Noções de Processos

Marcelo Johann

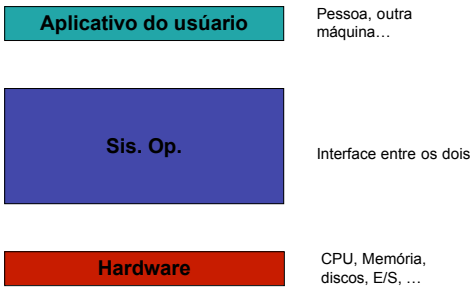
Plano da aula

- Introdução
- Histórico
- Multiprogramação
  - Noção de processo
    - Definição
    - Ciclo de vida do processo
  - Suporte de Hardware para multi-programação
    - Mecanismo de interrupção
    - Modo de execução do processador
- Linguagem C, Linux, Módulos e Makefile

Sistemas Operacionais : introdução

- Definição & Objetivos
- Serviços oferecidos
  - API – chamadas de sistema – programas de sistema
- Histórico
  - Lotes – monitor – multi-programação – timesharing
- Categorias de Sis. Op. atuais
- Exemplos de Sis. Op.

Definição & Objetivos



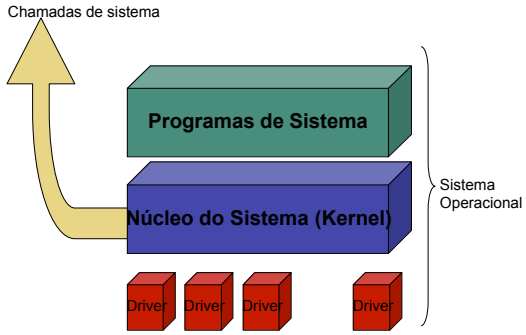
Definição & Objetivos

- O Sistema Operacional é uma interface HW / SW aplicativo
- Duas formas de vê-lo:
  - É um "fiscal" que controla os usuários
  - É um "juiz" que aloca os recursos entre os usuários
- Objetivos contraditórios:
  - Conveniência
  - Eficiência
  - Facilidade de evolução
  - A melhor escolha sempre DEPENDE de alguma coisa…

Serviços oferecidos

- O Sis. Op. deve fornecer uma interface aos programas do usuário
  - Quais recursos de HW?
  - Qual seu uso?
  - Tem algum problema? (Segurança, falha…?)
  - É preciso de manutenção?
  - Chegou um email?
  - Etc…
- Chamadas de sistema – programas de sistema

Organização e serviços do Sis. Op.



Núcleo, chamadas de sistema, programas de Sistemas

- Núcleo: é o conjunto mínimo de serviços executados pelo Sis. Op.
  - Definição de processos, escalonamento,…
- Chamadas de sistema: são funções que os programas dos usuários podem usar para acessar os serviços do núcleo
  - Exemplo: ls, mkdir, cd, format, CTRL-C…
  - O núcleo assume a execução.
- Programas de sistema: são serviços menos críticos
  - Compiladores, editores de texto, shell, GUI (Windows), Navegador…

Histórico – sistemas em lotes

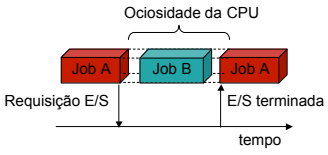
- Nos primórdios da Computação, não tinha Sis. Op…
  - O programador interagia diretamente com o HW;
  - A alocação dos recursos de HW era feita por planilha.
- Nos anos 50, automatizou-se a execução dos *jobs*
  - Definição de categorias de programas (filas) com uso parecido dos recursos = lotes (*batches*);
  - Possibilidade de definir bibliotecas especializadas;
  - Um operador profissional opera o HW para executar os jobs.
  - O mesmo fiscaliza a atribuição do HW e o andamento dos jobs.

Histórico – monitor residente

- Evolução natural: automatizar o andamento dos batches
  - O controle passa a ser feito por um programa, o monitor, residente na memória;
  - O monitor carrega o *job* na memória;
  - Passa o controle do fluxo de execução ao *job*;
  - Volta a exercer o controle ao terminar o *job*;
  - Centraliza os acessos aos periféricos (fitas, discos…)
- Melhor, porém possibilita a execução de apenas um *job* por vez!

Histórico – multi-programação

- Idéia seguinte: poupar o desperdício de CPU devido às Entradas/Saídas



- Mantém-se mais de um *job* em execução!

Histórico – multi-programação

- Mais de *um job*?
  - Terá um só na CPU, em execução…
  - Onde guardar o job que não estará executando?
    - Em fita? Muito lento!
    - Em disco (acesso randômico!)
- Como fazer com que o "monitor" saiba que um *job* acessa a dispositivos de E/S?
  - Mecanismo de interrupção.

time-sharing, multi-usuário, multi-tarefa

- Evoluções naturais da multi-programação:
  - **Compartilhamento de tempo:** cada usuário possui um terminal próprio e acessa à mesma CPU;
  - **Multi-usuário**:mais de uma sessão podem ser abertas em um computador só, por vários usuários (Windows NT/2000, Unix…)
    - **Sistemas mais antigos eram mono-usuários (MS-DOS)**
  - **Multi-tarefa**:cada usuário pode usar mais de um job "simultaneamente"

Categorias de Sis. Op. modernos

- Sistemas distribuídos
  - Distribuição de uma (ou mais) tarefa entre vários computadores;
  - O usuário não enxerga a distribuição (visão única)
  - Útil para tolerança a falhas.
  - Fracamente acoplados.
- Sistemas Paralelos
  - Um computador possui mais de um processador
    - Com ou sem memória compartilhada (SMP)
    - Fortemente acoplados!
- Sistemas embutidos
  - Celulares, Palm, carros, satélites…
- Sistemas de tempo real
  - *Hard real-time* vs. *soft real-time*

Exemplos: Windows

- Windows NT 3.1, 1993
  - Multi-tarefas, mono-usuário
  - 32 bits
  - "casca gráfica" (janela) em cima de MS-DOS e/ou OS/2
- Windows NT 4.0, 1995: mudanças sobretudo na API gráfica, suporte a SMPs
- Windows 2000, 1999: serviços distribuídos
  - Cliente/servidor
  - Organização em "micro-núcleo" e orientada a objetos
  - Multi-usuários
  - NTFS
- Windows XP, 2001
  - API gráfica integrada com Web
  - Melhor segurança (firewalls)
  - 32-64 bits
- Windows VISTA, 2007.

Exemplos: Linux

- 1991 com o trabalho de Linus Torvalds sobre o kernel para i386 (kernel 0.01, maio 1991)
- Março de 94: kernel 1.0 com suporte de rede
  - Março de 95: kernel 1.2
    - suporte a novo HW (Sparc, Alpha).
- Junho de 96: versão 2.0
  - suporte a SMP, Sparc,
  - melhora na memória virtual e no sistema de arquivos,
  - *threads* no kernel,
  - módulos
- 2002 versão 2.4.x
  - melhora nos algoritmos de escalonamento das *threads*.
- Agora versão 2.6.y

O conceito de processo

- Um programa é:
  - Uma seqüência finita de instruções;
  - Uma entidade ***passiva*** (que não se altera com o passar do tempo).
  - Armazenado em disco.
- Um processo é:
  - Uma abstração que representa um programa ***em execução***;
  - Uma entidade ***dinâmica***: seu estado se altera conforme for executando.
  - Armazenado na memória.
- Pode-se encontrar mais de um processo instanciando um programa único.

O processo do ponto de vista do S.O.

- Imagem de um programa
  - Segmento de código
  - Espaço de endereçamento
- Conjunto de recursos de HW alocados pelo Sis. Op.:
  - Registradores (PC, *Stack Pointer*…);
  - Memória;
  - Espaço no disco (arquivos de E/S).
  - = ***Contexto*** do processo.
- Unidade de escalonamento
  - Estado;
  - Algoritmos de escalonamento para otimizar o uso do HW.
  - Alocar a CPU a um processo implica em uma ***troca de contexto***

Tabela de Processos

- Para manter as informações relativas aos processos, o núcleo deve manter:
  - Uma estrutura de dados relativa a um dado processo
    - struct proc
    - Process Control Block (PCB)
  - Uma estrutura de dados que gerencia o conjunto de processos
    - Tabela de processos.
- As chamadas de sistema que manipulam os processos irão interagir com essas estruturas de dados.

Relacionamento entre processos

- Caso mais simples: os processos são independentes.
  - Sem relacionamento
- Grupo de processos
  - Compartilhamento de recursos
  - Baseados em ***hierarquia*** de processos:
    - Um processo pai cria processos filhos;
    - Os filhos podem executar o mesmo código, ou trocá-lo;
    - Obtem-se uma árvore de processos.
- Implica na definição da semântica de termino de um processo:
  - Só o processo morre;
  - Toda sua descendência morre.

Sinalização de processos

- Uma das formas de interagir entre processos é através de sinais.
  - A recepção é assíncrona
  - Ao receber um sinal, o processo para sua atividade,
  - Ele executa um tratamento de sinal adaptado (*signal handler*),
  - Ao se encerrar o tratamento, o processo pode voltar ao estado onde estava antes.
- Um sinal é uma versão em nível de software das interrupções de hardware.

Suporte de Hardware: interrupções

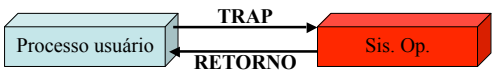
- Erros e eventos são detectados por hardware
  - Exemplos de evento: inserir um pendrive na porta USB, escrever um bloco em disco, receber um pacote pela rede, escrever numa área proíbida…
  - O HW emite uma interrupção.
- São tratados pelo Sis. Op.
  - Identifica a interrupção (número);
  - Verifica sua prioridade;
  - Acha no vetor de interrupções qual procedimento é apropriado (*handler*).

Suporte de Hardware: modos de execução

- O HW provê no mínimo dois modos de execução diferentes:
  - Modo privilegiado (protegido, sistema…) – todo o conjunto de operações é disponível. É o modo de execução do Sis. Op.
  - Modo usuário: uso limitado. Os processos usuários operam neste modo.
- ***Chaveamento*** de modos:
  - É o fato de passar de um modo para o outro.
  - Usuário -> protegido: por interrupção.
  - Protegido -> usuário: por instrução clássica (yield, return, …).

Exemplo importante de uso dos 2 modos

- Para proteger os periféricos, as instruções de E/S são privilegiadas.
- Logo, um processo usuário não pode acessá-los
  - E.g.: escrita em disco, leitura de um CD…
- O usuário deve passar pelo Sis. Op. através de uma chamada de sistema, que gera uma interrupção (trap).



Chamada de sistema com interrupção

- A chamada de sistema oferece o serviço ao processo usuário, em modo "seguro".
- Ela gera uma interrupção
  - Identificação, prioridade, *handler*…
- Ela implica em uma troca de contexto
  - O processo chamador deve deixar o lugar para o código do núcleo!
  - A troca de modo implica em uma troca de contexto.
- Conforme for a prioridade e o tipo de escalonador, a troca de contexto pode ser imediata ou atrasada.
- O que acontece se houver uma interrupção durante o tratamento de uma interrupção?
  - Comparam-se as prioridades
  - Possibilidade de desabilitar as interrupções em casos críticos.

Processos e Disco

- Os processos devem interagir com o disco para armazenar e recuperar dados não voláteis.
- O disco físico é abstraído pelo Sistemas de Arquivos, de acordo com uma hierarquia:
  - Diretórios
  - Arquivos.
- Os diretórios estão freqüentemente organizados de acordo com uma hierarquia em árvore
  - Raiz ('/')
  - Diretório de trabalho de um processo ('.')
  - Caminho relativo / absoluto
- Deve ter chamadas de sistema para acessar o Sistema de Arquivos!

Exemplos de chamadas de sistema

- Minix 2 provê 53 chamadas de sistema no total
- Gerenciamento de processos:
  - fork(), waitpid(), exit(), execve(...), getpid()...
- Sinais
  - sigaction(), sigreturn(), sigprocmask(), kill()...
- Gerenciamento de arquivos
  - open(), close(), mknod(), read(), write(), pipe(),...
  - mkdir(), mount(),...
- Direitos de acesso
- Gerenciamento de tempo
  - Time()

Chamadas de sistema (Linux) para gerenciar processos

- Criação de processo: fork()
  - Cria um novo processo
  - Igual ao pai (clone!)
  - No processo pai, fork() retorna o pid do filho;
  - No processo filho, fork() retorna 0.
- Mudar o segmento de código: exec()
  - Executa o binário apontado em argumento.
  - Em geral, chamado logo após o fork() ("fork-exec")
- Recuperar o identificador: getpid()
  - Retorna um int, que identifica o processo.
- Terminar o processo: kill()
  - Manda um sinal (e.g. TERM) para o processo cujo pid é dado em argumento.

Próxima Aula

Laboratório de Programação C / UNIX