Frequência I

Introdução

Função dos Sistemas Operativos

- O **Sistema Operativo** (**SO**) é um dos principais componentes de qualquer sistema informático;
- A principal função de um SO é apresentada de três pontos de vistas distintos:
  - **Gestor de recursos que fornece às aplicações um conjunto de recursos lógicos**:
    - Trata os pedidos das mesmas para manipular tais recursos lógicos fazendo executar operações que efetuam a gestão dos recursos físicos correspondentes.
  - **Fornecer uma interface simples e uniforme para a máquina física**:
    - Interface operacional: direcionada aos utilizadores de maneira a manipularem diretamente os recursos lógicos;
    - Interface de programação: corresponde à biblioteca de chamadas de sistema (denominadas ***syscalls***).
  - **Uma máquina virtual que encapsula todos os detalhes da máquina física numa abstração que virtualiza o *hardware* e os mecanismos de baixo nível:**

Critérios de Qualidade de um Sistema Operativo

- Existem vários critérios para avaliar a qualidade de um SO:
  - Desempenho;
  - Segurança;
  - Tolerância a falhas;
  - Qualidade da interface;

Evolução Histórica

1. Os primeiros SO eram simples **monitores de controlo** que geriam sessões:
   a. Cada utilizador tinha o uso exclusívo da máquina.

2. Os **sistemas de tratamentos por *batch*** permitiam uma melhor rentabilização da máquina, armazenando vários programas que, posteriormente, eram executados consecutivamente:
    a. Paralelização de diferentes fases da execução:
        i. **e.g.** efetuar a saída de um programa para um periférico ao mesmo tempo que o programa seguinte efetua o seu processamento.
    b. A paralelização é possível devido ao mecanismo de **interrupções**:
        i. Permite que um periférico notifique **assincronamente** o processador da conclusão de uma operação.
3. Nos **sistemas multiprogramados**, diferentes programas competiam pela utilização do processador, que era libertado sempre que esses processos tivessem necessidade de se bloquear;
4. Nos **sistemas interativos**, vários utilizadores partilhavam o mesmo computador:
    a. Estes sistemas levaram a considerar novos aspetos do SO:
        i. Proteção contra acessos indevidos;
        ii. Interface com o sistema de ficheiros.
5. Com o aparecimento dos **sistemas de memória virtual**, surge a abstração de um espaço de endereçamento virtual de elevada dimensão e independente da memória física disponível:
    a. **Princípio da Localidade de Referência:** os programas tendem a aceder a regiões de memória próximas das regiões acedidas recentemente.
6. Os **computadores pessoais** democratizaram a utilização dos sistemas informáticos:
    a. Permitiu a descentralização da informática;
    b. Reimplementação de mecanismos de segurança;
    c. Evolução da interface com o utilizador (e.g. GUI, ligações a redes).
7. A massificação de sistemas em redes levou a surgirem **sistemas distribuídos.**

Classificação de Sistemas Operativos

- Os SO podem ser classificados de acordo com vários vetores.

Sistemas de Tempo Real e Tempo Virtual

- **Relação da execução com o tempo cronológico:**
  - Inexistente em **sistemas de tempo virtual** (e.g. Unix, Windows);
  - Crucial nos **sistemas de tempo real:**
    - Utilizados em aplicações de controlo que têm de oferecer garantias de cumprimento de determinadas metas temporais.

Dimensão

- SO tradicionais:
  - Dimensão muito grande;
  - Exigem recursos significativos de memória para a sua execução;
- Sistemas Embebidos:
  - Sistemas de dimensão reduzida;
  - Forte integração do *software* com o *hardware*.

Política de Divulgação e Comercialização

- Sistemas proprietários:
  - Impõem restrições à interoperabilidade e à portabilidade entre diferentes plataforma de *hardware*;
- Sistemas *open-source*:
  - Normalização de interfaces;
  - É possível modificar o seu código fonte.

Arquitetura do Sistema Operativo

- O SO é composto por três entidades:
  - **Kernel;**
  - **Biblioteca de *syscalls*;**
  - **Processos sistema;**

Kernel

- Implementa mecanismos de base do SO:

- Gestão de processos;
- Gestão de memória;
- GEstão de ficheiros;
- Gestão de I/O;
- Comunicação entre processos (*inter-process communication*, **IPC**).

Suporte *Hardware* à Execução do *Kernel*

- O funcionamento do *kernel* é influenciado por mecanismos de *hardware:*
  - Permitem garantir o seu funcionamento **seguro:**
    - Isolamento entre processos do utilizador e o *kernel* do SO;
      - O isolamento é garantido pela gestão da memória que permite que apenas certas posições de memória sejam acedidades por um utilizador.
- ***User Mode***:
  - Restringe o acesso a determinadas posições de memória e instruções que podem quebrar o referido isolamento;
- ***Kernel Mode****:*
  - Não existem as restrições acima;

Interrupões e Exceções

- A passagem de um modo para outro acontece através de exceções/interrupções:
  - Forçam a transição de modo;
  - Colocam em execução uma rotina de tratamento previamente definida para cada tipo de evento:
    - O contexto do processo em execução é guardado à *priori* de forma que seja possível retomar o processamento posteriormente.

Chamadas de Sistema

- Implementadas por uma **função** que invoca a exceção — *trap*:
  - Transfere o controlo para o *kernel* e automaticamente coloca o processador no *kernel mode*;

    - No *kernel*, é feita a escolha do código apropriado que implementa a chamada de sistema em causa;
    - No final da chamada, a instrução de retorno da interrupção devolve o controlo para a função de biblioteca que, por sua vez, retorna para o código do utilizador.
- O código das aplicações não têm acesso às estruturas de dados mantidas no *kernel.*

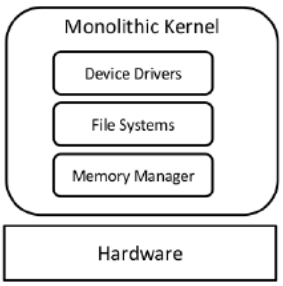
Processos Sistema

- Executam funções que podem ser delegadas em processos autónomos:
    - Reduzem a complexidade do *kernel*;
    - Aumentam a sua robustez.

Organização do *Kernel* do Sistema Operativo

- A arquitetura do *kernel* afeta a capacidade de adaptação do sistema a novos periféricos, a segurança, a fiabilidade e o seu desempenho;

Sistemas Monolíticos



Kernel monolítico.

- Consistem apenas num programa:
    - Estruturas de dados globais;
    - Eventualmente organizado por módulos;
- A extensibilidade do SO a novos periféricos é conseguida ao adicionar, ao *kernel*, novos gestores de periféricos que adaptam as funções de I/O do *kernel* às caraterísticas de um periférico específico.

Sistemas em Camadas

- Divide o *kernel* em várias camadas que implementam diferentes funcionalidades;

- Obriga que camadas mais externas invoquem funções das camadas internas;
- É garantindo um maior isolamento e extensibilidade do *kernel*;
- Perda de desempenho devido à sobrecarga do processamento introduzido pelos mecanismos supracitados.

Micro-*kernels*

- Oferecem apenas serviços de gestão de processos, memória e IPC.
- As restantes funcionalidades são fornecidas por processos sistema executados fora do *kernel*:
  - Aumenta a flexibilidade do sistema.
- Organização mais segura e rebosta;
- O mecanismo de IPC entre processos sistemas e entre estes o *kernel* é significativamente mais lento por ter de ser efetuado através de *syscalls;*

Máquinas Virtuais

- Componente de *software* capaz de executar aplicações, fazendo crer que se executam sobre a máquina real;
- Não são SO:
  - Pretendem simplesmente oferecer a interface da máquina física a diferentes aplicações que podem ser SO.

Processos

> 💡 A multiprogramação consiste na execução pseudoparalela de vários programas na mesma máquina, onde os vários fluxos de execução são multiplexados no tempo, cada fluxo como um programa independente designado por **processo**.

Conceito "Processo"

- **Um processo é um fluxo de atividade no sistema, correspondendo a uma instância de um programa em execução**:
  - Os elementos principais constituintes de um processo são:

        - Espaço de endereçamento:
            - *Code;*
            - *Stack*;
            - *Heap.*
    - Tem ao seu dispor várias operações:
        - Intruções máquina do processador;
        - Operações exportadas pelo SO (as *syscalls*).
    - Tem um estado:
        - Salvaguardado e posteriormente reposto, de forma a permitir a comutação entre processos.

Modelo de Segurança

- Baseia-se:
    - No isolamento entre processos:
        - Os processos não podem interferir uns com os outros;
    - Na existência de utilizador que é dono do processo:
        - A associação de um processo a um utilizador é precedida de uma operação de autenticação perante o SO:
            - e.g. *password.*

Hierarquia de Processos

- Um processo, ao criar outro, estabelece a relação pai-filho;
- Algumas informações são herdadas do processo pai (*parent process*, PP) para os processos filho (*child processs*, CP):
    - Identificação do dono;
    - Ficheiros abertos;
    - Periféricos de I/O.

Recursos Associados a um Processo

- Cada processo tem de se associar aos recursos que pretende utilizar:

        - Permite ao SO fazer validações de segurança antes da utilização;
        - Permite ao SO implementar quotas de gestão de recursos.

Objeto "Processo"

- Um processo pode ser visto como um objeto:
    - Tem **propriedades** e **operações**;
    - **Propriedades fundamentais:**
        - ID;
        - Ficheiro com o programa executável;
        - Espaço de endereçamento;
        - Prioridade;
        - Estado de execução;
        - PP;
        - Canais de I/O;
        - Ficheiros abertos;
        - Quotas de utilização dos recursos;
        - Contexto de segurança;
        - Ambiente utilizador;
    - As operações definem a interface do SO para manipular processos e incluem operações como:
        - Criar um novo processo;
        - Auto-terminação de processos;
        - Eliminação de processos;
        - Esperar pelo término de um processo;
        - Ler estado do processo.

Rotínas Assíncronas

- Permitem a um processo associar uma rotina a um determinado evento:
    - A rotina é colocada em execução pelo SO quando o evento ocorre:

        - Não é necessário haver um teste sistemático.
- **Particularmente úteis para tratar de I/O ou condições de exceções**.

Modelo Multitarefa

- Tarefas (*threads*) são fluxos de execução independentes dentro do mesmo processo;
- Ao contrário dos processos, as *threads* partilham o mesmo espaço de endereçamento e recursos;
    - **A comutação entre *threads* é mais rápida do que entre processos.**
- As *threads* podem ser implementadas pelo *kernel* (**tarefas reais**) ou por uma biblioteca a nível do utilizador (**co-rotinas/pseudotarefas**):
    - As co-rotinas são mais eficientes na criação e comutação:
        - Não obrigam a uma passagem a *kernel mode* para as operações de criação e comutação;
        - Obrigam, no entanto, a uma comutação explícita;
        - Não permitem o paralelismo quando uma *thread* se bloqueia;

Objeto *Thread*

- **Propriedades:**
    - ID;
    - Procedimento de arranque;
    - Prioridade.
- **Operações**:
    - Criação;
    - Eliminação;
    - Transferência de controlo (i.e. comutação);
    - Esperar pelo término de uma *thread*.

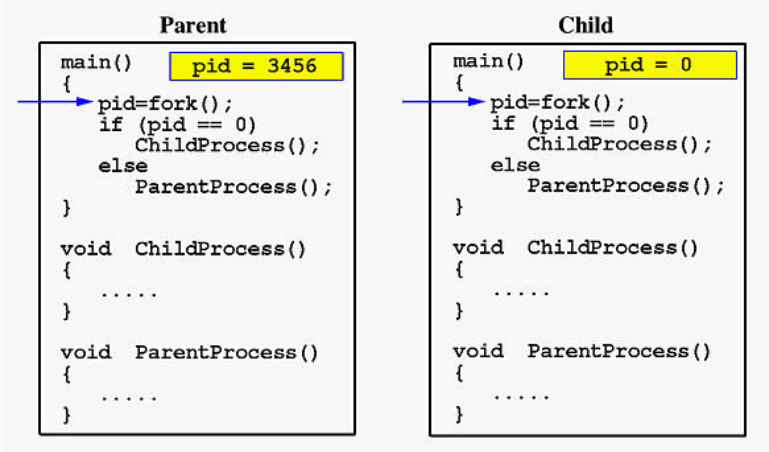
Processos em Unix

Modelo de Segurança

- Baseia-se na associação de um identificador de utilizador e de um identificador de grupo a cada processo:
  - Permite validar a autorização de uso de um dado recurso;

Operações sobre Processos

- A criação de processos é feita com a *syscall* `fork` :
  - Cria um novo processo absolutamente idêntico ao PP:
    - Para executar um programa diferente, é necessário invocar a *syscall* `exec` .
  - **Valores retornados numa chamada a `fork` :**
    - `-1` : erro;
    - `0` : CP;
    - qualquer outro: PP;

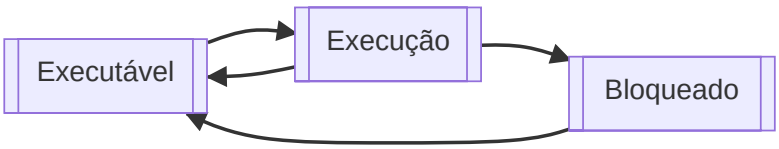


Gestor de Processos

> 💡 O gestor de processos implementa os mecanismos que permitem criar e comutar processos.

Representação de Processos

Estados de Execução



- **Executável → Execução:** o processo foi selecionado para execução pelo despacho porque é mais prioritário do que o processo corrente ou porque este último deve abandonar o CPU (i.e. bloqueou-se ou terminou);
- **Em Execução → Executável:** um processo na lista de Executáveis tem maior prioridade, pelo que o despacho efetua a comutação;
- **Em Execução → Bloqueado:** o processo tem de interromper a sua execução porque espera um acontecimento sem o qual não pode continuar a execução do programa;
- **Bloqueado → Executável:** o acontecimento que mantinha o processo bloqueado ocorreu e este passa novamente a poder ser executado.

Contexto

- O núcleo mantém um contexto que tem toda a informação associada a cada processo:
  - Permite retomar a execução de um processo interrompido;
  - Informações guardadas: acumuladores, registos de uso geral, *program counter*, *stack pointer* e *status flag register*.
- O núcleo mantém uma lista de processos executáveis, com o contexto dos processos que competem pelo processador:
  - A lista procura refletir a prioridade dos processos;
  - O processo na cabeça da lista é o que se encontra em execução; o processo seguinte é o que o deverá substituír na próxima comutação.

Comutação de Processos

Despacho

> 💡 A rotina de despacho é invocada sempre que o processo em execução possa ser comutado.

- A comutação pode acontecer em duas situações:

    - Após um processo mudar de estado;
    - Após uma interrupção de *hardware*.
- A comutação consiste em armazenar o estado atual do processador no contexto do processo atual, e repor no processador os valores previamente armazenados no contexto do processo que se irá executar de seguida.

Escalonamento

> 💡 O escalonador determina quais os processos que devem estar em execução em que instantes de tempo.

- Controla a prioridade dos processos:
    - Faz com que os processos mais prioritários tenham uma maior probabilidade de vir a dispor do CPU;
- Objetivos:
    - Otimizar a execução dos programas;
    - Procurar equilibrar a carga no sistema;
    - Dotar o sistema de grande reatividade a condições externas.
- Existem diversos tipos de algoritmos de escalonamento:
    - ***Shortest Job First*:** O sistema executa primeiro os processos que menos usam o processador, otimizando a produtividade do computador.
    - **Tempo de Execução Partilhado:**
        - Procura dar a cada utilizador um serviço equitativo que distribua o CPU de forma justa, mesmo que para tal os programas não se executem da forma mais célere;
        - O escalonador permite apenas que o processo em execução utilize o CPU durante um intervalo de tempo limitado: ***time-slice*** ou ***quantum***;
        - Findado um ***quantum***, o despacho é chamado e a lista de processos executáveis é analisada para determinador o processo a ser processado:
            - Poderá ser o mesmo se continuar a ser o mais prioritário.
    - **Tempo de Execução Partilhado com Prioridades:**

        - A evolução do algoritmo anterior consistiu na criação de mecanismos de prioridades que procuram dar primazia aos processos **I/O-bound (interativos) sobre os CPU-bound**:
            - Os primeiros tendem a executar-se por períodos curtos, bloqueando-se de seguida, pelo que libertam o processador para outros processos.
    - **Prioridades Dinâmicas:**
        - Existem sistemas de prioridades fixas, onde cada processo é colocado numa sublista de processos executáveis, consoante a prioridade;
        - Existem sistemas de prioridades dinâmicas, em que o **algoritmo aumenta o valor da prioridades aos processos que se bloqueiam frequentemente**, pois têm maior probabilidade de ser I/O-bound.
    - ***Quantum* variável:**
        - O valor do *quantum* é ajustado de forma a diminuir o número de comutações:
            - Aumenta o *quantum* dos processos I/O-bound, permitindo que estes terminem a sua execução mais rapidamente.
    - **Preempção**:
        - Permite que um processo mais prioritário reaja mais rapidamente a um acontecimento;
        - Consiste em retirar um processo menos prioritário de execução logo após um processo mais prioritário passar ao estado **Executável**;
        - O despacho é invocado sempre que um processo muda de estado.

Sincronização

Necessidade de Secções Críticas

- A multiprogramação implica que os processos podem ser comutados em qualquer momento durante a execução de um programa;
- A partilha de variáveis entre programas pode conduzir a que pressupostos lógicos na programação sequencial sejam violados, e.g. uma variável pode mudar de valor entre leitura e teste.

> 💡 Em programação concorrente, sempre que se testam ou se modificam estruturas de dados partilhadas é necessário efetuá-lo dentro de uma secção crítica.

Requisitos de Uma Secção Crítica

1. **Exclusão mútua**: se uma tarefa está a executar a secção crítica, então nenhuma outra a pode estar a executar;
2. **Progresso** (i.e. não haver *deadlocks*): se nenhuma tarefa está a execução Secção Crítica e existem outras que a pretendem executar, então apenas estas devem participar no protocolo de decisão — que não deverá prolongar-se infinitamente — que seleciona uma delas. A exclusão mútua deve ser resolvida entre as tarefas que estão em competição e não afetar as restantes;
3. **Míngua**: se uma tarefa pretende executar a secção crítica, então alguma vez no futuro irá conseguir o acesso à mesma e não ficará sempre impedida de o fazer;
4. **Eficiência**: Na ausência de contenção — conflito no acesso a um recurso partilhado —, se uma única tarefa quiser entrar na secção crítica, consegui-lo-á de forma eficiente.

Exclusão Mútua Baseada em *Software*

- Recorre a simples leituras e escritas de variáveis;
- Implementam um trinco lógico com operações `fechar` e `abrir`:
  - Soluções complexas e não são fáceis de generalizar a um número qualquer de tarefas.

Exclusão Mútua Baseada em *Hardware*

- Implementada através da inibição de interrupções;
  - Tal mecanismo só pode utilizado pelo núcleo e em trechos de código de reduzida dimensão, não funcionando em multiprocessadores;
- Solução mais habitual é **implementar trincos lógicos recorrendo a instruções especiais que permitam fazer testes e atribuição de uma posição de memória de forma atómica (de** tal forma que não possa ser intercalada pelo acesso de outra tarefa a essa posição de memória):
  - Implica a existência de instruções especiais (*test-and-set*);

- Implica uma espera ativa em que o processo que não consegue entrar fica em ciclo a testar a variável trinco;

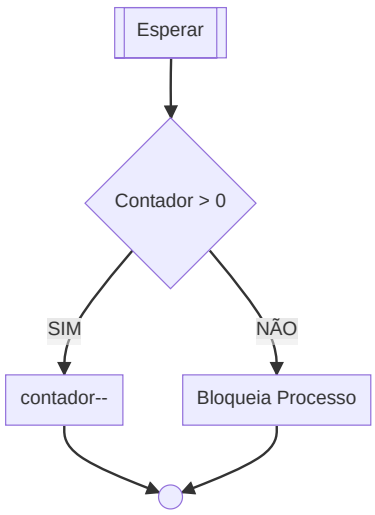
Exclusão Mútua com Objetos do Sistema Operativo

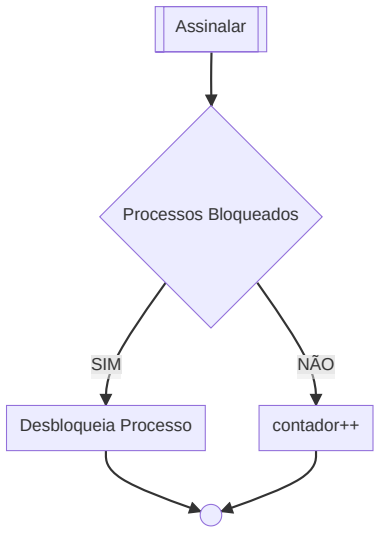
- Permitem evitar espera ativa;
- Consistem na (i) utilização de um objeto *mutex*, que tem um valor do trinco, atualizado de forma atómica e (ii) uma lista de tarefas bloqueadas no objeto.
- Quando uma tarefa tenta aceder à exclusão mútua e encontra o trinco fechado, o sistema operativo altera o estado da tarefa para **Bloqueado:**
  - Evita que a tarefa não entre em competição pelo CPU até outra tarefa abrir o trinco e desbloquear a primeira tarefa.

Programação Concorrente

- A sincronização estabelece-se através de um objeto do sistema operativo.

Semáforos





- Objetos do sistema operativo que têm na sua representação interna um contador e uma fila de tarefas bloqueadas no semáforo e exportam dois métodos:
  - `esperar`: bloqueia a tarefa que o invoca, caso não haja recursos disponíveis;
  - `assinalar`: desbloqueia uma tarefa caso haja alguma bloqueada nesse semáforo.

- **Os semáforos precisam de ser inicializados.**