Resumo – Sistemas Operativos

Tema 1 – INTRODUÇÃO

- função SO: gerir componentes e usá-los de forma correcta; providenciar interface simples para HW
- SO → conjunto de programas que permitem uma interacção simplificada entre o utilizador e a máquina
- SO → porção do SW k funciona em modo núcleo (MN); ex. liberdade para modificar compilador, mas não para modificar Clock Interrupt Handler; ex. gerir passwords (MU) → ajudam SO

1.1. O que é um Sistema Operativo?

- **Máquina virtual**: Dar ao utilizador a ilusão de dispor de uma máquina muito mais fácil de utilizar e programar do que o hardware.
- **Gestor de recursos**: Efectuar a gestão dos diversos componentes da arquitectura de um computador, impondo ordem na atribuição de recursos aos programas. Tirar máximo partido dos recursos disponíveis (Tempo de CPU, memória, etc) ; time multiplexing → turnos ; space multiplexing → cada um fica com parte dos recursos

1.2. História dos Sistemas Operativos

➢ 1º computador → Charles Babbage (1792-1871) → realizou k precisaria de SW → Ada Lovelace → 1ª programadora → linguagem Ada

1) Primeira geração (1945-1955): Válvulas e relés mecânicos

- computadores k enchiam salas inteiras
- + de 20000 válvulas
- ligar plugboard ao computador
- cálculos numéricos, tabelas, etc

2) Segunda geração (1955-1965): transistores, sistemas por batchs

- programador traz card para IBM 1401 → 1401 grava card em tape → operador leva tape para IBM 7094 → 7094 faz computação → operador leva tape para 1401 → 1401 imprime output

3) Terçeira geração (1965-1980): ICs e multiprogramação

- System/360 → computador k daria para fazer as 2 coisas → 1º a usar ICs → + potente
- SO do 360 demasiado complexo
- **Multiprogramação**: dividir memória em várias partições → enquanto um trabalho espera por I/O, outro pode ser processado → CPU ocupado quase 100% do tempo
- **Spooling**: ler trabalhos das cards para o disco assim k eles chegam (1401 desnecessário)
- tempo de resposta enorme → pequenos bugs dificéis de corrigir → timesharing → variante da multiprogramação → computador fornece serviço interactivo rápido para quem precisa com mais urgência
- **MULTICS** → poder computacional através de tomadas (tipo energia eléctrica)
- Minicomputadores → série PDP
- UNIX desenvolvido a partir de um minicomputador por um trabalhador da MULTICS

4) Quarta geração (1980-presente): Computadores pessoais

- **LSI circuits** → chips com milhares de transistores num cm^2 → era dos PCs (microcomputadores) → muito barato → abre a possibilidade de toda a gente comprar um PC
- 1974 → Intel → 8080 → precisava SO → Garz Kildall → CP/M → K. pediu direitos e formou Digital Research
- 1980s → IBM → IBM PC → precisava SO → Bill Gates → DOS → MS-DOS

- Gates vende SO a fabricantes MAS Kildall vende SO a utilizadores finais
- invenção do GUI
- Steve Jobs → computador Apple → Apple com GUI → Lisa (fracasso) → Macintosh (sucesso) → Mac + user friendly e + barato k Lisa
- Microsoft → SO com GUI → Windows → funcionava sobre o MS-DOS → Win95 freestanding (SO) → Win98 (16 bits) → Win NT → 32 bits → WinNT5 = Win2000 → WinMe
- UNIX → Linux → GUI = X Windows
- Network OS → existência de múltiplos computadores k se podem ligar uns aos outros e partilhar ficheiros
- Distributed OS → OS k gere vários processadores

➢ Ontogeny recapitulates Phytogeny → cada nova espécie passa pelo desenvolvimento de todos os antecessores

1.3. A Diversidade de Sistemas Operativos (OS Zoo)

1) **SO de Mainframes**: processar vários trabalhos simultâneamente; número considerável de I/O
2) **SO de Servidores**: funcionam em servidores; servem muitos utilizadores ao mesmo tempo; permitem partilha de recursos de HW e SW
3) **SO de Sistemas Multiprocessador**: gerem vários processadores em termos de escalonamento, etc
4) SO de Computadores Pessoais: providenciar interface amigável ao utilizador; Windows, MacOS, Linux, etc
5) **SO de Tempo-Real**: cumprir metas de tempo; gerir evente k TÊM de ocorrer em determinado momento → hard real-time system; falhar meta pode ser aceitável → soft real-time system (áudio digital, sistemas multimédia)
6) **SO Embebidos**: PDAs, SO de aparelhos k não são computadores (TV, micro-ondas, télémoveis, etc); PalmOS, Windows CE (Consumer Electronics)
7) **SO de Smart Card**: cartões de crédito; uma única função: pagamentos; orientados a Java

1.4. Revisão do Hardware dos Computadores

1) **Processadores**
   - **Processador (CPU)**: Elemento activo do sistema que executa processos. Obtém instruções da memória, descodifica-as e executa-as.
   - multiprogramação → gurdar registos para posterior reutilização
   - **pipeline**: fetch, decode, execute separados
   - **superscalar CPU**: fetch, decode → holding buffer → execute
   - **Modo núcleo (Kernel Mode)**: Modo privilegiado do processador, para o qual todas as instruções estão disponíveis. Só o Sistema Operativo é que tem acesso a este modo.
   - **Modo utilizador (User Mode)**: Disponível um subconjunto das instruções do CPU. É neste modo que correm as aplicações.
   - **chamada ao sistema (system call)**: mecanismo usado para requisitar um serviço do modo núcleo do SO
2) **Memória**
   - **Hierarquia da memória**
     — Registos: tão rápidos qto a CPU
     — Cache
     — Memória Principal: RAM
     — Discos Magnéticos: + barato; lento; aparelho mecânico
     — Tapes Magnéticas: baratíssimo, pode-se remover; usado para backups

- **Memória e Gestão de memória**: Divisão estruturada da memória de modo a ser possível o carregamento de diversos programas na memória principal.
- **Realocação e Protecção**: Existência de mecanismos que permitam o crescimento da memória de dados de um programa. (Base/Limit Registers)

3) Dispositivos I/O

- dispositivos I/O → controlador, próprio dispositivo
- controlador apresenta interface simples ao SO
- driver do dispositivo (device driver) → SW k "fala" com o controlador enviando instruções e aceitando respostas; funciona em MN
- SO → driver → controlador → dispositivo
- **Formas de colocar o Driver no Núcleo:**
  - ligar o núcleo ao driver e reiniciar (UNIX)
  - entrar num ficheiro do SO dizendo k o driver é necessário e reiniciar (Windows)
  - SO aceita novos driver sem reiniciar (USB, etc)
- **Formas de realizar Input e Output:**
  - busy waiting
  - interrupt
  - uso de um chip DMA (Direct Memory Access)

4) Buses

- **Principais:**
  - IBM PC ISA (Industry Standard Architecture)
  - PCI (Peripheral Component Interconnect): usado com a maioria dos dispositivos I/O de alta velocidade
- **Especializados:**
  - IDE: ligação de dispositivos como discos e CD-ROMs
  - USB (Universal Serial Bus): ligação dos dispositivos I/O de baixa velocidade, como teclado, rato; driver único; ligados sem necessidade de reiniciar
  - SCSI (Small Computer System Interface): ligações de alto desempenho como discos rápidos, scanners e outros dispositivos que requerem muita potência
- **Outros**
  - IEEE 1394 (FireWire):ligação de câmeras digitais e outros dispositivos multimédia
- **Plug and Play**: dispositivos I/O recebem automaticamente níveis de interrupt e endereços I/O
- A tecnologia Plug and Play (PnP) , que significa "ligar e usar", foi criada com o objectivo de fazer com que o computador reconheça e configure automaticamente qualquer dispositivo que seja instalado, facilitando a expansão segura dos computadores e eliminando a configuração manual.

1.5. Conceitos dos Sistemas Operativos

1) Processos

- Um **processo** é basicamente um programa em execução. Num sistema multi-programado, vários processos podem estar a correr simultaneamente. Contudo, quando existe um só processador, apenas um processo pode utilizá-lo em cada instante temporal. Os processos concorrem pelo processador e cooperam entre si para realizar tarefas mais complexas.
- **Espaço de endereçamento**: lista de localizações na memória; contém o programa executável e os seus dados
- **Tabela de processos (process table)**: tabela k guarda todas as informações de um processo quando este é suspenso de modo a poder continuar o trabalho onde parou
- **Processo-filho**: processo criado por outro processo

- Comunicação entre processos (ver Tema 2)

2) **Impasses (Impasses)**
   - **Impasse**: situação em k 2 processos estão bloqueados pk cada um espera por algo k o outro já tem
3) **Gestão de Memória**
   - memória pode conter vários programas ao mesmo tempo → gestão e protecção
   - memória virtual (ver Tema 3)
4) **Input/Output**
   - I/O gerido pelo SO (ver Tema 5)
5) **Ficheiros (Sistemas de Ficheiros)**
   - Ficheiros → modelo abstracto do disco
   - Directoria → forma de agrupar ficheiros
   - Sistema de Ficheiros → hierarquia
   - Outros conceitos: path name, root directory, working directory, mounted file system, block special files, character special files, pipe (ver Tema 4)
6) **Segurança**
   - Exemplo: rwxr-x--x → Owner / Other Group Members / Everyone else (directory x search)
7) **Shell** → esta parte tem k ser treinada no Ubuntu
   - SO pode ser visto como um código que executa as chamadas ao sistema;
   - Interpretador de comandos, apesar de não ser parte do sistema operacional, faz uso das chamadas
   - ao sistema para interfacear o usuário e o SO
   - Exemplo: dir, ls, clear, cls.
8) **Reciclagem de Conceitos**
   - Exemplo: "contiguously allocated files" em CD-ROMs

1.6. Chamadas ao Sistema (System Calls)

- Uma **chamada ao sistema** (system call) é o mecanismo usado pelo programa para requisitar um serviço do SO, ou mais especificamente, do modo núcleo do SO.
- Fazer uma chamada ao SO é como realizar uma chamada a um procedimento, contudo chamadas ao sistema são executadas em MN e chamadas a procedimentos em MU
- **Chamadas ao Sistema para Gestão de Processos**: fork, waitpid, execve, exit
- **Chamadas ao Sistema para Gestão de Ficheiros**: open, close, read, write, lseek, stat
- **Chamadas ao Sistema para Gestão de Directorias**: mkdir, rmdir, link, unlink, mount, unmount
- **Chamadas ao Sistema Diversas**: chdir, chmod, kill, time

1.7. Estrutura de um Sistema Operativo

1) **Sistemas Monolíticos**
   - exemplo mais comum de SO, no qual aparentemente não existe estrutura
   - qualquer função do SO pode comunicar com qualquer uma das outras
2) **Sistemas em Camadas (Layered Systems)**
   - SO estruturado segundo um conjunto de camadas funcionais
   - cada camada utiliza serviços de camadas que lhe são interiores
   - cada camada é uma máquina virtual com uma interface bem definida
   - camada mais baixa (gestão de processos) corresponde ao núcleo do sistema operativo
   - Camadas
     - Camada 0: Gestão de Processos

- Camada 1: Gestão de Memória
- Camada 2: Comunicação Operador-Processo
- Camada 3: Gestão de Input/Output
- Camada 4: Programas do Utilizador
- Camada 5: Operador

3) **Máquinas Virtuais**
   - máquina implementada através de SW que executa programas como um computador real
   - cópia isolada, e totalmente protegida, de um sistema físico
   - computador fictício criado por um programa de simulação
   - cada processo recebe uma cópia exacta do computador actual
4) **Exokernel**
   - cada utilizador recebe um clone exacto do computador actual
   - VM1: blocos 0-1023; VM2: blocos 1024-2047; etc
   - exokernel → organiza recursos das várias VM e impede-as de interferir umas com as outras
   - cada VM acha k tem o seu próprio disco
5) **Modelo Cliente-Servidor**
   - mover código para camadas superiores
   - retirar o máximo de código do núcleo (kernel) deixando um microkernel
   - implementar maioria do SO em processos
   - processo cliente comunica com processo servidor através de mensagens k passam pelo núcleo
   - SO encontra-se organizado segundo módulos à volta de um núcleo (kernel)
   - núcleo geralmente pequeno (Micro-kernel), comunicando com o hardware e estabelecendo a comunicação entre os diversos módulos
   - processo cliente e de processo servidor correm em modo utilizador
   - facilmente adaptável a sistemas distribuídos ; estrutura mais estável (teoricamente...)

Tema 2 – PROCESSOS, TAREFAS E IMPASSES

2.1. Processos

2.1.1. O Modelo de Processo

- **Processo**: Entidade activa, que corresponde a um programa em execução. Cada processo tem um espaço de endereçamento próprio. A gestão de processos é da responsabilidade do sistema operativo, que utiliza estruturas de dados (process tables) que descrevem o contexto de execução de cada processo. O próprio sistema operativo é também um conjunto de vários processos.
- **Programa**: Sequência de instruções sem actividade própria (não confundir com processo)
- Multi-programação: Num sistema multi-programado, mesmo que só exista um processador é possível vários processos estarem activos simultaneamente. Contudo, em cada instante temporal, apenas um deles pode utilizar o processador. A esta ilusão de vários processos correrem aparentemente em paralelo, dá-se o nome de pseudo-paralelismo. Não devem ser feitas assunções em relação à ordem de comutação do processador, devido a: Existência de interrupções, Falta de recursos, Entrada de processos prioritários. Depois de uma comutação do processador, o próximo processo a utilizá-lo é escolhido pelo sequenciador de processos do SO. Os vários processos "competem" entre si pela atenção do processador. Mas também podem trabalhar em conjunto para a realização de tarefas mais complexas. Esta cooperação exige ao SO a existência de mecanismos de sincronização e comunicação entre processos.

2.1.2. Criação de Processos

- **Principais eventos que causam a criação de processos:**
  1. Inicialização do sistema
  2. Execução de uma chamada ao sistema de criação de um processo por parte de outro processo
  3. Criação de um processo a pedido do utilizador
  4. Inicialização de um trabalho batch (batch job)
- **daemons**: processos de fundo k aguardam sinais (e-mail, impressão, etc)
- fork → cópia exacta do processo pai → mas espaço de endereçamento diferente → modificações no filho não são visíveis nem para o pai → filho executa execve para mudar a sua imagem de memória

2.1.3. Terminação de Processos

- **Condições que causam a terminação de processos:**
  1. **Saída normal (voluntária)**: acaba o seu trabalho → exit ; clique na cruz no canto da janela
  2. **Saída de erro (voluntária)**: compilação de um ficheiro k não existe → pop-up dialog box
  3. **Saída fatal (involuntária)**: erro causado pelo processo; bug; instruções ilegais; dividir por 0
  4. **Terminado (killed) por outro processo (involuntária)**: Processo A "mata" Processo B → kill

2.1.4. Hierarquias de Processos (UNIX)

- prcesso pai e processo filho continuam associados
- processo filho pode criar + processos → hierarquia
- processo e seus descendentes → Grupo de Processos
- envio de um sinal (teclado) → todo o grupo
- UNIX → processo init → árvore de processos com init na raiz
- Windows → sem hierarquia de processos

2.1.5. Estados de um Processo

- **Em Execução (Running)**: O processo está a utilizar o processador
- **Executável (Ready)**: O processo está activo, mas está à espera de ter a atenção do processador, que nesse instante está dedicado a outro processo
- **Bloqueado (Blocked)**: O processo está inactivo …
  - — à espera que termine uma operação de I/O
  - — à espera que outro processo liberte recursos
  - — devido à ocorrência de uma page fault – não possui recursos na memória principal
- process-structured OS → processos sequênciais e camada inferior com escalonador (Scheduler)

2.1.6. Implementação de Processos

- Cada vez que ocorre uma comutação de processos, o SO salvaguarda e actualiza informação relevante na Process Table do processo que "perdeu" a CPU
- os interrupts é k causam as comutações de processos
- **Passos de uma Interrupção:**
  1. HW guarda program counter (entre outros) na pilha (stack)
  2. HW carrega o novo programa indicado no vector de interrupção
  3. procedimento em assembly guarda os registos
  4. procedimento em assembly gera nova pilha
  5. procedimento em C lê input
  6. escalonador decide qual o próximo processo a trabalhar

7. controlo devolvido ao procedimento em assembly
8. procedimento em assembly inicia novo processo

2.1.7. Escalonamento

2.1.7.1. Introdução ao Escalonamento

- Quando ocorre uma comutação de processos, o escalonador (scheduler) escolhe um processo para o qual se atribui a CPU
- A escolha é feita de acordo com um dado algoritmo de escalonamento
- Após a escolha do sequenciador, o despachante (dispatcher) encarrega-se de colocar o processo em execução.
- timesharing trouxe complexidade aos algoritmos de escalonamento
- **Comportamento dos Processos**
  - → impulso de computação (CPU burst) → CPU é utilizado
  - → impulso de I/O (I/O burst) → espera-se por resposta de dispositivo externo
  - → processos de computação (compute-bound) → impulsos de computação longos
  - → processos de I/O (I/O-bound) → impulsos de computação curtos → prioritários
- **Quando escalonar?**
  - → quando um processo é criado → lançar pai ou filho
  - → quando um processo termina → escolher outro
  - → quando um processo bloqueia → devia considerar-se razão do bloqueio, mas escolanador burro
  - → quando ocorre uma interrupção I/O → considerar processo k esperava pelo I/O
  - → algoritmos de escalonamento não-preemptivos → deixam funcionar processo até k bloqueie
  - → algoritmos de escalonamento preemptivos → deixam funcionar durante um tempo máximo → interrupção do relógio (clock interrupt)
- **Categorias de Algoritmos de Escalonamento** (dependem do tipo de sistema)
  1. Sistema batch → utilizador à espera → alg. não-preemptivos → aumento do desempenho
  2. Sistema interactivo → preempção é essencial
  3. Sistema em tempo real → preempção nem sempre é necessária → processos rápidos
- **Objectivos dos Algoritmos de Escalonamento**
  - Todos os Sistemas
    - → Justiça → garantir que todos os processos terão direito a tempo de CPU
    - → Prioridades → dar maior tempo de CPU aos processos com maior importância
    - → Equilíbrio → manter os recursos do sistema com uma taxa de ocupação equilibrada
  - Sistemas Batch
    - → Throughput → maximizar o número de trabalhos por hora
    - → Turnaround time → minimizar tempo entre submissão e terminação
    - → Utilização da CPU → manter CPU sempre ocupada
  - Sistemas Interactivos
    - → Tempo de Resposta → deve ser curto (interacção)
    - → Proporcionalidade → responder às expectativas dos utilizadores (ex. ligar/desligar Internet)
  - Sistemas em Tempo-Real
    - → Cumprir metas (deadlines) → evitar perda de dados
    - → Previsibilidade → um mesmo programa deve ser correctamente executado, independentemente da carga do sistema (Sistemas Multimédia)

2.1.7.2. Escalonamento em Sistemas Batch

- **First-Come First-Served**
  - alg. não preemptivo; tem uma fila de espera
  - em caso de bloqueio, processo colocado no fim da fila
  - Vantagem: simples, justo
  - Desvantagem: processos de computação demoram muito tempo a concluir
- **Shortest Job First**
  - Vantagem: diminui turnaround time
- **Shortest Remaining Time Next**
  - Vantagem: trabalhos novos e curtos são tratados rapidamente
- **Three-Level Scheduling**
  - admission scheduler → selecção de uma mistura de processos de computação e de I/O; admitir trabalhos curtos primeiro
  - memory scheduler → espaço insuficiente → trabalho enviado para disco
  - grau de multiprogramação: quantidade de processos em memória
  - CPU scheduler → escolhe um processo para usar a CPU

2.1.7.3. Escalonamento em Sistemas Interactivos

- **Round-Robin Scheduling**
  - a cada processo é atribuído o seu quantum → alg. preemptivo
  - **quantum**: intervalo de tempo em k um processo é autorizado a funcionar
  - quantum usado ou processo bloqueado → processo colocando no fim da fila
  - ideia implícita → todos os processos têm a mesma importância
  - Desvantagens:
    - quantums curtos → muitas comutações → desperdÍcio da CPU
    - quantums longos → poucas comutações → tempo de resposta longo
  - Solução: quantum > impulso de CPU → preempção raramente acontece → eliminação da preempção → comutações acontecem "apenas" quando são necessárias
- **Priority Scheduling**
  - existe prioridade
  - evitar monopolização por parte dos processos de alta prioridade → diminui-se as suas prioridades a cada interrupção do relógio OU atribui-se um quantum máximo
  - alg. k beneficia processos de I/O → prioridade = $\frac{1}{f}$, onde f é a fracção do último quantum k um processo usou
  - agrupar em classes de prioridade e usar Round-Robin dentro de cada classe → prioridades baixas são tratadas quando prioridades altas estão vazias
- **Multiple Queues**
  - divisão em classes
  - classe N → N quantums
  - processo gasta quantum todos → movido para classe seguinte
- **Shortest Process Next**
  - não se conhece o tempo de cada processo → estimativas → comportamentos anteriores
- **Guaranteed Scheduling**
  - N processos → cada um recebe 1/N dos ciclos da CPU → funciona com rácios
- **Lottery Scheduling**
  - distruição de bilhetes de lotaria

- decisão de escalonamento → sorteia-se um número
- processos prioritários recebem mais bilhetes → aumento da probabilidade
- processos k cooperam podem partilhar bilhetes (bloqueio)

➢ **Fair-Share Scheduling**
- tem-se em consideração o dono (utilizador) do processo
- cada utilizador recebe uma fracção da CPU
- exemplo (50%): User 1 tem processos: A,B,C,D; User 2 tem processo X → A,X,B,X,C,X,D,X,A,X,…

2.1.7.4. Escalonamento em Sistemas em Tempo-Real
- tempo → papel essencial
- dispositivos físicos geram estímulos → computador tem k reagir adequadamente num determinado intervalo de tempo
- hard real time → metas absolutas
- soft real time → é aceitável perder-se uma meta ocasionalmente
- dividir programa em vários processos → comportamento conhecido previamente → muito curtos → evento externo → escalonador gere processos de modo a k todas as metas sejam cumpridas
- eventos periódicos → mesmo periodo
- eventos aperiódicos → periodos diferentes
- um sistema em tempo-real é escalonável só se $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$
- alg. estáticos → decisões previamente tomadas
- alg. dinâmicos → decisões tomadas na hora do escalonamento

2.2. Tarefas

2.2.1. O Modelo de Tarefa
- Tal como os processos, as **tarefas** são também entidades activas. Um processo pode ser composto por várias tarefas que partilham o mesmo espaço de endereçamento. Processos diferentes possuem recursos diferentes, mas um conjunto de tarefas dentro do mesmo processo partilha os mesmos recursos.
- Cada tarefa tem registos, program counter, stack e estado próprios (running, ready, blocked, terminated)
- não existe protecção entre tarefas pk: (1) é impossível ; (2) é desnecessário, as tarefas cooperam entre si e não lutam entre si
- cada tarefa tem a sua própria pilha (stack)
- uma tarefa pode criar outra → com ou sem hierarquia → thread_create
- tarefa fecha-se a si própria → thread_exit
- uma tarefa pode esperar por outra → thread_wait
- não há interrupções de relógio para tarefas → escalonamento é feito através da chamada thread_yield → tarefas desistem voluntáriamente da CPU para permitir k outras trabalhem
- Algumas complicações que as tarefas introduzem:
- fork → processo filho deve ter as mesmas tarefas k processo pai? tarefas bloqueadas?
- partilha de dados → se uma tarefa fecha um ficheiro enquanto outra o está a ler? getão dos pedidos de alocação de memória?

2.2.2. Utilização de Tarefas

- **Razões principais da existência de tarefas**
  - Decomposição de uma aplicação em várias tarefas que se executam quase simultâneamente
  - Facilidade de criação e destruição das tarefas
  - Divisão em tarefas de computação (CPU-bound) e tarefas de I/O (I/O-bound)
- **Exemplos de utilização de tarefas:**
  - Processador de texto → podem existir tarefas para:
    - Ler input do teclado
    - Reformatar o documento
    - Salvar o documento automaticamente, etc.
  - Web Server → dois tipos de tarefas
    - "dispatcher" → sempre que chega um pedido de página, a tarefa "dispatcher" lança uma tarefa "worker"
    - "worker" → procura a página pedida na cache de páginas, caso não a encontre, terá que ir buscá-la ao disco
- **Formas de construir um servidor:**
  - Threads → Parallelism, blocking system calls
  - Single-threaded process → No parallelism, blocking system calls
  - Finite-state machine → Parallelism, nonblocking system calls, interrupts

2.2.3. Implementar Tarefas no Espaço do Utilizador

- gerir as tarefas em MU
- núcleo não sabe da existência das tarefas
- geridas pelos procedimentos thread_create,exit,wait,yield,etc.
- cada processo tem a sua própria tabela de tarefas (thread table)
- Vantagens:
  - podem-se usar tarefas num SO k não suporta tarefas
  - comutação de tarefas muito rápida
  - cada processo tem o seu próprio algoritmo de escalonamento
- Desvantagens:
  - chamadas de bloqueio podem interferir e bloquear outras tarefas
  - quando ocorre uma falta de página (page fault), o processo é bloqueado apesar de outras tarefas ainda poderem funcionar
  - como as interrupções não têm influência no escalonamento de tarefas, uma tarefa pode trabalhar indefinidamente se não usar thread_yield
  - programadores querem tarefas que fiquem bloqueadas com frequência

2.2.4. Implementar Tarefas no Núcleo

- núcleo gere tarefas
- tabela de tarefas única situada no núcleo
- criação/destruição de tarefas → chamada ao núcleo → núcleo actualiza tabela de tarefas
- chamadas de bloqueio de tarefas → chamadas ao sistema
- qdo 1 tarefa bloqueia, núcleo escolhe outra (do mesmo ou de outro processo)
- alto custo de criação/destruição de tarefas → reciclagem de tarefas → desactivada → reactivada

2.2.5. Implementações Híbridas

- objectivo → reunir vantagens das tarefas em espaço do utilizador e das tarefas de núcleo
- mistura desses dois métodos

2.2.6. Activações do Escalonador

- implementação híbrida
- imitar funcionalidades das tarefas de núcleo
- qdo 1 tarefa bloqueia → deixar funcionar 1 tarefa do mesmo processo
- eficiência → evitar transições desnecessárias entre MU e MN
- uso de processadores virtuais
- bloqueio → núcleo notifica processo → upcall → escalonamento de tarefas (dentro do processo)

2.2.7. Tarefas Pop-Up

- tarefa pop-up → tarefa cuja criação é causada pela chegada de uma mensagem
- criação instantânea, devido à tarefa ser nova e não ter histórico → nenhuma informação precisa de ser carregada

2.2.8. Tornar Código de Tarefas-Únicas em Multitarefas

- maioria dos programas escritos para tarefas-únicas → torná-los utilizáveis para multitarefas
- **1º problema → Variáveis Globais:**
  - possível solução → proibir variáveis globais → quase impossível
  - outra solução → atribuir a cada tarefa a sua própria variável global → acesso complicado
  - introduzir novos procedimentos de biblioteca (library procedures) para gerir essas variáveis globais privadas
  - tarefa usa procedimento → acessa apenas os seus próprios dados
- **2º problema → Procedimentos de Biblioteca podem ser usados só por uma tarefa de cada vez:**
  - solução 1 → procedimento em uso deixa procedimento inválido k leva a crash
  - solução 2 → atribuir a cada procedimento um bit adicional k indica se o procedimento está a ser usado ou não → elimina grande parte do paralelismo
- **3º problema → Sinais** (núcleo não sabe para k tarefa os deve enviar)
- **4º problema → Pilhas** (Stacks) → núcleo não sabe quando é k as tarefas precisam de mais espaço
- problemas corrigíveis, mas soluções difíceis e soluções têm k suportam tarefas-únicas também

2.2.9. Escalonamento de Tarefas

- 2 níveis de paralelismo → processos e tarefas
- **Tarefas no Espaço do Utilizador**
  - escalonamento de processos e de tarefas separados
  - alg. de escalonamento de tarefas comuns → round-robin, priority
  - ex: A1,A2,A3,B1,B2,B3,A1,A2,A3,…
- **Tarefas no Núcleo**
  - núcleo escolhe tarefa (não tem k considerar o processo ao qual pertence)
  - ex: A1,B1,A2,B2,A3,B3,A1,B1,…
- é preferível deixar trabalhar tarefas do mesmo processo para evitar comutações entre processos

2.3. Comunicação entre Processos (Interprocess Comunnication – IPC)

- **Modos de comunicação entre processos:**
  - Passagem de informação de um processo para outro
  - Processos não devem atrapalhar uns aos outros em regiões críticas
  - Ordem quando existe dependência entre processos
- tudo isto tb é válido para tarefas

2.3.1. Condições de Disputa (Race Conditions)

- Uma **condição de disputa** é uma situação que ocorre quando dois ou mais processos lêem e escrevem (n)os mesmos dados e o resultado final depende do proceso que acedeu aos dados por último.
- Exemplo: impressora, spooler directory, printer daemon

2.3.2. Regiões Críticas

- **região crítica** → secção do programa onde são efectuados acessos (para leitura e escrita) a recursos partilhados por dois ou mais processos
- é necessário assegurar que dois ou mais processos não se encontrem simultaneamente na região crítica → exclusão mútua
- assegura-se a exclusão mútua recorrendo aos mecanismos de sincronização fornecidos pelo SO
- Estas afirmações são válidas também para as tarefas (é ainda mais crítico, pois todas as tarefas dentro do mesmo processo partilham os mesmos recursos)
- **Regras para programação concorrente (evitar condições de disputa):**
  1. Dois ou mais processos não podem estar simultaneamente dentro de uma região crítica
  2. Não se podem fazer assunções em relação à velocidade e ao número de CPUs
  3. Um processo fora da região crítica não deve causar bloqueio a outro processo
  4. Um processo não pode esperar infinitamente para entrar na região crítica

2.3.3. Exclusão Mútua com Espera Activa (Busy Waiting)

1) Disabling Interrupts

- assim k 1 processo entra na sua região crítica, desliga as interrupções de modo a não ser incomodado por outro processo. qdo sair, volta a ligar as interrupções
- não se deve dar essa liberdade aos processos → risco: um processo pode não reactivar as interrupções
- desligar interrupções é permitido pelo núcleo e pode ser muito útil, mas não é aconselhável como mecanismo de exclusão mútua

2) Lock Variables

- considerar uma variável de trancar partilhada
- 0 → pode entrar na região crítica ; 1 → não pode entrar
- contém mesmo erro fatal de "spooler directory" → pode levar a condições de disputa

3) Strict Alternation

- variável k indica qual dos processos pode entrar na região crítica
- qdo um processo sai, muda o valor e outro (k corresponde áquele valor) pode entrar
- processos k estão fora testam variável em espera activa (busy waiting)
- esperas activas devem ser evitadas pk reduzem a eficiência do processador
- método não aconselhável qdo temos processos lentos e rápidos → viola Regra 3

4) Peterson's Solution

- alg. consiste em 2 procedimentos: enter_region, leave_region ; var.glob.: turn, interested[N]

- enter_region[0] → turn=0 → interested[0]=TRUE → leave_region[0] → interested[0]=FALSE
- processo 1 em espera activa até processo 0 sair da região crítica

5) **The TSL Instruction (Test and Set Lock)**
   - uma instrução do processador carrega num registo o valor lido de uma posição e de seguida escreve nessa posição um valor diferente de zero (e.g. 1) → compara valor lido com 0 → se for 0, entra; se não for 0, espera activa
   - enter_region → TSL → busy waiting ; leave_region → lock=0

2.3.4. Sleep and Wakeup (Adormecer e Acordar)

- Duas chamadas ao sistema que funcionam do seguinte modo:
- Sleep() → causa bloqueio ao processo que a invoca
- Wakeup(PID) → desbloqueia o processo identificado por PID
- A utilização destas duas chamadas evita esperas activas, e em conjunto com outros mecanismos (e.g. TSL) consegue-se garantir a exclusão mútua
- Problema → lost Wakeup signal → processo manda "acordar" o outro sem este ter "adormecido" ainda → possível solução → wakeup bit → esta solução não funciona sempre → inválida

2.3.5. Semáforos (Semaphores)

- Um **semáforo** consiste basicamente num número inteiro não negativo
- Foram originalmente sugeridas duas operações atómicas (indivisíveis) sob o ponto de vista do SO:
- UP(Sem) – Incrementa em uma unidade o valor do semáforo Sem
- DOWN(Sem) – Tenta decrementar em uma unidade o semáforo Sem. Caso o semáforo esteja a "0", o processo que invoca DOWN bloqueia até que o valor do semáforo permita o decremento e a operação seja finalizada
- acções atómicas (atomic actions)
- implementação de forma indivisível → considerar UP & DOWN como chamadas ao sistema → desligar interrupções durante as suas execuções
- no caso de múltiplas CPUs → proteger semáforo com Instrução TSL → permite apenas 1 CPU
- semáforos resolvem problemas de exclusão mútua e problemas de sincronização

2.3.6. Mutexes

- qdo a funcionalidade de contagem dos semáforos não é necessária, usa-se uma versão simplificada de semáforo → **mutex**
- mutexes apenas servem para exclusão mútua
- mutex → variável k pode ter 2 estados: destrancado, trancado (unlocked, locked)
- representação em integer
- 2 procedimentos: mutex_lock, mutex_unlock
- mutex_lock → LOCKED → thread_yield → sem espera activa (no busy waiting)
- **Como é que processos, que têm obviamente espaços de endereçamento diferentes, podem partilhar um semáforo?**
  - semáforos guardados no núcleo → acesso através de chamadas ao sistema
  - a maioria dos SOs oferece maneira de os processos partilharem uma parte do seu espaço de endereçamento com outros processos
  - usar um ficheiro partilhado

2.3.7. Monitores

- **monitor** → primitiva de sincronização de alto nível; colecção de procedimentos, variáveis e estruturas de dados k são agrupadas num tipo especial de módulo ou pacote
- propriedade importante → dentro de um monitor, apenas um processo pode estar activo em cada instante
- tornar regiões críticas em procedimentos de monitor
- condition variables → wait ou signal (parecido com sleep, wakeup)
- diferença entre wait/signal e sleep/wakeup → w/s não são interrompidos por escalonamento
- nem semáforos nem monitores funcionam em troca de informação entre máquinas pk estas não partilham memória umas com as outras

2.3.8. Troca de Mensagens (Message Passing)

- procedimentos de biblioteca: send, receive
- **Design de Mensagens**
  - rede → mensagens perdidas → solução: receptor envia mensagem de confirmação
  - mensagem recebida, mas confirmação perdida → evitar reenvio → atribuir número (código) a cada mensagem → de modo a poder reconhecer e ignorar uma transmissão jà recebida
  - autenticação tb é um assunto muito importante

2.3.9. Barreiras

- mecanismo de sincronização para grupos de processos
- algumas aplicações são divididas em fases
- regra → só se segue para a fase seguinte quando todos os processos terminaram a fase actual
- barreira → limite entre fases (fronteira)
- ex: cálculo de matrizes de grande ordem

2.4. Problemas Clássicos de Comunicação entre Processos (Classical IPC Problems)

1) Problema do Jantar dos Filósofos (The Dining Philosophers Problem)
2) Problema do Leitor e Escritor (The Readers and Writers Problem)
3) Problema do Barbeiro Adormecido (The Sleeping Barber Problem)

2.5. Impasses (Deadlocks)

➔ Um conjunto de processos está num **impasse** se cada um dos processos está bloqueado à espera de um sinal dependente de outro processo nesse conjunto.

2.5.1. Recursos

- **recurso** → dispositivo de HW, pedaço de informação, etc
- cada recurso só pode ser usado por um processo num determinado instante

1) **Recursos Preemptíveis e Não-preemptíveis (Preemptable and Nonpreemptable Resources)**
   - recurso preemptível: pode ser tirado do processo que o possui sem prejudicar o resultado da computação (memória)
   - recurso não-preemptível: se o recurso for tirado do processo antes de este o libertar, o resultado da computação será incorreto (impressora, gravador de CD, etc)
   - **Sequência de eventos necessários para usar um recurso:**
     1. Requisitar o recurso → disponível → continuar em 2. ; indisponível → sleep
     2. Usar o recurso
     3. Libertar o recurso

2) Aquisição de Recursos

- associar um semáforo a cada recurso → DOWN → usa recurso → UP
- ordem de requisição dos recursos deve ser sempre a mesma senão potencial impasse

2.5.2. Introdução aos Impasses

1) Condições para um impasse (as 4 têm k estar presentes → impasse)

1. Exclusão Mútua: cada recurso está atribuído a um único processo em um dado intervalo de tempo
2. Espera com Retenção: um processo pode solicitar novos recursos quando ainda está a reter outros recursos
3. Não-preempção: um recurso concedido a um processo não lhe pode ser retirado á força; somente pode ser libertado pelo próprio processo
4. Espera Circular: existe uma cadeia circular de dependência entre os processos

2) Modelação de Impasses

- quadrado → nó de recurso
- círculo → nó de processo
- seta R-P → recurso conedido ao processo
- seta P-R → processo bloqueado à espera do recurso
- ciclo → impasse

3) Estratégias para lidar com impasses

1. Ignorar totalmente a existência dos impasses
2. Detectar o impasse e recuperar o sistema após a ocorrência deste impasse
3. Evitar a ocorrência dos impasses em tempo de execução, ao alocar os recursos aos processos
4. Impedir ocorrência de impasses, definindo regras que impedem a existência de uma das quatro condições necessárias.

2.5.3. Algoritmo da Avestruz (The Ostrich Algorithm)

- ignorar o problema → fingir que não existe
- aceitável se a probabilidade de impasse for muito baixa; custo da solução for elevado
- compromisso entre correcção e conveniência/eficiência análogo a muitos em engenharia.

2.5.4. Detecção de Impasse e Recuperação

➢ sistema não tenta prevenir impasses → deixa-os acontecer para lhes tratar da saúde :D

A. Detecção

- Algoritmo e detecção de Impasses (ciclos) → analisa cada nó → segue setas → caso encontre um nó pelo qual jà passou → impasse

B. Recuperação

a) Recuperação através da Preempção → remove-se o recurso manualmente
b) Recuperação através de Rollback → checkpoint → recomeçar a partir de um checkpoint
c) Recuperação através da Terminação (Killing) de Processos → "matar" processos até sair do impasse (até destruir o ciclo) → escolha cuidadosa do(s) processo(s) a "matar"

2.5.5. Prevenção de Impasses (Deadlock Prevention)

➢ ideia → garantir k uma das 4 condições de impasse nunca ocorre → impasses impossíveis

1) Atacar a Exclusão mútua

- recorrer a processos auxiliares → printer spooler
- nem todos os recursos se prestam a spooling

- a contenção no acesso a outros recursos (ex. disco, no caso do printer spooler) pode também dar origem a impasses
- ideias importantes → atribuição um recurso apenas quando necessário ; minimizar o número de processos que partilham um recurso

2) **Atacar a Espera com Retenção**
   - pedir todos os recursos necessários antes de iniciar tarefa → problemas: (1) processo não sabe, previamente, quais os recursos k vai precisar; (2) processo pode bloquear recursos desnecessáriamente durante muito tempo
   - processo deverá libertar todos os recursos que possui, quando o pedido de um recurso conduziria a um
   - bloqueio; quando desbloqueado, o processo terá que readquirir todos os recursos de novo
3) **Atacar a Não-preempção**
   - retirar recursos a um processo (sem a sua cooperação) é inviável na maioria dos casos
4) **Atacar a Espera Circular**
   - a um processo só pode ser atribuído um recurso de cada vez; tem k libertar recurso para poder requisitar outro → não aconselhável → exemplo: imprimir a partir de uma tape
   - numeração global de todos os tipos de recursos → requisição feita por ordem → elimina impasses → MAS ordenação pode ser complicada pk nunca satisfaz todos

2.5.6. Outros Asuntos

1) **Two-Phase Locking**
   - 1ª fase → processo "tranca" (locks) todos os recursos k necessita
   - se encontrar algum recurso jà utilizado por outro processo → liberta todos os recursos → reinicia 1ª fase → sem impasses
   - 2ª fase → trabalho normal do processo
   - método não aplicável em geral → sistemas de tempo-real → há processos k não podem ser reiniciados de qualquer maneira
2) **Nonresource Deadlock**
   - 2 processos podem bloquear-se mutuamente (mesmo sem recursos) → impasse
   - ex: ordem de DOWNs em semáforos
3) **Starvation (Privação)**
   - processos k nunca recebem atenção → não é um problema de impasse MAS sim de privação
   - ex: Shortest Job First → se entram sempre trabalhos curtos → os longos nunca são atendidos
   - evitar privação → First-Come First-Served

Tema 3 – GESTÃO DE MEMÓRIA

- limitações da memória → hierarquia de memória → Registos, Cache, Memória principal (RAM), Discos (memória secundária), Tapes
- gestor de memória → parte do SO k gere a hierarquia de memória
- **É da responsabilidade do sistema operativo gerir a memória disponível no sistema:**
  - Representação do estado da memória
  - Atribuição de memória aos processos
  - Libertação da memória
  - Conjugação entre a memória principal e secundária

3.1. Gestão Básica de Memória

1) Monoprogramação sem Swapping nem Paging

- Programas/processos executados um de cada vez
- **Formas simples de organizar a memória com um SO e um processo (de baixo para cima):**
  - SO em RAM, Pr. em Mem. → mainframes e minicomputadores
  - Pr. em Mem., SO em ROM → Palmtop Computers, sistemas embutidos
  - SO em RAM, Pr. em Mem., Drivers dos disp. em ROM → 1ºs PCs (MS-DOS) → ROM (BIOS)
- Mediante um comando, o SO carrega o programa em memória principal, e este corre até terminar

2) Multiprogramação com Partições Fixas

- dividir memória em N partições fixas (de tamanhos diferentes)
- cada programa é carregado numa partição com uma dimensão adequada
- parte da partição não utilizada é perdida
- múltiplas filas de espera → desvantagem: pode haver uma partição vazia → desperdício
- fila de espera única → partição livre → carrega-se 1º trabalho da fila k lá cabe
- manter interactividade → manter sempre pelo menos uma partição pequena → trabalhos pequenos nunca esperam muito

3) Modelar a Multiprogramação

- processos passam uma fracção P do tempo da sua existência bloqueados em operações de I/O
- se N processos estiverem a correr, a probabilidade da CPU estar desocupada será P^N
- taxa de ocupação do CPU será então dada por $1 - P^N$ → grau de multiprogramação

4) Análise do Desempenho de Sistemas Multiprogramados

- modelo anterior pode ser usado para analisar sistemas batch
- exemplo: 4 jobs ; 1º job não perde muito qdo 2º é adicionado, …

5) Realocação e Protecção

- A multi-programação trouxe dois problemas a resolver:
  - realocação → garantir que os endereços referenciados por um programa sejam os correctos independentemente da posição de memória a partir da qual é carregado
  - protecção → impedir que um programa aceda aos endereços de um outro programa
- Soluções:
  - Registo Base → a todos os endereços referenciados pelo programa soma-se o endereço base da partição onde este é carregado. Os programas são escritos como se o primeiro endereço fosse 0.
  - Registo Limite → verifica-se se os endereços referenciados pelo programa se encontram dentro da partição que lhe é atribuída

3.2. Swapping

- em sistemas batch, a gestão básica de memória é suficiente
- em sistemas de timesharing e GUI, há demasiados processos → não cabe tudo na memória → tem k haver trocas (swapping) conforme os programas vão sendo utilizados
- equema que envolve a memória principal e a memória secundária (disco)
- **swapping** → consiste em transferir processos da memória principal para o disco e vice-versa
- transferir para o disco um processo bloqueado, e trazer para a RAM um processo pronto
- estratégia utilizada em conjunto com partições de dimensões variáveis
- conduz à proliferação de buracos (fragmentação) → compactação da memória → demora demasiado tempo
- processos com tamanho fixo → alocação de memória simples

- processos k crescem → alocação dinâmica → buraco adjacente : fácil → caso não haja um buraco adjacente, troca-se (swap) outro(s) processo(s) OU o processo é terminado (killed)
- alocar um pouco de memória extra para permitir expansão do processo
- se um processo tem 2 segmentos de expansão (data segment & stack segment), coloca-se data segment em baixo a expandir-se para cima e stack segment em cima a expandir-se para baixo

1) **Gestão de Memória com Bitmaps**
   - estrutura-se a memória em unidades de alocação (blocos) de dimensão fixa
   - representa-se o estado da memória através de bitmaps
   - 1 representa bloco ocupado
   - 0 representa bloco livre
   - tamanho dos blocos ↘ tamanho do bitmap ↗
   - desvantagem → procurar buracos é uma tarefa lenta
2) **Gestão de Memória com Listas Ligadas**
   - listas ligadas → indicam o bloco inicial e o número total de blocos de um processo (P) ou de um buraco (H, de hole)
   - Vantagem: fácil actualização da lista → ex: swap out → substituir P por H → fundir 2 Hs
   - **Algoritmos**:
     - First Fit: procura o primeiro buraco da lista com dimensão suficiente
     - Next Fit: variante do anterior, procura a partir do ponto em que foi encontrado um buraco anteriormente
     - Best Fit: procura o buraco com a dimensão que melhor se ajusta
     - Worst Fit: procura o maior buraco disponível, na esperança de que o espaço que sobra possa ainda ser utilizado por outros programas a carregar no futuro
   - aumento do desempenho dos algoritmos se se usar listas separadas: uma para Ps, outra para Hs
   - ordenar listas → do + pequeno ao maior bloco → ↗ desempenho de Best Fit
   - outro algoritmo → Quick Fit → mantém várias listas de buracos, agrupados de acordo com as suas dimensões → desvantagem: encontrar vizinhos (para possível fusão de buracos) é difícil

3.3. Memória Virtual

- programas demasiado grandes → não cabem na memória
- dividir programas em overlays → trabalho muito chato efectuado pelo programador
- dar esse trabalho ao computador → memória virtual
- não se usa o programa inteiro (visto k não cabe), usa-se apenas as partes necessárias

1) **Paginação (Paging)**
   - **Espaço de endereçamento virtual**: Espaço de endereçamento que engloba a memória primária e secundária, tirando partido da sua dimensão pode ser muito superior à da memória RAM
   - O endereçamento virtual difere do swapping visto anteriormente; cada processo pode ter partes carregadas em memória principal, partes em memória secundária, ou em ambos os lados
   - **Endereços reais**: Endereços físicos que correspondem ao acesso aos dispositivos de uma forma directa
   - **Endereços virtuais**: Endereços utilizados internamente pelo sistema, e que não estão ligados aos dispositivos físicos de uma forma directa; um endereço virtual pode ser muito diferente de um endereço real
   - endereços virtuais são convertidos em endereços reais através de estruturas e algoritmos nos quais intervém o SO e também uma unidade de hardware designada MMU (Memory Management Unit)

- A função da **MMU** é converter endereços virtuais em endereços físicos; notifica o sistema se for feito um acesso a um endereço virtual que não corresponde fisicamente à memória principal (page fault → trap to kernel)
- **Paginação**: Método mais comum de gestão da memória virtual
- **Páginas (pages)**: O espaço de endereçamento virtual é divido em blocos de dimensão fixa designadas por páginas. A dimensão de cada página é uma potência de 2
- **Molduras de página (page frames)**: A memória principal é dividida em blocos com a capacidade de alojarem uma página
- **Faltas de página (Page faults)**
  - qdo é acedida uma página que não se encontra na memória principal, ocorre uma page fault
  - uma page fault é uma excepção que causa bloqueio ao processo em questão
  - carregamento da página em falta, da memória secundária para a memória principal
  - efectuam-se as alterações necessárias na page table, de modo a esta continuar consistente
  - pode ser necessário transferir uma outra página para a memória secundária, de modo a libertar-se uma page frame → nesse caso corre-se o algoritmo de substituição de páginas

2) Tabelas de Páginas (Page Tables)

- 16 bits → 4 bits → número da página ; 12 bits → offset
- MMU → troca 4 bits iniciais (endereço virtual) por outros bits (número da moldura de página) → offset é apenas copiado
- tabela de páginas é como uma função → f(nº da página) = nº da moldura de página
- Cada processo tem associado ao seu espaço de endereçamento uma tabela de páginas
- A tabela de páginas de cada processo tem que estar carregada em memória
- Bit de presença: Indica se a página se encontra carregada na memória principal ou não
- **Assuntos importantes**
  1. a tabela de páginas pode ser extremamente grande
  2. o mapeamento tem de ocorrer com muita rapidez
- problema: guardar tabelas de páginas enormes na memória → solução: TP Multi-Nível
- **Tabelas de páginas multi-nível**
  - Guardam-se na memória uma tabela principal (directoria) e as tabelas dos restantes níveis, que contém os descritores das páginas que estão a ser utilizadas pelo processo
  - Estas tabelas têm uma dimensão muito mais pequena do que se fosse utilizado um esquema com um só nível
- **Estrutura de uma entrada de tabela de páginas**
  - Nº da página: indica nº da moldura de página (principal razão da existência de TPs)
  - Bit de presença: indica se a página se encontra carregada na memória principal ou não
  - Protecção: Bits de protecção da página (ex: read-only, rwx)
  - Modificação: bit k indica se a página foi modificada (dirty bit)
  - Referência: bit k indica se a página foi usada (grau de utilização)
  - Caching: bit k activa/desactiva caching
- **TLBs – Translation Lookaside Buffers**
  - conversão end.virt. para end.fís. → consultar page table → na memória → acessos extra
  - para minorar desperdício de tempo feito por estes acessos → MMUs contém geralmente uma espécie de cache para páginas, designada TLB
  - TLB mantém um conjunto de descritores das páginas acedidas mais recentemente
  - cada processo tende a utilizar mais exaustivamente um pequeno conjunto de páginas
  - quando uma página é descartada da memória principal, é também libertada a entrada correspondente na TLB

- faltas de TLB geridas pelo HW da MMU ou por SW (pelo SO)

- **Tabelas de páginas invertidas**
  - Uma entrada para cada página de memória
  - **Cada entrada contém:**
    - → o endereço virtual da página
    - → o endereço físico
    - → o processo que é dono da página
  - Vantagem: diminui a memória necessária para armazenar a tabela
  - Desvantagem: pode aumentar o tempo necessário para procurar na tabela

3.4. Algoritmos de Substituição de Páginas (Page Replacement Algorithms)

1) O algoritmo ideal

- sempre que for necessária uma substituição de páginas, a que deveria sair será aquela que só será utilizada daqui a mais tempo → impossível → SOs não conseguem prever o futuro
- aproximação → tentar descartar as páginas que são pouco utilizadas, ou que já não são utilizadas há muito tempo, na esperança de que não venham a ser utilizadas brevemente

2) Algoritmo "Not Recently Used" (NRU)

- quando ocorre uma page fault, este algoritmo classifica as páginas carregadas em memória
- Utilização de dois bits:
  - R (Referenced): indica k a página foi acedida desde a última interrupção do relógio
  - M (Modified): indica k a página foi modificada desde que foi carregada na memória principal
- Estabelecem-se 4 classes diferentes, de acordo com R e M
  - Classe 0: R=0 e M=0
  - Classe 1: R=0 e M=1
  - Classe 2: R=1 e M=0
  - Classe 3: R=1 e M=1
- A página a substituir será uma página aleatória pertencente à classe mais baixa encontrada

3) Algoritmo "First-In, First-Out" (FIFO)

- A página a substituir é a que se encontra carregada há mais tempo na memória principal
- fácil implementação → lista com índices de páginas → a + antiga no topo e a + recente no fim
- à medida que as páginas vão sendo carregadas na memória, os seus índices vão também sendo acrescentados ao fim da lista mantida pelo algoritmo
- Problema: a página que foi carregada há mais tempo pode estar a ser utilizada

4) Algoritmo da Segunda Chance (Second Chance)

- Algoritmo baseado no FIFO, mas que utiliza o bit de referência (R)
- Antes de uma página ser descartada, analisa-se o bit R e, caso este se encontre a "1", então é posto a "0", mas a página não é descartada, analisando-se a próxima.
- A página a descartar será a primeira que for encontrada com R=0

5) Algoritmo do Relógio

- Semelhante ao algoritmo da segunda chance, mas a lista de páginas é circular
- Deste modo ganha-se eficiência pois deixa de ser necessário retirar estruturas do topo da lista para as inserir no fim

6) Algoritmo "Least Recently Used" (LRU)

- A página a substituir será a que não é acedida hà mais tempo
- Para tal, guarda-se para cada página uma marca temporal com o tempo do último acesso
- Teoricamente este algoritmo é o que efectua a melhor escolha na página a substituir
- Bom desempenho a um custo elevado

- Na prática, este algoritmo obriga à existência de um temporizador extra (pois as interrupções do relógio são demasiado "lentas")
- Para além disso, exige bastante espaço para guardar as marcas temporais (e.g. 64 bits) sempre que uma página é acedida
- ex: N molduras de páginas → matriz de ordem N

7) Algoritmo "Not Frequently Used" (NFU)

- simulação do LRU em SW
- Associado um contador a cada página carregada em memória, inicializado a zero quando a página é carregada
- Sempre que ocorre uma interrupção do relógio, e para cada página, soma-se o valor do bit R ao contador
- Problema: uma página muito acedida no início, mas que depois deixe de ser acedida fica com um valor elevado no contador, pelo que poderá persistir na memória

8) Algoritmo "Aging"

- Variante do algoritmo NFU, que tenta resolver o problema descrito anteriormente
- Em vez de somar o bit R ao valor do contador, desloca-se o seu conteúdo para a direita com entrada série do bit R
- Deste modo consegue-se que uma página muito acedida no passado, mas que já não está a ser utilizada, fique com o valor do contador a zero após algumas interrupções do relógio
- Algoritmo com melhor relação custo/desempenho

9) Algoritmo "Working Set"

- **Paginação a pedido (Demand paging)**: As páginas de um processo vão sendo carregadas à medida que ocorrem page faults → esta abordagem faz com que ocorram page faults inicialmente, até ser estabelecido o Working Set do processo
- Quando um grupo de processos começa a gerar page faults a um ritmo muito elevado diz-se que se entrou numa fase de thrashing → esta situação deve ser evitada a todo o custo
- O **Working Set** é o conjunto de páginas utilizadas nos K últimos acessos de memória
- Se todo o Working Set de um processo está carregado em memória → não ocorrem page faults
- Tirando partido deste facto, existem algoritmos de substituição de páginas que funcionam tendo em conta o working set de um processo
- A ideia será substituir páginas que não se encontrem dentro do working set de um processo
- qdo ocorre uma comutação de processos → prepaging → antes do processo correr, o SO carrega para a memória o Working Set
- tempo virtual corrente (current virtual time) → tempo k 1 processo usou desde k foi iniciado
- Algoritmo → encontrar página k não está no WS e retirá-la

10) Algoritmo WSClock

- mistura entre Working Set e Relógio

3.5. Questões de Concepção de Sistemas de Paginação

1) Políticas de alocação local vs global

- exemplo: 3 processos → A, B, C → falta de página em A → 2 hipóteses:
  - → trocar página do próprio processo (A) → alg. de substituição de páginas "local"
  - → trocar página de qquer processo (A,B,C) → alg. de substituição de páginas "global"
- alg. globais funcionam melhor qdo o Working Set pode variar
- alocar um nº de páginas "proporcionais" a cada processo → no entanto, deve dar-se um mínimo de páginas a cada processo → processos pequenos tb devem ter hipótese de funcionar
- gestão da alocação → alg. PFF → indica qdo ↗ ou ↘ alocação de páginas de um processo

- atribuição de páginas ↗ faltas de página ↘ → manter equilíbrio entre estes 2 conceitos
- por vezes, escolha alg. local/global é independente do próprio alg. escolhido (FIFO, LRU, etc)
- alg. do Working Set (ou WSClock) só faz sentido se for local

2) Controlo de Carga (Load Control)

- Problema: qdo um processo precisa de + mem. mas nenhum outro precisa de – mem. → tendência para faltas de página → thrashing
- Solução: parar um (ou +) processo(s) → trocá-lo (swap) para o disco → libertar mem.
- escolher processo a parar de modo a k o processador não fique parado → I/O-bound
- mesmo com o Paging, o Swapping ainda é necessário

3) Tamanho das Páginas (Page Size)

- Argumentos a favor de páginas pequenas:
  - programa pequeno → desperdício de espaço na página → fragmentação interna
  - páginas grandes → grandes partes do programa em memória, mas não utilizadas
- Argumentos a favor de páginas grandes:
  - páginas pequenas → muitas páginas → tabela de páginas enorme
- overhead: processamento ou armazenamento em excesso
- s =tamanho médio dos processos em bytes
- p = tamanho das páginas
- e = entrada na tabela de páginas
- $overhead = \frac{se}{p} + \frac{p}{2}$ → derivada(p): $-\frac{se}{p^2} + \frac{1}{2}$ → iguala-se a 0 → valor óptimo: $p = \sqrt{2se}$
- ex: s=1MB, e=8bytes → p=4KB

4) Instruções Separadas e Espaços de Dados (Separate Instruction and Data Spaces)

- maioria dos computadores tem espaço de endereçamento suficiente para programa e dados
- problema: esp.ender. pequeno demais
- solução: esp.ender. separados para instruções (texto do programa) e dados
- I-space & D-space
- paginação indepenente → cada uma tem a sua tabela de páginas
- este método não traz complicações e duplica o esp.ender. disponível

5) Páginas Partilhadas (Shared Pages)

- partilhar programas → + eficiente partilhar as páginas
- I/D-spaces facilita partilha → processos partilham a mesma TP para o seu I-space → cada um tem TP diferentes para o seu D-space
- ex: A,B partilham progr. → A termina → MAS programa não deve ser fechado pk B ainda o usa
- dados podem ser partilhados em READ-ONLY

6) Política de Limpeza (Cleaning Policy)

- daemon de paginação: processo de fundo k verifica o estado da memória periodicamente e escreve as páginas sujas (dirty) no disco
- gravar páginas modificadas na última hora pode ser pouco eficiente → paging daemon acelera
- paging daemon tenta manter um certo número de molduras de página livres
- usa lista circular com apontador (tipo Relógio)

7) Interface da Memória Virtual

- programadores podem ter controlo sobre a memória
- pode permitir k 2 processos partilhem a msm página → ↗ do desempenho
- memoria partilhada distribuída → partilha de páginas por rede (várias máquinas)

8) Mapeamento de Ficheiros de Memória (Memory-Mapped Files)

- novas chamadas ao sistema → map & unmap

- mapear ficheiros na memória
- Vantagem: elimina necessidade de I/O → programação + fácil
- Desvantagens:
  - sistema desconhece o comprimento do ficheiro de output
  - ficheiro mapeado por processo A e aberto para leitura pelo processo B → modificações feitas por A não são vistas por B enquanto o ficheiro não é re-escrito no disco
- ficheiro pode ser maior k esp.ender.virtual → possível solução: mapear apenas parte do ficheiro (não-satisfatório)

Tema 4 – SISTEMAS DE FICHEIROS

- Problema: como armazenar grandes quantidades de informação e de forma permanente, num suporte que o permita: disquete, disco, CD, etc. ?
- **Requisitos essenciais para o armazenamento de informação a longo-prazo:**
  - possibilidade de armazenar grandes quantidades de informação
  - a informação tem k ser permanente (sobreviver à terminação de processos)
  - possibilidade de acesso por vários processos simultâneamente
- Solução: sobrepor à organização física do "meio" (sectores, ...) uma organização em "peças" de informação lógica: ficheiros
- é da responsabilidade do SO criar esta organização lógica

4.1. Ficheiros

➔ **ficheiro**: mecanismo de abstracção k poupa o utilizador aos detalhes dos discos apresentando uma forma organizada, e de fácil compreensão, para armazenar informação

1) Nomeação de Ficheiros

- nome → forma de identificação do ficheiro; 1º aspecto visível de utilização de um ficheiro
- case sensitive → distinguir letras maiúsculas e minúsculas
  - UNIX: case sensitive
  - Windows : case insensitive (razões: compatibilidade com versões anteriores e MS/DOS)
- dimensão do nome:
  - UNIX → tamanho variável (limite de 255 caracteres)
  - MS/DOS → tamanho fixo → 8+3 (extensão)
- extensão → formal ou informalmente indica a natureza (ou conteúdo) do ficheiro
  - UNIX → apenas para indicar tipo de ficheiro ao utilizador ; não-atribuída pelo SO
  - Windows → extensões têm significado formal no SO

2) Estrutura dos Ficheiros

- SO vê ficheiros como mera sequência de bits → utilizador atribui significado → flexibilidade máxima → UNIX, Windows funcionam assim
- estruturação por records
- organização + lógica → por key-fields → árvore estruturada

3) Tipos de Ficheiros

- ficheiros comuns
  - → ficheiros de texto (character special files)
    - sequência de caracteres ASCII, incluindo alguns caracteres especiais: linhas de texto & carriage returns
    - podem ser visualizados e impressos tal como são

- relacionados com input/output

→ ficheiros binários (block special files)
- conteúdo "físico" não interpretável como um conjunto de caracteres ASCII
- tratáveis apenas por programas (ou os próprios ficheiros são programas)

- directorias (ver 4.2. Directorias)

4) Acesso aos Ficheiros
- acesso sequencial → localizar lendo o ficheiro, desde o início até à posição pretendida
- acesso aleatório → possibilidade de localizar uma posição ("seek"), com base numa chave ou outra indicação

5) Atributos dos Ficheiros
- atributos → características do ficheiro (além do nome e conteúdo) que o SO gere, para efeitos de controlo e administração
- **Alguns atributos significativos (de maior utilidade):**
  - Dono (Owner) → utilizador dono / criador do ficheiro
  - Protecção → permissões de acesso ao ficheiro
  - Datas → data de criação: outras datas de acesso / alteração
  - Dimensão → dimensão actual do ficheiro
  - Flags extra → bit de backup, bit de ficheiro oculto/temporário, etc

6) Oprerações sobre Ficheiros
- operações → chamadas ao sistema
- **Algumas operações significativas:**
  - Create → cria um ficheiro vazio
  - Delete → remove um ficheiro
  - Open → abre um ficheiro para determinada operação
  - Close → fecha um ficheiro depois de concluído um conjunto de operações
  - Read → lê o conteúdo de um ficheiro
  - Write → escreve (altera) o conteúdo de um ficheiro
  - Append → acrescenta ao conteúdo anterior
  - Seek → posicionamento para acesso directo
  - Get/Set attributes → obter/alterar atributos
  - Rename → altera o nome de um ficheiro

4.2. Directorias

➔ **directoria**: estrutura utilizada para organizar ficheiros; contém referências a outros fich./direct.

1) Sistemas de Um Nível de Directoria
- uma única directoria (mesmo com vários utilizadores) → root directory
- Vantagens: simplicidade de implementação, facilidade de localização dos ficheiros
- Desvantagens: conflitos de nomes, confusão devida ao grande nº de ficheiros

2) Sistemas de Dois Níveis de Directoria
- 2 níveis → root directory → user directory
- cada utilizador tem a sua própria directoria
- SO identifica o utilizador através de um Login
- forma básica → cada utilizador só pode ver e acessar conteúdos da sua directoria
- extensão → pode-se acessar ficheiros de outros utilizadores (ex: open("nancy/x"))

3) Sistemas de Directorias Hierárquicas
- queremos + organização → árvore de directorias → hierarquia
- pode-se criar um nº arbitrário de subdirectorias

4) Nome [Caminho] (Path Name)

- nome do ficheiro → absoluto/relativo
- nome absoluto (absolut path name): nome único na árvore de ficheiros; caminho desde a directoria principal (root) até à directoria onde o ficheiro se encontra
- nome relativo (relative path name): caminho a partir de uma directoria qualquer (corrente)
- directoria de trabalho [corrente] (working/current directory): directoria na qual se está a trabalhar; usada para aceder + facilmente a certos ficheiros; elimina a necessidade de escrever os longos nomes absolutos
- nome absoluto funciona sempre independentemente da directoria de trabalho
- cada processo usa a sua própria directoria de trabalho → sem interferir com outros processos
- entradas especiais → "." → directoria de trabalho & ".." → directoria-mãe (parent directory)

5) Operações sobre Directorias

- operações → chamadas ao sistema
- **Algumas operações significativas:**
  - Create → cria uma directoria vazia (contém apenas "." e "..")
  - Delete → remove uma directoria (somente se estiver vazia)
  - Opendir → abre uma directoria para determinada operação (listagem dos seus ficheiros)
  - Closedir → fecha uma directoria depois de concluído um conjunto de operações
  - Readdir → retorna entrada seguinte numa directoria aberta
  - Rename → altera o nome de uma directoria
  - Link → técnica k permite a aparição de um ficheiro em mais k uma directoria; cria uma entrada para um ficheiro na directoria
  - Unlink → remove a entrada (se for a única, ficheiro é removido tb)

4.3. Implementação de Sistemas de Ficheiros

1) Esboço de Sistemas de Ficheiros (File System Layout)

➢ **Organização de um disco:**

- Master Boot Record (MBR) → inicia o computador
- Tabela de partições → contém endereços das partições; inicia partição marcada como activa
- Partições → pode conter 1 ou +
  - bloco de boot → executa SO
  - super bloco → contém parâmetros chave do SF (nº mágico, tipo de SF, nº de blocos, etc)
  - gestão do espaço livre → em forma de bitmap ou lista de apontadores
  - I-nodes → contêm a informação sobre os ficheiros
  - directoria raíz
  - ficheiros e directorias

2) Implementar Ficheiros

➢ questão + importante → associar blocos ao ficheiro correspondente (vice-versa)

a) Alocação contígua (Contiguous Allocation)

- armazenamento de cada ficheiro num conjunto de bloco contíguos
- Vantagens:
  - implementação simples → para cada ficheiro, basta SO saber o bloco inicial nº de blocos
  - eficiente em termos de leitura → sempre bloco contíguos → leitura feita de 1 só vez
- Desvantagem: fragmentação resultante da remoção/criação de novos ficheiros, que só pode ser eliminada com compactação
- usada em CD-ROMs

b) Alocação por Listas Ligadas (Linked List Allocation)

- mantém-se uma lista ligada dos blocos ocupados por cada ficheiro; em cada bloco, para além dos dados, guarda-se também um apontador para o bloco seguinte do mesmo ficheiro
- Vantagens:
  - todos os blocos podem ser ocupados (a fragmentação não é problemática)
  - ao SO basta saber a localização do 1º bloco
- Desvantagens:
  - acesso sequencial → chegar a um bloco → passar pelos anteriores → acesso lento
  - tamanho real de cada bloco é diminuído pelo espaço ocupado pelo apontador (e jà não é uma potência de 2)

c) Alocação por Listas Ligadas usando uma Tabela na Memória (LLA using a Table in Memory)

- eliminar desvantagens da ALL → tirar apontador do bloco e colocá-lo numa tabela na memória → tabela de alocação de ficheiros → FAT (File Allocation Table)
- manter em memória uma tabela com uma representação da lista ligada de blocos; em cada posição da tabela indica-se o bloco seguinte do ficheiro
- Vantagens:
  - tal como no modelo anterior, a fragmentação não é problemática
  - cada bloco é utilizado integralmente para armazenamento de dados
  - facilita o acesso directo → para obter um bloco basta percorrer a FAT (mais rápido, pois percorre-se a memória e não o disco)
- Desvantagem: dimensão da FAT pode ser demasiado grande

d) I-nodes

- associar a cada ficheiro uma estrutura de dados contendo a localização em disco e os atributos do ficheiro
- i-node contém um número limitado de blocos do ficheiro → para ficheiros de maior dimensão são atribuídos ao i-node outros blocos que contém tabelas com nºs de bloco extra
- O i-node contém todas as características do ficheiro, excepto o nome que figura no (ou nos) directórios onde o i-node é incluído
- Vantagens:
  - fragmentação não é problemática
  - para aceder a um ficheiro basta ter o respectivo i-node em memória (não é necessário dispor de toda uma tabela de alocação)
  - facilita a partilha de ficheiros através de hard links

3) Implementar Directorias

- para ler (read) um ficheiro, temos k abrí-lo (open) 1º → SO usa nome (caminho) → localiza entrada da directoria → esta fornece a informação sobre os blocos do ficheiro → AC, ALL, I-node
- **directoria** → ficheiro especial → lista de nomes (em ASCII), a cada um dos quais se associam os respectivos atributos e localização em disco
- Atributos e Localização:
  - Windows → directoria contém atributos e localização dos ficheiros
  - UNIX → directoria contém nºs de i-nodes
- **Nomes:**
  - → Comprimento fixo (255 caracteres)
    - estabeler limite no comprimento dos nome
    - 255 caracteres reservados para cada nome
    - desperdício de espaço no caso de nomes curtos
  - → Nomes em linha (in-line)

- abandonar ideia k entradas de directoria têm todas o mesmo tamanho
- exemplo: FE1, FA1, FN1, FE2, FA2, FN2, …
- preenchimento para acabar linhas → filler characters
- mesma desvantagem k alocação contígua

→ Nomes numa pilha [numa fila] (in a heap)

- separar entrada e atributos do nome
- colocam-se os nomes no fim da directoria
- nomes todos seguidos com apontadores no seu início
- Vantagens:
  - remover uma entrada → nova entrada cabe la
  - acaba-se com os "filler characters"
- pesquisa de nomes linear → acelerada por "hashing" ou "caching"

4) Ficheiros Partilhados (Shared Files) [Linking]

- **Ligação real (Hard link)**
  - incluir o mesmo ficheiro em mais que uma directoria
  - replicar em cada directoria os atributos e localização no disco
  - implementação simples com i-nodes → basta replicar o nº de i-node
  - implementação complexa se os atributos estiverem contidos na directoria → as alterações têm que ser replicadas em cada ligação
  - i-node contém contador de ligações (link counter)
- **Ligação simbólica (Soft link [symbolic linking])**
  - incluir numa directoria o nome de outro ficheiro (do tipo LINK) que contém o caminho para o ficheiro original
  - através do caminho acede-se à entrada de directoria do ficheiro original e, por essa via, aos seus atributos e localização em disco

5) Gestão de Espaço de Disco (Disk Space Management)

a) Dimensão do bloco (Qual será a melhor dimensão para os blocos ?)

- Eficácia de leitura: relação entre o tempo de leitura e a informação efectivamente obtida do disco (blocos grandes)
  - aumenta com a dimensão do bloco
  - menor overhead de posicionamento em cada leitura
  - menor número de leituras necessárias para obter os dados
- Eficácia de ocupação: relação entre o espaço físico ocupado e o respectivo aproveitamento em termos de dados (blocos pequenos)
  - aumenta com a diminuição da dimensão do bloco
  - diminuição do desperdício devido ao ajuste da dimensão do ficheiro para um número fixo de blocos (blocos pequenos → pouco desperdício)
- compromisso → blocos de 2 KB

b) Controlo da lista de blocos livres → estrutura de dados de controlo dos blocos de disco não ocupados por ficheiros

- Lista ligada (com o número de cada um dos blocos livres)
  - Criação de um ficheiro: obter 1ºs blocos da lista (dimensão do fich.) e retirá-los da lista
  - Remoção de um ficheiro: acrescentar os respectivos blocos à lista
  - método simples, mas dispersão dos ficheiros por vários blocos não contíguos
  - dimensão da lista poderá ser bastante grande se o disco estiver pouco ocupado
- Bitmap (sequência com um número de bits igual ao número de blocos)
  - bit a "0" significa bloco livre ; bit a "1" significa bloco ocupado

- dimensão fixa e quase sempre menor que com lista (excepto disco quase cheio)
- facilita a procura de blocos contíguos/próximos (procurar 0's contíguos no bitmap)

- apontadores mantidos na memória para gestão rápida de ficheiros temporários

c) Quotas de Disco (Disk Quotas)

- em SOs multi-utilizador, administrador atribui a cada utilizador uma porção de blocos
- tabela "Open File" → utilizador abre ficheiro → tabela contém atributos e endereços de disco → aumento no ficheiro → tabela aumenta a quota do dono
- tabela de quota → contém dados gerais sobre a quota de um determinado utilizador
  - soft limit: limite k pode ser excedido, mas utilizador recebe avisa para libertar espaço
  - hard limit: limite k não pode ser excedido

6) Fiabilidade dos Sistemas de Ficheiros (File System Reliability)

a) Backups

- Problemas essenciais:
  - recuperação de desastre (crash, fogo, etc)
  - recuperação de estupidez (recycle bin)
- Evitam-se geralmente backups de:
  - Programas (pois podem-se reinstalar)
  - Ficheiros que modelizam dispositivos I/O (ex: directoria /dev)
- backup incremental → apenas as alterações desde o backup anterior (1º backup completo)
- backup físico → cópia igual ao disco (bloco por bloco)
  - Vantagens: (1) fácil de programar e de implementar; (2) rápido
  - Desvantagens: (1) copia blocos vazios/danificados; (2) backup incremental impossível
- backup lógico → inicia na directoria raíz e copia uma directoria de cada vez com todos os seus ficheiros → mantém-se a ordem lógica
- Algoritmo de Backup (Fases …):
  1. marca i-node de ficheiros modificados e marca todas as directorias
  2. desmarca directoria k não contêm ficheiros modificados
  3. verifica ordem numérica dos i-nodes e faz o backup de todas as directorias marcadas
  4. faz o backup de todos os ficheiros marcados
- Questões importantes (backup & restauro):
  - lista dos blocos livres tem k ser reconstruida a cada restauro
  - restaurar links → evitar cópias do mesmo ficheiro

b) Consistência de Sistemas de Ficheiros

- quando ocorre um crash, pode haver inconsistências
- reinício após crash → verificação de inconsistências e reparação (ex: fsck, scandisk)
  - 1ª contagem → qtas vezes é k 1 bloco está presente num ficheiro?
  - 2ª contagem → qtas vezes é k 1 bloco está presente na lista (bitmap) de blocos livres?
  - 2 zéros → missing block → reportado como bloco livre
  - obter um "2" na lista de blocos livre → reconstruir adequadamente a lista
  - obter um "2" na lista de blocos em uso → criar cópia do bloco
- efectuar a mesma verificação para as directorias (verificar links)

7) Desempenho de Sistemas de Ficheiros (File System Performance)

a) Caching

- leitura de um ficheiro → carregar bloco no cache → leitura
- manter alguns blocos no cache → acesso + rápido
- referências de cache não frequentes → alg. LRU modificado
- Factores novos do alg. LRU modificado:

- o bloco vai ser usado em breve? → divisão em categorias de blocos
- o bloco é essencial à consistência do SF? → escrever bloco imediatamente no disco

- UNIX → chamada ao sistema "update" → grava todos os 30s → crashes afectam pouco

b) Leitura antecipada (Block Read Ahead)

- leitura e caching, por antecipação, de vários blocos
- vantajoso para acesso sequencial
- bit de acesso (sequencial/aleatório)

c) Armazenamento contíguo (Reducing Disk Arm Motion)

- objectivo → tentar reduzir o tempo de overhead relativo ao posicionamento no disco
  - tentar colocar o ficheiro em blocos de disco contíguos
  - dividir disco em grupo de cilindro cada um com os seus próprios i-nodes

8) Sistemas de Ficheiros LFS (Log-Structured File Systems)

- Com CPUs mais rápidas e memórias maiores
  - caches de disco podem também ser grandes
  - número maior de pedidos pode ser resolvido pela cache
  - maior parte de acessos a disco será para escritas
- LFS (Log-structured File System) organiza o disco como se fosse um log
  - escritas são guardadas em memória (buffered)
  - escreve periodicamente no fim do log
  - i-nodes, blocos de dados, diretórios etc estão todos misturados ocupando um segmento
  - abrir ficheiro → localizar i-node → encontrar blocos

4.4. Exemplos de Sistemas de Ficheiros

1) Sistemas de Ficheiros de CD-ROM

a) Sistema de Ficheiros ISO9660

- objectivo: tornar CD-ROMs legíveis em todos os computadores
- sem cilindros → uma única espiral contínua
- todo o CD-ROM começa com 16 blocos a serem geridos pelo fabricante (bootstrap, etc)
- 1 bloco → primary volume descriptor → informação geral, identificadores, dimensão dos blocos, localização da directoria raíz, etc
- Formato de uma entrada de directoria:
  - comprimento da entrada de directoria
  - localização do ficheiro (bloco inicial)
  - dimensão do ficheiro
  - data e hora (da gravação)
  - flags
  - interleaving
  - nº do CD-ROM em k o ficheiro está localizado
  - dimensão do nome do ficheiro
  - nome do ficheiro → nome base (8) + extensão (3) → limitações MS-DOS
  - padding
  - System use
- Três Níveis (Three Levels)
  - Level 1: limitação de nomes 8(+3) caracteres
  - Level 2: aumenta comprimento a 31car., mas mesmo conjunto de caracteres
  - Level 3: blocos não têm k ser contíguos

b) Extensões Rock Ridge (UNIX)

- PX → bits de protecção (rwxrwxrwx)
- PN → permite representação de dispositivos no CD-ROM (directoria /dev)
- SL → ligações simbólicas k podem referir ficheiros de outro SF
- NM → permite nomes segundo as normas UNIX
- CL, PL, RE → permite contornar limite de (sub)directorias (8)
- TF → carimbos de tempo UNIX

c) Extensões Joliet (Microsoft)

1. longos nomes de ficheiros → até 64 caracteres
2. set de caracteres Unicode → útil para países com outros alfabetos
3. ultrapassar 8 níveis de directorias → permite contornar limite de (sub)directorias (8)
4. nomes de directorias com extensões → não serve para nada!!!

2) Sistema de Ficheiros do MS-DOS

- deriva do CP/M
- nomes: 8+3 caracteres (upper case)
- nº infinito de subdirectorias → árvore → sem ligações (só existem no UNIX)
- permite nomes absolutos & relativos e directorias de trabalho
- directorias têm dimensão variável MAS entradas de directoria têm dimensão fixa (32 bytes)
- Formato de uma entrada de directoria:
  - nome → 8 caracteres (8 bytes)
  - extensão → 3 caracteres (3 bytes)
  - atributos → bits: read-only, archived, hidden, system (1 byte)
  - resevado → inutilizado (10 bytes)
  - hora (time)
  - data
  - nº do 1º bloco
  - tamanho (size)
- MS-DOS usa FAT na memória →

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	3 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Tamanhos máximos de partições para dimensões de blocos diferentes.

3) Sistema de Ficheiros do UNIX V7

- deriva do MULTICS
- hierarquia de directorias com ligações (links)
- nomes com 14 caracteres em ASCII (excepto / e 0)
- entrada de directoria → nome (14 bytes) + nº do i-node (2 bytes)
- limite de ficheiros num SF → 64K
- ficheiros grandes (i-node pequeno) → single/double/triple indirect block(s)

Tema 5 – INPUT/OUTPUT

➢ **Uma das mais importantes funções do SO é controlar os dispositivos periféricos**
- Enviar comandos aos dispositivos
- Receber/enviar dados
- Aperceber-se das interrupções
- Tratar erros

➢ O SO estabelece uma interface entre os dispositivos e o resto do sistema

5.1. Príncipios do Hardware de I/O

1) Dispositivos I/O

- Dispositivos orientados ao bloco (block devices):
  - Guardam a informação em blocos de dimensão fixa, cada um com o seu endereço
  - O acesso aos blocos é feito de forma independente uns dos outros
  - Exemplos: discos-rígidos, CD-ROMs
- Dispositvos orientados ao caracter (character devices):
  - Aceita ou entrega um conjunto contínuo de bytes (stream)
  - Exemplos: teclados, ratos, modems, impressoras
- Outros dispsitivos:
  - Relógio do sistema: apenas envia uma interrupção periodicamente

2) Controladores de Dispositivos

- Interface (em HW) que o dispositivo apresenta ao resto do sistema
- interface entre o exterior de um dispositivo e o seu funcionamento interno (def. Wikipedia)
- O controlador é geralmente constituído por:
  - Conjunto de registos programáveis
  - Conjunto de registos para dados (escrita/leitura)
  - Lógica de controlo
- No fundo é um micro-processador, com ligações aos barramentos (buses) do sistema

3) "Memory-Mapped I/O"

- O acesso aos dispositivos pode ser feito de duas formas:
  - → **Portos I/O**
    - a cada registo dos diversos controladores é atribuído um número designado Porto I/O
    - o acesso é feito utilizando instruções em linguagem de baixo nível (habitualmente assembly: IN & OUT)
    - ex: IN → copia conteúdo do registo do controlador de periférico para um registo da CPU
    - espaço de I/O e de memória separados
  - → **Memory-mapped I/O**
    - deste modo o acesso aos dispositivos é feito como se tratasse de um acesso à memória
    - cada registo do controlador é mapeado para uma posição de memória
    - uma escrita ou leitura nessa posição de memória corresponde na realidade a uma escrita/leitura no registo do controlador
    - espaço de I/O e de memória único
- **Vantagens do "Memory-mapped I/O":**
  1. registos de controlo são apenas variáveis → driver programado em C (sem assembly)
  2. mecanismos de protecção desnecessários pk dispositivos são colocados cada um na sua página e assim não interferem uns com os outros → manter apenas cuidado de não colocar espaço de endereçamento do registo de controlo num espaço de endereçamento virtual
  3. instruções k referenciam memória também podem referenciar registos de controlo
- **Desvantagens do "Memory-mapped I/O":**
  1. caching pode tornar-se num desastre → solução: HW deve poder desligar caching
  2. todas as referências à memória têm k ser verificadas (memória ou I/O) → solução: durante o boot, identificar as regiões k NÃO são de memória e depois usar um filtro no chip PCI

4) Acesso Directo à Memória (Direct Memory Access (DMA))

- DMA permite que os dispositivos transfiram dados sem sobrecarregar a CPU

- **leitura de disco "sem" DMA**: controlador lê blocos do disco e coloca-os no buffer interno → controlador faz computação e verificação dos blocos → controlador causa interrupção → SO lê blocos do buffer para a memória
- **leitura de disco "com" DMA**: CPU programa DMA (registos) → CPU ordena controlador do disco a fazer transf./comp./verif. dos blocos (buffer) → DMA dá início à transferência → controlador transfere do buffer para a memória → controlador envia aviso (de fim) a DMA → (loop: DMA verifica contador de bytes até este estar a 0) → DMA interrompe CPU indicando o fim da transferência → CPU não necessita de transferir nada, já está tudo ne memória
- **Modos de operação dos "buses":**
  - word-at-a-time mode: controlador DMA pede e recebe "bus" para transferir uma palavra, CPU espera → **cycle stealing** → desvantagem: demasiados pedidos atrasam CPU
  - block mode: aquisição do "bus" para transferir várias palavras → **burst mode** → vantagem: transfere + dados de uma vez → desvantagem: pode parar CPU por muito tempo
- **Razões de existência do buffer interno:**
  - controlador faz verificação antes da transferência para a memória
  - controlador faz o trabalho ao seu ritmo, sem tempos críticos
- **Argumento contra DMA**: por vezes a CPU é tão rápida k faz o trabalho + depressa do k qdo espera pelo controlador DMA

5) Interrupções [Revisão] (Interruptions Revisited)

- dispositivo I/O acaba o seu trabalho → causa interrupção enviando um sinal pelo seu "bus" até ao controlador de interrupções → se não houver interrupções pendentes, controlador processa a interrupção → controlador de interrupções coloca nº nas linhas de endereço para identificar o dispositivo I/O k requer atenção → nº usado como indicador na tabela "vector de interrupções" para obter um novo program counter → interrupção é notificada (pode fazer-se outra)
- Onde/Como guardar a imformação dos processos interrompidos?
  - em registos internos do SO → atrasos; possível perda de dados
  - na pilha (stack) → qual pilha?; pode calhar num fim de página e gerar uma falta de página
  - na pilha do núcleo → MN, invalidar MMU e TLB, recarregá-los atrasa CPU
- Outros problemas:
  - CPUs são "pipelined" e "superscalar" → várias informações são interrompidas → não se sabe onde continuar após a interrupção
  - precise interrupt: interrupção k deixa a máquina num estado bem definido
  - Propriedades dos "precise interrupts":
    1. program counter é guardado num local conhecido
    2. todas as instruções antes da k está apontada estão executadas
    3. nenhuma instrução depois da k está apontada está executada
    4. o estado de execução de uma instrução apontada é conhecido
  - uma interrupção k não cumpre ests requisitos é um "imprecise interrupt"

5.2. Princípios do Software de I/O

1) Objectivos do Software de I/O

- Conceito chave na concepção de SW de I/O → **Independência do Dispositivo**: possibilidade de usar o mesmo programa para vários (ou mesmo todos) os dispositivos I/O
- Nomeação uniforme: regras de nomeação de ficheiros independentes do dispositivo
- Mount/unmount: colocar dispositivo (floppy-disk) na hierarquia do SF
- Tratamento de erros: deve ser feito ao nível de HW (controlador), se possível
- Transferências síncronas/assíncronas: maioria assíncronas (interrupções)

- Buffering
- Dispositivos partilháveis/dedicados
- **Existem basicamente três formas diferentes de efectuar operações de I/O:**
  - I/O programada
  - I/O por interrupções
  - I/O por DMA

2) I/O Programada

- CPU efectua todo o trabalho de I/O, vai enviando/recebendo os dados dos dispositivos
- após despachar cada dado, verifica se o periférico está pronto para continuar (espera activa)
- Desvantagem: processador passa maior parte do tempo em espera activa

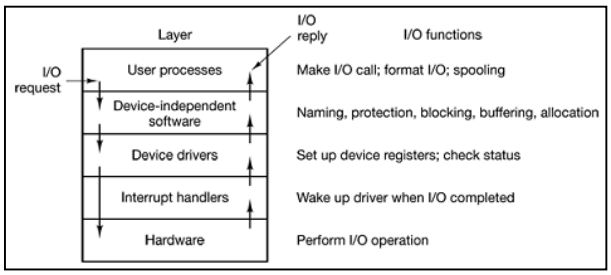
3) I/O por Interrupções

- Com este modelo, o processador envia/recebe dados do periférico, mas depois pode-se dedicar a outro processo
- Entretanto quando o periférico está pronto para continuar, interrompe o processador
- Após a interrupção o processador envia/recebe mais dados e assim sucessivamente até a operação de I/O estar concluída
- Vantagem: Maior rendimento → o processador pode-se ocupar de outros processos, enquanto não chega uma interrupção
- Desvantagem: As interrupções ocorrem com demasiada frequência

4) I/O por DMA

- Semelhante a I/O programada, mas o controlador DMA substitui o processador
- O processador limita-se a dar as instruções necessárias ao controlador de DMA para iniciar a transferência de dados
- Quando o controlador termina a transferência de dados, notifica o processador através de uma interrupção
- Vantagem: Uma só interrupção após toda a operação de I/O terminar
- Desvantagem (só em alguns casos): Este esquema pode não funcionar se o periférico for demasiado rápido em relação ao controlador de DMA

5.3. Camadas de Software de I/O (I/O Software Layers)



➔ O modelo de I/O para um sistema operativo, pode ser descrito por várias camadas funcionais

1) Tratamento de Interrupções (Interrupt Handlers)

- Quando ocorre uma interrupção, o SO tem diversas tarefas a efectuar:
  1. Salvaguardar o conteúdo dos registos do processador
  2. Estabelecer um contexto para o procedimento de tratamento da interrupção (ISP)
  3. Enviar sinal de notificação ao controlador de interrupções (ou reactivá-las, se não existir controlador)
  4. Executar o procedimento de tratamento da interrupção (ISP)
  5. Escolher um novo processo para correr

2) Drivers dos Dispositivos (Device Drivers)

- Código que permite o controlo de um dado dispositivo (contém o código para programação do controlador do dispositivo correspondente)
- Geralmente o fabricante do dispositivo fornece também o driver do mesmo
- drivers correrem em modo núcleo (modelo mais fácil de implementar)

- drivers situados no fundo do SO → por baixo de todo o resto
- SO tem k estar preparado para aceitar drivers → interface dedicada aos dispositivos orientados aos blocos & outra interface para disp.o.aos caracteres → cada uma com os seus procedimentos
- Em tempos idos, o código de específico para controlo de cada periférico era compilado juntamente com o núcleo do SO → era rara a ligação de novos periféricos
- Actualmente SOs suportam o carregamento dinâmico de device drivers
- **Funções de um driver:**
  - aceitar/tratar dos pedidos de leitura/escrita do SW independente dos dispositivos (DISW)
  - inicializar dispositivo
  - gerir energia (power) do dispositivo
- **Estrutura geral dos drivers (passos k efectuam):**
  - verificar validade dos parâmetros de input
  - verificar se o dispositivo está a ser usado (pode ser necessário ligá-lo)
  - controlar dispositivo → sequência de comandos
  - enquanto controlador trabalha, driver espera (tempo longo) ou continua (tempo curto)
  - driver envia dados para DISW

3) Software de I/O Independente dos Dispositivos (Device Independent I/O Software)

a) Interface Uniforme para os Drivers dos Dispositivos

- Apresentar ao programador (utilizador) uma interface de acesso aos dispositivos sempre da mesma forma, independentemente do dispositivo
- Exemplo: escrever num disco, disquete ou CD-RW sempre da mesma forma
- Uniformizar os nomes pelos quais os dispositivos são referenciados pelo sistema

b) Buffering

- sem buffering → demasiadas interrupções
- buffering no espaço do utilizador → buffer cheio → falta de página
- buffering no núcleo + cópia para o esp.util. → buffer a esvaziar, não há buffer no núcleo
- double buffering → 2 buffers do núcleo → turnos
- buffering é importante no input "e" no output
- exemplo (envio de dados de A para B): userA-kernelA → kernelA-controllerA → controllerA-controllerB (network) → controllerB-kernelB → kernelB-userB

c) Reportar Erros (Error Reporting)

- Efectuar o tratamento dos erros originados pelos dispositivos
- Classes de erros:
  - erros de programação → qdo 1 processo pede algo impossível
  - erros no próprio disp. I/O → ex: escrever num bloco danificado

d) Alocar e Libertar Dispositivos Dedicados (Allocating and Releasing Dedicated Devices)

- dispositivos dedicados só podem ser utilizados por 1 processo de cada vez
- abrir (alocar) e fechar (libertar) ficheiros especiais k representam os dispositivos
- se dispositivo ocupado, pedido falha ou bloqueia (fica em fila de espera)

e) Dimensão de Blocos Independente dos Dispositivos (Independent-Device Block Size)

- discos diferentes podem ter dimensões de sectores diferentes
- cabe ao SO esconder esse facto e apresentar uma dimensão igual para todos

4) Software I/O do Espaço do Utilizador (User-Space I/O Software)

- Chamadas ao sistema que iniciam as operações de I/O são normalmente organizadas em bibliotecas acessíveis ao utilizador (programador) → bibliotecas na directoria lib no Unix/Linux
- Spooling:
  - técnica para controlar o acesso a periféricos dedicados

- utilizadores diferentes podem utilizar o mesmo periférico dedicado, mas tem que ser à vez (ex: impressora)
- existe uma directoria especial (spooling directory) para onde os utilizadores enviam os documentos que pretendem imprimir
- um processo (daemon) do SO encarrega-se de ir despachando os documentos para a impressora, mas de uma forma ordenada (eventualmente com prioridades)

5.4. Discos

1) Hardware de Discos

➢ grande variedade de discos → úteis para memória secundária; armazenamento

a) Discos Magnéticos

- organizados em cilindros → pistas (tracks) → sectores
- discos simples → apenas enviam sequência de bits → controlador faz tudo
- discos IDE → contêm microcontrolador k faz algum trabalho
- overlapped seeks: disco por fazer seek num local enquanto lê/escreve noutro
- geometria especificada pelo driver pode ser diferente do formato físico do disco → controlador faz esse mapeamento
- endereçamento lógico de blocos (logical block addressing) → numeração consecutiva dos sectores sem olhar à geometria do disco

b) RAID

- ↗ velocidade dos discos → I/O paralelo → novo dispositivo: RAID
- RAID → caixa com vários discos
- substituir controlador de disco por controlador RAID
- dados distribuídos pelos discos → permite operações paralelas
- **Organizações possíveis de RAID:**
  - → **RAID level 0**
    - disco dividido em faixas (stripes) de K sectores cada
    - striping → distribuição de dados por faixas
    - o desempenho do RAID é óptimo com pedidos grandes e mau qdo se requisita um sector de cada vez
    - falha num disco causa falha na transferência toda
  - → **RAID level 1**
    - duplica os discos → 4 discos primários & 4 discos de backup
    - desempenho de escrita é igual, mas desempenho de leitura duplica
    - falha num disco → usa-se o backup
  - → **RAID level 2**
    - baseado em palavras (word), baseado em bytes
    - dividir cada byte num par de 4 bits e adicionar 3 bits de Hamming code, doa quais os bits 1, 2 e 4 são bits de paridade (parity bits)
    - Hamming word → 7 bits → 7 drives
    - Vantagens: throughput enorme; perder uma drive não é problemático (Hamming code repara)
    - Desvantagem: drives têm k estar sincronizadas rotacionalmente
  - → **RAID level 3**
    - versão simplificada de RAID level 2
    - um único bit de paridade → são escritos na drive de paridade (parity drive)
    - crash → controlador mete bit a 0 → erro de paridade → bit deve ser 1

→ **RAID level 4**
  — idêntico ao RAID level 0, mas com uma drive de paridade (faixa por faixa)
  — Vantagem: drive de paridade trata erros de crash
  — Desvantagem: pouco eficiente em pequenas actualizações

→ **RAID level 5**
  — baseado no level 4, mas bits de paridade estão distribuídos por todas as drives
  — Vantagem: não sobrecarrega uma única drive de paridade
  — Desvantagem: reconstituição de uma drive k sofre um crash é complicada

c) CD-ROM

➢ **CD (Compact Disc)**

- disco óptico desenvolvido em 1980 pela Philips & Sony
- gravação → fazem-se buracos de 0,8 µm com um laser infravermelho de alta-potência num disco de vidro → faz-se um molde → injecta-se polycarbonato no molde → coloca-se uma camada fina de alumínio → CD pronto
- pits (buracos); lands (espaços sem buracos)
- reprodução → laser de luz infravermelha → comprimento de onda das reflexões determinam pits/lands
- transição pit/land 1 ; ausência de transição 0
- uma única espiral contígua
- velocidade constante de bits → velocidade de rotação ↘ conforme se vai para o exterior

➢ **CD-ROM**

- 1984 → guardar dadoa informáticos
- usa Hamming codes
- divisão em sectores
- preambúlo (preamble) usado para procurar (seek)
- sistema de ficheiros → ISO 9660

d) CD-Recordables

- graváveis a partir de gravadores de CD domésticos
- camada dourada & camada de dye
- dye → fica com manchas qdo gravado → reflexão na camada dourada impossível
- simula pits & lands
- CD-ROM XA → gravação incremental → ex: Kodak PhotoCD
- gravação incremental → cada pista tem a sua própria VTOC
- bits têm k chegar a velocidade constante → criação de uma imagem de CD → lento

e) CD-Rewritables

- laser com 3 níveis de potência: gravar, apagar, ler

f) DVD

- **Vantagens de discos ópticos sobre tapes para filmes:**
  → maior qualidade de imagem
  → mais baratos
  → duram + tempo
  → ocupam menos espaço
  → não necessitam de ser rebobinados
- DVD (Digital Versatile Disk) → para filmes
- **Novidades em relação ao CD:**
  1. pits menores

2. espiral mais apertada
3. laser vermelho (mais exacto)

- DVDs → maior capacidade (4,7GB) e mais rápidos k CDs
- **Tipos de DVDs:**
  1. Single-sided, single-layer (4.7 GB)
  2. Single-sided, dual-layer (8.5 GB) → Philips & Sony
  3. Double-sided, single-layer (9.4 GB) → Toshiba & Time Warner
  4. Double-sided, dual-layer (17 GB)

2) Formatação de Disco (Disk Formatting)

- antes de utilizar um disco → formatação de baixo nível → criação dos sectores (preamble)
- preamble → indica início de um sector, nº de cilindro, nº de sector, etc
- ECC → informação redundante → recuperação de erros de leitura
- alguns sectores adicionais para substituir sectores danificados
- cilinder skew → posição do sector 0 é diferente de pista para pista → ↗ desempenho → permite leitura contínua de múltiplas pistas
- formatação de baixo nível reduz capacidade do disco (até 20%)
- single interleaving → numeração dos sectores: 1,x,2,x,3,x,4,…
- double interleaving → numeração dos sectores: 1,x,x,2,x,x,3,x,x,4,…
- form.BN → partições
- sector 0 → master boot record
- formatação de alto nível → criação: bloco de boot, directoria raíz, SF vazio, etc

3) Algoritmos de Escalonamento do Braço do Disco (Disk Arm Scheduling Algorithms)

- **Factores que determinam a velocidade de um disco:**
  1. tempo de busca (seek time)
  2. demora de rotação (ratational delay)
  3. tempo de transferência
- seek é o k reduz mais a velocidade do disco
- **Possíveis algoritmos:**
  - FCFS (Forst-Come First-Served) → fraco desempenho
  - SSF (Shortest Seek First) → serviço pobre nos pedidos situados longe do meio
  - Algoritmo do Elevador → UP até último pedido → DOWN até último pedido → etc
  - Algoritmo do Elevador Modificado → UP até último → início → UP até último → etc
- se seek é mto + rápido k rotational delay → ordenar pedidos por nº de sector de forma a serem executados qdo sector seguinte passa por baixo do leitor (head), mesmo k mova o braço
- controlador costuma ler múltiplos sectores da mesma pista em cache
- cache do controlador costuma conter ficheiros não requisitados, mas k foram guardados pk passaram debaixo do leitor (head) ≠ cache da memória

4) Tratamento de Erros (Error Handling)

- **Como/Onde lidar com os blocos danificados (bad blocks)?**
  - → no controlador → bad blocks colocados em blocos de reserva
    - apenas substituir → 1,2,3,x,5,6,7,4
    - substituir e remapear tudo → 1,2,3,x,4,5,6,7
  - → no SO
    - faz a mesma coisa, mas em termos de SW, usa tabela de remapeamento
    - mantem lista com blocos danificados para evitar k apareçam na lista de blocos livres
- programas de backup não devem copiar blocos danificados
- erros de seek → recalibrar braço (arm)

5) Armazenamento Estável (Stable Storage)

- stable storage → sistema k ou escreve os dados correctamente ou não faz nada mantendo intactos os dados existentes → implementado em SW
- Escrita-estável: primeiro escrever um bloco na unidade 1 e em seguida ler o mesmo dado de volta para verificar se ele foi escrito correctamente
- Leitura-estável: Lê o primeiro bloco da unidade 1. Se essa unidade produz um ECC incorrecto, a leitura é tentada novamente n vezes.

5.5. Relógios (Clocks)

- essenciais para o funcionamento de qualquer sistema multiprogramado
- mantêm a hora do dia e evitam que um processo monopolize a CPU

1) Hardware do Relógio

- contruído a partir de 3 componentes: oscilador de cristal, contador, registo de apoio
- contador decrementado a cada pulsação
- qdo contador=0 → interrupção
- **Modos de operação:**
  - one-shot mode → processo de contagem só é accionado por SW
  - square-shot mode → após atingir o zero e causar a interrupção, o registo de apoio é automaticamente copiado para dentro do contador e o processo todo é repetido → interrupções periódicas → clock ticks
- Para evitar a perda do horário actual quando a energia do computador é desligada, a maioria dos computadores tem um relógio de segurança mantido por uma bateria

2) Software do Relógio

- **Obrigações exactas do driver do relógio:**
  1. Manter a hora do dia → incremento do contador em cada tique do relógio
     - usando um contador de 64 bits
     - manter o tempo em segundos (≠ tiques)
     - contagem em tiques desde k o PC foi ligado (adiciona-se tempo guardado)
  2. Evitar que um processo monopolize a CPU → escalonador inicializa o contador com o valor do quantum do processo em tiques de relógio
  3. Contabilizar o uso da CPU → uso de um temporizador secundário
  4. Tratar a chamada ao sistema alarm feita pelos processos do utilizador
  5. Fornecer temporizadores watchdog para partes do próprio sistema
  6. Gerar o perfil da execução, da monitorização e das colheitas de estatísticas
- watchdog timers → exemplo do floppy disk a ser ligado

3) Temporizadores por Software (Soft Timers)

- A maioria dos computadores tem um segundo relógio programável, que pode ser ajustado para causar interrupções em qualquer taxa que um programa precisar. Os temporizadores por software evitam interrupções.