Cap. 1 – CONCEITOS INTRODUTÓRIOS

1. Descreva as 2 perspectivas de definição de um S.O.. Mostre claramente em que circunstâncias cada uma delas é relevante.

Perspectiva "top-down": o S.O. liberta o programador do conhecimento preciso dos detalhes do "hardware" subjacente, fornecendo um modelo funcional do S.C., designado por máquina virtual, que é mais simples de compreender e programar, e onde são ocultadas situações relacionadas, por exemplo, com interrupções, temporizadores e gerenciamento da memória.

Perspectiva "bottom-up": o S.O. é visto como um gerenciador de recursos. É o programa que gere o S.C., fazendo a atribuição controlada e ordeira dos seus diferentes recursos aos programas que por eles competem. Como objectivo, visa a rentabilização máxima do S.C., garantindo uma utilização tão eficiente quanto possível dos recursos existentes.

2. O que são chamadas ao sistema? Dê exemplos válidos para o Unix (recorde que o Linux não é mais do que uma implementação específica do Unix). Explique qual é a sua importância no estabelecimento de um interface de programação de aplicações (API).

As chamadas ao sistema são rotinas que o S.O. fornece aos programadores para acederem a dispositivos de I/O, por exemplo, "read" (ler um ficheiro, independentemente do suporte físico onde ele se encontra), ou a funcionalidades do S.O., por exemplo, "fork". As chamadas ao sistema têm uma interface bem definida que faz com que o uso delas seja independente da arquitectura do S.C., permitindo que haja uma portabilidade das aplicações. O S.O. fornece ao utilizador uma máquina virtual que é mais simples e conveniente do que a máquina real para o programador. Essa máquina virtual é constituída pelas chamadas ao sistema.

Devido ao facto de em Unix os dispositivos de I/O serem representados por ficheiros, ao utilizar o "read", por exemplo, num destes ficheiros, estamos a ler do próprio dispositivo de I/O (por exemplo, /dev/ttyS0, /dev/hda).

3. Os S.O.'s actuais apresentam um ambiente de interacção com o utilizador de características eminentemente gráficas. Contudo, quase todos eles fornecem em alternativa um ambiente de interacção baseado em linhas de comandos. Qual será a razão principal deste facto?

Os ambientes gráficos têm 4 elementos essenciais: janelas, ícones, menus e apontador. As janelas são blocos rectangulares da área da tela de vídeo usados para executar os programas; os ícones são símbolos pequenos que podem ser clicados, fazendo com que ocorra alguma acção; os menus são listas de acções das quais uma pode ser escolhida; o apontador é implementado com um rato, usado para mover o cursor na tela para seleccionar itens.

A razão do uso de linhas de comando é que estas fornecem uma metalinguagem de programação que permite a automatização de tarefas e construção de comandos complexos, que é feita à custa do redireccionamento de I/O entre vários comandos, o que muitas vezes não é possível num ambiente gráfico.

4. Distinga multiprocessamento de multiprogramação. Será possível conceber-se multiprocessamento sem multiprogramação? Em que circunstâncias?

Multiprocessamento: é quando temos 2 ou mais programas a serem executados em simultâneo, isto é, estão a ser executadas mais que 1 instrução de diferentes processos no mesmo instante (1 processador por cada programa, um "program counter" para cada programa); ou seja, é quando temos mais do que 1 processador (paralelismo).

Multiprogramação: é quando se estão a executar mais do que 1 programa, não simultaneamente mas sim alternados no tempo, sendo que a atribuição do processador é multiplexada no tempo entre os diferentes programas (concorrência).

Os casos possíveis são:

multiprogramação	multiprocessamento
0	0
0	1
1	0
1	1

00: um único processo a ser executado num único processador;
01: vários processos executados em simultâneo, um por cada processador;
10: um único processador a executar diferentes processos, multiplexados no tempo;
11: vários processadores, cada um com vários processos que são multiplexados no tempo;

Logo, é possível conceber-se multiprocessamento sem multiprogramação, quando o número de processos em execução é menor ou igual ao número de processadores (por exemplo, uma estação meteorológica).

5. Considere um S.O. multiutilizador de uso geral. A que níveis é que nele se pode falar de multiprogramação?

1º nível: um único processador e vários utilizadores, o que significa que vamos ter solicitações por parte destes ao processador, as quais têm que ser multiplexadas no tempo em intervalos muito pequenos, de modo a criar-se a ilusão de que o processador lhes está inteiramente dedicado ("time-sharing");

2º nível: quando um comando seguido de '&' é executado e, enquanto isso, podemos continuar a trabalhar (temos as linhas de comando disponíveis); aqui tem-se multiprogramação; os resultados do comando são desviados (para, por exemplo, um ficheiro);

3º nível: "ongap"; quando, como utilizadores, entramos no nosso ambiente, pomos um processo a decorrer e queremos que este continue a ser executado (exemplo de um processo computacionalmente intensivo), mesmo depois de desligarmos a nossa conta, quando voltamos, termos a continuação ou resultado do processo.

6. Os S.O.'s de tipo "batch" são característicos dos anos 50 e 60, quando o custo dos S.C.'s era muito elevado e era necessário rentabilizar a todo o custo o "hardware". A partir daí, com a redução progressiva de custos, os S.C.'s tornaram-se interactivos, visando criar um ambiente de interacção com o utilizador mais confortável e eficiente possível. Será que hoje em dia ainda se justificam sistemas deste tipo? Em que circunstâncias?

Os sistemas de tipo "batch" são caracterizados por se atribuir para execução uma sequência de tarefas, isto é, pelo fornecimento em bloco de um conjunto ("batch") de "jobs" a serem processados sequencialmente e sistematicamente. Este tipo de sistemas optimizam a ocupação do processador e aproveitam os tempos mortos durante a execução de um programa enquanto o processador aguarda pela realização de operações de I/O, para execução de outro programa.

Ainda se justificam hoje em dia estes sistemas onde as tarefas a executar não exigem a intervenção do utilizador, as quais apresentam uma linguagem própria. Como exemplos, temos os processos aritméticos intensivos, muito complicados e que exigem muitos recursos. Um exemplo prático é o "rendering" de filmes de vídeo (aperfeiçoamento de imagem), onde se tem processamento puro e muito específico, em que a intervenção humana é pouco eficiente.

7. Quais são as semelhanças e as diferenças principais entre um S.O. de rede e um sistema distribuído?

A principal semelhança entre estes dois tipos de sistema reside no facto de ambos tirarem partido das facilidades actuais de interligação de S.C.'s, partindo-se então do princípio de que em ambos existe um canal de comunicação entre os S.C.'s.

No que respeita a diferenças, enquanto que o S.O. de rede fornece um conjunto de serviços para uma comunidade (por exemplo, acesso à Internet, partilha de dispositivos remotos, Telnet, etc.), permitindo a existência de individualidade, no sistema distribuído esta partilha resulta de uma interligação de S.C.'s distintos para estabelecer um ambiente integrado de interacção com os utilizadores, no sentido de garantir uma transparência o mais completa possível no acesso aos processadores ou aos diferentes recursos dos S.C.'s interligados, isto é, tira-se partido de vários S.C.'s para se obter um de maior capacidade de processamento (um exemplo de sistema distribuído é um motor de pesquisa na Internet, tal como o "Google").

8. Os S.O.'s de uso geral actuais são tipicamente S.O.'s de rede. Faça a sua caracterização.

Os S.O.'s actuais são S.O.'s de rede pois permitem o acesso de um modo quase indistinto a S.F.'s e a dispositivos de I/O locais e remotos, e incluem aplicações que permitem, entre outras coisas, o "login" em máquinas remotas, o correio electrónico e a navegação na Internet.

9. Os S.O.'s dos "palmtops" ou "personal digital assistants" (PDA's) têm características particulares face ao tipo de situações em que são usados. Descreva-as.

Portabilidade: são leves, pequenos, com mínimo consumo de energia, o que limita a memória (memória de massa quase inexistente); o número de aplicações disponíveis é reduzido (agenda, calculadora, processamento de texto…).

Compatibilidade: os trabalhos executados (edição de texto, folha de cálculo, …) devem ter um formato compatível com outros S.O.'s de modo a podermos ler o que foi feito, por exemplo, num vulgar PC.

10. O S.O. Linux resulta do trabalho cooperativo de muita gente, localizada em muitas partes do mundo, que comunica entre si usando a Internet. Mostre porque é que este facto é relevante para a arquitectura interna do sistema.

As funcionalidades básicas que um S.O. deve ter estão implementadas ao nível do "kernel" ("scheduler" e gestão de memória). Todas as restantes funcionalidades e "device drivers" são implementados em módulos que têm uma interface bem conhecida e definida *a priori*, o que permite a comunicação entre estes vários componentes (módulos) do sistema. Esta abordagem modular do Linux tem uma desvantagem em relação à abordagem monolítica, que é a de ser menos eficiente devido à possibilidade de "overhead" de comunicação entre os vários componentes do sistema.

O Linux é um sistema desenvolvido por muita gente localizada em muitas partes do mundo, e isso contribui decisivamente para a modularidade deste S.O.. É possível assim a qualquer pessoa desenvolver um novo módulo de sistema, testá-lo e modificá-lo isoladamente e acrescentá-lo ao Linux via Internet, aumentando o seu número de funcionalidades. Além disso, porque é um S.O. desenvolvido por pessoas em todo o mundo, pode dizer-se que o Linux é um S.O. construído à imagem do utilizador comum.