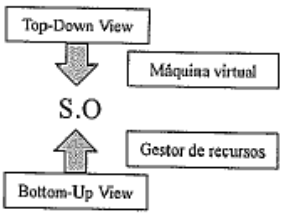
2019/2020

Resumos Sistemas Operativos

Slides Teóricos

Capítulo 1 - Evolução Histórica

Objetivo de um sistema operativo – Fornecer um ambiente no qual o utilizador possa executar tarefas de maneira conveniente e eficiente.



- S.O como **máquina virtual** (Top-Down View)
  - O utilizador não pretende dominar os detalhes próprios ás características da máquina.
  - Virtualiza H/W. A abstração apresentada ao utilizador deseja-se simples e de fácil compreensão.
  - Apresenta ao utilizador um interface que trata de um modo uniforme operações sobre entidades parecidas.
  - Garante fiabilidade e segurança.

- S.O como **gestor de recursos** (Bottom-Up View)
  - Garante uma gestão de recursos.
  - Otimiza o desempenho.

O que é um sistema operativo

- **Gestor de Recursos** – Gere e aloca recursos
- **Programa de controlo** – Controla a execução dos programas do utilizador e as operações dos dispositivos de entrada e saída.
- **Kernel** – único programa que está sempre a ser executado.

Sistemas Batch

Ideia – Agrupar um conjunto de *jobs** e executá-los. Otimizar a utilização de recursos, tempo de set-up do sistema, entre outros.

**Job* – Unidade que combina uma sequência predefinida de comandos, programas e dados. Durante a execução do job não é necessária interação com o utilizador.

- As operações de entrada e saída podem permanecer em paralelo.
- Permite a utilização de mecanismos de otimização de gestão de memória
- Concorrência limitada entre programas e tarefas input/ouput.
- São possíveis através de mecanismos de interrupções.

Multiprogramação

- **Multiprogramação**
  - Executa vários programas em simultâneo.
  - Vários programas são mantidos em memoria ao mesmo tempo e o CPU é multiplexado entre eles.
- **Sistemas interativos – Tempo partilhado**
  - O programador em acesso frequente à máquina.
  - Multiplexagem entre vários utilizadores.
  - Cada utilizador pensa que suspensa sempre recursos.
  - Memoria virtual – ultrapassa as restrições físicas impostas pela memoria.

Tipos de sistemas

- **Sistemas paralelos**
  - Sistemas com mais do que um CPU e que partilham entre eles o bus, relógio, memoria (por vezes) e devices de periféricos.

    <u>Vantagens</u>
    - Maios capacidade de trabalho em tempos curtos.
    - Economia (a partilha de alguns recursos pelos CPU pode diminuir custos).

        - Aumento da segurança (tolerância a falhas).

- **Sistemas distribuídos**
    - A carga computacional é distribuída por vários processadores, mas estes não partilham relógio nem memória. Cada processador tem a sua memoria local e comunicam-se através de linhas de comunicação. Os processadores podem variar em complexidade e função.
    - Estes sistemas têm capacidade para trabalha em rede.

      Vantagens
        - Partilha de recursos
        - Velocidade de computação
        - Segurança – comunicação

- **Sistemas de tempo real**
    - Um sistema de tempo real deve obedecer a restrições de tempo.
    - O processamento deverá ser feito de acordo com as restrições definidas, caso contrário o sistema irá falhar.

Componentes do sistema

- **Gestão de processos** – Um processo é um programa em execução
    - Um processo, para realizar a sua tarefa, necessita de recursos como: tempo de CPU, memória, ficheiros, dispositivos de I/O, entre outros.

      Funções do sistema operativo na gestão de processos:
        - Criar e eliminar processos.
        - Suspender e ativar processos.
        - Garantir os mecanismos de sincronização de processos e comunicação entre processos.

- **Gestão de memória central (primária)**
    - O processador e os dispositivos de entrada e saída armazenam os dados de acesso rápido na memória.
    - A memória central é volátil.

      Funções do sistema operativo na gestão de memória:

        - Monitorizar que parte da memoria está a ser utilizada e por quem.
        - Decidir que processo deve ser carregado quando existe espaço em memória.
        - Alocar e desalocar espaço da memoria conforme as necessidades.

- **Gestão de memória secundária**
    - Uma vez que a memória central (primária) é volátil e pequena é necessária uma memória secundária por forma a armazenar os dados e os programas de forma permanente.
    - Discos rígidos

      Funções do sistema operativo na gestão de memória:

        - Gestão do espaço livre.
        - Alocação do espaço para armazenamento.
        - Escalonador de disco.

- **Gestão de input / output**
    - O sistema de I/O:
        - Buffer/cache.
        - Interface geral de device-driver.
        - Drivers para dispositivos de hardware específicos.

- **Gestão de ficheiros**
    - Um ficheiro é uma coleção de informação relacionada definida pelo seu criador (programas e dados).

      Funções do sistema operativo na gestão de ficheiros:

        - Criação e eliminação de ficheiros.
        - Criação e eliminação de diretorias.
        - Primitivas para manipulação de ficheiros e diretorias
        - Mapeamento dos ficheiros
        - Backups.

- **Proteção do sistema**
    - Mecanismo de proteção para controlar os acessos dos programas, processos e ou utilizadores ao sistema e aos recursos.

      Funções do sistema operativo na proteção do sistema:

        - Distinguir acessos autorizados e não autorizados a recursos.
        - Especificar controlos.
        - Mecanismos de imposição.

- **Rede** – Sistema distribuídos

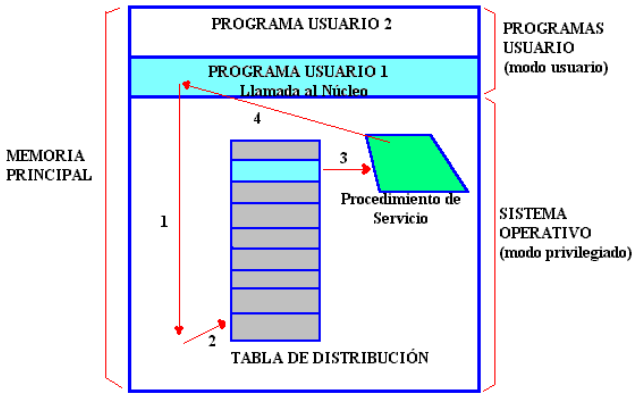
- **Gestão do interpretador de comandos**
    - Muitos dos comandos entregues ao S.O são instruções de controlo que lidam com:
        - Gestão e criação de processos, gestão I/O, gestão de memoria central e/ou secundaria, acessos ao sistema de ficheiros, proteção, rede, entre outros.
    - O programa que lê e interpreta essas instruções de controlo é o interpretador de comandos.

      Funções do sistema operativo na gestão de interpretador de comandos:

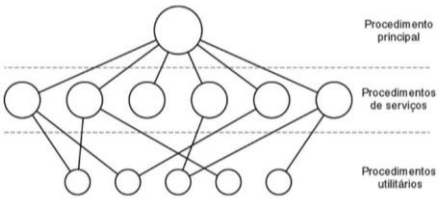
- Ler e executar o próximo comando

Estrutura do sistema operativo

- **Sistemas Monolíticos (“The Big Mess”)**
  - Cada procedimento obedece a uma especificação bem definida no que respeita aos parâmetros e resultado.
  - Cada procedimento é visível por todos os outros, não possuindo uma estrutura modular.



- **Sistemas Monolíticos: estrutura básica**
  - É um programa principal que chama as funções de serviço
  - É um conjunto de funções de serviço que sustentam as chamadas ao sistema operático (system cal).
  - É um conjunto de funções auxiliares (apoio às funções de serviço).



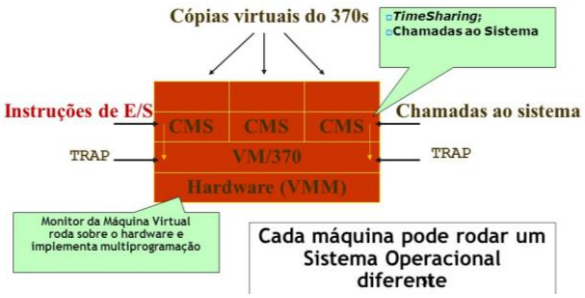
Modelo simples de estruturação de um sistema monolítico

- **Sistema dividido em camadas**
  - O sistema operativo é estruturado em camadas hierárquicas.
  - A modularidade pressupõe que cada camada apenas use funções e serviços de camadas de níveis inferiores.

Nível	Função
5	Operador
4	Programas do utilizador
3	Entrada e saída
2	Comunicação dos processos do utilizador
1	Gestão de memória
0	Gestão da CPU (alocação de processos e multiprogramação)

- **Máquina virtual**
  - Estruturado por camadas lógicas e onde o hardware e o kernel do sistema operativo é encapsulado por forma a ser visto como hardware.
  - Uma máquina virtual proporciona um interface parecido ao do hardware físico embora mais agradável de se usar.
  - O sistema operativo cria a ilusão de múltiplos processos, cada qual executando num processador individual e com memoria individual.

- Os recursos físicos do computador são partilhados de forma a criar as máquinas virtuais.



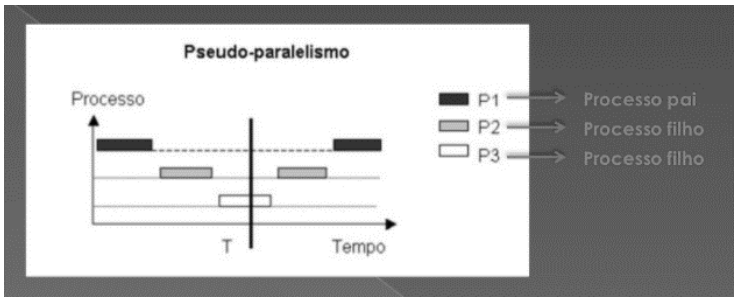
- **Cliente/Servidor**
  - A tendência dos sistemas operativos atuais é de mover o máximo de código possível para camadas superiores.
  - Remover o código ao sistema operativo por forma a simplificar o kernel.
  - Para obter um serviço (leitura de um bloco de um ficheiro) o processo cliente envia um pedido ao processo servidor.ve
  - O kernel encarrega-se das comunicações entre os clientes e os servidores.
  - Divisão logica do sistema operativo, em unidades pequenas e de fácil utilização.
  - O kernel gere as comunicações.
  - Os servidores correm em modo utilizador evitando um crash total do sistema na eventualidade de erro interno ao servidor.
  - Vocacionados para sistemas distribuídos (em rede).

Capítulo 2.1 - Gestão de Processos; Escalonamento

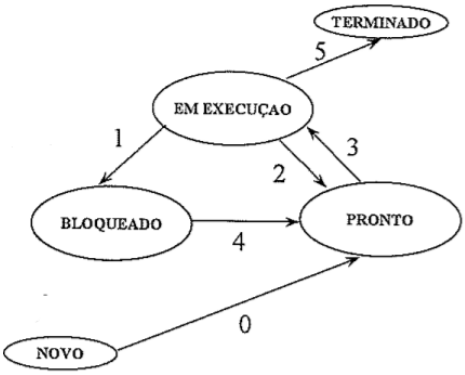
Processo – programa em execução. Um processo tem associado um programa, dados de entrada, saída e um estado.

Pseudoparalelismo é a execução de sistemas operativos multiprogramados com um único processador. Cada processo tem o seu próprio CPU virtual, mas na realidade o CPU troca várias vezes de um processo para outro.

Um programa é executado (pai), dentro desse programa podem decorrer vários outros processos (filho).



Estados dos Processos



Legenda:

NOVO – criação do processo

EM EXECUÇÃO – processo a correr

BLOQUEADO – processos em espera

PRONTO – o processo tem todos os recursos e apenas espera uma vez no processador

TERMINADO - o processo conclui a sua execução

0 – criação de um processo
1 – o recurso não disponível, entra em espera
2 – escalamento - seleciona outro processo
3 - escolhe um processo para execução
4 – recurso disponível
5 – termina o processo

BCP – Bloco de Controlo de Processos

Estrutura de dados no núcleo do sistema operativo que serve para armazenar a informação necessária para tratar um determinado processo. As informações contidas neste bloco são:

- Estado do processo
- Contador de programa
- Registos da CPU
- Informação do despacho do CPU -> Escalonador
- Informação de gestão de memória
- Contabilização de utilização de recursos
- Estado dos recursos de entrada e saída

Threads

Uma Thread é a unidade básica de utilização do CPU.

Consiste num:

- Contador de programa
- Conjunto de registos
- Espaço da stack

Uma thread partilha com threads parceiras: **(também conhecida como task)**

- Zona de código
- Zona de dados
- Recursos do Sistema Operativo

Um processo clássico é equivalente a uma task composta por uma thread.

Com a utilização de Threads é possível um mecanismo em que processos sequenciais efetuem chamadas ao sistema de bloqueio e mantenham o funcionamento paralelo.

(falta, porque está em inglês)

Escalonamento

A gestão do recurso de CPU é conhecida por escalonamento – determina como o gestor de tarefas retira e coloca processos a executar na CPU.

Política de escalonamento -> define a forma como o tempo de CPU é repartido pelos vários processos e a ordem com que esta partilha é realizada.

Estratégias de Seleção

- Dar valor á prioridade

- Sem-preempção e com preempção da CPU:

- ✓ Algoritmos não-preemptivos são projetados de forma que sempre que um processo ocupa a CPU ele não é removido até que esgote o tempo necessário á sua total execução. (FIFO; SJN; SJF)
- ✓ Algoritmos preemptivos estão associados a sistemas que utilizam interrupções para forçar a partilha da CPU. Interrupção involuntária da execução do processo.(RR;

- Grau de liberdade, neste caso há um temporizador define um intervalo de tempo ("time slice" ou "time quantum".

Modelo das estratégias

Service time – t(pi) - Tempo necessário ao processo para completar, estado no estado "running" .

Wait time – W(pi) – Tempo gasto pelo processo em estado "ready" antes da primeira transição para o "running".

Turnaround time - T(pi) – Quantidade de tempo entre o momento em que o processo entra pela primeira vez no estado "ready" e o processo termina.

Estratégias sem preempção

➢ FCFS – FIFO – First Come First Served

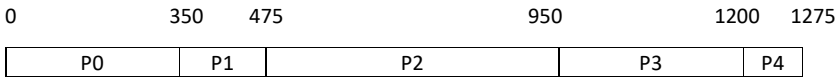
A prioridade de execução é feita pela ordem de chegada do processo.

A prioridade de um processo é calculada através de uma marca temporal.

Exemplo:

i	t(pi)
0	350
1	125
2	475
3	250
4	75

(acho que se vê pelo valor de chegada que não está neste slide)



TurnaRound

TtRnd(p0) = t(p0) = 350

TtRnd(p1) = t(p1) + TtRnd(p0) = 125 + 350 = 475

TtRnd(p2) = t(p2) + TtRnd(p1) = 475 + 475 = 950

TtRnd(p3) = t(p3) + TtRnd(p2) = 250 + 950 = 1200

TtRnd(p4) = t(p4) + TtRnd(p3) = 75 + 1200 = 1275

Tempo médio de “turnaround” é: TturnRound= (350 + 475 + 950 + 1200 + 1275) : 5 = 850

Tempos de Espera

W(p0) = 0

W(p1) = TtRnd(P0) = 350

W(p2) = TtRnd(P1) = 475

W(p3) = TtRnd(P2) = 950

W(p4) = TtRnd(P3) = 1200

Tempo médio de espera é: (0+350+475+950+1200):5 = 595

➢ SJF OU SJN – Shortest Job Next

O algoritmo dá prioridade ao processo que requer o menor tempo de execução para completa. Minimiza o tempo médio de espera, mas penaliza os processos que necessitam de mais tempo de execução.

Exemplo:

i	t(pi)
0	350
1	125
2	475
3	250
4	75

0 75 200 450 800 1275

P4	P1	P3	P0	P2

TurnaRound

TtRnd(p0) = t(p0) + t(p3) + t(p1) +t(p4) = 350 +250 +125 +75 = 800

TtRnd(p1) = t(p1) + t(p4) = 125 + 75 = 200

TtRnd(p2) = t(p2) + t(p0) + t(p3) + t(p1) +t(p4) = 1275

TtRnd(p3) = t(p3) + t(p1) + t(p4) =450

TtRnd(p4) = t(p4) = 75

Tempo médio de "turnaround" é: TturnRound= (800 + 200 + 1275 + 450 + 75) : 5 = 560

Tempos de Espera

W(p0) = 450

W(p1) = 75

W(p2) = 800

W(p3) = 200

W(p4) = 0

Tempo médio de espera é: (450 + 75 + 800 + 200 +0) : 5 = 305

➢ Escalonamento por Prioridade

Neste caso, o escalonamento é realizado através da prioridade de execução definida (maior prioridade -> menor número associado) . A prioridade define a importância do processo.

Exemplo:

i	t(pi)	Priority
0	350	5
1	125	2
2	475	3
3	250	1
4	75	4

0 250 375 850 925 1275

P3	P1	P2	P4	P0

TurnaRound

TtRnd(p0)= t(p0)+t(p4)+t(p2)+ t(p1)+t(p3) = 1275

TtRnd(p1) = t(p1) +t(p3)= 375

TtRnd(p2) = t(p2)+ t(p1)+t(p3) = 850

TtRnd(p3) = t(p3) = 250

TtRnd(p4) = t(p4)+t(p2)+ t(p1)+t(p3) = 925

Tempo médio de "turnaround" é: TturnRound= (275 + 375 + 850 + 250 + 925) : 5 = 735

Tempos de Espera

W(p0) = 925

W(p1) = 250

W(p2) = 375

W(p3) = 0

W(p4) = 850

Tempo médio de espera é: (925 + 250 + 375 + 0 + 850) : 5 = 480

➢ Escalonamento por Limite Temporal

Aplica a sistemas cujos processos devem terminar antes de um dado tempo. São usados em sistemas com tempos críticos "**de tempo real**. Este escalonamento não utiliza medidas de turnaround ou Wait time.

Exemplo:

i	t(pi)	DeadLine
0	350	575
1	125	550
2	475	1050
3	250	(none)
4	75	200

0 125 200 550 1025 1275

P1	P4	P0	P2	P3

0 75 200 550 1025 1275

P4	P1	P0	P2	P3

0 75 425 550 1025 1275

P1	P0	P1	P2	P3

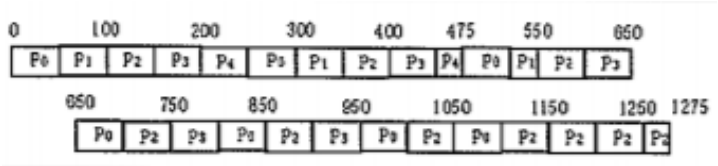
[Estratégias com preempção]

➢ Round – Robin

Algoritmo mais usado. Distribui de forma igual o tempo de processamento pelos processos.

Exemplo:
Time Quantum = 50

i	t(pi)
0	350
1	125
2	475
3	250
4	75



TurnaRound

TtRnd(p0) = 1100

TtRnd(p1) = 550

TtRnd(p2) = 1275

TtRnd(p3) = 950

TtRnd(p4) = 475

Tempo médio de "turnaround" é: TturnRound=(1100 + 550 + 1275 + 950 + 475) : 5 = 870

Tempos de Espera

W(p0) = 0

W(p1) = 50

W(p2) = 100

W(p3) = 150

W(p4) = 200

Tempo médio de espera é: (0 + 50 + 100 + 150 + 200) : 5 = 100

Multi -Lista

- Uma lista para cada tipo de processo em execução.
- Surgiu a necessidade de conciliar processos pouco interativos (CPU-bound) com processos interativos (I/O -bound).
- Selecionada um processo da lista mais prioritária
- Cada lista tem o seu próprio algoritmo de escalonamento
- Dá mais tempo de execução a processos CPU-bound, pois reduz swapping
- Os processos poderão ser transferidos entre diversas listas.
- Prioridades por envelhecimento.

Prioridades Dinâmicas – critério de alteração de prioridades é baseado na utilização dos recursos, como exemplo, o tempo de CPU ou ocupação de memória.

Capítulo 2.2 - Gestão de processos - Sincronização

Interação entre processos

- **Sincronização** – é um conjunto de protocolos usados para manter a integridade do sistema e para controlar o acesso a recursos que não podem ser usados por mais do que um processo em simultâneo.
- **Sinalização –** é a troca de sinais de temporização entre processos concorrentes para coordenar o processo coletivo, sendo uma forma de sincronização rudimentar, mas usada frequentemente.
- **Comunicação** – os processos concorrentes necessitam muitas vezes de trocar dados entre si. Uma das formas mais simples é através do uso de memória partilhada, mas o uso requer cuidados na sincronização dos acessos.

Mecanismos de sincronização

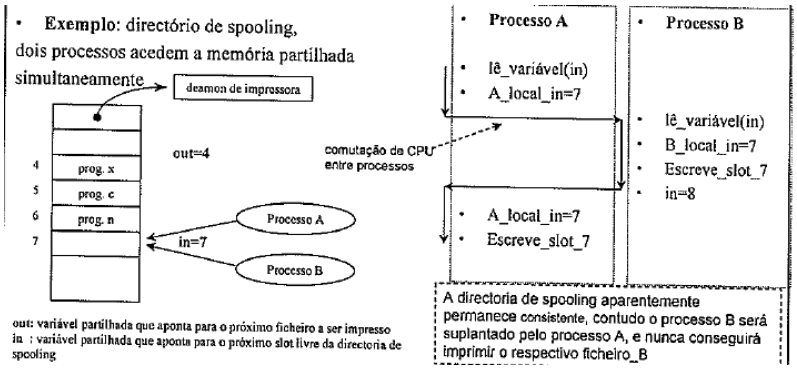
- **Cooperação** – resulta da necessidade de interação entre diversas atividades para obtenção de um objetivo comum.

  Exemplo: quando um processo requisita uma operação a outro e não pode prosseguir enquanto não obtiver resposta.

- **Competição por um recurso** – quando vários processos pretendem a obtenção de um recurso único (periférico, ficheiro) ou limitado (memória).

- **Exclusão mútua** – caso particular de competição por um recurso, quando este representa uma estrutura de dados.

Exclusão Mútua



- **Secção crítica** – como evitar estas situações:

    - Garantir que quando um processo usa uma variável partilhada, os outros ficam excluídos de o fazer.
    - A manipulação de estruturas de dados partilhados deve ser realizada de forma atómica.
    - A parte do programa onde a leitura das estruturas são realizadas chama-se de **secção critica.**

    - O acesso a uma variável partilhada pode ser considerado como uma zona critica.
    - Uma secção critica é um conjunto de instruções bem definido que normalmente protegem dados partilhados. Quando um processo entra numa zona critica e até completar a execução do código desta, mais nenhum outro processo tem permissão para entrar – **exclusão mútua**.
        - Exclusão mútua acontece quando um processo, durante determinado período, utiliza um recurso partilhado impedindo o acesso a qualquer outro.

          <u>Qualquer método de exclusão muta deverá:</u>

            - Garantir exclusão muta entre processos que partilham o recurso.
            - Não deve depender de velocidades, número de CPU's ou prioridades relativas a processos em conflito.
            - Deverá garantir que a terminação (ou crash) de um processo fora da zona critica não afete a possibilidade dos restantes processos acederem ao recurso partilhado (um processo bloquear outro).
            - Quando mais que um processo deseja entrar na secção critica, pelo menos um deverá poder fazê-lo num período finito.

- **Desativar interrupções** – soluções possíveis para mecanismos de exclusão mútua:
    - Cada processo desativa as interrupções ao entrar na secção critica, reativando-as à saída. O processo evita assim que a interrupção de relógio ative o escalonador, fazendo com que a alternância de processos não aconteça.

      <u>Problemas:</u>

- Os processos “utilizador” ficam com poder para desativar interrupções. Um processo de prioridade baixa pode bloquear todo o sistema.

Conclusão:

- Esta técnica é útil por vezes dentro do kernel, mas não é apropriada para processos de utilizador.

- **Uma só variável de lock**
  - Usar uma variável inicializada a zero – Um processo ao entrar na secção critica testa a variável.
    - **Se = 0**, o processo coloca-a a 1 e entra.
    - **Se =1**, o processo espera.

    Problemas:

    - Não funciona, entre o teste e a entrada de um processo, o CPU pode ser alterado para outro.

- **Alternância estrita**
  - Solução que obriga que a região crítica seja dada a um dos processos por vez, numa alternância estrita. O teste contínuo de uma variável em espera de um certo valor é chamado de espera ocupada, e representa um grande desperdício de UCP (unidade central de processamento).

Processo A

```
while(true) {
  while(turn!=0);          /*espera*/
    secção_critica();
    turn=1;
    resto_do_algoritmo();
}
```

Processo B

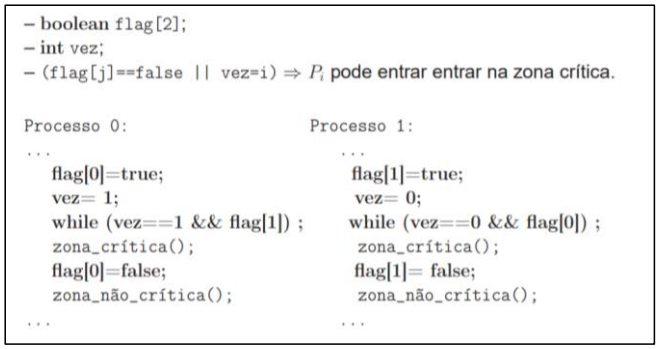
```
while(true) {
  while(turn!=1);          /*espera*/
    secção_critica();
    turn=0;
    resto_do_algoritmo();
}
```

Problemas:

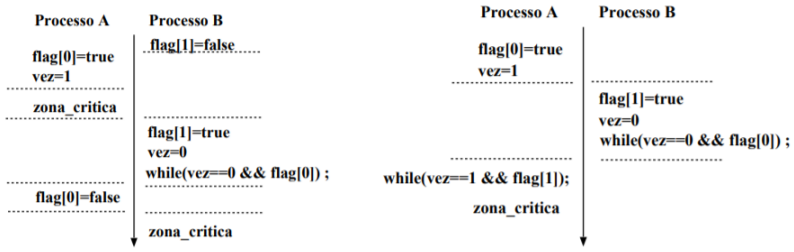
- Um processo pode bloquear outro.
- Obriga a uma alternância estrita entre processos.
- Espera, ocupando CPU.

- **Solução Peterson**

- O algoritmo de Peterson é um algoritmo para exclusão mútua, que permite a dois ou mais processos ou subprocessos compartilharem um recurso sem conflitos, utilizando apenas memória compartilhada para a comunicação.



Dois exemplos do funcionamento do algoritmo:



O processo que executa **vez = valor** em último, fica sem conseguir entrar na zona critica.

- **Trinco Lógico**
  - As soluções apresentadas são complexas para um programador.
  - Convém incorporar no sistema operativo mecanismos para implementação da exclusão mútua.
  - Primitivas do tipo:

  - **Instrução "Test and Set" (Lock)**

- Testa e modifica o conteúdo de uma posição de memoria de forma atómica.
- A instrução corresponde à função:

```
int TSL(int *m) {
  int r;

  r= *m;
  *m= 1;
  return r;
}
```

- A execução da função **TSL (m)** tem de ser indivisível (atómica), pois nenhum outro processo pode aceder à posição de memória **m** até que a instrução tenho sido executada.
    - Muitos computadores (principalmente com vários CPU's) possuem uma instrução denominada por **TSL**. Esta instrução permite de uma forma indivisível:
        - Ler conteúdo de uma palavra para um registo.
        - Escrever um valor diferente de zero nesse endereço.

Problemas:

- Um processo que perdeu o CPU enquanto estava na região critica deixa o trinco fechado.
- Quando um processo tem que ficar à espera, ele irá ficar em ciclo a testar a variável trinco – **espera ativa** (desperdiçando tempo de CPU desnecessariamente).
- Starvation - é quando um processo não consegue ser executado, de forma alguma, pois sempre existem processos de prioridade maior para serem executados, de forma que o processo "faminto" nunca consiga tempo de processamento (devido a prioridades).

Soluções:

- Suspender os processos que têm de esperar.

Semáforos

- Mecanismo de sincronização que **não requer espera ativa**.
- Baseia-se em **duas primitivas** wait e signal que operam sobre uma variável de forma atómica.
    - **Wait**(s):
        - Decrementa o semáforo s.
        - Se semáforo < 0 bloqueia o processo.
    - **Signal**(s):
        - Incrementa o semáforo s.
        - Se semáforo <= 0 retira um dos processos da fila dos bloqueados.

Semáforo s – variável interna

Exemplo: n processos com **secção critica**

- variável partilhada
  semaphore int mutex=1;
- Processo Pi

```
while (1){
        wait(&mutex);
          secção crítica;
        signal(&mutex);
          resto da secção;
        }
```


Comentado [JH1]:

Sincronização e comunicação de processos

- **Semáforos**
    - Os semáforos são mecanismos simples e poderosos para sincronização, exclusão mútua e sinalização de processos.
        - Os semáforos **não são estruturados**:
            - A aquisição de sincronização depende da criação de protocolos específicos.

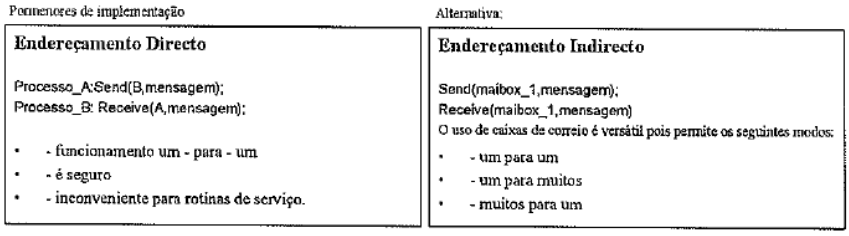
        - Uma troca incorreta das funções wait/signal ou até mesmo o esquecimento de uma delas pode bloquear todo o sistema.
    - **Não suportam abstração** de dados:
        - Mesmo utilizando corretamente, os semáforos protegem apenas o acesso a secções críticas.
        - Os semáforos encorajam a utilização de variáveis globais que apenas são protegidas de acessos concorrentes não havendo proteção a erros.
        - Não têm controlo sobre a proteção de recursos.

- **Monitores**
    - Mecanismo de estruturação de dados e controlo de concorrência.

        <u>Critica aos monitores:</u>
        - Restritivo.
        - Sinalização difícil.
        - "Deadlocks".

- **Comunicação interprocessos**
    - É uma forma de um processo partilhar informação entre processos.
    - Existem formas de passar mensagens (blocos de informação) entre espaços de endereçamento dos processos - são realizadas usando os protocolos IPC do sistema operativo.

- **Mensagens**
    - Resolvem alguns problemas dos métodos apresentados anteriormente, tendo vantagens em sistemas distribuídos.
    - A troca de mensagens é um padrão de comunicação entre redes de computadores.
    - O formato das mensagens é flexível e negociável pelos pares emissor – recetor.
    -



Passagem síncrona / assíncrona

- **Síncrona**
  - Ambos os processos têm de se encontrar.
  - Quando um processo envia uma mensagem é suspenso até o destinatário a receber.

    Consequência: poderia existir no máximo uma mensagem por par emissor/recetor.

    Vantagens:
    - Baixo "overhead" (despesa continua).
    - Implementação simples.
    - Confirmação da receção implícita da instrução send.

    Desvantagens:
    - Obriga a execução síncrona mesmo quando não é desejada.

- **Assíncrona**
  - O emissor não é bloqueado quando não existe recetor em espera.
  - A implementação da função **send** é conseguida colocando o sistema operativo como intermédio.
  - Aumenta a concorrência.

    Problema 1: Pode esgotar os buffers bloqueando outras comunicações

    Solução: Colocar limites a cada emissor/recetor.

Problema 2: "Crash" de um interveniente pode levar a que o outro fique eternamente é espera de uma mensagem.

Solução: "Receive" não bloqueante ou "Receiveve" temporizado.

Tamanho das mensagens

- **Tamanho fixo**
  - Baixa sobrecarga.
  - Desperdício de espaço com mensagens curtas.
- **Tamanho variável**
  - Reservar buffers de tamanho variável impõe sobrecarga e leva à fragmentação de memória.

Problema de "deadlocks"

- "Deadlock" é quando dois ou mais processos entram num estado em que cada processo está com recursos em que o outro quer requisitar.
- Se existirem vários processos em execução e os recursos forem finitos, vão competir para adquirirem os recursos.
- Os processos que precisam de um recurso que não está disponível ficam bloqueados – **estado wait**.

Exemplos de "deadlocks"

Considere dois processos P1 e P2 que fazem a actualização de uma lista L por duas acções distintas:

- Insere/retira elemento da lista
- Actualiza número de elementos na lista.
- Ambas as acções são secções críticas controladas por variáveis partilhadas "lock1" e "lock2".

```
boolean lock1 = FALSE;                  /* shared variables */
boolean lock2 = FALSE;
shared int I, J;

Program for P1                          Program for P2
  ...                                     ...
/* Enter list critical section */       /* Enter descr critical section */
  enter(lock1);                           enter(lock2);
  <delete element>;                       <update length>;
  <intermediate computation>;             <intermediate computation>;
/* Enter descr critical section */      /* Enter critical section for I */
  enter(lock2);                           enter(lock1);
  <Update length>;                        <add element>;
/* Exit both critical sections */       /* Exit both critical sections */
  exit(lock1);                            exit(lock2);
  exit(lock2);                            exit(lock1);
  ...                                     ...
```

Caracterização dos "deadlocks"

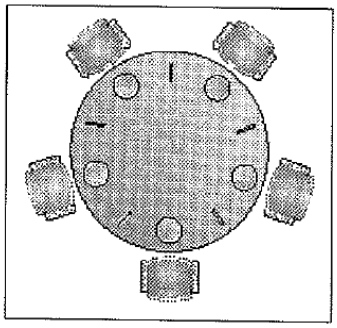
- Condições necessárias para possível existência de "deadlocks"

- **Exclusão mútua** – recurso que pode ser usado sem possibilidade de partilha.
- **“Hold and wait”** – processo que pode manter um recurso e esperar por outro recurso.
- **Impede apropriação** – o sistema não pode retirar recursos que está a ser usado num processo.
- **Espera de círculo fechado** – processos que estão á espera de recursos que estão a ser usado noutros processos.

Prevenção dos “deadlocks”

- Para prevenir a existência de “deadlocks” basta impedir as condições ditas a cima.

IPC – Problemas estáticos



- Jantar dos filosofos

```
#include "prototypes.h"

#define N 5                     /* number of philosophers */

void philosopher(int i)         /* i: which philosopher (0 to N-1) */
{
  while (TRUE) {
        think();                /* philosopher is thinking */
        take_fork(i);           /* take left fork */
        take_fork((i+1) % N);   /* take right fork; % is modulo operator */
        eat();                  /* yum-yum, spaghetti */
        put_fork(i);            /* put left fork back on the table */
        put_fork((i+1) % N);    /* put right fork back on the table */
  }
}
```

- **História do Jantar dos Filósofos** (só para perceber)

  Considere 5 filósofos que passam a vida a pensar e a comer. Partilham uma mesa redonda rodeada por 5 cadeiras sendo que cada uma das cadeiras pertence a um filósofo. No centro da mesa encontra-se uma taça de arroz e estão 5 garfos na mesa, um para cada filósofo.

  Quando um filósofo pensa não interage com os seus colegas. De tempos em tempos, cada filósofo fica com fome e tenta apanhar os dois garfos que estão mais próximos (os garfos que estão ou à esquerda ou à direita). O filósofo apenas pode apanhar um garfo de cada vez e como o leitor compreende, não pode apanhar um garfo se este estiver na mão do vizinho. Quando um filósofo esfomeado tiver 2 garfos ao mesmo tempo ele

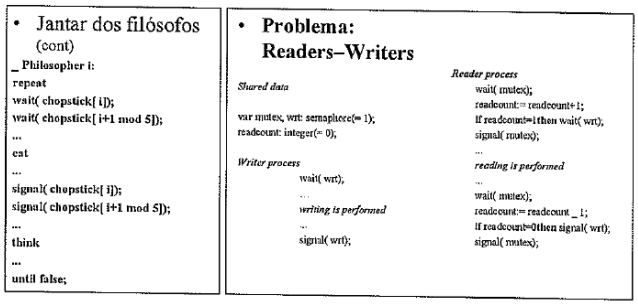
come sem largar os garfos. Apenas quando acaba de comer, o filósofo pousa os garfos, libertando-os e começa a pensar de novo. O nosso objetivo é ter uma representação/implementação que nos permita simular este jantar sem que haja problemas de deadlock ou starvation.

Para isso, o jantar será modelado usando uma thread para representar cada filósofo e usaremos semáforos para representar cada garfo. Quando um filósofo tenta agarrar um garfo executa uma operação wait no semáforo, quando o filósofo larga o garfo executa uma operação signal nesse mesmo semáforo. Cada filósofo (thread) vai seguir o algoritmo, ou seja, todos fazem as mesmas ações. Como deve estar já o leitor a pensar, o facto de seguirem o mesmo algoritmo pode dar azo à situação de deadlock, dai a utilização das primitivas de sincronização wait e signal. Uma outra possibilidade de deadlock seria o facto de mais do que um filósofo ficar com fome ao mesmo tempo, os filósofos famintos tentariam agarrar os garfos ao mesmo tempo. Isto é outro ponto que uma solução satisfatória terá que ter em atenção, devendo ser salvaguardado o facto de um filósofo não morrer à fome. Devemos recordar o leitor que uma solução livre de deadlock não elimina necessariamente a possibilidade de um filósofo morrer esfomeado.

Por todos estes motivos, o jantar dos filósofos é um algoritmo que deve ser implementado com algum cuidado por parte do programador.

Com o recurso à rede das redes, o leitor pode facilmente encontrar vários exemplos de implementação deste algoritmo em várias linguagens, contudo, também este artigo seguidamente apresenta um exemplo de uma implementação simples em C e em Java para que o leitor possa comparar os respetivos códigos.

IPC - Problemas clássicos



- Jantar dos filósofos (cont)

```
_ Philosopher i:
repeat
wait( chopstick[ i]);
wait( chopstick[ i+1 mod 5]);
...
eat
...
signal( chopstick[ i]);
signal( chopstick[ i+1 mod 5]);
...
think
...
until false;
```

- **Problema: Readers–Writers**

```
Shared data

var mutex, wrt: semaphore(= 1);
readcount: integer(= 0);

Writer process
        wait( wrt);
        ...
        writing is performed
        ...
        signal( wrt);

Reader process
        wait( mutex);
        readcount:= readcount+1;
        if readcount=1 then wait( wrt);
        signal( mutex);
        ...
        reading is performed
        ...
        wait( mutex);
        readcount:= readcount _ 1;
        if readcount=0 then signal( wrt);
        signal( mutex);
```

Capítulo 3 – Gestão de Memória

Os Mecanismos de Gestão de Memória determinam a organização da memória.

Os Algoritmos de Gestão de Memória determinam as decisões que devem ser tomadas.

A Memória é um importante recurso e por isso deve ser gerido (Gestão de Memória). Num sistema operativo há memória primária e memória secundária.

Espaço de Endereçamento de um processo -> Conjunto de células de memória que podem ser referenciadas por um processo.

- Um programa deve ser carregado para a memória para ser executado pelos processos
- Vários processos esperam no disco até serem carregados para a memória afim de serem executados.

Etapas de associação

Compilação -> Se se souber a priori a localização da memória onde irá residir o programa, pode gerar -se endereços absolutos.

Carregamento -> Há que gerar código recolocável caso não se saiba qual a localização da memória durante o processo de compilação.

Execução -> Associação de endereços do programa á memória faz se antes de o processo entrar em execução, podendo este ser movido de um segmento de memória para outro (H/W).

-----//-----

Carregamento Dinâmico – Só se carrega determinada rotina para a memória quando for chamada.

Otimização do espaço de memória (rotinas não usadas, nunca são carregadas).

Útil quando o código é extenso.

Implementa-se através da programação.

Linkagem Dinâmica – é feita através da execução.

Endereçamento Real

Endereços gerados pelo programa correspondem diretamente aos acedidos de memória.

- Dimensão dos programas limitados á memória física
- Programa só funciona nos endereços físicos para os quais foram escritos
- Dificulta a Multiprogramação – não é possível executar ao mesmo tempo dois programas preparados para correr no mesmo endereço

Sistemas Monoprogramados – Em cada momento só existe um programa, tendo disponíveis para si todos os recursos da máquina.

Overlays - Técnica de Sobreposição – Esta ultrapassa em parte o problema da dimensão da memória física. (Processo maior que a memória disponível). O programa é dividido por: parte residente e overlays.

- Vantagem – Simplicidade. O sistema operativo gere a memória física.
- Desvantagens – Deve ser o próprio programador a indicar quando deve ser carregado um novo overlay; Difícil de se aplicar a programas que não são modulares; A comutação entre overlays torna-se lenta; Não resolve os problemas da limitação física.

Proteção – Uma instrução, poderá referencia um endereço de memória fora do seu espaço de endereçamento interferindo com o espaço do outro. Em geral fica na parte inferior da memória.

Sistemas Multiprogramados – Interesse em ter vários processos de memória por forma a otimiza a utilização do CPU. O tamanho da memoria e o número de processos encontram-se relacionados e influenciam as performances do sistema.

- **Sistemas Multiprogramados com partições fixas:**

  Dimensão dos programas:
  Limitada pelo tamanho da maior partição.
  Utiliza as técnicas de overlays dentro das partições.

  Fragmentação Interna:
  Existe uma parte que nunca é usada devido á dimensão do programa não corresponder á partição.

Proteção:
Feita á custa de dois registos, onde são armazenados o endereço inferior e superior da partição em uso

- **Sistemas Multiprogramados com partições variáveis:**

Funcionamento:
Número e dimensão das partições varia dinamicamente.
Resolve o problema da fragmentação interna.
O número de partições cresce até esgotara memória.

Dimensão dos programas:
Limitada pela memória física, mas não é necessário reconfigurar as partições.

Fragmentação Externa:
A partição onde o programa se encontrava é libertada e associada ao bloco livre.
Caso exista um novo programa em espera de tamanho inferior ao bloco livre, usará parte desse bloco.
Ao fim de alguma tempo existem fragmentos espalhados pela memória -> solução -> **recompactação da memória.**

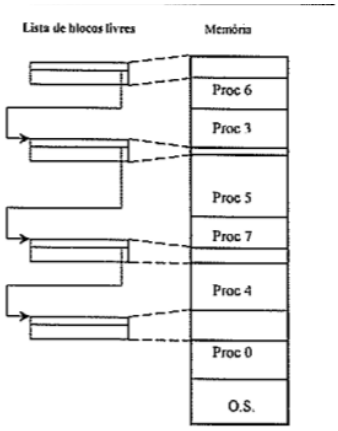
Proteção:
Métodos semelhantes aos aplicados nos sistemas de partição fixa.

A alocação da memória vai variando á medida que os processos vão chegando e saindo.

Swapping – Um processo pode ser temporariamente retirado de memória para o disco e posteriormente carregado para a memória para continuar a sua execução.

Alocação Dinâmica – Um processo pode requisitar mais memória do que a que está livre.

- Soluções:
  - Bloquear o processo até o espaço adjacente ficar livre (pode obrigar a tempos de espera prologados)
  - Descobrir um novo e maior espaço de memória (necessidade de ajustar os endereços)
  - Um sistema que suporta alocação dinâmica da memória (necessita de saber quais os blocos livres e seus tamanhos)



Algoritmos de alocação de memória:

- Algoritmos de alocação dinâmica de memória
  - tendo uma lista de buracos livres com diferentes tamanhos qual será seleccionado para satisfazer o pedido de um processo?

FIRST-FIT

Pesquisa a lista e devolve o primeiro bloco livre com dimensão suficiente para satisfazer o pedido

Vantagens

- Optimiza o tempo de alocação. Pesquisa mais rápida
- Não é necessário ordenar a lista por dimensões
- Um dos extremos da memória, em média, contém blocos suficientemente grandes para satisfazer qualquer pedido

Desvantagens

- Blocos tendem a ser colocados num dos extremos da memória
- Gera blocos livres num dos extremos da mémoria de difícil reaproveitamento.
- Gera maior fragmnetação

NEXT-FIT

Pequena variação do first-fit, na qual a pesquisa começa no ponto onde terminou a anterior(lista circular)

Vantagens

- Evita a pesquisa dos blocos pequenos do inicio lista

Desvantagens

- espalha fragmentos pela memória
- ligeiramnete pior que o first-fit

BEST-FIT

Pesquisa na lista e selecciona o bloco livre de dimensão mais aproxiamda da requerida. Mantém uma lista ordenada pela dimensão dos blocos livres

Vantagens

- Optimo quando encontra um bloco de dimensão igual à requerida

Desvantagens

- Acaba por derperdiçar mais memória ao criar fragmento muito pequenos de difícil reutilização
- Mais demorado (manter a lista ordenada sempre que surge um bloco livre)
- Pior que o first e next fit

WORST-FIT

Procura na lista o maior bloco livre, com a intenção de deixar livre um bloco de dimensão considerável. Mantém uma lista ordenada pela dimensão dos blocos livres

Vantagens

- O maior bloco é o primeiro da lista
- produz grandes blocos

Desvantagens

- esgota os blocos grandes rapidamente
- Mais demorado (manter a lista ordenada sempre que surge um bloco livre)

QUICK-FIT

Mantém listas separadas com os tamanhos mais requesitados

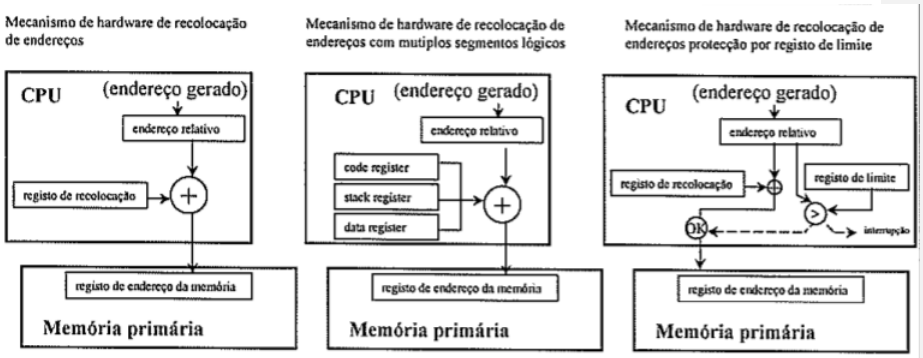
- é extremamnete rápido, mas tem a desvantagems dos algoritmos que requerem ordenamento da lista

O First-fit e best-fit é melhor que o worst-fit em termos de velocidades e alocação da memóra

Buddy System

- *Procura melhorar o desempenho (através de um mecanismo mais eficiente de junção de blocos separados) quando um processo termina ou è trasferido para o disco. Mantém uma lista de blocos de dimensão 1, 2,4,8,…, 2^n (1Mb – 21 entradas). Inicialmente toda a memória está livre e atribuída a um bloco. Ex: se fôr pedido um bloco de 70K , é colocado num bloco de 128K*

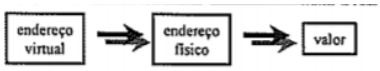
Recolocação da Memória



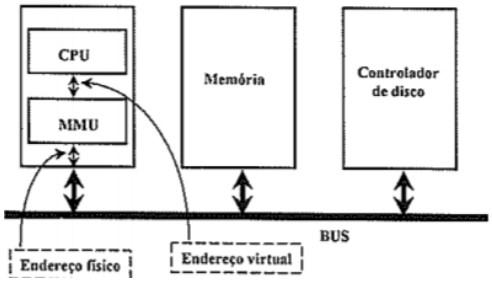
Endereçamento Virtual

Endereços gerados pelo programa são convertidos, em tempo de execução, em endereços físicos.

- Cria um espaço de endereçamento independentemente e maior que a memória física disponível.
- Acaba com a limitação do tamanho dos programas.
- Dissocia os endereços gerados pelos programas.
- Mecanismo de Endereçamento:



Unidade de gestão de memória (MMU) – unidade que efetua a tradução dos endereços virtuais em físicos.



Um endereço virtual possui:

- [bloco, descolamento]
- Bloco : identifica o bloco.
- Deslocamento : identifica o deslocamento dentro do bloco.

O bloco é usado para indexar a tabela de traduções, obtendo o endereço físico do bloco que somado ao deslocamento indica o endereço físico.

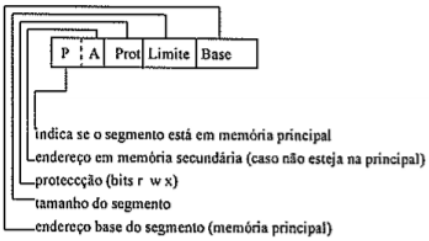
- Blocos de tamanho variável -> **Segmentação**
- Blocos de tamanho igual e fixo -> **Paginação**

Segmentação

Divisão funcional dos programas em segmentos lógicos.

O espaço de endereçamento virtual é dividido em segmentos

Endereço virtual: [segmento, deslocamento]



Existem dois registos da unidade de gestão: BTS (base da tabela de segmentos) e LTS (limite da tabela de segmentos).

- Fases da tradução de endereços
  - verifica se o num. de segmento é inferior a LTS
  - o num. de segmento é adicionado ao registo BTS, referenciando-se a tabela de segmentos
  - verifica se o segmento está em memória (senão gera um *segmentation fault*)
  - testa os bits de protecção
  - verifica se o deslocamento pretendido não é superior à dimensão pretendida
  - adiciona o deslocamento ao endereço base obtido

Tamanho do segmento pode variar. Dimensão máxima depende da arquitetura da máquina.

Partilha de memória entre processos é simplificada.

Otimização do mecanismo de tradução para evitar a duplicação de acessos á memória, são guardados em registos o descritor do segmento em uso.

Fragmentação externa – para evitar pode-se copiar todos os segmentos para um dos extremos da memória. Quando um segmento cresce, copia-se para a memória secundária, liberta-se espaço no disco, aloca um novo segmento com a dimensão pretendia e copia-se de novo para a memória primária.

Proteção feita a 3 níveis:

I. Processos diferentes têm tabelas de segmentos diferentes
II. O tipo de acesso ao segmento é verificado

III. O deslocamento dentro de um segmento é verificado

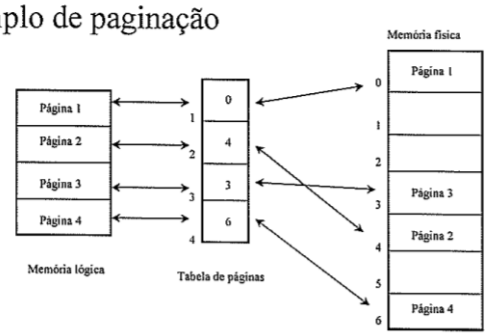
Paginação

Cria um espaço de endereçamento maior que a dimensão da memória física.

O espaço logico de endereçamento não é continuo.

Mecanismo de paginação:

- Exemplo de paginação



Otimização dos mecanismos de traduções – para evitar um acesso suplementar á memória os descritores das últimas n páginas acedidas são guardadas em memória associativa na tabela de traduções. A pesquisa TLB é feita em paralelo com acesso á tabela de páginas.

Faltas de Páginas – se a página não se encontra em memória primária é gerada uma exceção que interrompe a instrução a meio, carrega a página para a memória e recomeça a instrução.

Partilha de memória entre processos – basta que os descritores das tabelas de páginas de dois ou mais processos apontem para o mesmo endereço físico.

Fragmentação interna – o bloco lógico do programa corresponde a um conjunto de páginas, resultando fragmentação interna na última página.

Dimensão das páginas:

- **Pequenas –** diminuem a fragmentação, aumenta o número de falta de páginas, aumenta a dimensão das tabelas de páginas e das respetivas listas
- **Grandes –** aumentam a fragmentação, diminuía a dimensão das tabelas de páginas (pesquisas mais rápidas).

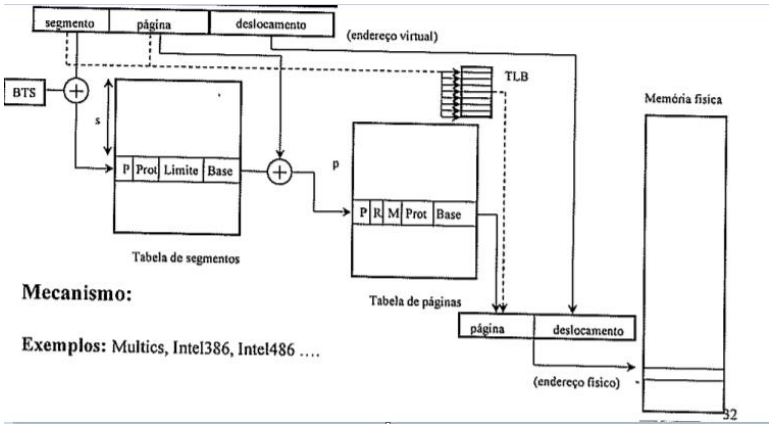
Proteção – Semelhante á usada na segmentação – para proteger um bloco logico de um programa é necessário proteger várias páginas **(Desvantagem face a segmentação) .**

Híbridos

Os mecanismos de segmentação traduzem melhor a funcionalidade logica dos programas, no entanto a sua implementação é complexa.

Os mecanismos de paginação são mais simples de implementar que os de segmentação sendo também mais rápidos ----- **solução ----- Sistemas Híbridos**

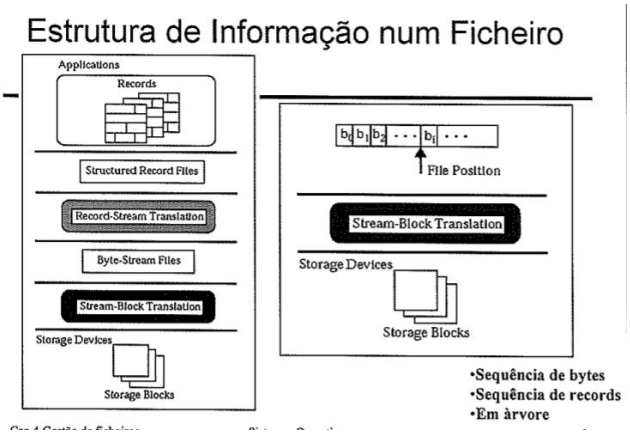
Estes têm como objetivo tirar partido das vantagens dos dois mecanismos, por um lado a eficiência da paginação e por a outro a estrutura logica seguida pela segmentação.



Capítulo 4 – Gestão de Ficheiros

Ficheiros são abstrações que representam dispositivos de armazenamento secundário (tapes, drives) e no caso do S.O. servir como suporto para realizar a memória virtual. Cada ficheiro apresenta uma coleção de dados armazenados num dispositivo.

O gestor de ficheiros é o modulo de software do S.O. que fornece a capacidade ao programador de manipular o ficheiro e o seu conteúdo.



Estrutura do Descritor

File Descriptor Entries

- **External name.**
- **Current state.**
- **Sharable.**
- **Owner..**
- **User.**
- **Locks.**
- **Protection settings.**
- **Length.**
- **Time of creation.**
- **Time of last modification.**
- **Time of last access.**
- **Reference count.**
- **Storage device details..**

Estrutura do Descritor UNIX

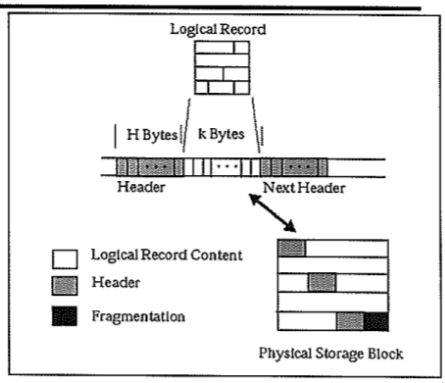
Field Description

- *Mode* - Especificação das permissões de acesso do dono e dos outros utilizadores.
- *UID* - ID do utilizador que criou o ficheiro.
- *Group ID* - ID associado com o ficheiro para identificar uma colecção de utilizadores com direitos de acesso ao ficheiro.
- *Length in bytes* - Número de bytes no ficheiro.
- *Length in blocks* - Número de blocos utilizados para realizar o ficheiro.
- *Last modification* - Altura em que o ficheiro foi escrito pela última vez.
- *Last access* - Altura em que o ficheiro foi lido pela última vez.
- *Reference count* - Número de directorias em que o ficheiro aparece. Este campo é utilizado para detectar quando é que o ficheiro é apagado e o espaço de armazenamento pode ser recuperado.
- *Block references* - Ponteiros e ponteiros indirectos para blocos no ficheiro.

Tradução "block-stream"

Operações típicas num ficheiro sequência de bytes (byte stream)

- open (fileName).
- close(fileID)..
- read(fileID, buffer, length).
- write(fileID, buffer, length).
- seek(fileID, filePosition).



Ficheiros Estruturados – vocacionados para records.

- Estrutura sequencial de records:
  - open(filename); close(fileID);getRecord(fileID,record); putRecord(fileID,record); seek(fileID,position);
- Estrutura sequencia de records com index:
  - getRecord(fileID,index); putRecord(fileID,record); delete(fileID,index);

Exemplos de "mailboxes para e-mail"

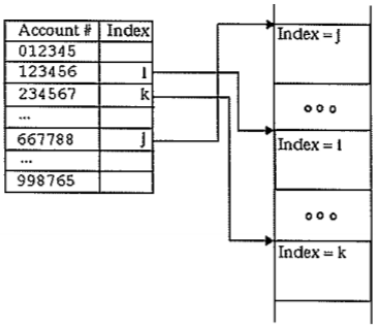
Uma declaração define a estrutura.

Existem as operações sobre a estrutura.

```
struct message {
      /* The mail message */
      address to;
      address from;
      line subject;
      address cc;
      string body;
};
struct message *getRecord(void) {
      struct message *msg;
      msg = allocate(sizeof(message));
      msg->to = getAddress(...);
      msg->from = getAddress(...);
      msg->cc = getAddress(...);
      msg->subject = getLine();
      msg->body = getString();
      return(msg);
}
putRecord(struct message *msg) {
      putAddress(msg->to);
      putAddress(msg->from);
      putAddress(msg->cc);
      putLine(msg->subject);
      putString(msg->body);
}
```

Uso de uma "Index table"

Existem aplicações em que um dado item pode ser procurado sem ser necessário pesquisar todo o ficheiro.



Sistemas de Base de Dados

A definição da estrutura dos dados e das linguagens que permitem a sua manipulação não fazem parte do S.O.

Realização dos Ficheiros

Necessita de um suporte para a tradução da sequência de bytes no bloco lógico para o registo do bloco no dispositivo físico.

Existem vários métodos de gerir a atribuição de blocos a um dado ficheiro:

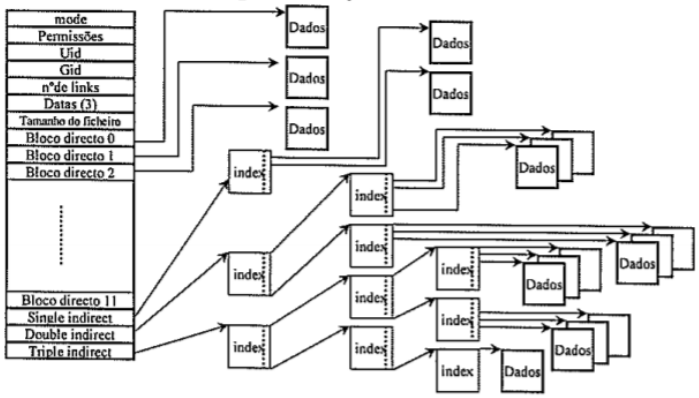
- Alocação de blocos contíguos
  Vantagem: implementação simples, excelente performance;
  Desvantagem: é necessário conhecer o tamanho do ficheiro; desperdiça espaço ;

- Lista Ligada
  Vantagem: todos os blocos podem ser usados;
  Desvantagem: acesso aleatório muito lento;

- Lista Ligada com index – lista ligada mantida usando tabela de memoria
  Vantagem: resolve as desvantagens do método anterior porque não são precisos acessos ao disco para chegar ao fim do ficheiro;
  Desvantagem: tabela acaba por ocupar espaço em memória;

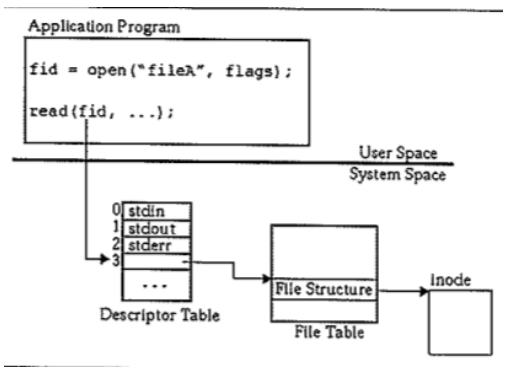
- I-nodes – associa-se a cada ficheiro um pequena tabela.

- UNIX i-node (4kbytes por bloco)



Operações em ficheiros

1. Localiza o descritor de ficheiro.
2. Obtém informação relativa ao ficheiro no descritor.
3. Verifica autorização de acesso ao ficheiro.
4. Cria entrada na tabela de estados de ficheiros.
5. Afeta os recursos necessários para manter a utilização do ficheiro.



Diretorias

Estrutura de dados que regista a existência de ficheiros e subdiretorias.

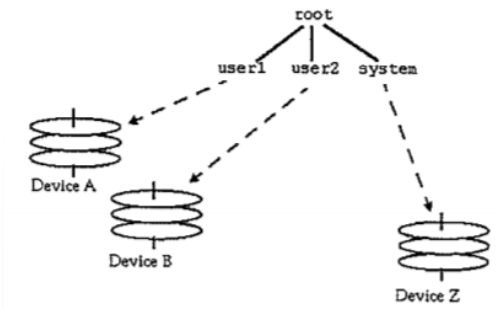
Tem como função, mapear o nome do ficheiro. Caracteres ASCII com a informação necessário para localizar os dados .

Existem diferentes estruturas.

Exemplos: Diretorias DOS/Windows – Associa a cada dispositivo físico de armazenamento um dispositivo logico de armazenamento.

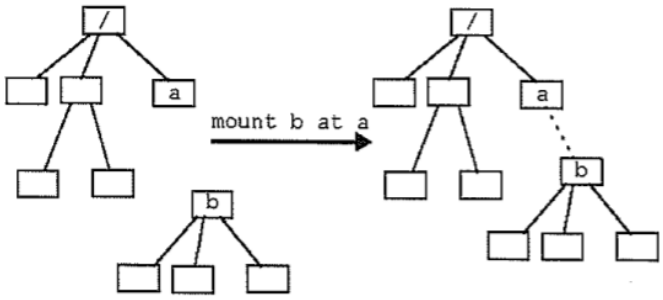
Realização de Diretorias

As diretorias têm estrutura de garfo. É possível ter dispositivos físicos afetados em diferentes ficheiros ou subdiretorias.



"Mount" – Montagem de ficheiros

Consiste numa operação sobre a estrutura de dados.



Capítulo 5 – Proteção e Segurança

Políticas e Mecanismos de Segurança

- **Política de segurança** – define regras para conceber autorização de acesso ao computador ou recurso informático. São selecionadas por forma a realizarem a segurança de um sistema de responsabilidade dos administradores.
- **Mecanismos de segurança** – ferramentas que permitem realizar políticas de segurança. Computadores do mesmo tipo podem aplicar diferentes políticas. São realizados pelo sistema operativo quando é necessário garantir a sua realização com determinado comportamento.

Kerberos – autenticação de rede insegura

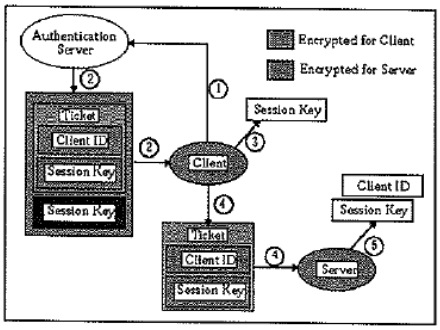
- É um conjunto de protocolos de rede que pode ser usado para autenticar o acesso a um computador por um utilizador num computador diferente e usando uma rede insegura.
- Assume que a informação circulando na rede pode ser corrompida durante a transmissão.

- Assume que as duas máquinas (computadores) não têm sistemas operativos necessariamente seguros.
- Assume também que o computador (cliente) deseja obter serviços de outro computador (servidor) usando rede insegura.

Sistemas Kerberos

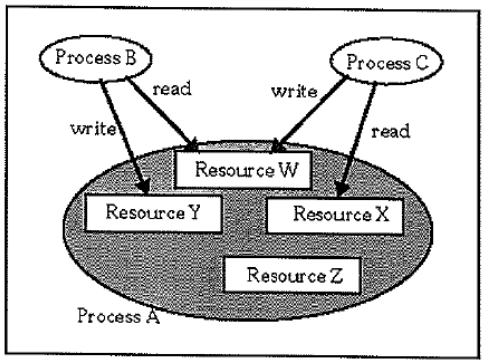
Operação

•1º Pede credenciais.
•2º (Ticket + Session Key) com chaves de descodificação diferentes.
•3º O cliente fica com uma chave da sessão (só descodificável por ele).
•4º O cliente envia o ticket (só descodificável pelo servidor).
•5º O servidor descodifica a identificação do cliente e a chave da sessão.
•O server de autenticação é confiável neste protocolo porque: tem uma copia do ID do cliente, sabe como codificar para o cliente, sabe como codificar para o servidor.



Autenticação interna (para partilha controlada de recursos)

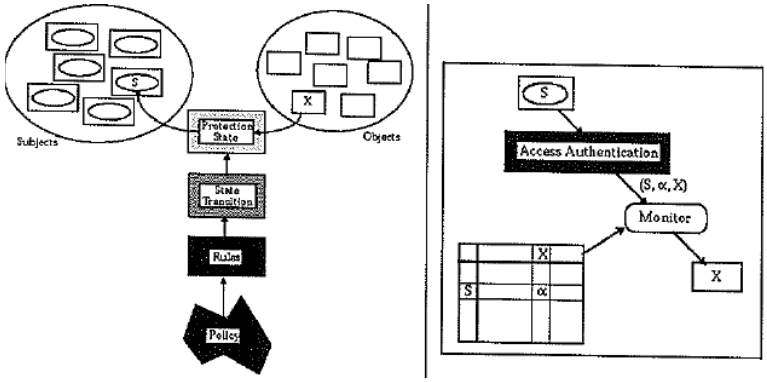
•A autorização interna é parte da tarefa de gestão da partilha de recursos, sendo o objectivo proteger os recursos dos de acções executadas por outros processos



Modelo básico de proteção de recursos

- A atividade num computador operando em tempo partilhado é constituído por **partes ativas** (processos e threads) e **partes passivas** ou objetos (recursos).
- Os processos têm diferentes direitos sobre um objeto em diferentes tempos e depende da tarefa a ser executada.
- Chama-se "**subject**" um processo executado num espaço de proteção.
- Um sistema de proteção é composto por: objetos, "subjects" e regras que especificam a política de proteção.

Sistemas de proteção – matrizes de acesso

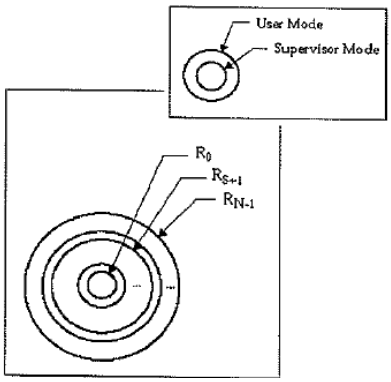


Sistemas de proteção por níveis

Sistema por Níveis

• O sistema por níveis - maior autorização para aneis mais concêntricos.

•Supervisor Mode/ User Mode (2 níveis).



Proteção de memoria