**2. Compartilhamento de informações:** como diversos usuários podem estar interessados na mesma informação é necessário prover um ambiente para acesso concorrente as mesmas informações.

**Agilidade na computação e modularidade:** dividir uma tarefa em sub-tarefas -> execução mais rápida.

**Conveniência:** trabalha com diversos processos ao mesmo tempo. Usuário pode editar, imprimir e compilar o mesmo programa ao mesmo tempo.

**3.** **Memória compartilhada:** requer a comunicação entre processos para estabelecer uma região de memória que é compartilhada pelos processos cooperativos. Os processos então escrevem ou lêem dados dessa região compartilhada, sendo esses responsáveis por garantir que não escreverão no mesmo local simultanemante.

**Troca de mensagens:** o Sistema Operacional oferece meios para que os processos cooperativos se comuniquem e sincronizem por meio de um recurso de troca de mensagem. É útil para trocas de quantidades menores de dados pois não precisam ser evitados conflitos.

**4. Comunicação direta:** os processos precisam nomear explicitamente um ao outro.

**Comunicação Indireta:** as mensagens são enviadas e recebidas por caixas de correio ou portas.

**Sincronismo:** as mensagens podem ser trocadas com as primitivas send/ receive.

**6. Capacidade zero:** fila de tamanho 0. Emissor bloqueado até o destinatário receber a mensagem.

**Capacidade limitada:** fila com tamanho N- N mensagens podem residir nela

Se fila não estiver cheia: mensagem vai pra fila

Se fila cheia: emissor bloqueado até conseguir enviar

**Capacidade ilimitada:** fila de tamanho infinito. Emissor nunca é bloqueado.

**7. Fork():** duplica um processo filho.

**Wait():** essa chamada de sistema faz o processo pai ficar esperando até que seus filhos terminem.

**8.** Pois permite gerenciar o acesso concorrente a recursos do sistema operacional de forma controlada por parte dos processos, de maneira que um recurso não seja modificado em simultâneo, ou que os processos não fiquem em espera que o recurso seja libertado. É utilizado quando há compartilhamento de recursos.

**9. Condição de corrida:** condições onde 2 ou mais processos estão lendo ou escrevendo algum dado compartilhado e cujo resultado final depende de quem e quando executa.

**Seção crítica:** é a região onde o processo pode alterar variáveis comuns.

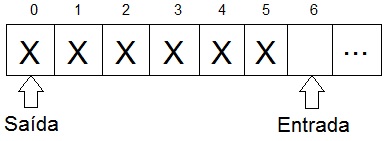
**Espera ocupada:** testar continuamente uma variável até que algum valor apareça.

**10.** Exemplo: Spool de impressão tem vagas numeradas 0,1,..

Cada uma capaz de ter um nome de arquivo. Há 2 variáveis:

Saída: que aponta para o próximo arquivo a ser impresso.

Entrada: aponta para a próxima vaga na fila.



1. Processo A quer imprimir.

2. Processo A entra com o nome do arquivo no spool de impressão.

3. Outro processo verifica periodicamente se há algum arquivo para ser impresso, se houver, imprime e então os remove da fila.

**11. Exclusão mútua:** nunca dois processos simultaneamente em uma região crítica.

**Progresso:** nenhum processo executando fora de sua região crítica pode bloquear outros processos.

**Espera limitada:** nenhum processo deve esperar eternamente para entrar em sua região crítica.

**12.** **Desabilitando interrupções:** a solução mais simples consiste em que cada processo desabilita todas as interrupções logo depois de entrar em sua região critica e as realibilita imediatamente antes de sair dela. Assim, um processo pode verificar e atualizar memória compartilhada sem temer a intervenção de um outro processo.

**TSL:** Operações seguramente indivisíveis

Lê o conteúdo da memória, a palavra lock, no registrador RX;

Armazena um valor diferente de zero no endereço de memória Lock

A CPU que está executando a instrução TSL impede oacesso ao barramento de memória para proibir queoutras CPUs tenham acesso a memória enquanto ela nãoterminar.

Desvantagens seriam a espera ocupada e starvation.

**13. Solução de Peterson:** É a combinação da idéia de TRAVA e TURN. É restrita a 2 processos que alternam a execução de suas regiões críticas.

Antes de usar as variáveis compartilhadas, cada processo chama ENTER\_REGION com seu número de processo e fica esperando até que seja seguro entrar.

Quando termina de executar o processo chama LEAVE\_REGION para indicar o término e permitir que outro entre.

**14. Semáforos:** ferramenta de sincronismo que não requer espera ocupada. As operações DOWN e UP são atomicas, ou seja, ou todas as operações executam ou nada executa, por isso funciona de forma mais eficiente.

#deifne N 100

Typedf int semaphore;

Semaphore mutex =1;

Semaphore empty =N;

Semaphore full = 0;

Void producer(void){

Int item;

While(true){

Item =produce item();

Down(&empty);

Down(&mutex);

Insert\_item(item);

Up(&mutex);

Up(&full);

}

}

Void consumer(void){

Int item;

While(true){

Down(&full);

Down(&mutex);

Item= remove\_item();

Up(&mutex);

Up(&empty);

Consume\_item(item);

}

}

**Monitor:** Unidade básica de sincronização de alto nível, todo monitor tem associado uma fila de threads

esperando para entrar.

Monitor ProducerConsumer

Condition full, empty;

Intereger count;

Procedure insert(item: integer);

Begin

If count = N then wait(full);

Insert\_item(item);

Count = count + 1;

If count =1 then signal(empty);

End;

Function remove: integer;

Begin

If count = 0 then wait(empty);

Remove = remove\_item;

Count = count – 1;

If count = N – 1 then signal(full);

End;

Count = 0;

End monitor;

Procedure producer.

Begin

While true do

Begin

Item = produce\_item;

ProducerConsumer.insert(item);

End

End;

Procedure consumer;

Begin

While true do

Begin

Item = ProducerConsumer.remove;

Consume\_item(item);

End

End;

**15. Jantar do filósofos:** Cinco filósofos sentam-se à mesa circular, passam a vida pensando ou comendo, para comer usam dois garfos: um de cada lado, cada garfo é compartilhado por dois filósofos.

**Leitores e escritores:** Alguns processos lêem o valor das variáveis compartilhadas: leitores, outros processos lêem e escrevem: escritores, eitores podem ler de modo concorrente, mas escritores só podem executar em exclusão mútua.

**Barbeiro:** A barbearia tem: um barbeiro, uma cadeira de barbeiro ealgumas cadeiras para os fregueses esperarem. Se o movimento estiver fraco (nenhum freguês), o barbeiro senta na sua cadeira e dorme. Quando um freguês chega no salão vazio, ele tem que acordar o barbeiro. Se outros fregueses chegam enquanto o barbeiro está ocupado, eles: ou sentam na cadeira de espera, ou vão embora, se não houver cadeira vazia.

**16. Deadlock:** um conjunto de processos está em situação de deadlock se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer.

**17. Condição de exclusão mútua:** todo recurso está ou associado a um processo ou

disponível (um recurso só é utilizado por 1 processo).

**Condição de posse e espera:** processos que retêm recursos podem solicitar novos

Recursos.

**Condição de não preempção:** recursos concedidos previamente não podem ser

forçosamente tomados.

**Condição de espera circular:** deve ser uma cadeia circular de 2 ou mais processos, cada um está à espera de recurso retido pelo membro seguinte dessa cadeia.

**19. Ignorar:** simplesmente ignora. É razoável quando deadlocks ocorrem raramente e o custo de prevenção é alto.

**Detecção e recuperação:** não tenta previnir, mas detecta na medida que ocorrem. Um algoritmo examina o estado do sistema para determinar se ocorreu deadlock, e outro algoritmo para recuperar o sistema. Existem duas soluções terminar ou preemptar recursos.

**Prevenção:** Evitar deadlocks é praticamente impossível, requer informações sobre requisições futuras, o que geralmente não se sabe.Para isso devemos negar as 4 condições de deadlocks. Exclusão mutua: usar filas. Posse e Espera: requisitar inicialmente todos os recursos. Não preempção: retomar recursos alocados Espera circular: ordenar numericamente os recursos.

**20.** **Terminar:** é uma maneira simples e grossa, consiste em abortar todos os processos em deadlock, que gera um alto custo computacional, ou então abortar um processo de cada vez até que o ciclo de deadlock seja eliminado, a cada processo terminado usa o algoritmo detecção para verificar se resolveu. A ordem para abortar seria: Prioridade do processo, tempo que está em execução e quanto falta para terminar, recurso que ele ja utilizou e recursos que ainda precisa e por fim quantos processos são dependentes e precisarão ser terminados também.

**Preemptar recursos:** alguns casos pode ser possível retirar provisoriamente um recurso de seu proprietário para dá-lo a outro processo. Algumas vezes, em sistemas em lote, é necessário uma intervenção manual. É uma solução altamente dependente do RECURSO. Deve-ser ter cuidado com STARVATION.

**21. Starvation:** durante o tratamento de deadlocks pode acontecer de nunca um processo obter seu recurso.

**24.** Errado. O certo seria: uma seção (ou região) crítica é um conceito em sistema operacional que faz referência a um trecho de código que manipula os dados compartilhados pelos processos.

**25.** Certo.

**26.** A.

**27.** A.

**28.** Errado. Pois além da exclusão mútua, precisa das condições de não preempção, espera circular e posse e espera.

**29.** B.

**30.** B.

**31.** D.

**32.** B.

**33.** C.

**34.** A.

**35.** C.

**36.** C.

**37.** Certo.