TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PROGRAMAÇÃO DE COMPUTADORES I

Aula 01: Conceitos Iniciais / Sistema Operacional

O conteúdo deste documento tem por objetivo apresentar uma visão geral sobre Sistemas Operacionais e Terminal Linux, e é baseado no livro do ***Prof. Dr. Carlos Alberto Maziero, disponível no link:*** http://wiki.inf.ufpr.br/maziero/doku.php

Observações:

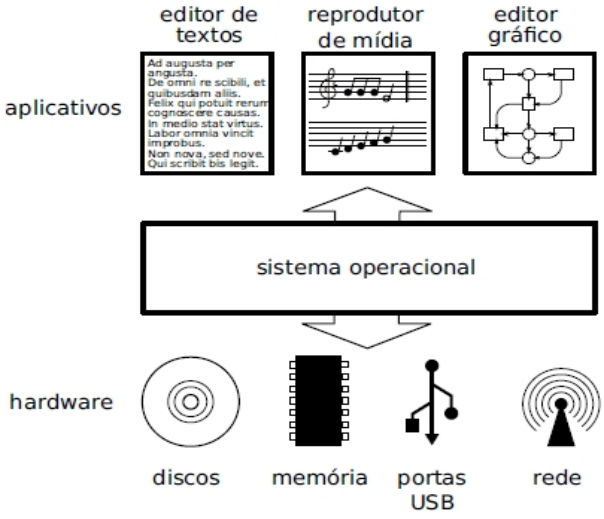
1) apresentar o plano de ensino da disciplina aos alunos.

2) efetuar o cadastro dos alunos no SETA.

Introdução

Um sistema de computação é constituído basicamente por hardware e software, existindo uma grande distância entre os circuitos eletrônicos e dispositivos de hardware e os programas em software. Os circuitos são complexos, acessados através de interfaces de baixo nível (geralmente usando as portas de entrada/saída do processador) e muitas vezes suas características e seu comportamento dependem da tecnologia usada em sua construção. Por exemplo, a forma de acesso em baixo nível, a discos rígidos IDE, difere da forma de acesso a discos SCSI ou leitores de CD. Essa grande diversidade pode ser um grande problema para o desenvolvedor de aplicativos. Com isso torna-se desejável oferecer aos programas uma forma de acesso homogênea aos dispositivos físicos, que possibilite abstrair as diferenças tecnológicas entre eles.

O sistema operacional é uma camada de software que opera entre o hardware e os programas aplicativos voltados ao usuário final. O sistema operacional é uma estrutura de software ampla, muitas vezes complexa que incorpora aspectos de baixo nível (como *drivers* de dispositivos e gerência de memória física) e de alto nível (como programas utilitários e a própria interface gráfica). A figura a seguir ilustra a arquitetura geral de um sistema de computação típico. Nela, podemos observar elementos de hardware, o sistema operacional e alguns programas aplicativos.



Estrutura de um Sistema de Computação Típico

Abstração de Recursos

Acessar os recursos de hardware de um sistema de computação pode ser uma tarefa complexa, devido às características específicas de cada dispositivo físico e a complexidade de suas interfaces. Por exemplo, a seqüência a seguir apresenta os principais passos envolvidos na abertura de um arquivo (operação open) em um leitor de disquete:

1. Verificar se os parâmetros informados estão corretos (nome do arquivo, identificador do leitor de disquete, buffer de leitura, etc);
2. Verificar se o leitor de disquetes está disponível;
3. Verificar se o leitor contém um disquete;
4. Ligar o motor do leitor e aguardar atingir a velocidade de rotação correta;
5. Posicionar a cabeça de leitura sobre a trilha onde está a tabela de diretório;
6. Ler a tabela de diretório e localizar o arquivo ou subdiretório desejado;
7. Mover a cabeça de leitura para a posição do bloco inicial do arquivo;
8. Ler o bloco inicial do arquivo e depositá-lo em um buffer de memória.

Assim, o sistema operacional deve definir interfaces abstratas para os recursos do hardware, visando atender os seguintes objetivos:

- Prover interfaces de acesso aos dispositivos mais simples que as interfaces de baixo nível, com intuito de simplificar a construção de programas aplicativos. Por exemplo: para ler dados de um disco rígido, uma aplicação usa um conceito chamado arquivo, que fornece uma visão abstrata do disco rígido.
- Tornar os aplicativos independentes do hardware. Ao definir uma interface abstrata de acesso a um dispositivo de hardware, o sistema operacional desacopla o hardware dos aplicativos e permite que ambos evoluam de forma mais autônoma. Por exemplo, o código de um editor de textos não deve ser dependente da tecnologia de discos rígidos utilizada no sistema.
- Definir interfaces de acesso homogêneas para dispositivos com tecnologias distintas. Através de suas abstrações, o sistema operacional permite aos aplicativos usar a mesma interface para dispositivos diversos. Por exemplo, um aplicativo acessa dados em disco através de arquivos e diretórios, sem precisar se preocupar com a estrutura real de armazenamento dos dados, que podem estar em um disquete, um disco IDE, uma máquina fotográfica digital conectada à porta USB, um CD ou mesmo um disco remoto, compartilhado através da rede.

Gerência de Recursos

Os aplicativos utilizam o hardware para executar suas funções: ler e armazenar dados, editar e imprimir documentos, navegar na Internet, tocar música, etc. Com diversas atividades em execução, e diversos possíveis acessos simultâneos ao hardware, conflitos podem surgir. Cabe ao sistema operacional definir políticas para gerenciar o uso dos recursos de hardware pelos aplicativos, e resolver eventuais disputas e conflitos. Exemplos:

- O uso desse processador deve ser distribuído entre os aplicativos presentes no sistema, de forma que cada um deles possa executar na velocidade adequada para cumprir suas funções sem prejudicar os outros.
- Ataques de negação de serviço (DoS – *Denial of Service*). Consistem em usar diversas técnicas para forçar um servidor de rede a dedicar seus recursos a atender um determinado usuário, em detrimento dos demais. Por exemplo, ao abrir milhares conexões simultâneas em um servidor de e-mail.

***Prática*: Sistema Operacional Linux – Comandos Básicos / Repositório (Linux)**
http://www.gileduardo.com.br/ifpr/pci/downloads/pc_pratica01.pdf

Gerência de Tarefas

Uma tarefa é definida como sendo a execução de um fluxo sequencial de instruções, construído para atender uma finalidade específica: realizar um cálculo complexo, a edição de um gráfico, a formatação de um disco, etc. Assim, a execução de uma sequência de instruções em linguagem de máquina, normalmente gerada pela compilação de um programa escrito em uma linguagem qualquer, é denominada "tarefa" ou "atividade" (do inglês *task*).

É importante ressaltar as diferenças entre os conceitos de tarefa e de programa:

- Um programa é um conjunto de uma ou mais sequências de instruções escritas para resolver um problema específico, constituindo assim uma aplicação ou utilitário. O programa representa um conceito estático, sem um estado interno definido (que represente uma situação específica da execução) e sem interações com outras entidades (o usuário ou outros programas). Por exemplo, os arquivos *C:\Windows\notepad.exe* e */usr/bin/nano* são programas de edição de texto.
- Uma tarefa é a execução, pelo processador, das sequências de instruções definidas em um programa para realizar seu objetivo. Trata-se de um conceito dinâmico, que possui um estado interno bem definido a cada instante (os valores das variáveis internas e a posição atual da execução) e interage com outras entidades: o usuário, os periféricos e/ou outras tarefas. Tarefas podem ser implementadas de várias formas, como processos ou threads, como veremos posteriormente.

Sistemas Operacionais e Tarefas

Em um computador, o processador tem executar todas as tarefas submetidas pelos usuários. Essas tarefas geralmente têm comportamento, duração e importância

distintas. Cabe ao sistema operacional organizar as tarefas para executá-las e decidir em que ordem fazê-lo.

Sistemas mono-tarefa

Os primeiros sistemas de computação, nos anos 40, executavam apenas uma tarefa de cada vez. Nestes sistemas, cada programa binário era carregado do disco para a memória e executado até sua conclusão. Os dados de entrada da tarefa eram carregados na memória juntamente com a mesma e os resultados obtidos no processamento eram descarregados de volta no disco após a conclusão da tarefa. Todas as operações de transferência de código e dados entre o disco e a memória eram coordenadas por um operador humano.

Nesse método de processamento de tarefas é possível delinear um diagrama de estados para cada tarefa executada pelo sistema, representado na figura a seguir.

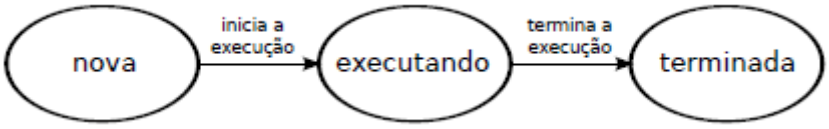


Diagrama de estados de uma tarefa em um sistema mono-tarefa

Com a evolução do hardware, as tarefas de carga e descarga de código entre memória e disco, coordenadas por um operador humano, passaram a se tornar críticas: mais tempo era perdido nesses procedimentos manuais que no processamento da tarefa em si. Para resolver esse problema foi construído um programa monitor, que era carregado na memória no início da operação do sistema com a função de coordenar a execução dos demais programas. O programa monitor executava continuamente os seguintes passos sobre uma fila de programas para execução, armazenados no disco:

1. Carregar um programa do disco para a memória;
2. Carregar os dados de entrada do disco para a memória;
3. Transferir a execução para o programa recém-carregado;
4. Aguardar o término da execução do programa;
5. Escrever os resultados gerados pelo programa no disco.

Percebe-se claramente que a função do monitor é gerenciar uma fila de programas a executar, mantida no disco. Na medida em que os programas são executados pelo processador, novos programas podem ser inseridos na fila pelo operador do sistema. Além de coordenar a execução dos demais programas, o monitor também colocava à disposição destes uma biblioteca de funções para simplificar o acesso aos dispositivos de hardware (teclado, leitora de cartões, disco, etc). Assim, o monitor de sistema constitui o precursor dos sistemas operacionais.

Sistemas multi-tarefa

O uso do programa monitor agilizou o uso do processador, mas outros problemas persistiam. Como a velocidade de processamento era muito maior que a velocidade de comunicação com os dispositivos de entrada e saída[1], o processador

ficava ocioso durante os períodos de transferência de informação entre disco e memória. Se a operação de entrada/saída envolvia fitas magnéticas, o processador podia ficar vários minutos parado, esperando. O custo dos computadores era elevado demais (e sua capacidade de processamento muito baixa) para permitir deixá-los ociosos por tanto tempo.

A solução encontrada para resolver esse problema foi permitir ao processador suspender a execução da tarefa que espera dados externos e passar a executar outra tarefa. Mais tarde, quando os dados de que necessita estiverem disponíveis, a tarefa suspensa pode ser retomada no ponto onde parou. Para tal, é necessário ter mais memória (para poder carregar mais de um programa ao mesmo tempo) e definir procedimentos para suspender uma tarefa e retomá-la mais tarde. O ato de retirar o processador de uma tarefa é denominado preempção.

Sistemas que implementam esse conceito são chamados sistemas preemptivos. A adoção da preempção levou a sistemas mais produtivos (e complexos), nos quais várias tarefas podiam estar em andamento simultaneamente: uma estava ativa e as demais suspensas, esperando dados externos ou outras condições. Sistemas que suportavam essa funcionalidade foram denominados monitores multi-tarefas. O diagrama de estados da figura a seguir ilustra o comportamento de uma tarefa em um sistema desse tipo.

Diagrama de estados de uma tarefa em um sistema multi-tarefa

***Prática*: Gerenciamento de Tarefas – Comando "Kill" (Linux)**
http://www.gileduardo.com.br/ifpr/pci/downloads/pc_pratica01.pdf

***Prática*: Cálculo de π com Threads (Linux)**
http://www.gileduardo.com.br/ifpr/pci/downloads/pc_pratica01.pdf

[1] Essa diferença de velocidades permanece imensa nos sistemas atuais. Por exemplo, em um computador atual a velocidade de acesso à memória é de cerca de 10 nanossegundos (10 x 109 s), enquanto a velocidade de acesso a dados em um disco rígido IDE é de cerca de 10 milissegundos (10 x 103 s), ou seja, um milhão de vezes mais lento!

Comunicação entre Tarefas

Muitas implementações de sistemas complexos são estruturadas como várias tarefas inter-dependentes, que cooperam entre si para atingir os objetivos da aplicação. Para que as várias tarefas que compõem uma aplicação possam cooperar, elas precisam comunicar informações umas às outras e coordenar suas atividades, para garantir que os resultados obtidos sejam coerentes.

Objetivos

Existem várias razões para justificar a construção de sistemas baseados em tarefas cooperantes, entre as quais podem ser citadas:

- Atender vários usuários simultâneos: um servidor de banco de dados ou de e-mail completamente sequencial atenderia um único cliente por vez, gerando atrasos intoleráveis aos demais. Sendo assim, servidores de rede são implementados com vários processos ou threads, para atender simultaneamente todos os usuários conectados.
- Uso de computadores multi-processador: um programa sequencial executa um único fluxo de instruções por vez, não importando o número de processadores presentes no hardware. Para aumentar a velocidade de execução de uma aplicação, esta deve ser "quebrada" em várias tarefas cooperantes, que poderão ser escalonadas simultaneamente nos processadores disponíveis.
- Modularidade: um sistema muito grande e complexo pode ser melhor organizado dividindo suas atribuições em módulos sob a responsabilidade de tarefas interdependentes. Cada módulo tem suas próprias responsabilidades e coopera com os demais módulos quando necessário. Sistemas de interface gráfica, como os projetos Gnome [Gnome, 2005] e KDE [KDE, 2005], são geralmente construídos dessa forma.
- Construção de aplicações interativas: navegadores Web, editores de texto e jogos são exemplos de aplicações com alta interatividade; nelas, tarefas associadas à interface reagem a comandos do usuário, enquanto outras tarefas comunicam através da rede, fazem a revisão ortográfica do texto, renderizam imagens na janela, etc. Construir esse tipo de aplicação de forma totalmente sequencial seria simplesmente inviável.

Para que as tarefas presentes em um sistema possam cooperar, elas precisam comunicar, compartilhando as informações necessárias à execução de cada tarefa, e coordenar suas atividades, para que os resultados obtidos sejam consistentes (sem erros).

Prática: Comunicação entre Tarefas – Chat (Linux)
http://www.gileduardo.com.br/ifpr/pci/downloads/pc_pratica01.pdf