SO – Anotações

# Sistema Operacional:

É um programa ou um conjunto de programas interrelacionados cuja finalidade é atuar como: Intermediário (interface) entre o usuário e hardware, e gerenciador de recursos. 2 funções: fornece abstrações para os programas de usuários e gerencia os recursos do computador.

# Visões

Uma visão mais moderna trata SO como uma máquina virtual, além de ter 3 tipos:

Resource Manager – gerencia e aloca recursos;

Control program – controla a execução de programas usuários e operações de E/S;

Command Executer – fornece um ambiente para execução de comandos.

# Kernel

O sistema operacional é o único programa em execução o tempo todo no computador.

Programa de sistemas: geralmente associados ao sistema operacional, mas não necessariamente parte do kernel (núcleo);

Programa de aplicações: que inclui todos os programas não associado à operação do sistema.

* É possível interagir com o SO de diversas maneiras:
  + Usuário: linguagem de comando, interface gráfica;
  + Programa de usuário: Invoca o SO por meio de syscalls.
* Compartilhamento de HW:
  + Monoprogramado: monotarefa, somente um programa ativo por vez em dado período.
  + Multiprogramados: multitarefa, mais de um programa ao mesmo tempo por meio de compartilhamento de recurso (CPU, RAM etc.).

# SO Monotarefa

“Monoprogramado”. Permite que a CPU, memória e periféricos permaneçam exclusivamente dedicados a um único programa. Alocação de memória para apenas um processo, SO fica em uma camada protegida da memória.

Desvantagem: ociosidade do processador e subutilização da memória (processador fica ocioso quando o programa espera uma E/S).

Vantagem: simples implementação.

Exemplo: MS-DOS.

# SO Multitarefa

“Multiprogramado”. Vários programas em memória ao mesmo tempo e várias tarefas simultâneas. Logo, quando uma roda, a outra espera, e requer mecanismo de troca rápidas. Podem ser classificados de acordo com a forma com que suas aplicações são gerenciadas, podendo ser divididos em sistemas batch, de tempo compartilhado e de tempo real.

Desvantagem: implementação muito mais complexa.

Vantagem: redução do tempo de resposta das aplicações, além dos custos reduzidos devido ao compartilhamento dos recursos do sistema entre as diferentes aplicações.

Exemplos: Windows, Linux, Android.

# Multitarefas em Batch

Sistema em batch (lote), consiste em submeter ao computador um lote de programas de um só vez. Os jobs são submetidos em ordem sequencial para a execução e não existe interação entre o job e o usuário durante a execução.

# Multitarefas de tempo de compartilhamento

SO’s interativos. Onde o sistema permite que os usuários interajam com suas computações na forma de diálogo, e podem ser projetados como sistemas monousuários ou multiusuários (usando conceitos de multiprogramação e timesharing).

Timesharing: variante da multiprogramação, cada usuário tem um terminal online, onde o SO vê quando a CPU pode ser alocada para as tarefas que demandam serviços. Tem um intervalo de tempo fixo para os diferentes processos. Seu objetivo principal é o tempo de resposta interativo.

* + Vantagem: resposta rápida, reduz tempo ocioso da CPU, todas as tarefas recebem um tempo específico e menor probabilidade de duplicação de software.
  + Desvantagem: consumo de muitos recursos, requer alta especificação de hardware, confiabilidade, segurança, integridade e comunicação de dados.

Multiprogramação: divisão da memória em várias partes, com uma tarefa diferente em cada partição. Permite executar vários processos monitorando seus estados de processo e alternando entre os processos. Não tem intervalo de tempo fixo para processos. Seu objetivo principal é a utilização de recursos.

* + Vantagem: sem tempo ocioso da CPU, tarefas são executadas em paralelo, tempo de resposta mais curto, maximiza o rendimento total do computador, aumenta a utilização de recursos.
  + Desvantagem: trabalhos de longa duração têm que esperar muito tempo, rastreamento de todos os processos é difícil, requer agendamento de CPU, requer gerenciamento de memória eficiente, nenhuma interação do usuário com qualquer programa durante a execução.

# Multitarefas de tempo real

Usados para servir aplicações que atendem processos externos, e que possuem tempos de resposta limitados. Geralmente sinais de interrupções comandam a atenção do Sistema e geralmente são projetados para uma aplicação específica.

# Estruturas de SO’s

São 6: sistemas monolíticos, micronúcleos, sistemas de camadas, sistemas cliente-servidor, máquinas virtuais e exonúcleos.

# Estrutura monolítica

É a forma mais primitiva. Consiste em um conjunto de programas que executam sobre o hardware, como se fosse um único programa. Os programas de usuário podem ser vistos como subrotinas, invocadas pelo SO, quando este não está executando nenhuma das funções do sistema.

# Estrutura micronúcleo

“Microkernel”. Incorpora somente as funções de baixo nível mais vitais. Fornece uma base sobre a qual é construído o resto do SO. A maioria destes sistemas são construídos como coleções de processos concorrentes e fornece serviços de alocação de CPU e de comunicação aos processos (IPC). O micronúcleo controla interrupções, processos, escalonamento, IPC.

# Estrutura em camadas

Apresenta modularização, que é a divisão de um programa complexo em módulos de menor complexidade. Os módulos interagem através de interfaces bem definidas. E, Informação Escondida, que é quando os detalhes das estruturas de dados e algoritmos são confinados em módulos. Externamente, um módulo é conhecido por executar uma função específica sobre objetos de determinado tipo.

# Estrutura máquina virtual

Apresenta a hierarquia de níveis de abstração, de modo que, a cada nível, os detalhes de operação dos níveis inferiores possam ser ignorados. Através disso, cada nível pode confiar nos objetos e operações fornecidas pelos níveis inferiores.

Exemplo Hierarquia:

Nível 1 – núcleo, responsável por alocação de CPU aos processos, HW: CPU.

Nível 2 – E/S básico, responsável por processos de E/S, HW: dispositivos de E/S.

Nível 3 – gerenciamento de memória, HW: memória principal.

Nível 4 – sistema de arquivos, HW: armazenamento secundário.

Nível 5 – interpretador de linguagem de comando, de mensagem, HW: controle do operador.

O Modelo de Máquina Virtual, cria um nível intermediário entre o hardware e o SO, denominado Gerência de Máquinas Virtuais. Cria diversas máquinas virtuais independentes, onde cada uma oferece uma cópia virtual do hardware, incluindo modos de acesso, interrupções, dispositivos de E/S etc. Como é independente, cada máquina pode ter seu próprio SO.

# Chamadas de sistema

Syscall. Porta de entrada para o modo kernel. Altera entre modo usuário e modo kernel. Feitas através de TRAPS (interrupções de sw), e após o término da chamada, a execução continua.

Normalmente, tem uma interface para esconder a complexidade das syscalls e é fornecida pelo SO. Geralmente escrita em linguagem de alto nível. As aplicações utilizam uma API, que é uma interface que encapsula o acesso direto às syscalls.

Motivações: Portabilidade, esconder complexidade e acréscimo de funcionalidades que otimizam o desempenho.

# Interrupções

Interrupções não são causadas por aplicações em execução, diferente de TRAPS. São modificações no fluxo de controle de um programa causadas por um evento externo ao processamento do programa, usualmente eventos relacionados a operações de E/S. As interrupções também param o processamento do programa atual e transferem o controle para uma rotina de tratamento de interrupção.

3 Tipos: Hardware, Software, Interna (exceção).

* + HW: Evento externo;
  + SW: Execução de uma instrução específica. Pode ser utilizada pelo programador para interrupção em algum ponto do programa.
  + Interna (Exceção): Uso inválido ou errado de uma instrução ou dado.

# Traps

É uma espécie de chamada a procedimento automática iniciada sempre que ocorrer alguma condição específica causada pela execução de um programa. Quando há ocorrência de uma trap, o fluxo de controle é alterado para um endereço fixo de memória. Nesse endereço fixo há uma instrução de desvio para um procedimento, conhecido como procedimento de tratamento de trap.

Exemplo: Overflow

# Interrupções vs Traps

Traps são síncronas e Interrupções assíncronas. Traps acontecem sincronamente porque resultam da execução do próprio programa, interrupções podem ser causadas por agentes externos.

# Processos

Instância de um programa em execução, único. É a forma pela qual o SO vê um programa e o executa. Cada processo tem seu espaço de endereçamento. Cada processo tem seu program counter, pilha de memória (contexto de hw). Cada processo tem seu id, escalonados pelo núcleo (contexto de sw).

* + 1º plano (foreground): Interação direta com o usuário.
  + 2º plano (background): Processos com funções específicas que independem do usuário.

Daemons: Processos que ficam em segundo plano para lidar com algumas atividades, como e-mail, páginas da web, notícias, impressão etc.

Processos são criados quando:

* Inicia o Sistema
* syscall para a criação de um processo, realizado por algum processo em execução
* há requisição do usuário para a criação de um novo processo
* há inicialização de um processo em batch

Processos são finalizados quando:

* Normal - A tarefa é finalizada
* Erro - O processo sendo executado não pode ser finalizado
* Erro fatal – não encontrou o arquivo
* Outro processo.

Mudança de contexto: salvar o conteúdo dos registradores do processo que está deixando a CPU e carregar os reg com os valores do novo processo que será executado, leva um overhead de tempo. (contexto de hw)

* + Dispacher: Armazena e recupera o contexto, atualiza as informações PCB, processo relativamente rápido (0,1ms).
  + Scheduler: Escalonador. Escolhe a próxima tarefa a ter acesso ao processador, parte mais demorada.

# Tabela de processos

Tabela com infos importantes de cada processo, incluindo o seu contador de programa, ponteiro de pilha, alocação de memória, estado dos arquivos abertos, informação sobre sua contabilidade e escalonamento, com o estado (em execução, pronto ou bloqueado).

Process Control Block (PCB): reside na memória principal e mantêm todas as informações sobre contexto de hardware, software e espaço de endereçamento de cada processo.

# Estados de um processo

* Executando: realmente usando a CPU naquele instante
* Pronto: executável, temporariamente parado para deixar outro processo ser executado.
* Bloqueado: incapaz de ser executado até que algum evento externo aconteça.

# Escalonador de processos

Responsável por realizar a troca de processos. O escalonador também é um processo. Diversas técnicas/algoritmos para realizar o escalonamento de processos. Escolhe qual é o próximo processo a ser executado. Nível mais baixo do SO.

Escalonamento controla o compartilhamento de recursos. Multiprogramação: permite a utilização simultânea de vários usuários. Maximizar utilização da CPU: Processos bloqueados não param a CPU.

* + Preemptivo: Processo pode perder uso da CPU, provoca uma interrupção forçada do processo para que outro possa usar.
  + Não Preemptivo: Permite que o processo que está sendo executado continue a execução. Para quando:
    - Termina de executar;
    - Solicita uma E/S;
    - Libera o processador voltando ao estado de pronto;

Throughput: números de processos terminados por unidade de tempo.

Turnaround: tempo transcorrido desde o momento em que o sw entra e o instante em que termina sua execução.

tempo de resposta: intervalo entre a chegada ao sistema e início de sua execução.

tempo de espera: soma dos períodos em que o processo estava no seu estado pronto.

# Algoritmos de escalonador

Para cada tipo de sistema (os 3 vistos anteriormente, batch, tempo real e compartilhado/interativo).

* Batch:
  + FIFO:

- Não preemptivo: processo executa na CPU até sua finalização, sem troca de contexto.

- Primeiro a chegar, primeiro a ser executado.

- Desvantagem: processos demorados atrasa td mundo.

- Vantagem: fácil entender e programar.

* + SJF:

- Não preemptivo.

- O menor processo da lista é executado primeiro, p isso tem que prever o tempo de execução do processo.

- Desvantagem: os Jobs precisam ser conhecidos de antemão, e muitos Jobs curtos chegarem, os mais longos demorarão p serem executados.

- Vantagem: algoritmo c menor turnaround.

* + SRNT:

- Preemptivo, versão melhorada do SJF.

- Processos com menor tempo de execução executam primeiro, se um processo novo chega e tem tempo menor do que o que está executando, a CPU interrompe p executar o novo processo.

- Desvantagem: starvation: processo espera liberação de recurso, mas nunca ocorre e o processo fica p sempre parado.

- Vantagem: processos menores executam mais rápido.

* Interativo:
  + Round-Robin:

- Preemptivo.

- Os processos ficam em uma fila circular e cada processo recebe um tempo de execução (quantum), ao final do tempo, o processo é suspenso, colocado no final da fila e o próximo é executado. Pode ser suspenso por E/S., todos os processos têm importância igual.

- Desvantagem: nenhum processo termina antes de todo mundo rodar um pouco.

* + Prioridade:

- Preemptivo.

- Cada processo tem uma prioridade que pode ser atribuída dinâmica ou estaticamente, maior prioridade executados primeiro, e executa como round-robin.

- Atualizar a prioridade: processos agrupados por classes de prioridade, e classes mais baixa podem morrer famintos (nunca executados), a atualização pode ser feita a cada mudança de clock.

- Desvantagem: se não feita atualização de prioridade correta, processos com prioridade baixa podem nunca executar.

* + Múltiplas filas:

- Preemptivo.

- São usadas várias filas de processos prontos para executar. Cada processo é colocado em uma fila. Cada fila tem uma política de escalonamento. Existe uma política de escalonamento entre filas. Cada vez que um processo é executado e suspenso ele recebe mais tempo p próxima vez que for executado.

- Desvantagem: processos longos tem quantum progressivos, é difícil de prever o comportamento, e os processos podem nunca mudar de fila.

- Vantagem: Reduz trocas de contextos, processos mais curtos terminam primeiro.

* + SPN:

- Preemptivo.

- Ideia do SJF, estimativa de tempo p prever quem é o próximo com base em execuções passadas.

- Vantagens e desvantagem: mesmas do SJF.

* + Garantido:

- Estimar (prometer) a um processo o tempo de sua execução e cumprir. Necessário conhecimento dos processos executando e a serem executados.

- Cada processo deve receber 1/n dos ciclos da CPU.

* + Lottery:

- Distribuição de bilhetes que dão acesso à CPU.

-Lembra o escalonamento com prioridade, mas bilhetes são trocados entre processos. Quando um escalonamento deve acontecer, escolhe-se aleatoriamente um bilhete. Processos mais importantes podem ter mais de um bilhete.

- P garantir que todos os processos chegarão à CPU, tem duas filas: 1 – bilhetes já sorteados e 2 – bilhetes ainda não sorteados.

* + Fair-share:

- Dono do processo é levado em consideração, se a um usuário foi prometido certa fatia de tempo, ele deverá receber independentemente do número de processos.

* Tempo real:

Há 2 tipos:

Hard Real Time: onde atrasos não são tolerados.

Soft Real Time: onde atrasos ocasionais são tolerados.

Algoritmos estáticos: tomam suas decisões de escalonamento antes de o sistema começar a ser executado. Funcionam apenas quando há uma informação perfeita disponível antecipadamente sobre

o trabalho a ser feito, e os prazos que precisam ser cumpridos.

Algoritmos dinâmicos: tomam suas decisões no tempo de execução, após ela ter começado.

# Threads

É a linha de execução de um processo, sendo pelo menos 1 ou mais, é usado para agrupar recursos e tem um único espaço de endereçamento de memória, ou seja, processos são usados para agrupar recursos, já threads são as entidades escalonadas para execução na CPU. Um processo pode ter mais de uma thread executando em quase paralelo, como se eles fossem (quase) processos separados (exceto pelo espaço de endereçamento compartilhado).

Cada thread possui seu próprio PC (contador de programa), registradores, pilha e estado. Mais rápidas que processos e mais rápidas de serem criadas e destruídas, mas requer mecanismos de sincronização e possuem acesso a qualquer recurso dentro daquele espaço de endereçamento.

Há 3 tipos de modelos: modo usuário, modo núcleo e híbrido. Há 4 estados de thread: pronto, executando, bloqueado e terminado.

# Modelos de Threads:

* Modo usuário:

- Totalmente implementadas no espaço usuário, onde cada processo tem sua própria tabela de threads similar à tabela de processos.

- Criação e escalonamento criadas sem kernel perceber

- Na visão do kernel, é um processo total monothread.

- O escalonamento funciona de forma que o núcleo escolhe o processo e passa o controle p ele, que escolhe a thread e a gerência da thread fica no espaço nível usuário.

- O escalonamento pode ser customizado e tem melhor resultados.

- Vantagem: pode ser implementado por um SO que não suporte a abstração de threads, trocas de contexto não envolve troca de modo usuário para modo núcleo, sendo mais rápida. Possibilita customizar o escalonador.

- Desvantagem: Syscalls bloqueadas, uma thread de usuário não é interrompida pelo escalonador, sendo assim, não atende interrupções do temporizador, e cada thread deve ceder à CPU para outras,

Ex: thread.h em C

* Modo núcleo:

- Criação, escalonamento e gerenciamento são realizados pelo kernel

- Núcleo possui tabela threads e processos separadas e as tabelas ficam no kernel

- Gerenciar, criar e destruir uma thread nesse modo é mais caro.

- Os algoritmos de escalonamento mais usados são: round-robin e por prioridade.

- Desvantagem: pode haver sobrecarga de gerenciamento.

* Híbrido:

- N threads de usuário sendo mapeadas pra M threads do kernel, sendo N >= M.

# Comunicação entre processos

Processos precisam se comunicar por diversas razões, compartilhar recursos, gerar produtos para outros processos. Interrupções para isso não é eficiente.

* Race Condition:

- Corrida pelo recurso, em caso de processos que se comunicam através de alguma área de armazenamento comum, podendo estar na memória principal ou pode ser um arquivo compartilhado.

* Exclusão mútua:

- Impede que o processo que está usando um arquivo ou variável compartilhado serão impedidos de realizar a mesma coisa.

- Seções do programa onde são realizados acessos a recursos compartilhados são denominadas de Regiões Críticas (RC).

- Pode gerar uma fila para acessar.

- Garantir que um outro processo tenha acesso a uma RC quando outro processo já está usando através de:

1. Dois processos NUNCA podem estar simultaneamente dentro de suas RCs

2. Não se pode fazer suposição em relação à velocidade e número de CPUs

3. Um processo fora da RC não pode causar o bloqueio de outro processo

4. Um processo não pode esperar eternamente para entrar em sua RC

# Algoritmos para de exclusão mútua:

* Espera Ocupada (Busy Waiting):

- Quando um processo deseja entrar em sua RC, ele verifica se sua entrada é permitida. Se não for, ele ficará em um loop de espera.

- Desvantagem: Desperdiça tempo de CPU, sendo alguns mecanismos bons e outros ruins.

* Primitivas Sleep/Wakeup:

- Sleep bloqueia o processo que chamou, suspende a execução até que outro processo o “acorde” (wakeup).

- Problema produtor-consumidor:

* + Dois processos compartilham um buffer de tamanho limitado
  + O processo produtor coloca dados no buffer
  + O processo consumidor retira dados do buffer

Problema: o produtor deseja colocar dados e o buffer está cheio, enquanto o consumidor deseja retirar dados, mas o buffer está vazio.

* Semáforos:

- Utiliza variável para controlar acesso a recursos compartilhados

- Sincronizar o uso de recursos em grande quantidade

- Conta o número de recursos ainda disponíveis no sistema

- Down (wait): Verifica se o semáforo é maior que 0 e se for, semáforo = semáforo – 1, então continua, se 0, o processo que executou o down é colocado para dormir, sem completar o down.

- Up: semáforo = semáforo + 1, se há processos dormindo nesse semáforo escolhe um deles e

desbloqueia (permite que complete o down), neste caso, o valor do semáforo permanece o mesmo (zero), e é executado sempre que um processo liberar um recurso.

- Ação atômica: confere o valor, modifica e possivelmente dorme são feitos como única ação indivisível. Essencial para solucionar problemas de sincronização e evitar condições de corrida.

- Semáforos binários: 1 variável (multex) 0 ou 1 para controlar acesso a RC. Cada processo dá um down antes de entrar e up logo depois de sair da RC.

- Semáforo geral.

- Desvantagens: Não são boas soluções para sistemas distribuídos e não permitem sincronização entre processos de máquinas diferentes

* Monitores:

- Somente um processo pode estar ativo dentro de um monitor em um mesmo instante e os outros permanecem bloqueados.

- 1 único módulo, composto de: Procedimentos, Variáveis e Estruturas de dados.

- O compilador é quem garante a exclusão mútua.

- Todos os recursos compartilhados devem estar implementados dentro de um monitor.

- Desvantagens: Dependem da linguagem de programação, não são boas soluções para sistemas distribuídos e não permitem sincronização entre processos de máquinas diferentes

* Troca de Mensagem:

- Processos enviam e recebem mensagens, garantida pela restrição de que uma mensagem só pode ser recebida depois de enviada

- Transferência só é realizada depois de ter ocorrido a sincronização.

# Deadlocks

Quando um processo fica parado sem continuar e assim ficarão para sempre. Ocorre tanto em HW quanto -em SW, e acontece quando um processo tem acesso exclusivo a um recurso.

Recursos:

- Preemptível: é aquele que pode ser retirado do processo proprietário sem causar-lhe prejuízo algum. Ex: Mem, CPU.

- Não preemptível: é aquele que se retirado do processo proprietário causa-lhe prejuízo. Ex: impressora, executar/gravar blu-ray.

Condições para um deadlock (deve ter todas):

* Exclusão mútua: cada recurso está atualmente associado a exatamente um processo ou está disponível
* Posse e espera (hold and wait): processos atualmente de posse de recursos que foram concedidos antes podem solicitar novos recursos.
* Não preempção: recursos concedidos antes não podem ser tomados à força de um processo. Eles precisam ser explicitamente liberados pelo processo que os têm.
* Espera circular: deve haver uma lista circular de dois ou mais processos, cada um deles esperando por um processo de posse do membro seguinte da cadeia. \*ciclos podem levar à um deadlock.

# Grafos de alocação

Há 4 estratégias.

1. Simplesmente ignorar o problema. Se você o ignorar, quem sabe ele ignore você.

- Ignorar se o esforço em solucionar for muito grande em relação a sua frequência.

* + Algoritmo do Avestruz: enfie a cabeça na areia e finja que não há um problema.

- Deve verificar a frequência do problema

- Alto custo para estabelecer as condições para o uso dos recursos

1. Detecção e recuperação. Deixe-os ocorrer, detecte-os e tome as medidas cabíveis.
   * Detecção com um recurso de cada tipo
   * Detecção com vários recursos de cada tipo:

- Vetor de recursos disponíveis

- C: matriz de alocação corrente.

- R: matriz de requisições.

* + Recuperação por meio da preempção
  + Recuperação por meio de rollback
  + Recuperação por meio da eliminação de processos

1. Evitar dinamicamente pela alocação cuidadosa de recursos.
2. Prevenção, ao não satisfazer uma das quatro condições

Sistemas Operacionais – Questões

1. Quais são as duas principais funções de um sistema operacional?
2. Qual é a diferença entre sistemas de compartilhamento de tempo e de multiprogramação?
3. Quais das instruções a seguir devem ser deixadas somente em modo núcleo?
   1. Desabilitar todas as interrupções.
   2. Ler o relógio da hora do dia.
   3. Configurar o relógio da hora do dia.
   4. Mudar o mapa de memória.