Resumo SO

Um **processo** é uma entidade ativa no sistema e representa um programa em execução.

Um processo é representado no SO por uma estrutura de dados chamada PCB. O PCB possui informações sobre o contexto de execução do processo. Algumas informações contidas no PCB:

-Id do processo
-Estado de execução
-Dados do contador de programas
-Dados dos registradores da CPU
-Info para o escalonamento de CPU, como prioridade por exemplo
-Informações sobre memória
-Informações sobre memória
-Informações de contabilidade
-Informações sobre as operações de entrada e saída

Cada SO possui chamadas de sistema que são responsáveis pela criação e destruição dos processos

-Fork () – cria processo novo
-Exit () – força destruição do processo
-Getpid () retorna id do processo
-Getppid() retorna o id do pai do processo
-Wait () aguarda execução do processo filho
-Sleep () aguarda um tempo sem segundos

Thread : Thread é o fluxo de execução de um processo. Processos podem ter mais de um thread (multithread). Cada thread pode executar uma tarefa específica dentro do processo. Threads são usadas para decompor um processo em partes menores com custo menor de troca de contexto e explorar múltiplos processadores.

MODELO N:1

Os primeiros SOs não reconheciam as threads, assim era necessário o uso de uma biblioteca de threads em nível de usuário. Neste modelo os threads implementados em nível de usuario são representadas por uma única thread no nível de kernel.

A vantagem deste modelo é que torna o SO mais eficiente no gerenciamento de threads.

A desvantagem é que se uma thread bloquear para realizar operações de entrada/saída, por exemplo, as demais também param.

MODELO 1:1

Neste modelo cada thread presente no nível de usuario tem uma correspondente no nível de kernel.

A vantagem deste modelo é que se uma thread bloquear, as demais continuam sendo escalonadas para serem executadas.

Desvantagem: torna o SO mais complexo em virtude do gerenciamento individual de cada thread.

MODELO N:M

Este modelo é uma junção dos dois anteriores. Neste caso, as N threads presentes no nível de usuario possuem M threads no nível de kernel M<N.

Vantagem: Thread continua sendo vista individualmente pelo SO

Desvantagem: Mais difícil implementar que 1:1 e N:1

Multithreading

Vantagens multithreading: Em um programa interativo é possível manter parte do programa executando enquanto outra parte dele estiver esperando bloqueada esperando uma operação E/S (capacidade de resposta).

Um sistema multithreading possui várias threads compartilhando o mesmo espaço de endereçamento (compartilhamento de recursos). A criação e o gerenciamento de threads pelo SO é mais econômico do que a criação e o gerenciamento de processos (economia). Se o computador possuir vários processadores (ou núcleos de processamento) é possível manter cada thread executando um processador distinto (escalabilidade). A troca de uma tarefa (processo ou thread) em execução, ou seja, troca de posse do processador que estiver com ela é definida como troca de contexto.

Os dados de contexto de uma tarefa são armazenados em sua PCB. A troca de contexto é realizada pelo despachante e a escolha de qual tarefa deverá assumir o processador é realizada pelo algoritmo de escalonamento de tarefas.

A tarefa (processo ou thread) pode estar em um dos seguintes estados:
-**Novo**: acabou de ser criada
-**Pronto**: pronta para ser executada
-**Executando**: está em execução
-**Aguardando**: aguardando um evento
-**Concluído**: execução concluída

Troca de contexto ocorre quando o tempo de execução do processo atual se esgotou, ocorrer uma interrupção de hardware, interrupção de software (execução de uma chamada de sistema chamada trap), ocorrer um erro de execução.

Em um sistema multiprogramado existe a necessidade de selecionar qual e por quanto tempo um determinado programa ocupará a CPU. Processos em execução alternam entre picos de CPU e picos de entrada/saída.

A CPU é escalonada quando um processo passa de execução para espera ou pronto, espera para pronto ou quando termina

Critérios para a escolha de um algoritmo escalonamento de recursos: utilização da cpu, quantidade de processos que são concluídos por unidade de tempo, intervalo de tempo a partir da criação do processo até a sua conclusão, tempo de espera e tempo de resposta

O escalonamento de threads pode ser realizado de duas maneiras:

No processo: a disputa pelo processador acontece no nível e processo, isto é, a biblioteca de gerenciamento de threads é responsável por determinar qual thrad entrará em execução

No sistema: A disputa pelo processador acontece no nível de sistema. O SO determinará qual thread entrará em execução no modelo 1:1 de gerenciamento de threads

Escalonamento em sistemas multiprocessadores: Abordagem assimétrica (mestre escravo): existe um processador mestre q manipula as estruturas de dados do sistema, os demais processadores (escravos) são destinados a execução de processos que o processador mestre determinar

Abordagem simétrica: cada processador é responsável pelo seu próprio escalonamento

Nesta abordagem pode existir uma única fila de prontos para todos os processadores ou cada processador pode ter sua própria fila de prontos. Deve haver mecanismos que impeçam que dois processadores não selecionem o mesmo processo para executar

Escalonamento em sistemas de tempo real: Um sistema de tempo real deve produzir resultados dentro de um limite de tempo (deadline)

Resultados após o deadline podem não ter valor real

Sistema de tempo real critico: possuem restrições de tempo mais perigosas, atrasos na execução de alguma tarefa não são permitidos. Uma tarefa que não cumpra o deadline pode ser considerada inútil, atrasos podem levar o sistema ao colapso

Sistema de tempo real não critico: Atrasos levam o sistema a degradação, mas não ao colapso, restrições de tempo mais brandas

Sistemas de tempo real, críticos ou não possuem: uso especifico, tamanho reduzido, produzidos em massa e requisitos de tempo especifico.

Tipos de tarefas em um sistema de tempo real

Periódicas: exigem a cpu em intervalos de tempos constantes. Possuem um tempo de processamento fixo t após adquirir a cpu, um deadline d durante o qual deverá ser atendido pela CPU, e um período p. 0<t<d<p. São usadas em sistemas de tempo real critico

Aperiódicas: Sua ativação reponde a eventos internos ou externos, definindo uma característica aleatória. São usadas em tempo real brando

Esporádicas: . Possuem como característica a restrição de um intervalo mínimo conhecido entre duas ativações consecutivas, por isso podem ter atributos de tarefas críticas. Usadas em sistemas de tempo real crítico.

Algoritmo de taxa monotônica: Define uma política de escalonamento baseada na alocação de prioridade estática com preempção. Cada tarefa recebe uma prioridade inversamente proporcional ao seu período, quanto menor o período, mais alta a prioridade. Considera o tempo de processamento de uma tarefa periódica é o mesmo em cada pico de CPU

Resumo SO pt2

Concorrência: é a condição de um sistema no qual múltiplas tarefas estão logicamente ativas em um determinado momento

Paralelismo: é a condição de um sistema no qual múltiplas tarefas estão realmente sendo executadas ao mesmo tempo.

Execução não paralela: alternância das tarefas, não são executadas ao mesmo tempo

Execução paralela: tarefas executadas ao mesmo tempo

Vantagens da programação concorrente: aplicações envolvendo grande quantidade de cálculos, aplicações criticas (deadlines), melhor uso de recursos computacionais (processador e memoria) e alta capacidade de processamento

Dificuldades programação concorrente: Desenvolvimento, corretude, debug

Ordem em que processos concorrentes executam não é determinística

Concorrencia: Processos cooperativos podem afetar outros processos em execução ou por ser eles afetados. Processos cooperativos pode: compartilhar um espaço de endereçamento lógico, compartilhar dados através de arquivos ou mensagens.

Processos produtor e consumidor:

Um recurso é compartilhado entre dois processos, o produtor que insere dados no buffer e o consumidor que retira dados do buffer

Um seção crítica é um segmento de código em que o processo pode estar: alternando variáveis comuns, atualizando uma tabela, gravando um arquivo...

Quando um processo estiver executando sua seção crítica, NENHUM, outro processo deve ter autorização para fazer o mesmo. Dois processos não podem estar simultaneamente nas seções críticas.

Cada processo deve solicitar permissão para entrar em sua seção crítica (protocolo da seção crítica)

Uma solução para o problema da seção crítica deve satisfazer os quatro requisitos a seguir:

-- Somente um processo por vez acessa a região critica

-- Ser independente do número e velocidade de CPU's

--Um processo fora da região crítica não bloqueia outro processo

--Nenhum processo espera indeterminadamente para acessar a região critica

Soluções região critica:

Soluções de algoritmo (algoritmo de lamport), hardware (inibição de interrupções) e SO (semáforos, variáveis condicionais, mutex)

Hardware de sincronização: Muitos sistemas de computação modernos fornecem instruções, de hardware especiais que nos permitem testar e modificar o conteúdo de uma palavra ou trocar os conteúdos de duas palavras atomicamente: TestAndSet: testa e modifica um valor de uma variável

Swap: Muda o valor de uma variável em uma única instrução

Abstração: muito usada para proteger regiões críticas, que evita a espera ocupada

Mecanismos de sincronização de processos

São variáveis que não armazenam valores específicos, mas representam condições que podem ser aguardadas por alguns processos.

Se um processo está aguardando uma condição, ele é inserido em uma fila de espera até que a condição seja verdadeira

As variáveis condicionais evitam espera ocupada

Monitores

Um monitor consiste de: recurso compartilhado, conjunto de variáveis ao monitor.

Um conjunto de procedimentos que permitem o acesso a essas variáveis

Um mutex ou semáforo para controlar a exclusão mutua. Cada procedimento de acesso ao recurso deve obter o semáforo antes de iniciar e liberar o semáforo ao concluir.

Um invariante (condição sobre as variáveis internas do monitor) sobre o estado interno do recurso

SO pt3

Processos independentes não compartilham dados com os demais processos

Processos cooperativos compartilham algum tipo de dado com um ou mais processos

A cooperação entre processo traz as seguintes facilidades:

compartilhamento de informações: acesso concorrente a dados, por exemplo, mais de um processo acessando um arquivo em disco.

Velocidade do processamento: a divisão de uma tarefa em subtarefas permite minimizar o tempo de processamento. O processamento paralelo acontece se houver mais de um processador disponível.

Modularidade: um sistema com muitas funcionalidades pode ser divididos em varias threads

Conveniência: um único usuario pode usufruir das vantagens do compartilhamento de informações entre processos

Modelos de Interação entre Produtores e Consumidores

Um modelo de interação entre processo é determinado por dois aspectos: Numero de processos interlocutores envolvidos na comunicação e papel desempenhado por cada um dos processos interlocutores

Modelo mestre escravo

Se baseia na associação estrita entre dois processos. O processo escravo (leitor) tem a sua atividade totalmente controlada pelo processo mestre (produtor). O canal de comunicação é fixo e a associação dos outros processos a este é pré-estabelecida. Isto é, cada um deve conhecer previamente a identificação do outro

Modelo Muitos para UM

Também conhecido como correio, se baseia na possibilidade de transferência de dados em modo assíncrono, sob a forma de mensagem. As mensagens são enviadas individualmente por um conjunto de processos produtores a um processo consumidor que está preparado para recebe-las. O canal de comunicação é criado previamente pelo processo consumidor e o seu nome é conhecido pelos processos produtores. O processo consumidor é visto como um servidor que atende as solicitações de vários clientes.

Modelo UM para UM de vários

Neste modelo é estabelecido um canal fixo entre dois processos, criado de forma dinâmica.

Um processo, normalmente um cliente, deve requisitar o estabelecimento da ligação enviando uma mensagem para um canal previamente criado pelo servidor. O resultado da ligação entre o cliente e o servidor é um novo canal ao qual o cliente e o novo processo (processo ou thread) servidor dedicado ficam automaticamente associados. A associação é temporária e durará apenas o tempo da interação entre o processo cliente e o processo servidor.

Modelo UM para muitos

Um processo produtor envia uma mesma informação para vários processos consumidores ou para um grupo de consumidores previamente identificado. Este tipo de comunicação é muito usado para notificações entre processos. O gerenciador de janelas utiliza esse mecanismo quando o usuario solicita o desligamento do computador, o gerenciador de janelas envia uma msg de encerramento para todas as aplicações em execução.

Modelo Muitos para muitos

Todos os processos podem ser produtores e consumidores de mensagens, alternando papeis.

Comunicação no modelo computacional:

A comunicação no modelo computacional é realizada por mecanismos disponibilizados pelo SO ou nos ambientes de programação.

A comunicação entre processos é suportada por um objeto do tipo canal de comunicação.

A transferência de informações entre processos pode ser vista como resultado da invocação de operações sobre um objeto canal.

As operações realizadas em um canal de comunicação podem ser:

Criar, associar, enviar, receber, terminar e eliminar.

As mensagens trafegadas em um canal de comunicação podem ser estruturadas de acordo com as vertentes**:**

Interna: determina a codificação dos dados trocados na comunicação. Nesta vertente, os canais podem ser: opacos; os dados tem de ser explicitamente codificados e interpretados pelos processo interlocutores. Estruturados: a comunicação impõe uma estrutura fixa para as mensagens trocadas ou então suporta a transferência de informação do tipo anexa aos dados no conteúdo das mensagens.

Externa: foca delimitação e a preservação das fronteiras entre as diferentes mensagens enviadas. Nesta vertente, os canais podem ser : mensagens pacote: a comunicação se realiza por meio de troca de mensagens individuais, cuja fronteira é preservada e imposta na recepção. Streams: a comunicação se processa através da escrita e da leitura de sequencias ordenadas de bytes.

A comunicação entre processos também determina um mecanismo para sincronizar as ações dos processos comunicantes. A semântica na comunicação de processos determina o comportamento do processo ao receber e enviar uma mensagem, e pode ser:

Síncrona: o produtor fica bloqueado até que o consumidor receba a mensagem e acesse o seu conteúdo

Assíncrona: o produtor envia a mensagem e continua a execução, assim que esta tenha sido armazenada no canal de forma temporária até que seja recebida pelo consumidor.

Cliente-servidor: o processo produtor (cliente) fica bloqueado até que o consumidor (servidor) tenha recebido a mensagem e enviado uma mensagem resposta.

MENSAGEM COMPARTILHADA:

Processos podem criar e associar áreas de memória compartilhada e mapeá-las em seu espaço de endereçamento. Um grupo de processos pode manipular a mesma área de memória sem que ocorram exceções devido a violação no espaço de endereçamento de cada um. As

áreas de memória compartilhada podem ser mapeadas no contexto de cada processo naturalmente, em diferentes espaços virtuais. Não é possível antecipar em qual endereço virtual o processo será alocado, desta fora o uso de memória compartilhada não permite trabalhar com estruturas do tipo listas encadeadas, por exemplo.

PIPE's

Um PIPE pode ser enquadrado como uma versão limitada das classes de mecanismos das conexões virtuais. O PIPE liga dois processo, o que permite o fluxo de informações de forma unidirecional. São adequados para a implementação de mecanismos mestre escravo (um para um). Os PIPEs podem ser anônimos ou nomeados (identificados). Um PIPE anônimo tem que ser usado entre processos que possuam relação hierárquica (pai e filho). Cada PIPE é representado por um arquivo especial no sistema de arquivos.

Resumo SO pt 4

Em sistemas multiprogramados, os processos solicitam acesso a recursos finitos. Um conjunto de processos está em deadlock se cada processo está bloqueado esperando um evento que somente pode ser causado por outro processo do conjunto. Deadlocks podem ocorrer quando vários processos recebem direito de acesso exclusivo a um recurso. Um processo solicita recursos obedecendo a seguinte ordem. Solicitação, uso e liberação.

Em uma situação de deadlock os processos envolvidos nunca concluem sua execução e os recursos ficam ocupados por tempo indeterminado, o que impede que outros processos os utilizem.

Situações que acarretam em deadlock: exclusão mutua, posse e espera (processos que usaram recursos anteriormente requisitam novos), não preempção (recursos liberados a força) e espera circular (um processo aguarda um recurso que está sendo usado pelo processo anterior na cadeia de processos)

Métodos para evitar deadlock

--Uso de protocolo de prevenção ou impedir a ocorrência de deadlocks

--Detectar um deadlock e executar um processo de recuperação

--Ignorar o deadlock

Estado seguro é aquele que não está em deadlock e que não corre risco de deadlock