

# Sistemas Operativos

LEI

2023/2024

fsm@di.uminho.pt

Sistemas Operativos - 2023/2024

.0

## Numa palavra...

Se fosse necessário dizer de que trata esta UC

- Numa só palavra: **CONCORRÊNCIA**
- Em mais do que uma:
  - Concorrência, concorrência...
  - Eficiência, rapidez, segurança, etc.
  - Boa gestão de recursos perante determinada carga
  - Conhecimento do funcionamento de sistemas (Apps, SO, HW)
  - Aplicações concorrentes, naturalmente!

Sistemas Operativos - 2023/2024

2

## Objectivo

- **Ajudar a perceber como funcionam os computadores**
  - Em termos físicos, o que é uma aplicação informática?
  - Que *recursos* necessita?
  - Como devem ser geridos esses recursos?
  - Como interage esta aplicação com outras?
  - Isto está lento. Que devo fazer?
  - O sistema bloqueou! Perdi tudo?
  - E muito mais...

Sistemas Operativos - 2023/2024

1

.1

## Programa

- Recapitulação de conceitos de **programação de sistemas**
- Gestão de processos, memória, ficheiros, periféricos
- Alguma **programação concorrente** (de baixo nível)
- E mãos na massa:
  - Aulas práticas em ambiente Linux
    - (quase) nada de janelas, nem ratos...
    - terminal com *Bash*, comandos, pipelines...
    - sem internet : basta consultar o manual
  - Programação de “baixo nível”: C, syscalls, libs ...



Sistemas Operativos - 2023/2024

3

.2

.3

.1

**GSD**  
Grupo de Sistemas Distribuídos

Ambiente de trabalho: bash + ...

Sistemas Operativos - 2023/2024

4

.4

**GSD**  
Grupo de Sistemas Distribuídos

Bibliografia recomendada

- Silberschatz, Galvin and Gagne, *Operating System Concepts*, 10th Ed, John Wiley & Sons, 2021.
- [Operating Systems: Three Easy Pieces](#), Remzi & Andrea Arpaci-Dusseau, 2018.
- **FSM 2004, Vou Fazer Exame de Sistemas Operativos!**

Sistemas Operativos - 2023/2024

5

.5

**GSD**  
Grupo de Sistemas Distribuídos

Bibliografia Adicional

- Man !!!!!!!!!
- R. Stevens, *Advanced Programming in the Unix Environment*, Addison Wesley, 1990.
- Beej Guides
- Google, Youtube, slashdot...

Sistemas Operativos - 2023/2024

6

.6

**GSD**  
Grupo de Sistemas Distribuídos

Slides

- Baseados nos slides originais dos livros recomendados (em especial Silberschatz), caso seja necessário identificar o respectivo capítulo
- Disponíveis no Blackboard, mas em permanente revisão
- Servem apenas de “âncora” ao estudo
 

Não chegam para responder aos testes!

Faltam os porquês, e.g. porquê usar “isto” com esta “carga”?

Sistemas Operativos - 2023/2024

7

.2

## Aulas práticas

- Cada aula prática tem um “guião” muito detalhado.
  - Normalmente só duram uma semana
  - Fazem sempre falta para a aula seguinte (+ trabalho práctico) →
  - Sempre que resolver uma alínea, deve parar e perguntar:  
*O que é que EU aprendi com este exercício?*
- Recomenda-se:
  - estudar o guião antes da respectiva aula
  - usar a aula para tirar dúvidas e não para colecionar resoluções; tente entender o raciocínio para lá chegar.
  - terminar em casa todas as alíneas não resolvidas durante a aula; se necessário, peça ajuda por mail



.8

## Avaliação

- Trabalho práctico em grupo de 3, + um teste
  - Não há nota mínima no TP 
  - Não se “descongela” nota TP do ano anterior, salvo excepções indicadas no RAUM
  - Nota **mínima** de 8 no teste ou exame(s)

$$\text{Classificação} = (\text{TP} + \text{teste\_ou\_exame}) / 2$$

.9

## Avaliação – datas importantes

- ???: Inscrição nos grupos TP do Blackboard
- 7 de Maio: Submissão do TP pelo Blackboard
- 27 a 29 de Maio: Defesa do TP
- 24 de Maio -- Teste
- 14 de Junho -- Exame de recurso

.10

## Avaliação

- Prova escrita (teste ou exame) individual e sem consulta.
- É preciso responder **ao problema proposto;**
  - Valoriza-se a capacidade de raciocínio e a concepção de algoritmos (por oposição à utilização de “padrões” de soluções)
  - Quase tudo tem a ver com concorrência
  - As perguntas da parte teórica insistem sempre nos “porquês”, na justificação, demonstração ou prova.

.11

## Programa

- Introdução (à *programação de sistemas*)
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos
- Gestão de ficheiros

Sistemas Operativos - 2023/2024

12

.12

Vamos começar pelo princípio...

## Para que serve um computador?

- Para executar programas (aplicações)
- Que facilitam a vida aos utilizadores

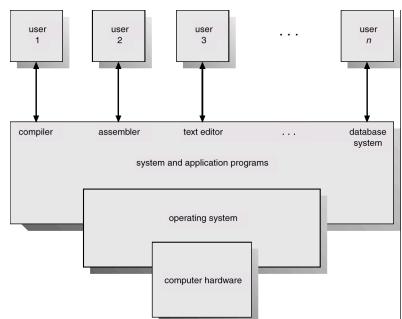
Sistemas Operativos - 2023/2024

13

.13

## O que é um Sistema Operativo?

- Programa que actua como **intermediário** entre os utilizadores e o hardware



Sistemas Operativos - 2023/2024

14

.14

Portanto...

- O SO deve gerir o hardware e colocá-lo à disposição dos programas e utilizadores, de uma forma
  - conveniente,
  - protegida,
  - eficiente,
  - justa,
  - ...

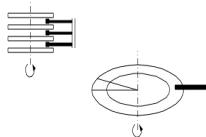
Sistemas Operativos - 2023/2024

15

.15

## O Sistema Operativo pode ser visto como...

- Extensão da máquina
  - simula uma *máquina virtual* "acima" da máquina real: **open(), read(), write()**...



Sistemas Operativos - 2023/2024

16

.16

## Objectivos (1)

- Conveniência
  - SO esconde os detalhes do hardware
    - e.g. [dimensão e organização da memória](#)
  - Simula máquina virtual com valor acrescentado
    - e.g. [cada processo executa numa "máquina"](#)
  - Fornece API mais fácil de usar do que o hardware
    - e.g. [ficheiros vs. blocos em disco](#)

Sistemas Operativos - 2023/2024

17

.17

## Na prática...

- É o Sistema Operativo quem define a "**personalidade**" de um computador
- Como se comporta o mesmo computador (hardware) após ter arrancado
  - MSDOS?
  - Windows 95?
  - Windows 10?
  - Linux (Ubuntu, Kali...)?



Sistemas Operativos - 2023/2024

18

.18

## Objectivos (2)

- Eficiência
  - SO controla a alocação de recursos
    - Se 3 programas usarem a impressora ao mesmo tempo → saí lixo?
    - Programa em ciclo infinito → [computador bloqueia?](#)
    - Processo corrompe a memória dos outros → [programas morrem?](#)
  - Multiplexação:
    - Tempo: cada processo usa o recurso à vez (impressora, CPU)
    - Espaço: recurso é partilhado simultaneamente por vários processos (memória central, disco)

Sistemas Operativos - 2023/2024

19

.19

## Objectivos (3)

- Recapitulemos então os objectivos gerais de um SO
  - Conveniência
  - Eficiência
  
- Então, os nossos critérios de avaliação serão...
  -  Dá jeito?
  -  É eficiente ou aumenta a eficiência geral do sistema?
  -  Nem uma nem outra?

Sistemas Operativos - 2023/2024

20

.20

## Tome nota:

- Este “filme” não é para decorar...
- É para perceber a evolução e os porquês
- Quando terminar, terá ficado a saber
  - os objectivos dos Sistemas Operativos
  - como estes foram sendo atingidos:
    - com muita massa cinzenta
    - e algum apoio do hardware!

Sistemas Operativos - 2023/2024

21

.21

## No início era assim...

- Acesso livre ao computador
  - Utilizador podia fazer tudo, mas
  - Também tinha de fazer tudo...
  
- Eficiência era baixa
  - Elevado tempo de preparação
  - Tempo “desperdiçado” com debug

Sistemas Operativos - 2023/2024

22

.22

## No início era assim...

Exemplo: HP 2114B (1968)



Sistemas Operativos - 2023/2024

23

.23

Sim, TUDO!!!!

## Altair 8800 (intel 8080)



Sistemas Operativos - 2023/2024

24

.24

## Acesso livre

- Utilizador é um faz-tudo (I):
  - Percebe o problema e idealiza a solução (algoritmo + dados)
  - Descreve o algoritmo em “alguma notação de alto nível”
  - Estuda o manual do CPU e memória, faz então a tradução para linguagem máquina (e decide que endereços vai usar)



```

1: S = 0
2: Ler N
3: Se N > 999 continuar no passo 6
4: S = S + N
5: Voltar ao passo 2
6: Mostrar S
    
```

```

    _malloc
    _calloc
    leaq    L_.str(%rip), %rdi    8945  83f3  03ff  0c25  0d48  0073
    movl   $4294967295, %rcx    0002  0000  01ac  0000  0005  000f
    xorl   %rdx, %rdi          0048  105b  00eb  f1f3  7502  4079
    %eax, %edi                5d89  4cd9  0312  0000  bff4  0000
    $0, -4(%rbp)              05400 40000 0000  07eb  ff48  48c3  5d89  0fd0  3bb6
    _malloc
    _calloc
    leaq    L_.str(%rip), %rdi    06420 84000 0000  07eb  ff48  48c3  5d89  0fd0  3bb6
    movl   $4294967295, %rcx    06420 84000 0000  410a  448b  3cbe  2144  ebfb  be0a
    xorl   %rdx, %rdi          06440 40000 0000  55e8  0001  8500  75c0  0fdf9  03b6
    
```

Sistemas Operativos - 2023/2024

25

.25

- A tradução manual é particularmente penosa e facilmente sujeita a erros
- É necessário construir a instrução a partir do “OP code”, flags, registos, endereços, modos de endereçamento...
- [8080 Instruction Set](#)

Inst	Encoding	Flags	Description
MOV D,S	01DDDS	-	Move register to register
MVI D,#	00DDDI0 db	-	Move immediate to register
LXI RP,#	00RP0001 lb hb	-	Load register pair immediate
LDA a	00111010 lb hb	-	Load A from memory
STA a	00110010 lb hb	-	Store A to memory
LHLD a	00101010 lb hb	-	Load H:L from memory
SHLD a	00100010 lb hb	-	Store H:L to memory
LDAX RP	00RP1010 *1	-	Load indirect through BC or DE
STAX RP	00RP0010 *1	-	Store indirect through BC or DE
XCHG	11101011	-	Exchange DE and HL content
ADD S	10000SS	ZSPCA	Add register to A
AND S	11000110 db	RSBDA	Add immediate to A

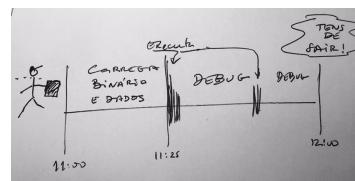
Sistemas Operativos - 2023/2024

26

.26

## Acesso livre

- Utilizador é um faz-tudo (II):
  - Chegada a hora, carrega manualmente o programa e dados
  - Executa o programa
  - Se não está correcto, tem de descobrir sozinho os erros
  - Eficiência é muito baixa devido ao desperdício de tempo de CPU



Sistemas Operativos - 2023/2024

27

.27

- Carregamento manual demorado
- Debug muito demorado...
  - Erro no algoritmo?
  - Ao traduzir para assembly?
  - Ao traduzir para binário?
  - Ao carregar programa e dados?

## Para aumentar a eficiência (I)

- Surgiu a ideia de reduzir a intervenção humana, fazendo a preparação de programas e dados *offline* e acelerando o seu carregamento posterior.
- Inventam-se periféricos de *input* e *output*, como a célebre Teletype, embora a velocidade ainda seja muito baixa (10 chars/s) para ler da fita ou escrever na fita/papel
- E aparece um **loader\*** residente

(\* ) Software que entende o que está na fita e carrega para o local pretendido da RAM os bytes que vai lendo



When apps were cards or tape...



## Card reader + line printer



## Para aumentar a eficiência (II)

- Automatizou-se uma parte do “procedimento”
  - Utilizador deixa de interagir com o seu programa, usa fita perfurada ou coloca os cartões num cesto e espera... horas
  - Operadores recolhem o cesto periodicamente e colocam programas e dados no leitor. O sistema executa os jobs e imprime os resultados, que são devolvidos a determinadas horas
- **Ganhou-se em eficiência, perdeu-se em conveniência**
  - Um job que antes demorava uma hora é agora executado em segundos
  - *Turnaround* time de horas: entrega às 9, recebe às 19

## Melhor do que um operador...

- É ter um *programa* que:
  - Automatize a operação do computador, passando ao job seguinte sempre que o job que detém o cpu chega ao fim
  - Operador apenas carrega / descarrega cartões e junta as listagens aos respectivos cartões, para os utilizadores verem os resultados
- Será o início de uma Job Control Language (JCL) e de um interpretador de comandos ?

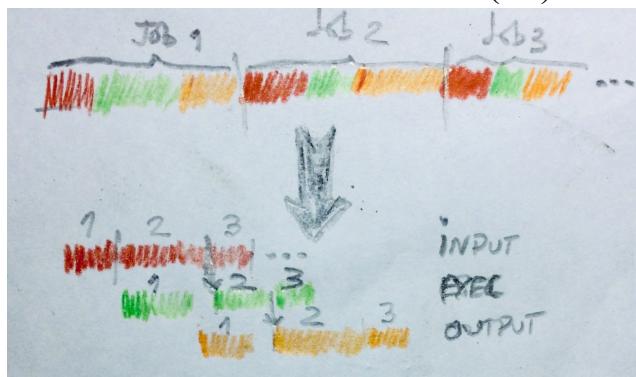
**Embrião de um sistema operativo?**

Sistemas Operativos - 2023/2024

32

.32

## Para aumentar a eficiência (IV)



Sistemas Operativos - 2023/2024

34

.34

## Para aumentar a eficiência (III) ...

Apesar de cada job “curto” demorar (dezenas de) segundos, o CPU está ocupado todo esse tempo, ora a executar o código ora em espera activa de IO.

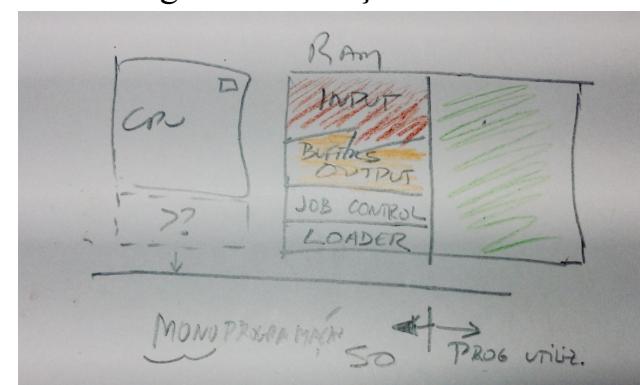
E se... input, execução e output pudessem ser **realizados em paralelo**?

Sistemas Operativos - 2023/2024

33

.33

## Como gerir a execução concorrente?



Sistemas Operativos - 2023/2024

35

.35

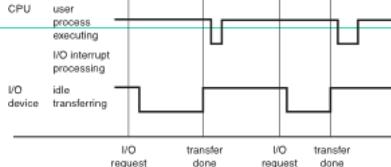
Mas havia o risco de...

- Perder eficiência devido a erros de programação
  - Ciclos infinitos
  - Espera activa por periféricos lentos
  - Erros na leitura ou escrita de periféricos
  - Programa do utilizador destruir o “programa de controle”

.36

Soluções (hardware)

- Interrupções



- Relógio de Tempo Virtual

- Instruções privilegiadas, 2 ou mais *modos de execução*, protecção de memória
- E aparecem as **system calls** !

.37

Exemplo: Polling IO

- Disk\_IO()
  - Carrega o controlador de disco com parâmetros adequados (pista, sector, endereço de memória, direcção...)
  - While (NOT IO\_done); /\* do nothing in a **busy** way\*/

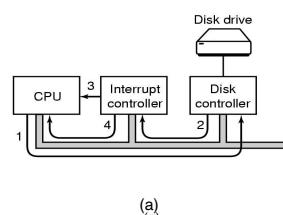
(Equivalente a:  
 Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste?  
 Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste?  
 Já acabaste? Já acabaste? Já acabaste? Já acabaste? Já acabaste?  
 ... )

- OK, regressa de disk\_io()

Resulta em **desperdício de tempo de CPU**

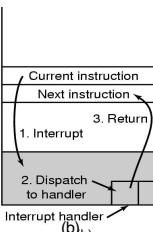
.38

Exemplo: Interrupt-driven IO



(a) OS inicia operação de IO e prepara-se para receber a interrupção; entretanto pode executar outras tarefas.

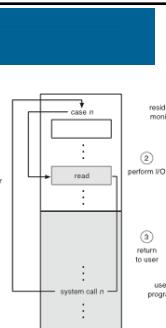
(b) No fim da operação de IO, o programa em execução é interrompido momentaneamente, SO trata o evento, e decide se o processo que pediu IO retorna à execução, se mantém o que foi interrompido ou até se muda para outro pronto a executar.



.39

## Soluções (software)

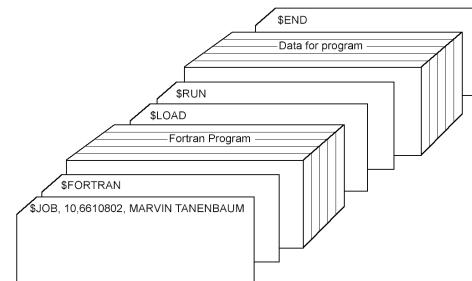
- Chamadas ao Sistema
- Virtualização de periféricos
  - Por exemplo o Leitor de cartões:
    - Programa pede para ler do periférico
    - SO devolve o conteúdo de um cartão que foi copiado para banda magnética ou lido anteriormente para memória (SPOOL)
- Mais tarde vai surgir a **multiprogramação**



Sistemas Operativos - 2023/2024

40

## Exemplo de um “job”

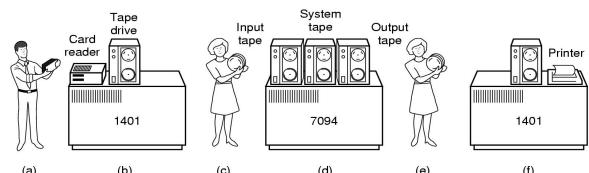


Sistemas Operativos - 2023/2024

41

.40

## Primeiros sistemas de batch



### Processador auxiliar faz IO de periféricos lentos (virtuais)

- Carregar cartões no 1401, que os copia para banda magnética
- Colocar banda no 7094 e executar os programas
- Recolher banda com resultados do batch e colocá-la no 1401, que os envia para impressora
- (Leitor e impressora seriam mais rápidos do que a imagem sugere... Ver slide 30)

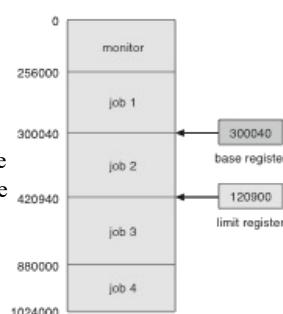
Sistemas Operativos - 2023/2024

42

.42

## Multiprogramação

- Memória central dividida em várias zonas (partições)
- Vários jobs simultaneamente em memória mas não “vêem” os outros
- Isto permite executar concorrentemente vários processos, repartindo o tempo de cpu entre eles



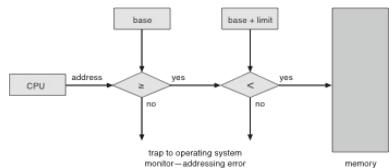
Já não obriga a encadear input, execução e IO:

Sistemas Operativos - 2023/2024

43

.43

## Protecção de memória

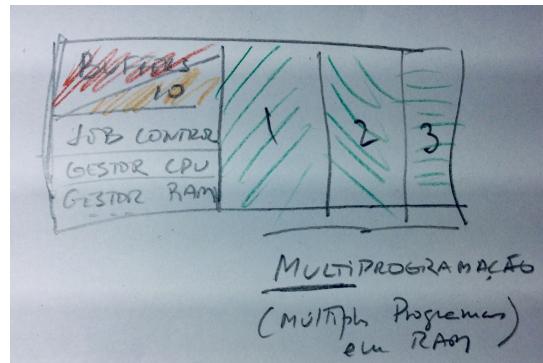


Note que estes testes têm de ser feitos sempre que há um acesso à memória...

2, 3 ou mesmo 4 vezes por instrução?

.44

## Multiprogramação



.45

## E a conveniência?



- Reaparece com os sistemas de **Time-Sharing**
- Terminais (consolas) ligados ao computador central permitem que os utilizadores voltem a interagir directamente
- Sistema Operativo reparte o tempo de CPU pelos programas prontos a executar (de preferência carregados em memória)

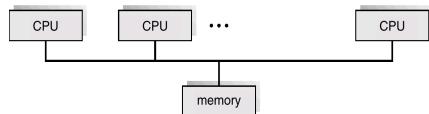
.46

## E desde aí?

- Com o computador pessoal volta tudo ao início...
  - Control Program for Microcomputers
  - Monoprogramação, baixa eficiência...
- Mas...
  - É muito conveniente para o utilizador
  - É barato, logo eficiência não é a prioridade

.47

## Multiprocessamento (1)



- A ideia é executar mais carga no mesmo intervalo de tempo (i.e. ter **throughput** maior). Não é executar um programa muito mais depressa (i.e. baixar o seu **tempo de resposta**). Para isso é preciso dividir uma aplicação em vários processos e executar cada um em seu CPU/core.
- No entanto, consegue reduzir o tempo de espera até ter acesso a um CPU

Sistemas Operativos - 2023/2024

48

.48

## Multiprocessamento (2)

- Arquitectura
  - Simétrico
    - qualquer CPU pode executar código do SO
    - cuidado com *race conditions*, (e.g. acesso à tabela de blocos de memória livres)
    - hardware mais sofisticado (e.g disco interrompe todos os CPUs?)
  - Assimétrico
    - Periféricos associados a um só CPU, o que executa o SO
    - CPUs podem estar parados porque o SO não “despacha”

Sistemas Operativos - 2023/2024

49

.49

## Sistemas Distribuídos (1)

- Nos anos 80 apareceram as redes locais para partilha de
  - recursos caros (e.g. impressoras) ou
  - inconvenientes de replicar (e.g. sistemas de ficheiros)
  - redirecionamento de IO
- Exemplo: cat fich.txt | rsh print\_server lpr
- Questões
  - protocolos de comunicação, modelo cliente-servidor?
  - como saber o estado de recursos remotos?

Sistemas Operativos - 2023/2024

50

.50

## Sistemas Distribuídos (2)

- Em breve se passou
  - dos *network aware OSs*, que já permitiam acesso remoto a discos, sistemas de ficheiros, impressoras...
  - para sistemas vocacionados para o trabalho em rede
- E chegou-se à Web...



Sistemas Operativos - 2023/2024

51

.51

E ainda...

- SOs para *mainframes*:
  - IBM MVS, IBM VM/CMS.
  - desenvolvidos nos anos 60 e ainda em operação (z/VM)!
- A caminho dos 60 anos, a virtualização de ambientes de execução mantém a sua importância: Vmware, Docker...

Sistemas Operativos - 2023/2024

52

.52

E ainda...

(apesar de não fazerem parte do programa)

- SO de Tempo Real
  - controlo de processos industriais, sistemas de vôo, automóveis, máquinas de lavar, etc.
  - SO normais não conseguem dar **garantias** de tempo de resposta.
- SOs para sistemas “restritos”:
  - microcontroladores, redes de sensores

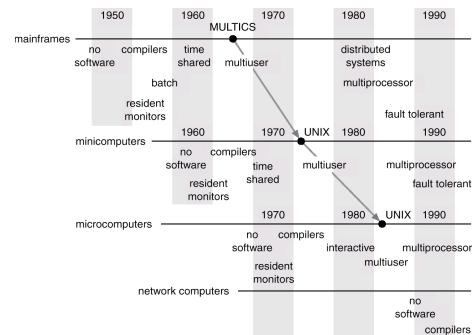


Sistemas Operativos - 2023/2024

53

.53

## Evolução de conceitos de SO



Sistemas Operativos - 2023/2024

54

.54

Antes de continuar...



Assegure-se que percebeu os conceitos anteriores, e que entendeu os problemas que as soluções indicadas procuram resolver...

- Sabe o que são e para que servem os 2 modos de execução?
- Modo de execução é **hardware** ou **software**?
- E multiprogramação? E multiprocessamento?
- E interrupção? Para que servem as interrupções?
- Para que servem as system calls? Qual a diferença em relação a uma função normal?
- O que é o tempo virtual?

Sistemas Operativos - 2023/2024

55

.55

## Recorde ainda....

- CPU
  - Registros (PC, SP, BP, CS, DS...) → “contexto volátil”
  - Instruções privilegiadas → só podem ser executadas em modo “protegido”; a forma de um programa do utilizador solicitar serviços ao SO é através das chamadas ao sistema (syscalls)
  - Interrupções (já agora, recordemos traps e exceções!)
- Memória (mas o que é um endereço? E modos de endereçamento?)
- Periféricos + formas de dialogar com eles

Sistemas Operativos - 2023/2024

56

.56

## Programas versus processos

- Programa executável:
  - Resultado da compilação, ligação, (re)colocação em memória
  - Normalmente dependerá de módulos externos, libs
  - Também pode ser um script interpretado (pela bash?)
- Processo em execução:
  - código já (re)colocado em memória central + dados +stack
  - Estruturas de gestão:
    - Processo: contexto, recursos HW e SO em uso (registos, ficheiros abertos...)
    - Utilizador (uid, gid, account...)

Sistemas Operativos - 2023/2024

57

.57

## Competição entre processos

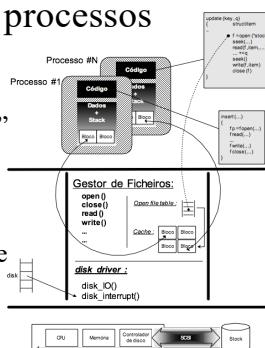
- Como surgem as “race conditions”
  - entre processos
  - dentro do SO
- Vantagens/desvantagens do uso de caches
  - Note que estamos a falar de caches por software, de cópias de dados em memória mas acessíveis em contextos diferentes



Sistemas Operativos - 2023/2024

58

.58



## Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos
- Gestão de ficheiros

Sistemas Operativos - 2023/2024

59

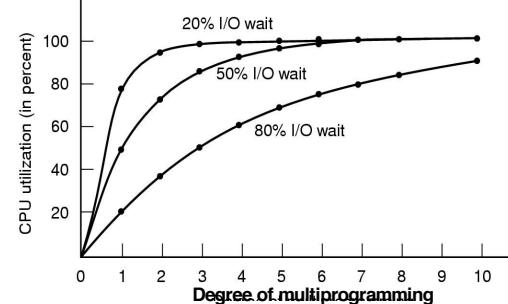
.59

## Porquê criar vários processos?

- Porque dá jeito... + conveniência
  - Estruturação dos programas
  - Para não estar à espera (spooling, background...)
  - Múltiplas actividades / janelas
  
- Porque é melhor + eficiência
  - Múltiplos CPUs
  - Aumenta a utilização de recursos (e.g multiprogramação)

.60

## Benefícios da multiprogramação



.61

## Processos

- **Processo**: um programa em execução, tem actividade própria
- **Programa**: entidade *estática*, **Processo**: entidade *dinâmica*
- Duas invocações do mesmo programa resultam em dois processos diferentes (e.g. vários utilizadores a usarem cada um a sua shell, o vi, browser, etc.)

.62

## Processos

- O contexto de execução de um processo (i.e. o seu **estado**) compreende:
  - código
  - dados (variáveis globais, *heap*, *stack*)
  - estado do processador (registos)
  - ficheiros abertos,
  - tempo de CPU consumido, ...

.63

## Exemplo de informação sobre um processo

Process management	Memory management	File management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Sistemas Operativos - 2023/2024

64

## Gestão de Processos

- Os programas em execução requerem tempo de CPU, memória, utilização de dispositivos, entradas livres em estruturas de dados do SO, canais de comunicação com outros processos, etc.:
  - COMPETEM POR RECURSOS (hw e sw)
- A competição é no entanto mediada pelo SO. Como os processos não podem aceder directamente ao código e dados do SO, necessitam de uma API para solicitar serviços através das *system calls*

Sistemas Operativos - 2023/2024

65

## Exemplo: (parte da) API de processos em Unix

- Para criar um novo processo:
  - **fork**: cria um novo processo (a chamada ao sistema retorna “duas vezes”, uma para o pai e outra para o filho)
  - A partir daqui, ambos executam o mesmo programa
- Para executar outro programa dentro do mesmo processo
  - **exec**: substitui o programa por um novo programa
- Para terminar a execução
  - **exit**

 Compare o **exec** com a invocação de uma função: são muito diferentes

Sistemas Operativos - 2023/2024

66

## fork/exec

```

pid = fork()
if (pid == 0) { // Sou o filho
    exec( novo programa )
} else {
    // Código do pai
}
  
```

Sistemas Operativos - 2023/2024

67

.66

.67

## Gestão de Processos

Cabe ao sistema operativo fazer o escalonamento dos processos, garantindo que a ordem de acesso ao CPU correspondente às políticas de escalonamento previamente definidas

É preciso definir **OBJECTIVOS**

Sistemas Operativos - 2023/2024

68

.68

## Objectivos

- Conveniência:
  - Justiça
  - Redução dos tempos de resposta
  - Previsibilidade
- Eficiência:
  - Débito (*throughput*), transacções por segundo ...
  - Maximização da utilização de CPU e outros recursos
  - Favorecer processos “bem comportados”, etc.

Sistemas Operativos - 2023/2024

69

.69

## Critérios de escalonamento

- IO-bound ou CPU-bound
- Interactivo ou não (batch, background)
- Urgência de resposta (e.g. tempo real)
- Comportamento recente (utilização de memória, CPU)
- Necessidade de periféricos especiais
- PAGOU para ir à frente dos outros...

Sistemas Operativos - 2023/2024

70

.70

## Estados de um processo (i)



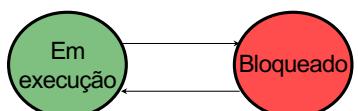
Sistemas Operativos - 2023/2024

71

.71

## Estados de um processo (ii)

Podemos para já admitir que durante a sua “vida” os processos passam por 2 estados:



Sistemas Operativos - 2023/2024

72

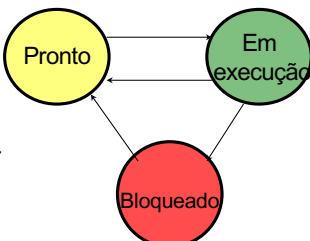
.72

## Estados de um processo (iii)

Na prática, há mais processos não bloqueados do que CPUs

Surge uma fila de espera com processos **Prontos a executar**

Processos em execução podem ser desafectados



Sistemas Operativos - 2023/2024

73

.73

## Estados de um processo (iv)

### • Em execução

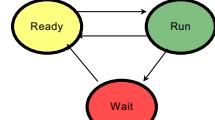
– Foi-lhe atribuído o/um CPU, executa o programa correspondente

### • Bloqueado

– O processo está logicamente impedido de prosseguir porque lhe falta um recurso ou espera por evento

– Do ponto de vista do SO, é uma transição **VOLUNTÁRIA!**

### • Pronto a executar, aguarda escalonamento

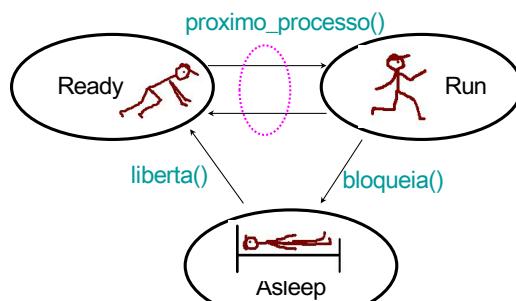


Sistemas Operativos - 2023/2024

74

.74

## Primitivas de despacho (i)



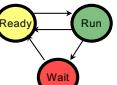
Sistemas Operativos - 2023/2024

75

.75

## Primitivas de despacho (ii)

- Bloqueia(evento)
  - Coloca **processo corrente** na fila de processos **parados** à espera deste “evento”
  - Invoca `próximo_processo()`
- Liberta(evento) ou `liberta(processo,evento)`
  - Se o **outro** processo não está à espera de mais nenhum evento, então coloca-o na lista de processos **prontos a executar**
  - Nesta altura pode invocar ou não `próximo_processo()`



76

Sistemas Operativos - 2023/2024

.76

## Primitivas de despacho (iii)

- `Proximo_processo()`
  - Seleciona um dos processos existentes na lista de processos prontos a executar, de acordo com a política de escalonamento
  - Executa a comutação de contexto
    - Salvaguarda contexto volátil do processo corrente
    - Carrega contexto do processo escolhido e regressa (executa o `return`)

Como o Stack Pointer foi mudado,  
"regressa" para o **processo escolhido!**

Sistemas Operativos - 2023/2024

77

.77

## Principais decisões

- Qual o próximo processo?
- Quando começa a executar?
- Durante quanto tempo?
- Por outras palavras,

Há **desafectação forçada** ou não?

Sistemas Operativos - 2023/2024

78

.78

## Escalonamento de processos

- Quando, uma vez atribuído a um processo, o CPU nunca lhe é retirado então diz-se que o escalonamento é **cooperativo** (non-preemptive).
  - Exemplos: Windows 3.1, co-rotinas, `thread_yield()`
- Quando o CPU pode ser retirado a um processo ao fim do quantum ou porque surgiu outro de maior prioridade diz-se que o escalonamento é com **desafectação forçada** (preemptive)

Sistemas Operativos - 2023/2024

79

.79

.20

## Escalonamento de processos

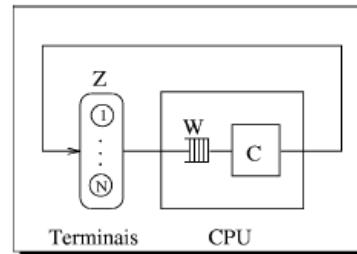
- Escalonamento **cooperativo** (non-preemptive).
  - “poor man’s approach to multitasking” ?
  - Sensível às variações de carga
- Escalonamento com **desafectação forçada**
  - Sistema “responde” melhor
  - Mas a comutação de contexto tem overhead

Sistemas Operativos - 2023/2024

80

.80

## Modelo de sistema interactivo



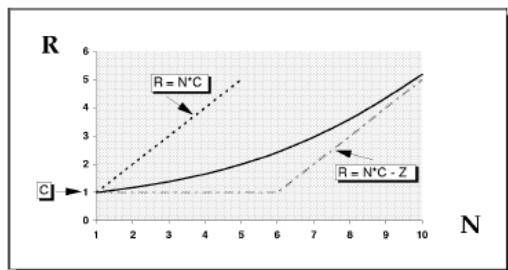
Z = Think time  
 C = Service time  
 W = Wait time  
 N = Number of users

Sistemas Operativos - 2023/2024

81

.81

## Tempo de Resposta (carga homogénea)

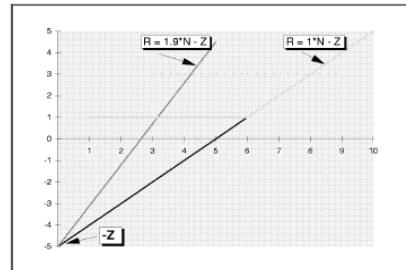


Sistemas Operativos - 2023/2024

82

.82

## Tempo de Resposta (carga heterogénea)



Sistemas Operativos - 2023/2024

83

.83

Assuma-se agora que uma em cada 10 interacções é muito longa, 10 vezes maior.  
 Veja-se a degradação de tempos de resposta

## Tempo de Resposta (carga heterogénea)

- Para evitar que as interações longas monopolizem o CPU e aumentem o tempo de resposta das restantes deve usar-se desafectação forçada.
- Neste caso deve atribuir-se um quantum (ou time slice) para permitir a troca rápida de processos:
  - Interacções curtas terminam dentro dessa fatia de tempo, logo não são afectadas pela política de desafectação.
  - Interacções longas executam durante um quantum e a seguir o processo correspondente regressa ao estado de **Pronto a Executar**, dando a vez a outros processos. Mais tarde ser-lhe-á atribuído nova fatia de tempo, e sucessivamente até a interacção terminar.

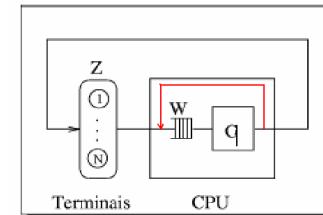
Sistemas Operativos - 2023/2024

84

.84

## Duração da fatia de tempo

- Maioria das interacções deve “caber” num quantum
- $R = W + C$
- Se precisar de 2 passagens pelo CPU,  $T_{Resposta}$  é quase o dobro!



Sistemas Operativos - 2023/2024

85

.85

## Escalonamento de processos

- Escalonadores de longo-prazo (segundos, minutos) e de curto-prazo (milisegundos)
- Processo CPU-bound: processo que faz pouco I/O mas que requer muito processamento
- Processo I/O-bound: processo que está frequentemente à espera de I/O.

Sistemas Operativos - 2023/2024

86

.86

## Escalonamento de processos

- Os processos prontos são seriados numa fila (*ready list*)
- A lista é uma lista ligada de apontadores para PCB's
- A lista poderá estar ordenada por vários critérios de forma a dar tratamento preferencial a alguns processos

Nesta UC deve evitar falar em *prioridade* sem descrever e justificar o critério que leva a uma oedenação particular.

Sistemas Operativos - 2023/2024

87

.87

## Escalonamento de processos

- Quando um processo é escalonado, é retirado da *ready list* e posto a executar
- Pode “perder” o CPU por 2 razões:
  - Faz um pedido de I/O que não pode ser servido imediatamente (e.g. teclado) ou pede ao SO para esperar: passa ao estado de **bloqueado**
  - É *desafectado* porque aparece um processo com maior "prioridade" ou o seu *quantum* expira: passa ao estado de **pronto**

Sistemas Operativos - 2023/2024

88

.88

## Escalonamento de processos

- A decisão de escalar um processo pode ser tomada em diversas alturas:
  - Quando o processo é bloqueado (óbvio!)
  - Quando o processo passa a pronto a executar
  - Quando se completa uma operação de I/O
  - Quando um processo termina

Sistemas Operativos - 2023/2024

90

.90

## Escalonamento de processos

- Pretende-se maximizar a utilização do CPU tendo em atenção outros aspectos:
  - Tempo de resposta para aplicações interactivas
  - Utilização de dispositivos de I/O
  - Justiça na distribuição do tempo de CPU

Sistemas Operativos - 2023/2024

89

.89

## Escalonamento de processos

- Diferentes algoritmos de escalonamento visam objectivos diferentes:
  - Diminuir o tempo de resposta (reduzindo o tempo de espera para determinados processos)
  - Maximizar a utilização do CPU

Sistemas Operativos - 2023/2024

91

.91

## Escalonamento de processos

- Alguns algoritmos de escalonamento:
  - FCFS (First Come, First Served)
  - SJF (Shortest Job First)
  - SRTF (Shortest Remaining Time First)
  - Preemptive Priority Scheduling
  - RR (Round Robin)

Sistemas Operativos - 2023/2024

92

.92

## First Come, First Served (FCFS)

- A *ready list* é uma fila FIFO
- Os processos são colocados no fim da fila e selecionado o da frente
- Método cooperativo
- Nada apropriado para ambientes interactivos

Sistemas Operativos - 2023/2024

93

.93

## FCFS

- Uma vantagem óbvia do FCFS é sua simplicidade de implementação: lista de processos por ordem de criação do processo (batch)
- Sujeito a tempos de espera com grandes flutuações, dependendo da ordem de chegada e das características dos processos: “efeito de comboio”
- Parece haver vantagens em escalar os processos mais curtos à frente...

Sistemas Operativos - 2023/2024

94

.94

## SJF (Shortest Job First)

- A ideia é escalar o processo mais curto primeiro
- Possibilidades:
  - Desafectação forçada (SRTF) – interrompe-se o processo em execução se aparecer um mais curto
  - Cooperativo – mesmo na presença de um processo mais curto, pode aguardar-se pela terminação ou bloqueio voluntário do processo em execução. E nessa altura escolhe-se o mais curto

Sistemas Operativos - 2023/2024

95

.95

## Preemptive Priority

- Associa uma prioridade (geralmente um inteiro) a cada processo.
- A *ready queue* é uma fila seriada por prioridades.
- Escalona sempre o processo na frente da fila.
- Se aparece um processo com maior prioridade do que o que está a executar faz a troca dos processos

Sistemas Operativos - 2023/2024

96

.96

## Preemptive Priority

- Problema: starvation
- Uma solução: envelhecimento – aumenta a prioridade dos processos pouco a pouco de forma a que inevitavelmente executem e terminem.
- Convém justificar quando e quanto aumenta...

Sistemas Operativos - 2023/2024

97

.97

## RR (Round Robin)

- Dá a cada processo um intervalo de tempo fixo de CPU de cada vez
- Quando um processo esgota o seu quanto retira-o da CPU e volta a colocá-lo no fim da fila.
- Ignorando os overheads do escalonamento, cada um dos  $n$  processos CPU-bound terá  $(1/n)$  do tempo disponível de CPU

Sistemas Operativos - 2023/2024

98

.98

## RR

- Se o quantum for (muito) grande o RR tende a comportar-se como o FCFS
- Se o quantum for (muito) pequeno então o overhead de mudanças de contexto tende a dominar degradando os níveis de utilização útil de CPU
- Favorece processos que libertem o CPU ao fim de pouco tempo“ (aproxima-se do “shortest”) mas sem exigir conhecimento rigoroso do tempo de cada “CPU burst”

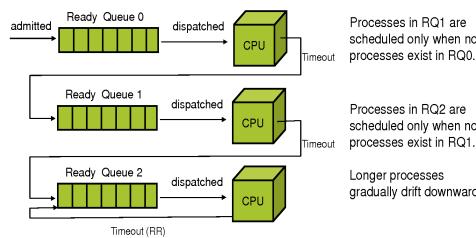
Sistemas Operativos - 2023/2024

99

.99

### Multilevel Feedback Queue Scheduling

- Another way to put a preference on short-lived processes
  - Penalize processes that have been running longer.
- Preemptive



Sistemas Operativos - 2023/2024

100

.100

### Níveis de escalonamento

- Uma vez que há inúmeros critérios de escalonamento e muitas variáveis a considerar para saber qual o “melhor” processo a escolher, é habitual dividir a questão em 2 ou 3 níveis:
  - Nível 0 --- só despacha o que está em RAM (RR ou MLQ)
  - Nível 1 --- Decide que processos são multiprogramados, por indicação do gestor de memória
  - Nível 2 --- Não deixa criar processos nas horas de ponta, por decisão se quem administra e conhece a carga diária.

Sistemas Operativos - 2023/2024

101

.101

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos
- Gestão de ficheiros

Sistemas Operativos - 2023/2024

102

.102

### Paralelismo versus concorrência

- Execução paralela => hardware
  - Vários computadores, eg. cluster
  - Multiprocessamento, eg. um processo em cada CPU
  - Hyper-threading / Dual core
  - CPU a executar instruções em paralelo com a operação de disco (que se manifesta através de uma **interrupção**, com prioridade superior à actividade no CPU)

Sistemas Operativos - 2023/2024

103

.103

## Em geral...

- Seja num ambiente de paralelismo real ou simulado pelo SO
- Existem várias “actividades” em execução “paralela”
- Normalmente essas actividades **não são independentes**, há **interacção** entre elas
- Este facto que levanta algumas questões...

Sistemas Operativos - 2023/2024

104

.104

## Papel do SO

- O sistema operativo tem a responsabilidade de
  - Fornecer **mecanismos** que permitam a criação e interacção entre processos
  - Gerir a execução concorrente (ou em paralelo), de acordo com as **políticas** definidas pelo administrador de sistemas

Sistemas Operativos - 2023/2024

105

.105

## Cooperação e Competição

- Há normalmente 2 padrões de interacção entre processos:
  - Cooperam entre si para atingir um resultado comum
    - . Processo inicia transferência do disco e aguarda passivamente que esta termine
    - . O disco interrompe e a rotina de tratamento avisa que o processo já pode prosseguir
  - Competem por recursos (CPU, memória, impressora, etc.)
    - . Há necessidade de forçar 1 ou mais processos a esperar até que o recurso pretendido fique disponível e lhes seja atribuído (seja a sua vez ).

Sistemas Operativos - 2023/2024

106

.106

## Sincronização

- Nos dois casos anteriores estamos perante uma questão de sincronização:
  - Cooperação (espera até que evento seja assinalado)
  - Competição (espera até que recurso esteja disponível)
- **Sincronizar** é fazer esperar, atrasar deliberadamente um processo até que determinado evento surja
  - Convém que a espera seja passiva.

Sistemas Operativos - 2023/2024

107

.107

## Comunicação

- Para haver interacção tem de haver de **comunicação**:
  - Processos podem requisitar ao SO memória partilhada, podem escrever/ler ficheiros comuns, enviar/receber *mensagens* através de “canais”, pipelines, sockets, etc.
  - Threads do mesmo processo podem comunicar através de variáveis globais
- A comunicação pode ser tão simples como o assinalar a ocorrência de um evento (de sincronização), ou pode transportar dados.

Sistemas Operativos - 2023/2024

108

.108

## Exemplos de concorrência

- Inspirados na vida real
  - Diálogo cliente/bar(wo)men
  - Actualização do saldo de uma conta bancária
  - Entrada num parque de estacionamento ou sala de cinema sem marcação de lugar
- São exemplos, respectivamente, de
  - Sincronização
  - Exclusão mútua (caso particular em que capacidade = 1)
  - Controlo de capacidade

Sistemas Operativos - 2023/2024

109

.109

## Na realidade...

- Os três exemplos anteriores são casos de **Sincronização**, pois em todos poderá ser necessário esperar até que se verifique determinada condição:
  - Cliente: existir pelo menos um copo no balcão
  - O recurso estar livre, não haver outro processo a actualizar “esse” registo do ficheiro
  - Sala de cinema ou parque de estacionamento não estar totalmente cheios.
- Como veremos a seguir, conduzem a 3 padrões de soluções

Sistemas Operativos - 2023/2024

110

.110

## Exemplo: num festival...

- Que algoritmo sugere para controlar a concorrência de largas dezenas de clientes ansiosos por retirarem um copo de cerveja do balcão?
  - Que algoritmo sugere seja usado pela pessoa ou pessoas que estão a encher os copos e a colocá-los no balcão?

Sistemas Operativos - 2023/2024

111

.111

Não basta dizer...



- **Cliente/servidor?**
  - Já temos, não?
  - Clientes pedem, barman/servidor responde?
- **Arranja-se uma fila**
  - Onde? Só uma? Porquê? Como a implementa?
- **Lista ligada?**
  - De quê? De copos, clientes ou barmen? Dos 3???

.112

Vamos recapitular

- **Comunicação:** *escrita/leitura de dados* passados através de vários mecanismos (ficheiros, memória partilhada...)
- **Sincronização:** *coordenação de eventos* entre processos (usando semáforos, variáveis de condição, locks, etc.), designadamente para garantir uma ordem na ocorrência desses eventos -- só leio depois de tu escreveres;
- Há mecanismos que tratam dos dois aspectos, e.g. pipelines, mensagens, sockets...

.114

Onde está a espera?

- Uma lista é apenas uma estrutura de dados, quando muito poderia servir para comunicação entre barmen e clientes (balcão)
- E depois de inserir um copo nessa fila, como/quem espera por um cliente? Se a fila for de clientes, como esperam pelo seu copo?
- É preciso perceber a diferença entre **comunicação** e **sincronização!**

.113

Arquitectura da solução

- Que processos temos, quem são os componentes activos que executam algoritmos?
  - processos Cliente e processos Barman
- Como comunicam através do “balcão”?
  - variável/array/lista/ficheiro de copos?
- Como sincronizam?
  - Cliente espera por copo, barman pousa copo

.115

## Sincronização com variáveis partilhadas

### CLIENTE

```
/* aguarda por copo cheio */
while (n_copos == 0);

n_copos = n_copos - 1;
tirar_copo_do_balcao();
...
```

Consegue perceber a ideia ?

Sistemas Operativos - 2023/2024

116

.116

## Sincronização com variáveis partilhadas

### CLIENTE

```
/* aguarda por copo cheio */
while (n_copos == 0);

n_copos = n_copos - 1;
tirar_copo_do_balcao();
...
```

### BARMAN

```
/* aguarda vaga no balcão*/
while (n_copos == MAX);

n_copos = n_copos + 1;
pousar_copo_no_balcao();
...
```

Acha que este pseudo-código está correcto?

Sistemas Operativos - 2023/2024

117

.117

## Infelizmente não está...

- Não garante que não haja copos partidos (?!)
- Nem clientes a pegarem no mesmo copo (?!?)
- E é muito ineficiente
  - esperas activas desperdiçam tempo de CPU
  - precisamos de primitivas capazes de *adormecer/acordar* processos

Sistemas Operativos - 2023/2024

118

.118

## Mecanismos de sincronização

- Existem muitas propostas, sendo o seu estudo diferido para unidades curriculares como programação concorrente, sistemas distribuídos e computação paralela.
- Umas são genéricas (semáforos, mensagens), outras vocacionadas para casos particulares nomeadamente
  - *exclusão mútua* (e.g. mutexes, fcntl, Linux leases, métodos *synchronized*, etc.).
  - para serem usadas dentro do kernel do SO (sleep/wakeup, spin locks)
  - em ambientes multithreaded (wait/signal/notify)...

Sistemas Operativos - 2023/2024

119

.119

## Mecanismos de sincronização

- Em sistemas operativos explora-se a partilha entre processos de estado global tipicamente guardado em ficheiros e, por outro lado, o encapsulamento desse estado em processo(s) gestor(es).
- Nas aulas teóricas usam-se semáforos como introdução e treino mental de controlo de concorrência, enquanto que nas práticas se usam pipes e fifos, que combinam comunicação e sincronização no mesmo mecanismo.

Sistemas Operativos - 2023/2024

120

.120

## Semáforos

- Servem para resolver problemas de sincronização, exclusão mútua e controlo de capacidade
- Com apenas 3 operações\*:
  - Inicialização :  
 $s = \text{cria\_semáforo}(\text{valor\_inicial})$
  - P (s) ou Down (s)
  - V (s) ou Up (s)

\* Na realidade há mais operações

Sistemas Operativos - 2023/2024

121

.121

## Semáforos

- Imagine uma caixa com bolas e as operações:
- P: Se há bola(s) na caixa, retiro uma e continuo, senão aguardo (passivamente) que alguém deposite uma
- V: Devolvo a bola à caixa; se há alguém bloqueado à espera, acordo-o

(P = PÁRA e V = VAI; podem bloquear e libertar processos, respectiv.)

Sistemas Operativos - 2023/2024

122

.122

## Semáforos

$P(s)\{$ $s = s - 1$ $\text{if } (s < 0)$ $\quad \text{then bloqueia}("S")$ $\}$	$V(s)\{$ $s = s + 1$ $\text{if } (s \leq 0)$ $\quad \text{then liberta}("S")$ $\}$
--	--

Quando  $s < 0$ , o seu valor absoluto  $|s|$  conta o número de processos bloqueados no P()

Sistemas Operativos - 2023/2024

123

.123

## Semáforos

- Bloquear significa retirar o processo corrente do estado RUN e inseri-lo na fila “S”
- “S” contém os processos BLOQUEADOS no semáforo S
- Libertar significa escolher um processo da fila “S” e inseri-lo na fila READY
- Normalmente essa escolha é FIFO
- **Mas pode não ser...**

Sistemas Operativos - 2023/2024

124

.124

## Sincronização com semáforos

- Para cada evento de sincronização, é criado um semáforo com **valor inicial zero**
- Um processo espera *passivamente* pelo evento, e só avança depois do evento acontecer
  - P(s) /\* se caixa vazia, espera; senão evento já ocorreu \*/
- Outro processo assinala ocorrência do evento
  - V(s) /\* se ninguém à espera, deixa bola na caixa para indicar que o evento já ocorreu\*/

Sistemas Operativos - 2023/2024

125

.125

### CLIENTE

```
/* aguarda por copo cheio */
P(copo)
```

```
tirar_copo_do_balcao();
```

### BARMAN

```
pousar_copo_no_balcao();
```

```
/* avisa que há +1 copo cheio */
V(copo)
```

```
....
```

**Falta inicializar o semáforo copo a Zero**

Sistemas Operativos - 2023/2024

126

.126

## Capacidade

- Se a capacidade do balcão fosse de apenas 1 copo, Cliente e Barman executariam à vez. (Seria má estratégia... porquê?)
- Para aguardar por espaço no balcão, usa-se um semáforo inicializado à capacidade do recurso partilhado (e não a Zero)
- É um caso particular de sincronização: só bloqueia se o recurso estiver esgotado naquele instante

Sistemas Operativos - 2023/2024

127

.127

**CLIENTE**

```
/* aguarda por copo cheio */
P(copo)

tirar_copo_do_balcao();

/* avisa que há vaga */
V(espaço)
```

**BARMAN**

```
/* aguarda por vaga no balcão */
P(espaço)

pousar_copo_no_balcao();

/* avisa que há +1 copo cheio */
V(copo)
....
```

**Falta inicializar o semáforo copo a Zero e espaço à capacidade do balcão**

Sistemas Operativos - 2023/2024

128

**Exclusão mútua com semáforos**

- Para cada região crítica, é criado um semáforo com valor inicial igual a **1 (um)**
- No início da região crítica  
 $P(s)$  /\* só avança se região está livre \*/
- No fim da região crítica  
 $V(s)$  /\* assinala que a região está livre \*/

Sistemas Operativos - 2023/2024

129

**“Receitas” com semáforos**

- Sabendo que  $\text{valor inicial} + \#V() \geq \#P()$  concluídos
  - Sincronização:  
 $\text{valor inicial} = 0$
  - Exclusão mútua:  
 $\text{valor inicial} = 1$
  - Capacidade:  
 $\text{valor inicial} = N = \text{capacidade do recurso}$

Sistemas Operativos - 2023/2024

130

**Produtor/consumidor com semáforos**

- Agora que resolvemos os aspectos de sincronização
- E já sabemos a “receita” da exclusão mútua
- É altura de reparar que o balcão é uma variável partilhada pelos vários processos ( $M$  barmen +  $N$  clientes)

=>Falta garantir **exclusão mútua** no acesso ao balcão!

Sistemas Operativos - 2023/2024

131

.130

.131

### Produtor(es)

```
P(espaço);
P(mutex);
Buf[p++ % N] = px;
V(mutex);
V(copo);
```

### Consumidor(es)

```
P(copo)
P(mutex);
cx = Buf[c++ % N];
V(mutex);
V(espaço)
```

**Falta inicializar os semáforos espaço a N, copo a ZERO, mutex a UM, e as variáveis p e c a ZERO.**

Sistemas Operativos - 2023/2024

132

.132

### Self-scheduling vs Client/Server

- As chamadas ao sistema são executadas dentro do kernel mas em ambiente protegido e controlado
- Na arquitectura cliente/servidor há separação de processos:
  - o cliente nunca tem acesso directo ao estado do servidor nem AOS RECURSOS que este gere: isto implica permissões diferentes nos ficheiros e periféricos!
  - cliente faz o pedido e aguarda pela resposta: são precisos mecanismos de comunicação e sincronização, bem como de autenticação de clientes

Sistemas Operativos - 2023/2024

134

.134

### E não temos cliente/servidor?

- O que aqui temos são dois conjuntos de processos, uns a “produzirem” dados e outros a “consumirem” esses dados, de forma coordenada, eficiente e justa (ordem FIFO)
- Os algoritmos tratam apenas da passagem de dados entre produtores/barmen e consumidores/clientes/utentes. Mas poderão vir a ser usados numa arquitectura cliente/servidor!
- E antes de chegarmos a essa arquitectura, há que responder a:
  - Que mal teria deixar os utentes/clientes servirem-se a eles próprios? Porque há um balcão e gente a servir?*

Sistemas Operativos - 2023/2024

133

.133

### E não há...

- Sistemas operativos com arquitectura C/S?
  - Há! E desde os anos 80... tipicamente com um micro-kernel que gere escalonamento e diálogo com periféricos e um conjunto de processos “gestores” de recursos (e.g. o MACH de Carnegie Mellon University)
  - Curiosidade: dois dos investigadores principais do MACH mudaram-se em meados de 90 para a Microsoft e Apple, presumivelmente influenciando a evolução para Windows NT e MAC OSX...

Sistemas Operativos - 2023/2024

135

.135

## Programa

- Introdução
- Gestão de processos
- Noções de programação concorrente
- Gestão de memória
- Gestão de periféricos
- Gestão de ficheiros

Sistemas Operativos - 2023/2024

136

.136

## Gestão de Memória

- Idealmente a memória seria:
  - grande
  - rápida
  - não volátil
- Na realidade, há uma hierarquia
 

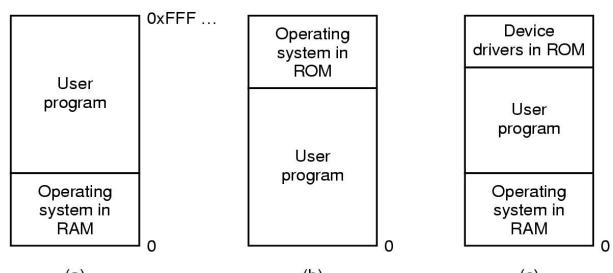
<ul style="list-style-type: none"> <li>– Pouca memória muito rápida</li> <li>– Velocidade média, custo aceitável</li> <li>– Giga/terabytes de memória auxiliar</li> </ul>	<ul style="list-style-type: none"> <li>– cache(s)</li> <li>– RAM</li> <li>– discos mag. /SSD</li> </ul>
---	---

Sistemas Operativos - 2023/2024

137

.137

## Monoprogramação



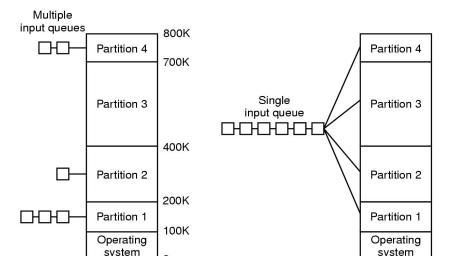
Três formas de organizar a memória com SO e apenas um processo

Sistemas Operativos - 2023/2024

138

.138

## Multiprogramação c/ partições de dimensão fixa (qq alteração implica reboot ?)



Filas separadas para cada partição

Fila única

Sistemas Operativos - 2023/2024

139

.139

## Recolocação e Protecção

- Para carregar um processo em qualquer partição livre...
  - O endereço de carregamento do programa não pode ser definido pelo compilador!
  - Isso é, o endereço de variáveis funções não pode ser *absoluto*
  - Tem de ser *recolocável*, para ser alterável no momento do carregamento
  - Para o compilador, *tudo começa no endereço 0!*
- CPUs passam a dispor de registos base e limite
  - Endereços saídos do CPU são adicionados à base para obter endereços físicos
  - Valores superiores ao registo limite são erros ( =>processo é terminado)

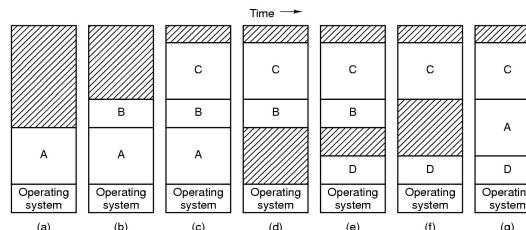
Sistemas Operativos - 2023/2024

140

.140

## Swapping

A alocação de memória muda sempre que um processo é criado ou terminado (óbvio!), e também sempre que um processo é *swapped out* ou *swapped in* de disco

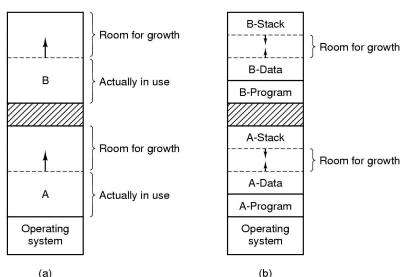


Sistemas Operativos - 2023/2024

141

.141

## Swapping



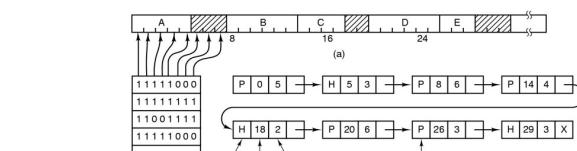
- (a) Alocação para segmento de dados crescente  
 (b) Alocação para segmentos de dados e stack crescentes

Sistemas Operativos - 2023/2024

142

.142

## Gestão de memória com bitmaps



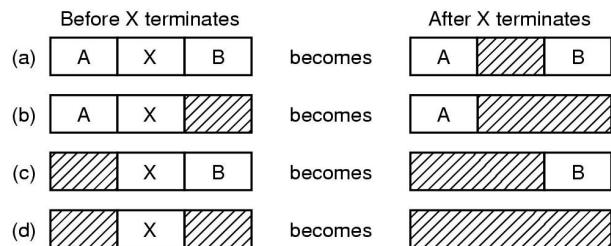
- Zona de memória
- Bitmap correspondente
- Mesma informação de uma lista ligada

Sistemas Operativos - 2023/2024

143

.143

## Gestão de memória listas ligadas



Quatro cenários para a terminação do processo X

.144

Em resumo, a GM baseada em partições...

- Aumenta a eficiência à custa de multiprogramação mas...

– É inconveniente:

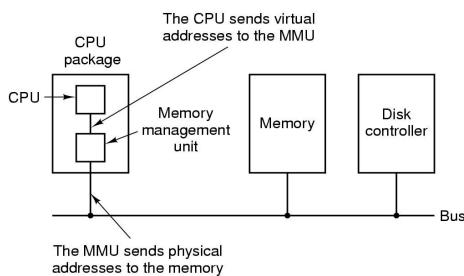
- Restringe a dimensão máxima dos processos
- Alocação contígua dificulta a atribuição de endereços
- Não permite protecção “fina” (e.g. read, write, exec)

– É ineficiente:

- Causa fragmentação e não permite partilha
- Nem tira partido da dispersão de referências

.145

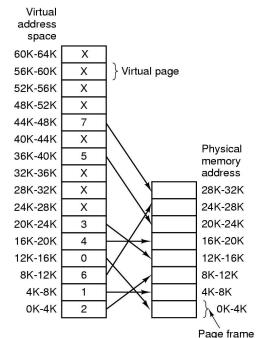
## Memória Virtual



.146

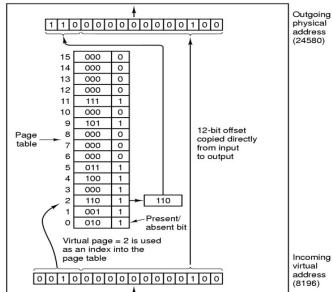
## Memória Virtual

A relação entre  
endereços virtuais e  
físicos é dada por uma  
tabela



.147

## Memória Virtual



Operação da MMU com 16 páginas de 4 KB

.148

## Memória Virtual

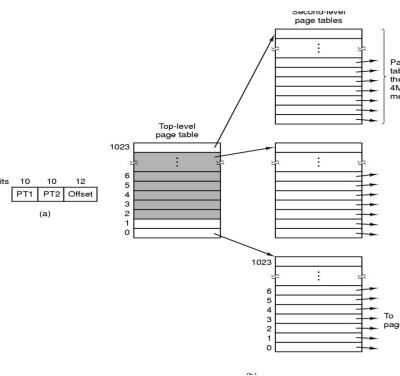
- Endereço de 32 bit c/ 3 campos:
  - PT1, PT2, Offset

- Tabelas de páginas de 2 níveis
  - Indexadas por PT1 e PT2

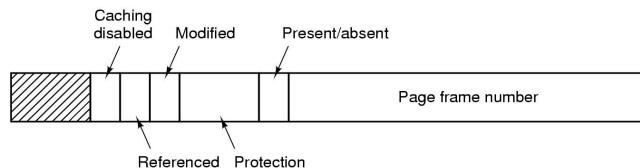
- Tabela de 2º nível tem endereço
  - base do frame respectivo
  - Ao qual se some o Offset

Porquê 2 ou mais níveis????

.149



## Memória Virtual



Entrada típica da tabela de páginas

.150

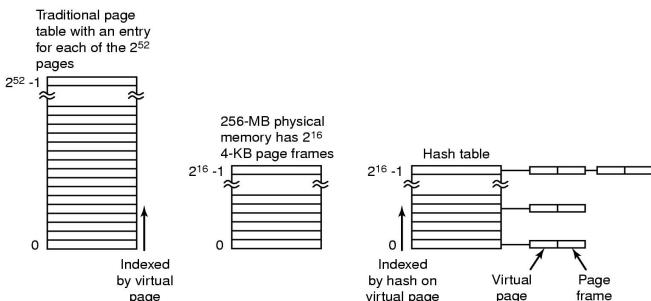
## Memória Virtual

- TLBs – Translation Lookaside Buffers – para melhorar o desempenho

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

.151

## Memória Virtual



- Comparação entre tabelas tradicionais e **tabelas invertidas**

Sistemas Operativos - 2023/2024

152

.152

## Aspectos de Implementação

### Tratamento da Page Fault:

1. MMU interrompe o processador.
2. Kernel salvaguarda os registos e invoca função de tratamento
3. GMV determina a página necessária e a razão para interrupção
4. SO valida endereço e procura/cria page frame
  - Page in, zero fill, in transit...
  - Pode implicar rejeição de outra página

Sistemas Operativos - 2023/2024

153

.153

## Aspectos de Implementação

O SO intervém na paginação em 4 situações:

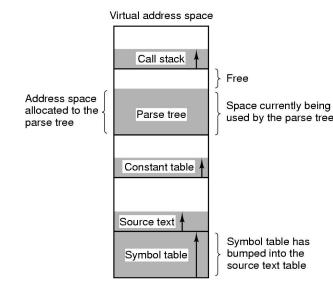
1. Criação do processo
  - Determina o tamanho do programa
  - Cria a tabela de página
2. Execução do processo *ie, na comutação para o próximo\_processo*
  - Re-inicializa a MMU para o novo processo
  - Limpar a TLB
3. Page Fault
  - Determina o endereço virtual causador da *page fault*
  - Coloca a página em memória, se endereço for legal
4. Fim da execução do processo
  - Liberta a tabela de páginas e as páginas/frames

Sistemas Operativos - 2023/2024

154

.154

## Segmentação



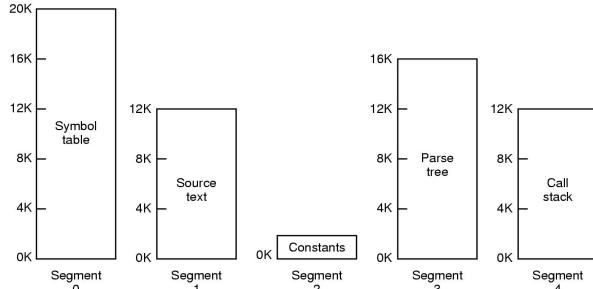
- Espaço de endereçamento único com tabelas crescentes
- Uma tabelas pode sobrepor-se a outra

Sistemas Operativos - 2023/2024

155

.155

## Segmentação



- Cada tabela pode crescer ou encolher independentemente

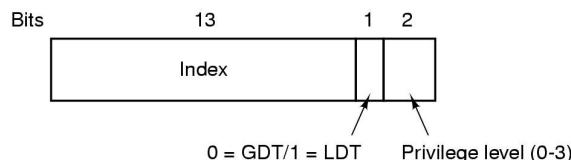
•156

## Segmentação vs Paginação

	Paginação	Segmentação
Transparente para o programador	Sim	Não
Número de espaços de endereçamento	1	Vários
O espaço de endereçamento pode ultrapassar o tamanho da memória física	Sim	Sim
O código e dados podem ser distintos e protegidos separadamente	Não	Sim
Tabelas de tamanho variável podem ser geridas facilmente	Não	Sim
A partilha de código é facilitada	Não	Sim

•157

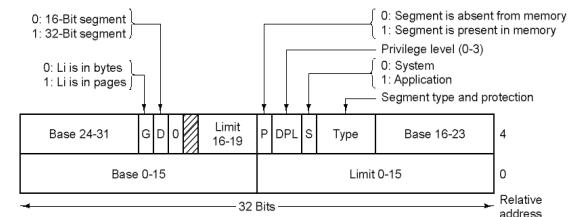
## Segmentação com Paginação: Pentium



Um **selector** no Pentium

•158

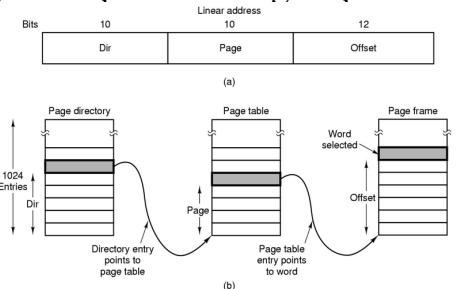
## Segmentação com Paginação: Pentium



Descriptor de segmento de código

•159

## Segmentação com Paginação: Pentium



Mapeamento de um endereço linear para o endereço físico

.160

## Rejeição de páginas

- Rejeitar a página que será usada mais tarde
  - Inexequível
  
- Aproximado por estimativa
  - Histórico de execuções anteriores do processo
  - Também isto é impraticável

.162

## Rejeição de páginas

- Um *page fault* leva:
  - A encontrar espaço para a nova página, a criar ou trazer de disco
  - A decidir que página em memória rejeitar, caso não haja espaço
  
- Uma página modificada tem que ser escrita (em disco)
  - Uma que não modificada pode ser imediatamente utilizada
  
- Convém não rejeitar uma página frequentemente usada
  - Pois provavelmente terá de ser carregada a seguir

.161

## Rejeição de páginas NRU

- Cada página tem 1 bit de acesso e 1 de escrita
- As páginas são assim classificadas:
  1. Não acedida, não modificada
  2. Não acedida, modificada
  3. Acedida, não modificada
  4. Acedida, modificada

NRU remove a página com menor “ranking”

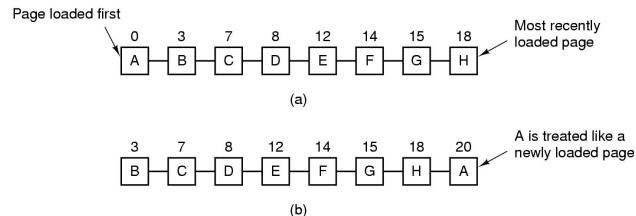
.163

## Rejeição de páginas FIFO

- Mantém uma lista das páginas em memória
  - Segundo a ordem em que foram carregadas
- A página no topo da lista é rejeitada
- Desvantagem
  - A página há mais tempo em memória poderá ser a mais usada

•164

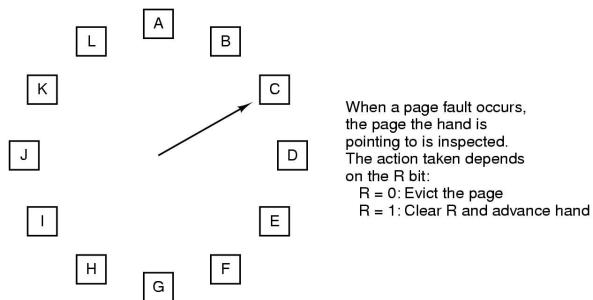
## Segunda Oportunidade



- Ordem FIFO
- Se a página mais antiga tiver sido acedida, não é rejeitada
- É limpo o bit de acesso e é colocada no fim da fila

•165

## Rejeição de páginas: Relógio



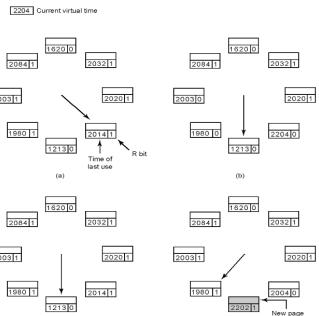
•166

## Rejeição de páginas LRU

- Assume que as páginas usadas recentemente serão usadas em breve
  - Rejeitar a página não usada há mais tempo
- Tem de gerir uma (enorme?) lista de páginas
  - Ordenada pela mais recente
  - Tem de actualizar em todos os acessos à memória! (talvez não...)
- Uma alternativa seria manter um contador em cada entrada da tabela de páginas
  - Escolher a página com o menor valor
  - Periodicamente zerar o contador (de todas?)

•167

## WorkingSetClock



Sistemas Operativos - 2023/2024

168

.168

## Rejeição de páginas

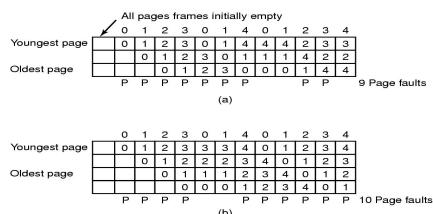
Algoritmo	Características
Óptimo	Inexequível. Padrão para comparação.
NRU (não usado recentemente)	Aproximação grosseira.
FIFO	Leva à rejeição de páginas importantes.
Segunda Oportunidade	Melhoramento do FIFO.
Relógio	Solução realista.
LRU (menos recentemente usado)	Muito bom. Implementação exacta difícil.
NFU (menos frequentemente usado)	Aproximação grosseira do LRU.
Aging (envelhecimento)	Aproximação boa e eficiente do LRU.
Working set	Implementação ineficiente.
WSClock	Aproximação boa e eficiente.

Sistemas Operativos - 2023/2024

169

.169

## Anomalia de Belady



- FIFO com 3 page frames
- FIFO com 4 page frames
- Os P's indicam ocorrência de page faults

Sistemas Operativos - 2023/2024

170

.170

## Gestão de Memória – recap

Como auto-avaliação, assegure-se que consegue acompanhar os vídeos seguintes:

<https://www.youtube.com/watch?v=qdkxXygc3rE>

<https://www.youtube.com/watch?v=qlH4-oHnBb8>

Sistemas Operativos - 2023/2024

171

.171

## Sistemas Paginados

- Aspectos de Concepção de
  - Alocação local e global
  - Controlo de carga / thrashing
  - Tamanho das páginas

.172

## Alocação Local e Global

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(b)

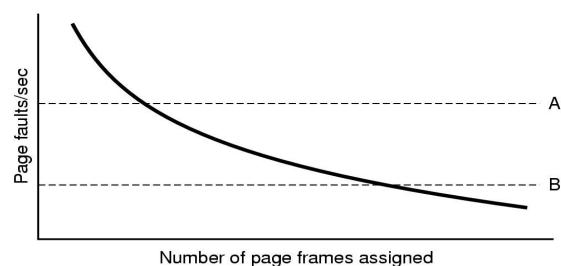
	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(c)

Config. original Subst. local Subst. global

.173

## Alocação Local e Global



.174

## Controlo de carga

- Apesar de um bom desenho, pode ainda ocorrer **thrashing**
- Quando a Frequência de Page Faults é elevada e indica que:
  - Alguns processos precisam de mais memória central
  - Mas nenhum pode ceder parte da memória que tem
- A solução passa por reduzir o número de processos que competem por memória
  - Passando um ou mais processos para disco, libertando os frames que lhes estavam atribuídas. Mas quais???

.175

## Tamanho das páginas

### Páginas pequenas

- Vantagens
  - Menos fragmentação interna
  - Melhor adequação a várias estruturas de dados e código
  - Menos partes de programas não usados em memória
- Desvantagens
  - Mais páginas, tabelas de páginas maiores

.176

## Tamanho das páginas

- Overhead estimado:

$$\text{overhead} = \frac{s \cdot e}{p} + \frac{p}{2}$$

Espaço da tabela de páginas

Fragmentação interna

- Em que

- s = tamanho médio dos processos em bytes
- p = tamanho das páginas
- e = entrada na tabela de páginas

Valor óptimo quando

$$p = \sqrt{2se}$$

.177

## Tamanho das páginas

- Hardware pode impor valores fixos (e.g. 4KB em Intel) mas o SO pode *olhar para a memória* de outra forma, permitindo tamanhos maiores a certas aplicações
- **Huge pages** de 2 MB... 2 GB ? Que não saem de RAM? **Porquê?**
- Quem se lembra do *sticky-bit* no início do Unix?

.178