Resumo Sistemas Operativos

Um SO (Sistema Operativo) é um programa que controla a execução de programas “aplicacionais”.

Um SO é uma interface entre aplicações e hardware.

Principais objetivos:

* Conveniência
* Eficiência
* Permitir a evolução

SO fornece abstrações sobre o hardware facilitando o uso por parte das aplicações. Conseguido através de interfaces e serviços.

## Infraestrutura de Hardware/Software

Uma imagem com diagrama

Descrição gerada automaticamente

## Serviços de um SO

* Desenvolvimento de programas
* Execução de programas
* Acesso I/O a dispositivos
* Acesso controlado a ficheiros
* Acesso ao sistema
* Deteção de erros e resposta a erros
* Accounting (recolha de estatísticas de utilização

## Sistema operativo como Gestor de Recursos

Um computador é um conjunto de recursos para armazenar dados e processá-los.

O SO é responsável por gerir estes recursos, fazendo este controlo através de software.

Um SO funciona da mesma forma que um software “normal”, ou seja, este é um programa, ou conjunto de programas, executados pelo processador. Frequentemente cede o controlo (depende do processador para voltar a ter controlo).

## Processamento em série

**Primeiras gerações de computadores**

* Não tinham SO, os programadores interagiam diretamente com o hardware.
* Computadores eram comandados através de consolas com luzes, switches, dispositivos de input e uma impressora.
* Utilizadores têm acesso ao computador “em série”

**Problemas**

* Escalonamento:
  + Maioria das instalações usavam uma “sign-up sheet” para reservar tempo de computação; as alocações de tempo podiam ser longas ou curtas, resultando em tempo de computação desperdiçado.
* Tempo de Setup:
  + Parte considerável do tempo era gasto apenas em fazer o setup do programa para o colocar a correr.

## Evolução dos SO

Serial Processing → Simple Batch Systems → Multi-programmed Batch Systems → Time Sharing Systems

## Sistemas de batch

Os primeiros computadores eram muito caros…

* Importante maximizar a utilização do processador
* Tempo perdido por causa do escalonamento e setup não era aceitável
* Criado o conceito sistemas de batch

Monitor:

* Utilizador deixa de ter acesso direto ao processador.
* Software faz a gestão dos jobs/programas.
* Programas/jobs são submetidos através de um operador, que junta todos os jobs num “lote”/”batch” sequencialmente, e coloca-os num device de input, usado pelo monitor.
* Programa volta ao monitor quando termina.
* O programa seguinte inicia de forma automática.

## Monitor point of view

* O monitor controla a sequência de eventos: *Monitor Residence*: software que está sempre em memória.
* O monitor lê o próximo programa/job e dá-lhe controlo (input - cartões ou fitas magnéticas).
* Job/programa devolve o controlo ao monitor.

## Process point of view

* Processador executa instruções da memória onde se encontra o monitor.
* Executa as instruções do programa do utilizador, até chegar ao fim ou encontrar um erro.
* “controlo é passado ao job/programa”, o processador faz fetch e executa instruções de um programa do utilizador.
* “controlo é devolvido ao monitor”, o processador faz fetch e executa instruções do programa no monitor.

## Job Control Language (JCL)

Linguagem de programação específica para fornecer instruções ao monitor (qual o compilador que deve ser usado; quais os dados que vão ser usados).

## Hardware: características desejáveis

* Proteção de memória para o monitor, um programa não deve alterar a área de memória que contém o monitor.
* Temporizador, previne que um programa/job monopolize o sistema.
* Instruções privilegiadas, apenas podem ser executadas pelo monitor.
* Interrupções, dá mais flexibilidade ao SO no controlo de programas do utilizador.

## Modos de operação

**User mode**

* Programas dos utilizadores são executados em user mode.
* Algumas áreas de memória estão protegidas contra o acesso dos utilizadores.
* Algumas instruções não podem ser executadas.

**Kernel mode**

* Monitor é executado em kernel mode.
* Instruções privilegiadas podem ser executadas.
* Áreas protegidas da memória podem ser acedidas.

## Simple batch systems overhead

* Tempo de processador alterna entre:
  + Execução entre programas pelo utilizador
  + Execução do monitor
* Sacrifícios:
  + Alguma da memória principal é alocada para o monitor
  + Algum do tempo do processador é consumido pelo monitor
* Apesar do overhead, os sistemas de batch melhoram a utilização do computador.

## Multi-programmed Batch Systems

Processador está frequentemente em “idle”.

Mesmo com execução automática de jobs em sequência/série.

Os dispositivos de I/O são lentos, em comparação com o processador.

## UniProgramming

* Deve existir memória para guardar o SO e um programa de utilizador.
* O processador passa algum tempo em execução, até que chega a uma instrução de I/O e tem de esperar que a instrução de I/O termine antes de continuar.

## MultiProgramming

* Assumindo que existe memória para guardar o SO (monitor residente) e dois programas de utilizador.
* Quando um job tem de esperar por I/O, o processador pode mudar para outro job que em princípio não está à espera de I/O.
* Assumindo que existe memória para guardar o SO e 3 programas de utilizador.
* Multiprogramming, também conhecido como multitasking, a memória é expandida para guardar 3, 4 ou mais programas e alternar entre todos eles.

## Sistemas time-sharing

* Podem ser usados para lidar com múltiplos jobs interativos.
* Tempo de processador é partilhado entre vários utilizadores.
* Vários utilizadores acedem simultaneamente ao sistema através de terminais. O SO intercala a utilização de cada programa de utilizador em pequenos “bursts”.

## Compatible Time-Sharing System (CTSS)

**Time Slicing**

* Relógio do Sistema gera **interrupts**, aproximadamente a cada 0.2 segundos.
* A cada **interrupt**, o SO ganha controlo e pode atribuir processador a outro utilizador.
* Em intervalos de tempo regulares, o utilizador atual é interrompido e outro utilizador é carregado.
* Programas e dados de utilizadores “antigos” são escritos em disco.
* Programas e dados de utilizadores “antigos” são restaurados para a memória e voltam a correr.

## Principais Avanços

Os sistemas operativos são uns dos tipos de software mais complexos, alguma vez desenvolvidos.

Principais avanços no seu desenvolvimento incluem:

* Processos
* Gestão de memória
* Proteção de informação e segurança
* Escalonamento e gestão de recursos

## Processo

Estrutura fundamental de um SO.  
Pode ser definido como:

* Um programa em execução.
* Uma instância de um programa em execução.
* Uma entidade que pode ser atribuída, e executada, num processador.
* Uma entidade de atividade caracterizada por: uma única linha (thread) sequencial de execução, um estado atual, e associado a um conjunto de recursos do sistema.

## Componentes de um processo

Um processo contém 3 componentes:

* Um programa executável
* Os dados associados necessários pelo programa (variáveis, espaço de trabalho buffers, etc…)
* O contexto de execução do programa, ou estado do processo

O contexto de execução é essencial:

* São os dados internos através dos quais o SO é capaz de supervisionar e controlar o processo.
* Inclui o conteúdo de vários registos do processo (e.g. Program Counter).
* Inclui informação como a prioridade do processo, e se o processo está à espera de um evento I/O.

## Gestão de processos

Todo o estado do processo, em qualquer instante, é contido no seu contexto.

Novas funcionalidades podem ser desenhadas e incorporadas no SO expandido o contexto para incluir nova informação necessária para suportar as novas funcionalidades.

## Diferentes propostas de arquiteturas

Novos requisitos/exigências obrigam a novas formas de organização do SO.

Diferentes abordagens foram testadas:

* Arquitetura Microkernel
* Multithreading
* Symmetric multiprocessing
* Distributed operating systems
* Object-oriented design

## Kernel monolítico

Maioria dos SO têm um kernel monolítico.

O kernel monolítico disponibiliza grande parte das funcionalidades:

* Escalonamento
* Sistema de ficheiros
* Rede
* Device drivers
* Gestão de memória
* Etc…

Tipicamente é implementado como um único processo, todos os elementos partilham o mesmo espaço de endereçamento.

## Microkernel

Inclui apenas funcionalidades essenciais:

* Gestão de memória
* Comunicação inter processo
* Escalonamento

Vantagens:

* Simplifica implementação
* Dá mais flexibilidade
* Indicado para sistemas distribuídos

## Kernel monolítico modular

A maioria dos sistemas UNIX são monolíticos.

Todas as funcionalidades do kernel têm acesso a todas as estruturas de dados internas e rotinas.

O Linux melhorou um pouco este aspeto, sendo estruturado como um conjunto de módulos.

**Loadable Modules**

* Blocos relativamente independentes.
* Objetos, código pode ser linked e unlinked do kernel em runtime.
* Executados em kernel mode, em nome do processo atual.
* Dynamic linking e stackable modules.

Freq 1 2020

1 – C

2 – D

3 – D

4 – B

5 – E

6 –

7 –

8 - C