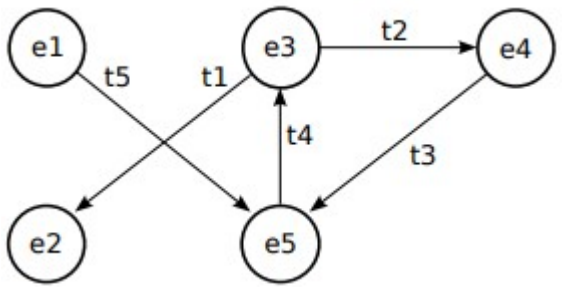
(dica: http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-03.pdf)

12.Além das arquiteturas clássicas (monolítica, em camadas, micronúcleo), recentemente surgiram várias propostas para organizar os componentes do sistema operacional. Descreva as diferenças de Máquinas Virtuais e Contêineres.
(dica: http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-03.pdf)

Nos contêineres o espaço do usuário no SO é dividido, e em cada divisão se localiza uma parte dos recursos do SO, e essas divisões são isoladas entre si. Processos não podem se comunicar entre divisões/domínios/containers.

As máquinas virtuais, são uma simulação de um sistema operacional sobre outro, elas dividem o hardware entre todos os sistemas operacionais criados, de modo que não haja sobreposição entre eles.

13.Considere o Diagrama de Estados dos processos e identifique todos os estados (e1 até e5), identifique e descreva quando ocorre as mudanças de estados (t1 até t5).
Indique se cada uma das transições de estado de tarefas a seguir é possível ou não. Se a transição for possível, dê um exemplo de situação na qual ela ocorre (N: Nova, P: pronta, E: executando, S: suspensa, T: terminada).
a) E → P ok
c) S → E ok
e) S → T não
g) N → S não
b) E → S ok
d) P → N não
f) E → T ok
h) P → S não



e1: novo
e2: terminada
e3: executando
e4: suspensa
e5: pronto

14.Relacione as afirmações abaixo aos respectivos estados no ciclo de vida das tarefas (N: Nova, P: Pronta, E: Executando, S: Suspensa, T: Terminada):
a) [N] O código da tarefa está sendo carregado.
b) [P] A tarefas são ordenadas por prioridades.
c) [S] A tarefa sai deste estado ao solicitar uma operação de entrada/saída.
d) [N] Os recursos usados pela tarefa são devolvidos ao sistema.
e) [S] A tarefa vai a este estado ao terminar seu quantum.S
f) [P] A tarefa só precisa do processador para poder executar.P
g) [S] O acesso a um semáforo em uso pode levar a tarefa a este estado.
h) [E] A tarefa pode criar novas tarefas.
i) [E] Há uma tarefa neste estado para cada processador do sistema.
j) [S] A tarefa aguarda a ocorrência de um evento externo.

15.O que significa “trocas de contexto”? Dê um exemplo com todos os passos en-

volvidos. (Dica:http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-05.pdf)
Contexto é o estado de uma tarefa em um determinado instante. Temos 5 classificações: Nova, pronta, executando, suspensa, terminada.

16.Qual a diferença de programa e processo?
Um programa é uma aplicação criada com o objetivo de facilitar o trabalho dos desenvolvedores na comunicação software-hardware. Um processo é a execução do programa na memória.

17.Explique o que é, para que serve e o que contém um TCB - Task Control Block.
Um TCB estrutura de dados em formato de lista encadeada que é um descritor de tarefas, e serve para armazenar informação importantes sobre as tarefas e seus estados. Um tcb armazena um identificador de tarefa, o estado da mesma, as informações de contexto do processador, a lista de áreas de memória utilizadas pela tarefa, a lista de arquivos abertos e as informações de gerência e contabilização da tarefa.

18.Cite os componentes de um processo.

19.O que são Threads e para que servem?

Threads é basicamente um fluxo de execução de tarefas que operam dentro de um mesmo processo, ou seja, tarefas que são executadas dentro de um mesmo contexto. Por exemplo, um aplicativo 'x' de edição de fotos terá várias tarefas sendo executadas simultaneamente no mesmo contexto.

20.Qual a relação de Thread com Processo? Existe vantagem/desvantagem? Cite.

R: Uma das vantagens que pode ser citada diz respeito ao ganho de desempenho do sistema. Já uma desvantagem é o aumento da complexidade de implementação.

21. Quais os tipos de threads?

many-to-one: várias threads de usuário para uma de kernel

one-to-one: uma thread de usuário para uma de kernel

many-to-many: várias threads de usuário para várias de kernel

22.Preencha a tabela abaixo:

Característica	Com processos	Com *threads* (1:1)	Híbrido
Custo de criação de tarefas			
Troca de contexto			
Uso de memória			
Compartilhamento de dados entre tarefas			
Robustez			
Segurança			
Exemplos			

(Dica: http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-05.pdf).

	alto	baixo	medio
	lenta	rápida	variavel
	alto	baixo	médio
	canais de comunicação e áreas de memoria compartilhada.	variáveis globais e dinâmicas.	ambos.
	alta, um erro fica con- tido no processo.	baixa, um erro pode afetar todas as threads.	média, um erro pode afetar as threads no mesmo processo.
	alta, cada processo pode executar com usuários e permissões distintas. baixa,	todas as threads herdam as permissões do processo onde exe- cutam.	alta, threads que necessi- tam as mesmas permis- sões podem ser agru- padas em um mesmo processo.
	Servidor Apache (ver- sões 1.*), SGBD Post- Gres	Servidor Apache (ver- sões 2.*), SGBD MySQL	Navegadores Chrome e Firefox, SGBD Oracle

LISTA2

1.Descreva brevemente as diferenças de um processo em relação a um programa.
Um programa é uma aplicação criada com o objetivo de facilitar o trabalho dos desenvolvedores na comunicação software-hardware. Um processo é a execução do programa na memória. IGUAL A 16^.

2.Cite e explique os estados possíveis de um processo.
• Novo: o processo está sendo criado
• Em execução: as instruções estão sendo executadas
• Em espera: o processo está esperando a ocorrência de algum evento
• Bloqueado: o processo foi bloqueado devido a algum evento
• Pronto: o processo está esperando para ser atribuído a um processador
• Encerrado: o processo terminou a execução
• Zumbi: o processo não consegue obter o uso de CPU

3.Para que servem os escalonadores? Cite no mínimo 4 motivos para utilizá-los.

- aumentar a vazão do sistema
- priorizar a execução de processos críticos
- manter o processador ocupado a maior parte do tempo
- oferecer tempos de respostas razoáveis

4.O que é preempção de processos?
Preempção de processos é quando o SO interrompe ou limita o tempo que o processo terá acesso à CPU.

5.Quais tipos básicos de escalonamentos?
Os tipos básicos de escalonamento são: CPU-bound e IO_bound, preemptivo e não preemptivo, processamento em lote, processamento interativo e processamento de tempo real.

6.Crie um quadro mostrando funcionamento, vantagens e desvantagens dos algoritmos First-
come Firts-served, Shortest Job First, Escalonamento por Prioridade, Round Robin e Múltiplas filas .

algoritmo	vantagem	desvantagem
FCFS	simples	um processo curto pode ficar atrás de um processo muito longo
SJF	maior vazão turnaround de processos curtos é menor do q em FCFS	starvation de processos mais longos
prioridade	os de alta prioridade dão a resposta logo	starvation de processos de baixa prioridade

RR	dá vez a todos os processos justamente na mesma fatia de tempo	processos de IO-bound curtos podem ser prejudicados se não usarem o quantum todo
múltiplas filas	fácil classificação de processos em grupos e baixo custo	inflexível

7.Descreva sobre escalonamento de tempo real crítico e não-crítico. Quais diferenças?
crítico: prioriza os críticos, se der para fazer no tempo, faz, se não, rejeita.
não crítico: tem que ter uma heurística de fora pra poder priorizar os críticos, se não, pode gerar starvation.

8.Qual a diferença de Escalonamento por Prioridades Fixas e Escalonamento por Prioridades Dinâmicas. Qual é o melhor escalonamento?
(Dica: http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-06.pdf)
O de prioridades fixas pode levar a starvation dos processos de menor prioridade. O de prioridades dinâmicas pode utilizar de técnicas, como por exemplo aging, para aumentar a prioridade de processos que estão há muito tempo na fila. Assim, procurando atender a todos os processos.

9.Ao definir Prioridades, quais fatores internos e externos podem influenciar?
Fatores internos: idade da tarefa, duração, tempo de espera.
Fatores externos: classe do usuário e importância da tarefa.

10.Explique o que é, para que serve e como funciona a técnica de aging.
A técnica de aging é uma técnica utilizada para amenizar a starvation de alguns processos que a utilização dos diferentes algoritmos disponíveis pode gerar. Ela tem o principio de aumentar a prioridade de uma tarefa ou processo à medida do aumento de tempo que ele vem esperando pelo acesso à CPU.

11. A tabela a seguir representa um conjunto de tarefas prontas para utilizar um processador:
Represente graficamente a sequência de execução das tarefas e calcule os tempos médios de vida (tourn-around time) e de espera (waiting time), para as políticas de escalonamento

Tarefa	t_1	t_2	t_3	t_4	t_5
ingresso	0	0	3	5	7
duração	5	4	5	6	4
prioridade	2	3	5	9	6

a seguir:
(a) FCFS cooperativa
(b) SJF cooperativa
(c) SJF preemptiva (SRTF)
(d) PRIO cooperativa
(e) PRIO preemptiva
(f) RR com tq = 2, sem envelhecimento
(g) SRTF
Considerações: todas as tarefas são orientadas a processamento; as trocas de contexto têm duração nula; em eventuais empates (idade, prioridade, duração, etc), a tarefa ti com menor i prevalece; valores maiores de prioridade indicam maior prioridade.
(Dica:http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-06.pdf)

12. Idem ao exercício anterior, para as tarefas da tabela a seguir:

Tarefa	t_1	t_2	t_3	t_4	t_5
ingresso	0	0	1	7	11
duração	5	6	2	6	4
prioridade	2	3	4	7	9

(n teve lista 3 da semana 4)
LISTA4 - semana 5

Cód questão 4:

```
/*Crie um programa para realizar a soma de duas matrizes 50x50 (números
aleatórios). Faça uma abordagem sem uso de pthread, depois crie uma
versão com
uso de 2 thread (cada thread irá tratar metade das linhas. Por exemplo,
impares x
pares ou as 25 primeiras para um e as restantes para a outra thread) .
Compare o
tempo de execução de cada uma..
```

```c
*/
#include <stdio.h>
#include <unistd.h> //Para utilizar sleep
#include <pthread.h>
#include <stdlib.h>
#include <time.h>

#define RANGE 3

int m1[6][6],m2[6][6];

void *somaMatrizes(void *threadid){
   int tid = *(int*)threadid, i, j;

   printf("\nlinhas de %d a %d :\n", (tid*RANGE)    ,   (RANGE*(tid+1))
);


   for(  i=(tid*RANGE)     ;i<(RANGE*(tid+1))     ;i++){
       for(  j=(tid*RANGE)     ;j<(RANGE*(tid+1))     ;j++){
       printf("%d, ", (m1[i][j] + m2[i][j]) );
       }
       printf("\n");
   }

   pthread_exit(NULL);
}


int main (void){
   pthread_t threads[2];
   int rc,t;
   srand(time(0));

   int i,j;
   for(i=0;i<6;i++){
       for(j=0;j<6;j++){
           m1[i][j] = rand()%100;
           m2[i][j] = rand()%100;
       }
   }

   printf("MAT1\n");
```

```
    for(i=0;i<6;i++){
        for(j=0;j<6;j++){
            printf("%d, ",m1[i][j]);
        }
        printf("\n");
    }
    printf("\n\nMAT2\n");
    for(i=0;i<6;i++){
        for(j=0;j<6;j++){
            printf("%d, ",m2[i][j]);
        }
        printf("\n");
    }

    printf("\nSOMA\n");
    for(t=0; t<2; t++){
        rc = pthread_create(&threads[t], NULL, somaMatrizes, (void
*)&t);

        if (rc){
            printf("ERRO; pthread_create() devolveu o erro %d\n", rc);
            exit(-1);
        }
    }


    for(t=0; t<2; t++){
        pthread_join(threads[t], NULL);
    }
}
```

(gambiaaaaarra, gambiaaaaaaaaarra)

fodam-se as outras questões

A lista da semana 6 e 7 são a mesma pq ele provavelmente colocou errado, e as duas falam sobre a matéria da semana 7.

Resumão semana 6 que n tem lista:

Região Crítica:

Região crítica é uma região compartilhada entre todos os processos em que, quando um processo está acessando, ninguém pode mais acessar. Mas todos os outros processos

ficam lá no busy waiting esperando pra poder acessar e não fazendo mais nada da vida, só comendo tempo da CPU.
Espera ocupada: Todas as soluções que nós temos de não deixar que um processo interfira na região crítica enquanto o outro tá usando não resolvem o busy waiting.

Além disso, não dá pra deixar a decisão na mão do processo em si de desbloquear a área quando for usar, bloquear quando tá usando, e desbloquear quando sair; pq tu não garante que depois q ele usar a CPU ele faça alguma coisa, ou seja a região crítica fica bloqueada a de eterno.

Todas as soluções que tentam resolver o problema tem que preencher pelo menos 3 requisitos para serem consideradas soluções ok:

- exclusão mútua: se um processo tá na região crítica ninguém mais tá.
- Progresso: nenhum processo bloqueia a região crítica para outro.
- espera limitada: nenhum processo pode ficar esperando pra sempre

Isso deve funcionar com qq nr de CPUs e threads.

CPU q bloqueia todo mundo quando um processo chega na região crítica.

Soluções

1. Variável de bloqueio: É uma bosta, não garante exclusão mútua. Flipa uma var pra true para o processo ter acesso, o cara acaba, era para flipar para false, n faz, e deixa a var em true para todo mundo usar a vontade.

2. Variável de comutação: adiciona turns. Se não tá no teu turno não entra, e muda o turno no final.

3. Comutação não alterada: Não precisa ficar alterando o turn depois de executar, add variável interested = true, e quando acabar deixa de ficar interessado. Satisfaz Exclusão mútua, progresso, espera limitada. (Algoritmo de Peterson é assim.)

4. TSL: A instrução TSL funciona como função e altera a var lock que dá acesso a região crítica. É uma solução atômica super específica para aquelas funções ali já que com as coisas modernas não se sabe a sequência das instruções pq se ordena da forma mais vantajosa.

Todas as soluções têm problema da espera ocupada.

Semáforos:

é uma variável pra ver se ajuda no caso das regiões críticas. É int >=0 e tem duas operações/funções pra sair e entrar na Região Crítica:

- wait() : quando quero entrar na reg. crítica. Faz semaforo = semaforo -1
- signal() : quando quero sair na reg. crítica. Faz semaforo = semaforo +1

Se o semáforo está em 0 bloqueia o processo pq tem gente lá dentro e tem q esperar a pessoa sair pra semáforo +=1. Aí fica em 1 tá liberado pode entrar, entrou semáforo -=1, virou 0 dnv. LOOP
não garante exclusão mútua.

Para garantir exclusão mútua usa o mutex e vira binário, 1 quando começa, 0 quando tem gente dentro, 1 quando sai.
Pode gerar:

- deadlock: um quer o que o outro tem e o outro quer oq o um tem.
- starvation, os dois esperam para sempre, acontece principalmente se n for FCFS.

Base para Comparação	Semáforo	Mutex
Conceitual	O semáforo é um mecanismo de sinalização.	Mutex é um mecanismo de bloqueio.
Existência	Semáforo é uma variável inteira.	Mutex é um objeto.
Função	O semáforo permite que vários encadeamentos de programa acessem uma instância finita de recursos.	O mutex permite que vários encadeamentos de programa acessem um único recurso, mas não simultaneamente.
Propriedade	O valor do semáforo pode ser alterado por qualquer processo de aquisição ou liberação do recurso.	O bloqueio de objeto mutex é liberado apenas pelo processo que adquiriu o bloqueio nele.
Categorizar	Semáforo pode ser categorizado em semáforo contador e semáforo binário.	Mutex não é categorizado.
Operação	O valor do semáforo é modificado usando a operação wait () e signal ().	O objeto Mutex é bloqueado ou desbloqueado pelo processo solicitando ou liberando o recurso.
Recursos Ocupados	Se todos os recursos estiverem sendo usados, o processo que solicita o recurso executa a operação wait () e bloqueia a si próprio até que a contagem do semáforo se torne maior que um.	Se um objeto mutex já estiver bloqueado, o processo que solicita recursos aguarda e é enfileirado pelo sistema até que o bloqueio seja liberado.

LISTA 6 - semana 7:

1. Explique as quatro condições necessárias para a ocorrência de impasses.
Exclusão mútua: pelo menos um recurso deve ser alocado em modo compartilhado
Posse e espera: o processo deve estar de posse de um recurso e esperando para adquirir recursos outro recurso, que é mantido por outro processo
Inexistência de preempção: o recurso só pode ser liberado voluntariamente
Espera circular: P0 espera por um recurso alocado a P1, P1 espera por P2, P2 por Pn-1 e Pn-1 espera por P0.

2. Na prevenção de impasses:
Se garantir q uma das 4 n ocorre não tem mais impasse.
a. Como pode ser feita a quebra da condição de posse e espera?
Pode gerar os problemas de starvation e/ou baixa taxa de utilização
– Usar um recurso de cada vez
– Obter todos os recursos antes de iniciar
b. Como pode ser feita a quebra da condição de exclusão mútua?
–Garantir que sempre exista recursos compartilháveis
– Reduzir áreas de exclusão ao mínimo
– Usar técnicas alternativas, como o spooling

c. Como pode ser feita a quebra da condição de espera circular?
algoritmos de ordenação
– Usar um recurso de cada vez
– Obter todos os recursos antes de iniciar
d. Como pode ser feita a quebra da condição de não-preempção?
quando o processo solicita um
recurso que está sendo usado, todos os recursos que ele já
possui sofrem preempção. O processo só entra na região
crítica quando consegue pegar todos os recursos
– “arrancar” os recursos dos processos
» Difícil de implementar, pode gerar inconsistências

3. Como pode ser detectada a ocorrência de impasses, considerando disponível apenas um recurso de cada tipo?
Vê se tem espera circular primeiro, depois vê o resto. (eu acho)

4.
Uma vez detectado um impasse, quais as abordagens possíveis para resolvê-lo? Explique-as e comente sua viabilidade.
Deve-se desfazer uma das condições como explicado na questão anterior. Essas soluções podem gerar starvation de processos ou diminuir a vazão.

5.
Sobre as afirmações a seguir, relativas a impasses, indique quais são incorretas, justificando sua resposta:

a.
Impasses ocorrem porque vários processos tentam usar o processador ao mesmo tempo.
processador n, recurso
b.
Sistemas operacionais atuais proveem vários recursos de baixo nível para o tratamento de impasses.
Normalmente eles ignoram, é muito caro tentar resolver caso a caso. Fodasse o programador.

c.
Podemos encontrar impasses em sistemas de processos que interagem unicamente por mensagens.
Sim exemplo cliente servidor.
d.
As condições necessárias para a ocorrência de impasses são também suficientes se houver somente um recurso de cada tipo no conjunto de processos considerado.
sim

6.
Sobre as afirmações a seguir, relativas às condições para ocorrência de impasses, indique quais são incorretas, justificando sua resposta:

a.
As condições necessárias para a ocorrência de impasses são: exclusão mútua, posse e espera, não-preempção e espera circular.
sim
b.
O principal problema com a quebra da condição de posse e espera é que a taxa de uso dos recursos pode se tornar bastante baixa.
sim, baixa vazão, casos extremos starvation.

c.
A condição de não-preempção indica que os processos envolvidos no impasse devem ser escalonados de forma não-preemptiva.
n, se for n preemptiva ele vai esperar pra sempre pq tá em um impasse. Arranca o recurso de todo mundo e pronto.

d.
A condição de não-preempção pode ser detectada graficamente, no grafo de alocação de recursos.
pode, todo mundo segurando todos os recursos e um esperando o outro soltar sendo q o outro está esperando o um soltar.
e.
A condição de exclusão mútua pode ser quebrada através do uso de processos gerenciadores de recursos ou de áreas de spool.
sim

f.
A quebra da condição de não-preempção só pode ser aplicada a recursos simples como arquivos e semáforos.
kinda, ye. É difícil de implementar.
g.
A quebra da condição de posse e espera consiste em forçar todos os processos a solicitar seus recursos em uma ordem global única e pré-fixada.
n, ou cada 1 pega 1, ou 1 pega todos no inicio

7. Sobre as afirmações a seguir, relativas à detecção e resolução de impasses, indique quais são incorretas, justificando sua resposta:
a.
A detecção e recuperação de impasses é bastante usada, pois as técnicas de recuperação são facilmente aplicáveis.
não é n, custa pacas.
b.
A resolução de impasses através de rollback só pode ser implementada em processos que executem I/O ou interação com o usuário.

nao, rollback n dá pra ser usada justamente em interações com o usuario. Se ficar voltando os pedidos de recurso dos processos até ficar num estado seguro e tu passar por um input de usuario n tem como salvar qual foi aquele input nem pedir dnv.

c.

Uma vez detectado um impasse, ele pode ser facilmente resolvido através da preempção dos recursos envolvidos.

dificil implementar

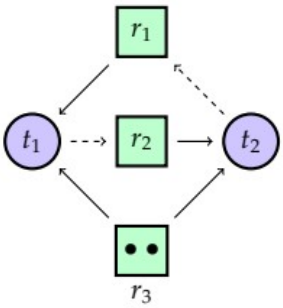
d.

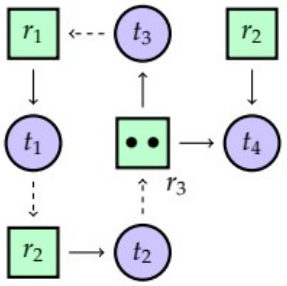
O algoritmo de detecção de impasses deve ser executado com a maior frequência possível, a fim de evitar que um impasse já formado se alastre.

acho q s

8.

Nos grafos de alocação de recursos, indique o(s) ciclo(s) com impasse:





a)

t2 quer r1 mas t1 tem r1

t1 quer r2 mas t2 tem r2

b)

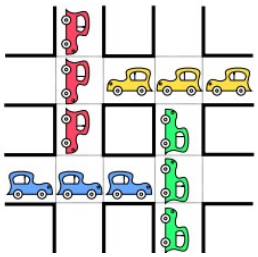
t3 quer r1 que é de t1

t1 quer r2 que é de t2

t2 quer r3 que é de t3 e de t4

t4 executa e libera r3, quebra o ciclo

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.



Exclusão mútua: pelo menos um recurso deve ser alocado em modo compartilhado

Exclusão mútua: vermelho tem pra baixo e amarelo quer ir pra baixo.

Posse e espera: o processo deve estar de posse de um recurso e esperando para adquirir outro recurso, que é mantido por outro processo

Posse e espera: vermelho tem pra baixo querendo pra direita, e é o azul que tem direita

Inexistência de preempção: o recurso só pode ser liberado voluntariamente
inexistência de preempção: só sai da pista quando conseguir pegar outra

Espera circular: P0 espera por um recurso alocado a P1, P1 espera por P2, P2 por Pn-1 e Pn-1 espera por P0.

Espera circular: vermelho espera recurso de azul, azul espera recurso de verde, verde espera recurso de amarelo, amarelo espera recurso de vermelho

Técnica de evitar impasse: é posse e espera, espera circular e preempção de uma vez só: vai um por vez. Vai vermelho, vai azul, vai verde, vai amarelo.

10.
O trecho de código apresenta uma solução para o problema do jantar dos filósofos, mas ele está sujeito a impasses. Explique como o impasse pode ocorrer. A seguir, modifique o código para que ele funcione corretamente e explique sua solução.

```
1  #define N 5
2
3  sem_t garfo[5] ; // 5 semáforos iniciados em 1
4
5  void filosofo (int i) // 5 threads (i varia de 0 a 4)
6  {
7     while (1)
8     {
9        medita ();
10       sem_down (garfo [i]) ;
11       sem_down (garfo [(i+1) % N]) ;
12       come ();
13       sem_up (garfo [i]) ;
14       sem_up (garfo [(i+1) % N]) ;
15    }
16 }
```

implementação de barreiras, faz alguém esperar, último slide de todos do último pdf.
Slide sobre barreira e ideia de solução do jantar de filósofos