Capítulo 2 – Processos e Threads

- Processo é uma abstração de um programa em execução.

2.1 – Processos

- Processos conferem simultaneidade.
- PseudoparalelismoXMultiprocessadores

2.1.1 – O modelo de processo

- Todos os softwares são organizados em processos sequenciais.
- Um processo é um programa em execução acompanhado de valores de registradores e variáveis.
- Multiprogramação é o modelo de troca rápida de processos.
- Processos não devem ser programados com hipóteses pré-definidas sobre o tempo de execução.

2.1.2 – Criação de processos

- Quatro eventos de criação de processos: Inicio do sistema, um systemcall cria um processo por um processo em execução, uma requisição do usuário, inicio de uma tarefa em lote.

2.1.3 – Término de processos

- Razões para término de processo: saída normal, saída por erro,erro fatal e cancelamento por outro processo.
- Processos podem dizer ao SO que querem tratar os erros, dessa forma não são finalizados quando um erro ocorre.

2.1.4 – Hierarquia de Processos

- Em alguns sistemas processos pais e processo filho continuam associados.
- Processos podem ser divididos em grupos relacionados À um dispositivo de IO por exemplo.
- Não há hierarquia de processos no Windows.

2.1.5 – Estados de processos

- Um processo bloqueia a espera de um recurso/ resultado ainda não disponível.
- Um processo pode está em 3 estados: Em execução, pronto e bloqueado.
- 4 transições são possíveis entre os estados:
    - Transição 1 ocorre quando o SO descobre que o processo não pode prosseguir. Sai de execução e entra em bloqueado.
    - Transições 2 e 3, de execução para pronto e vice versa. O escalonador de processo faz isso. A transição 2 ocorre com o fim do timeslice. A 3 ocorre quando chegou vez de um processo se executado.
    - Transição 4, ocorre quando o processo recebe o que estava aguardando.

2.1.6 – Implementação de processos

- O SO mantém uma tabela de processos com uma entrada par cada processo.
- Arranjo de interrupções contém o endereço das rotinas de tratamento de interrupções.

2.1.7 Modelando a multiprogramação

- Utilização de cpu = $1-p^n$

2.2 – THREADS

- Threads são fluxos de execução dentro de um processo eu podem executar em paralelo.
- Threads compartilham espaço de endereçamento e dados.
- Threads são mais fáceis de criar e destruir que processos, pois não tem muitos recursos associados a elas.
- Se só houverem processos CPU bound as threads não ajudam muito, mas em ambientes onde existe uma grande quantidade de processos IO bound as threads podem fazer a diferença.
- Threads são úteis em sistemas com múltiplas CPUS.

2.2.2 – O modelo de thread clássico

- O modelo de processos é baseado em agrupamento de recursos e execução.
- Podemos encarar um processo como um agrupamento de recursos comprtilhados por todas as threads daquele processo.

- Processos são utilizados para agrupar recursos e threads estão relacionadas com a execução na CPU.
- Threads = lightweight processes
- Não há proteção entre threads por ser impossível e não necessário.
- Um thread tem os mesmos estados do processo.
- Cada thread tem sua própria pilha(stack).
- Threads não tem interrupção de relógio.

2.2.3 - Threads Posix

2.2.4 – Implementação de threads no espaço do usuário

- Há dois modos de implementar threads, no espaço do usuário e no espaço do núcleo.
- Disfarçar as threads num processo e executar os processos como se fossem monothreads.
- Sistema de tempo de execução, conjunto de rotinas que gerenciam as threads.
- Cada processo tem sua própria tabela de threads.
- Fazer o chaveamento em threads é mais rápido que trocar o contexto completamente.
- As próprias threads chamam o escalonador sem precisar passar pelo núcleo.
- Threads de usuário permitem que algoritmos de escalonamento personalizado possam ser utilizados.
- Uma thread pode bloquear o processo inteiro.
- Threads não são boas para CPU bound.

2.2.5 – Implementação de threads no núcleo

- O núcleo gerencia as threads.
- O núcleo recicla threads.
- O custo de troca de contexto é alto.

2.2.6 – Implementações híbridas

- O núcleo ver apenas as threads de núcleo que escondem as threads de usuário.

2.3 – Comunicação entre processos

- Troca de mensagens, resolução de conflitos e ordem de execução.
- Problemas também aplicam-se as threads.

2.3.1 – Condições de corrida

- São situações que ocorrem quando um ou mais processo precisam ler ou escrever de dados compartilhados e isso potencialmente pode fazer um comportamento inesperado ocorrer.

2.3.2 – Regiões críticas

- Precisa-se ocorrer a exclusão mutua de códigos compartilhados.
- A região crítica é a área onde o acesso compartilhado ocorre.
- 4 condições precisam ser atendida das para uma boa solução da condição de corrida: dois processos nunca podem estar simultaneamente em suas regiões críticas, nada pode ser afirmado sobre o numero ou velocidade das cpus, nenhum processo fora da região critica pode bloquear outros processos, nenhum processo deve esperar eternamente para entrar na régio critica.

2.3.3 – Exclusão mútua com espera ociosa

Debilitando interrupções

- Cada processo desabilita todas as interrupções depois que entra na região crítica.
- O processo não será chaveado.
- Não é prudente permitir que processos do usuário desliguem as interrupções.
- Um processo pode não religar as interrupções.
- Não funciona para processadores co mais de uma CPU.
- O sistema desligando interrupções é uma boa abordagem.

Variáveis do tipo trava

- Um flag.
- Acontecerá condição de corrida nessa variável.

Chaveamento obrigatório

- Testa enquanto uma flag não muda de valor
- Aplica a espera ociosa.
- Spin lock é uma variável de trava que usa espera ociosa.

- Não é bom usar chaveamento quando os processos tem tempos de execução distintos.
- Além disso, viola o ponto 3 onde um processo não pode ser bloqueado por outro que não esteja na região crítica.

Solução de Peterson

- Parecido com semáforos

A instrução TSL

- Instrução que altera uma variável de trava atomicamente.

2.3.4 – Dormir e acordar

- As soluções de Peterson e TSL são boas ,mas tem o defeito da espera ociosa.
- Problema de inversão de prioridades.

O problema do produtor-consumidor

- Buffer limitado
- Dois processos compartilham um buffer comum de tamanho fixo.
- Problema quando o consumidor consumir tudo e precisar ir dormir ou quando o produtor produzir tudo e também precisar ir dormir.

2.3.5 - Semáforos

- Guarda os sinais de acordar.
- Propoe ações down e up atômicas

Resolvendo problema produtor-consumidor usando semáforos.

- Semáforos binários asseguram o uso de um trecho de código por apenas 1 processo de cada vez.
- Semáforos são usados para excluso mútua e sincronização.

2.36 – Mutexes

- Usados para gerenciar recursos compartilhados no qual você deseja a exclusão mútua.
- Mutex pode estar em, 2 estados: impedido e desempedido.
- Pode haver a possibilidade de processos compartilharem espaços de endereçamento

2.4 – Escalonamento

- O escalonador é a parte do sistema que escolhe qual o processo que será executado em seguida.

2.4.1 – Introdução ao escalonamento

- Antigamente o algoritmo era FIFO.
- Os algoritmos se tornaram mais complexos devido ao maior número de usuários esperando.
- Geralmente só existe um processo ativo, aquele que o usuário está usando em um determinado instante.
- O aumento da velocidade da CPU influenciou no modo como o escalonamento é feito.
- O escalonador deve se preocupar em fazer um uso eficiente da CPU, pois a troca de contexto é muito custosa.

Comportamento do processo

- Quase todos os processos alternam surtos de utilização de E/S e computação.
- CPU-bound X IO bound
- O fator determinante é o tamanho do surto de computação.
- A tendência da CPU ficar mais rápida muda muitos processos para IO bound, pois o tempo de leitura do disco se torna mais lento do que processar os dados lidos.

Quando escalonar

- Qual o momento certo para escalonar?
- Decisões sobre quem executar devem ser tomadas quando um novo processo é criado, quando um processo é terminado, e quando um processo é bloqueado.
- Existem duas categorias de algoritmos de escalonamento: não preemptivos e preemptivos.
- O algoritmo não preemptivos escolhe um processo e o deixa ser executado até terminar ou ser bloqueado.

- O algoritmo preemptivo usa o conceito de timeslice.

Categorias de algoritmos de escalonamento

- Para cada ambiente existe algoritmos diferentes. São 3 ambientes: Lote, interativo e tempo real.
- Nos sistemas em lote não há um usuário esperando imediatamente por uma resposta. Algoritmos não preemptiveis ou preempetiveis com grandes timeslices são boas táticas.
- Em ambientes imperativos a preempção é essencial.
- Em ambientes de tempo real a preempção é muitas vezes desnecessárias pois os processos são bem desenvolvidos e sabem quanto tempo devem executar.

Objetivos do algoritmo de escalonamento

- Todos os ambientes
    - Justiça – dar a cada processo uma porção justa da CPU.
    - Aplicação da política – Verificar se as políticas de sistema estão sendo aplicadas.
    - Equilibrio – Manter ocupada todas as partes do sistema
- Sistemas em lote
    - Vazão – maximizar o número de tarefas por hora
    - Tempo de retorno – minimizar o tempo entre a submissão e a entrega
    - Utilização de CPU – manter a CPU ocupada o tempo todo.
- Sistemas interativos
    - Tempo de resposta – responder rapidamente as requisições
    - Proposrcionalidade – Satisfazer as expectativas do usuário
- Sistemas de tempo real
    - Cumprimento de prazos – evitar perda de dados
    - Previsibilidade – evitar a degradação da qualidade em sistemas multimídia.

2.4.2 – Escalonamento em sistemas em lote

Primeiro a chegar, primeiro a ser servido (FIFO)

- O mais simples
- Não preemptivo
- A cpu é atribuída aos processos na ordem em que eles a requisitam.
- Processo executa até terminar ou bloquear. Quando ficar pronto volta para o fim da fila.
- Vantagem: Fácil de implementar e fácil de entender. Justo. Desvantagem: Processos com tempo de execução diferentes podem ser prejudicados. (Processos IO bound e CPU bound). Não leva em consideração o tipo de processo.

Tarefa mais curta primeiro

- Supõe como previamente conhecidos todos os tempos de execução dos processos envolvidos.]
- Executa a tarefa com menor tempo primeiro.
- Mais útil quando todas as tarefas estão disponíveis ao mesmo tempo (chegam ao mesmo tempo).

Próximo de menor tempo restante

- Versão preemptiva do tarefa mais curta primeiro.
- Interrompe a execução de uma tarefa em detrimento de uma outra mais curta.

2.4.3 – Escalonamento em sistemas interativos.

Escalonamento por chaveamento circular (round Robin)

- Atribui-se um timeslice ao processo e ele executa até o timeslice terminar.
- Usa uma fifo. É facíl de implementar.
- Um problema é o tempo gasto com troca de contexto.
- O tamanho do timslice é um ponto crucial nesse algoritmo, podendo diminuir o desempenho ou piorar a execução dos processos.
- Um timslice muito curto causa muita troca de contexto e degrada o desempenho, um timeslice muito longo pode causar uma má resposta (resposta lenta) às requisições feitas.

Escalonamento por prioridades

- Ao processo executável com prioridade mais alta é permitido executar.
- Para evitar que processos de alta prioridade executem indefinidamente, pode-se decrementar a prioridade dos processos em execução a cada intervalo de tempo.
- Pode-se usar um timeslice para evitar que processos menos prioritários fiquem sem executar

Filas múltiplas

- Separa os processos em classes de prioridades.

Próximo processo mais curto

- Estima-se o tamanho do tempo de execução do processo baseado em execuções passadas.

Escalonamento garantido

- A CPU é dividida justamente para cada processo.

Escalonamento por loteria

- Sorteio de quem vai executar.
- É altamente responsivo.

Escalonamento por fração justa (fair-share)

- Leva em consideração as propriedades e usuários dos processos.

2.4.4 – Escalonamento em sistemas de tempo real

- Sistemas e tempo real: crítico e não crítico.
- Algoritmo estáticos e dinâmicos.

2.4.5 – Política x mecanismo

- O escalonamento é parametrizados, mas os parâmetros podem ser decididos pelo o usuário.

2.4.6 – Escalonamento de threads

- Se as threads são de usuário
    - AS threads em um processo não influenciarão em outro.
    - Não há interrupção de relógio dentro do processo.
- Se as threads são de núcleo
    - Trata as threads como processos.
    - Threads de núcleo tem chaveamento mais complexo, pois são parecidas com processos.

2.5 – Problemas clássicos de IPC

2.5.1 – O problema do jantar dos filósofos

- Situações perigosa: Todos pegam 1 garfo e ninguém consegue pegar o próximo.
- Processos que competem para o acesso exclusivo à um número limitado de recursos.

2.5.2- Problema dos leitores e escritores

- Modelo o acesso à uma base de dados.