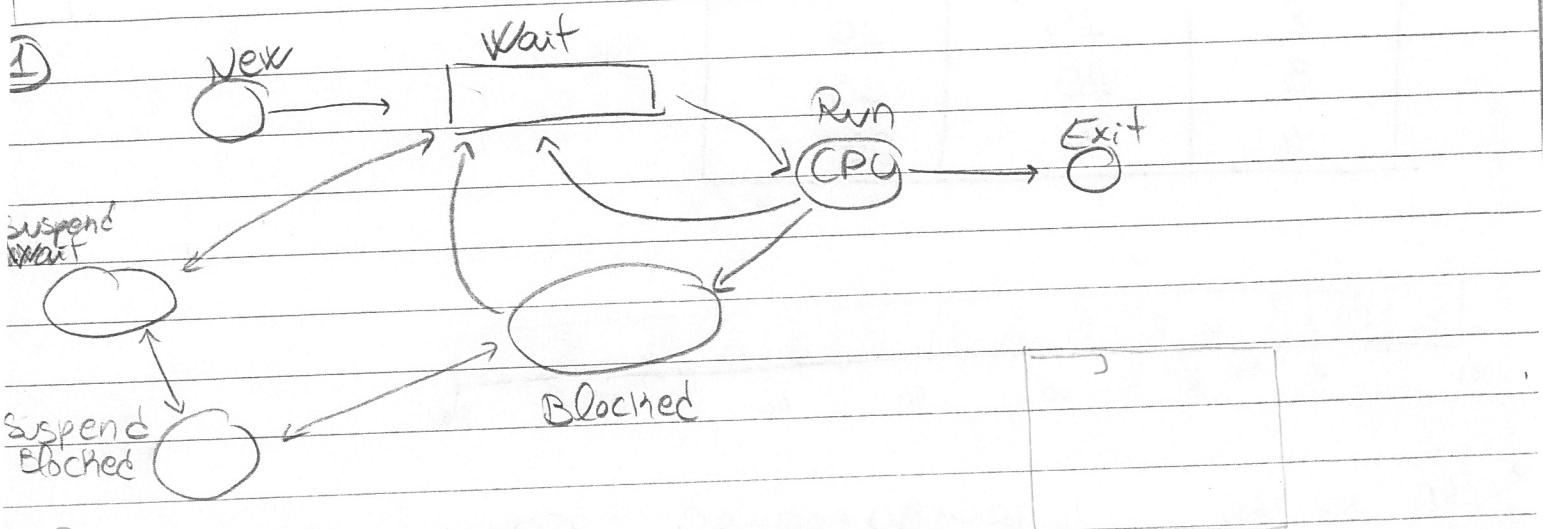


req 1 → 2016

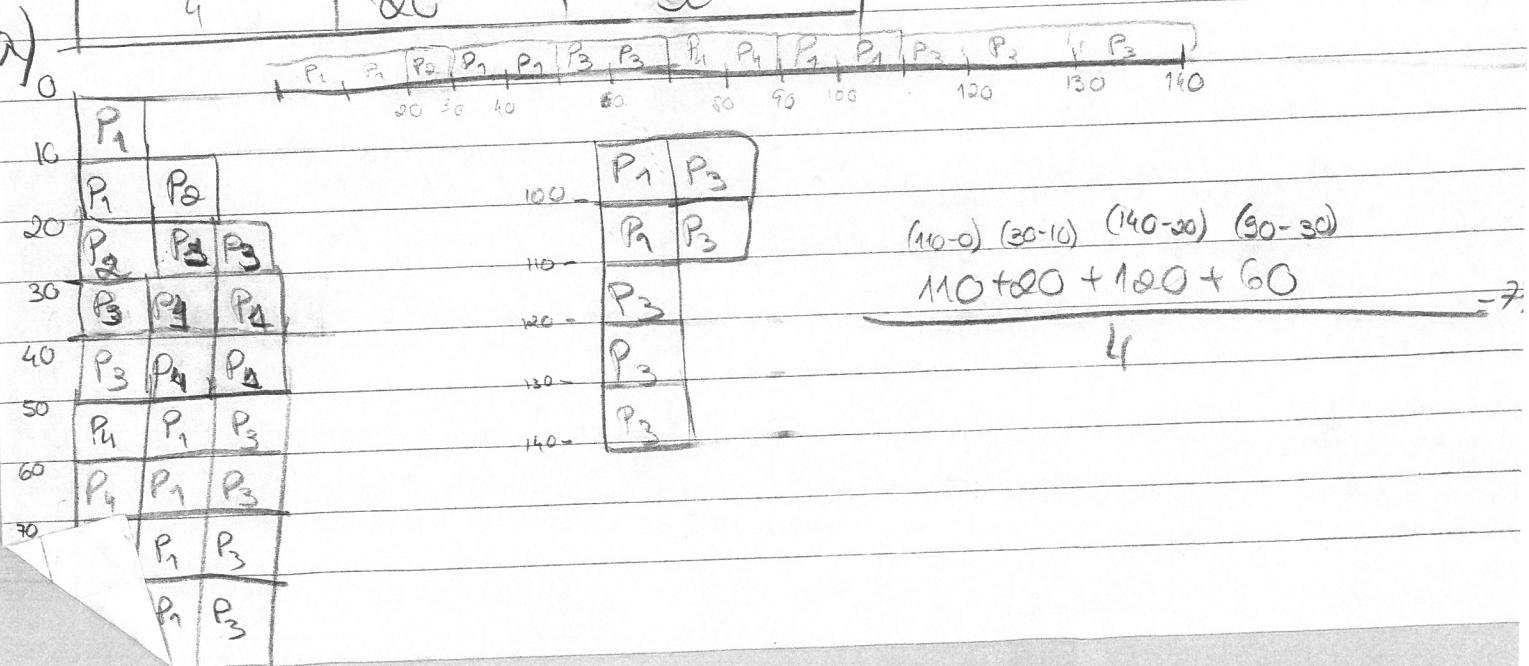
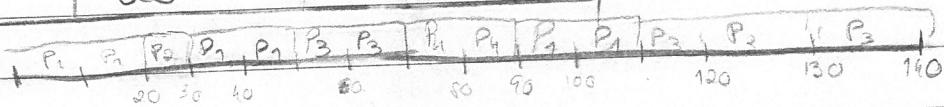


② C.

③ A.

④ Var. Globais, Código

Processos	T Serviço	T Chegada	
1	60	10	20 80
2	10	10	
3	50	20	60
4	80	30	



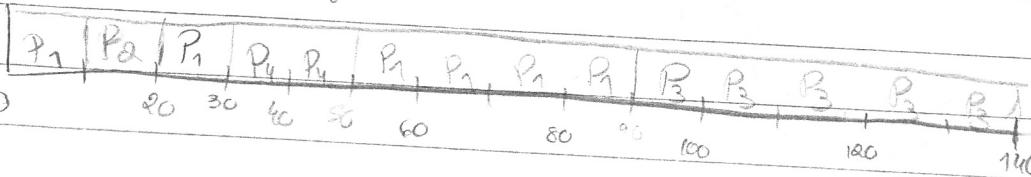
b)

Processo	Tchegada	T serviço
1	0	60
2	10	10
3	20	50
4	30	80

50 40

10

50



$$P_3 = 140 - 20 = 120$$

$$P_1 = 90 - 0 = 90$$

$$P_2 = 80 - 10 = 70$$

$$P_4 = 50 - 30 = 20$$

$$\frac{120 + 90 + 70 + 20}{4} = 60 \text{ ms}$$

⑥

A

26/5/2014

D	A	B	C	D
P ₁	1	2	3	1
P ₂	4	3	2	0
P ₃	0	5	0	2
P ₄	1	2	1	1

D	A	B	C	D
P ₁	1	1	2	0
P ₂	3	1	1	0
P ₃	0	4	0	0
P ₄	0	1	1	1

Rectâveis
6842

$$P_1 : RD = 2101 + 1120 = 3221$$

P₄ está em

$$6842 - 4741 = 2101$$

$$P_3 : RD = 3221 + 0400 = 3621$$

deadlock

$$P_2 : RD = 3621 + 0111 = 3732$$

b) Alloc Matrix

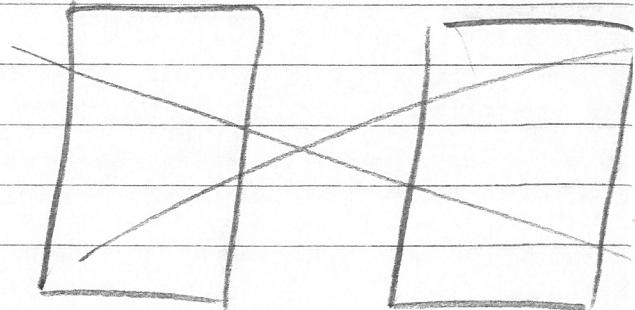
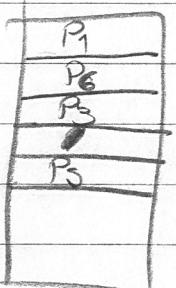
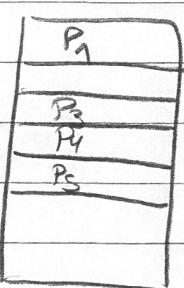
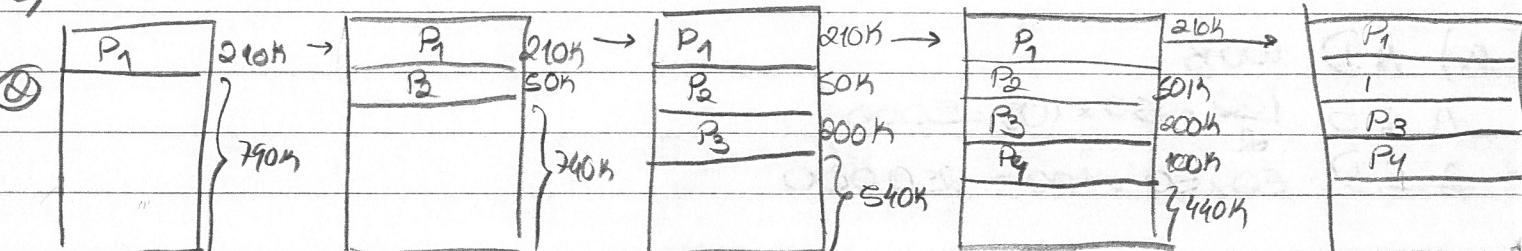
Falta

D	A	B	C	D
P ₁	1	2	2	0
P ₂	3	1	1	0
P ₃	0	4	0	0
P ₄	0	1	1	1

D	A	B	C	D
P ₁	0	0	1	1
P ₂	1	2	1	0
P ₃	0	1	0	2
P ₄	1	1	0	0

Como "Se consegue correr nenhum processo caso se aceitasse o pedido, não existe a possibilidade de haver deadlock então o pedido é Recusado."

c) -



③ 3 níveis $T_{LB} = 10\text{ns}$ $R_{AP} = 10\text{ns}$ Hit? $= 120\text{ns} \rightarrow$

Hit

Miss

T_{LB}

~~T_{LB}~~

$$6(10+100) + 4 - 2e(10 + 400) = 120$$

R_{AP}

~~R_{AP}~~

$$100e + 400 - 410e = 120$$

~~RAM~~ 3 níveis

$$-300e = -290$$

RAM

$$e = 0.97$$

RAM

$$x(T_{LB} + ram) + (1-x)[T_{LB} + (nível + 1) \times ram] = 120$$

$$100xe + (1-x)(410) = 120$$

④ 2 1 6 4 2 64 23 14 6 2 41 36 46

	②	2	2	2	②	2	2	2	⑥	6	6	6	③	3	3	3	
LRU	①	1	1	1	1	1	③	3	3	3	②	2	2	2	⑥	6	6
	⑥	6	6	⑥	6	6	6	①	1	1	1	1	①	1	1	1	
	④	4	4	④	4	4	4	④	4	4	④	4	4	4	④	4	

⑤ Blocos: $100B$

Endereços αB

Directório: $10B$ ($8B$ Nome, $2B$ endereço)

$$\frac{10}{10} = 1000 \Rightarrow 1000 \cdot 10 = 10.000$$

dimensão máxima

a) A.D: $100B$

$$A.I.S: \frac{100}{2} = 50 \times 100 = 5000$$

$$A.I.D: 50 \times 50 \times 100 = 250.000$$

19/Fev/2015

	A	B	C	D		A	B	C	D
P1	1	0	0	1					
P2	0	3	3	3					
P3	2	2	2	0					
P4	0	2	2	1					
	3	7	9	5					

$$\begin{array}{r}
 0101 \\
 +1021 \quad P_1 \\
 \hline
 3295 \\
 +0101 \\
 \hline
 3342 \\
 +0221 \quad P_2 \\
 \hline
 3563 \\
 +0333 \\
 \hline
 3896
 \end{array}$$

a) Ree Disp = $3896 - 3295 = 0101$

b) Com 0101 consigo satisfazer o pedido do processo 1

$$0101 + 1021 = 1122$$

Consigo satisfazer o pedido de P_3

$$1122 + 0220 = 3342$$

consigo satisfazer o pedido de P_4

$$3342 + 0221 = 3563$$

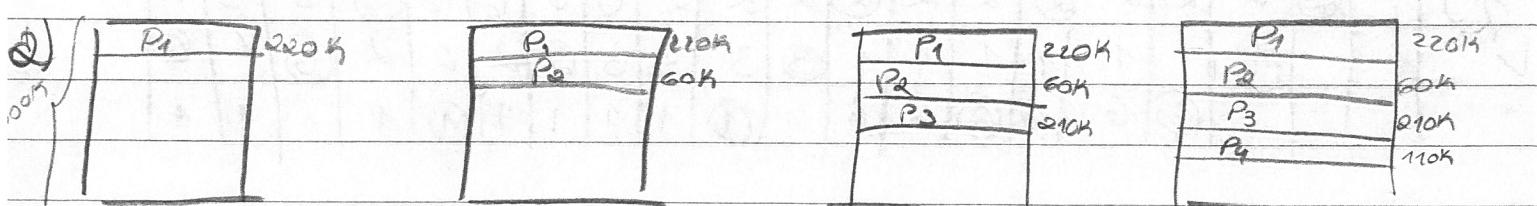
Consigo satisfazer o P_2

$$3563 + 0333 = 3896$$

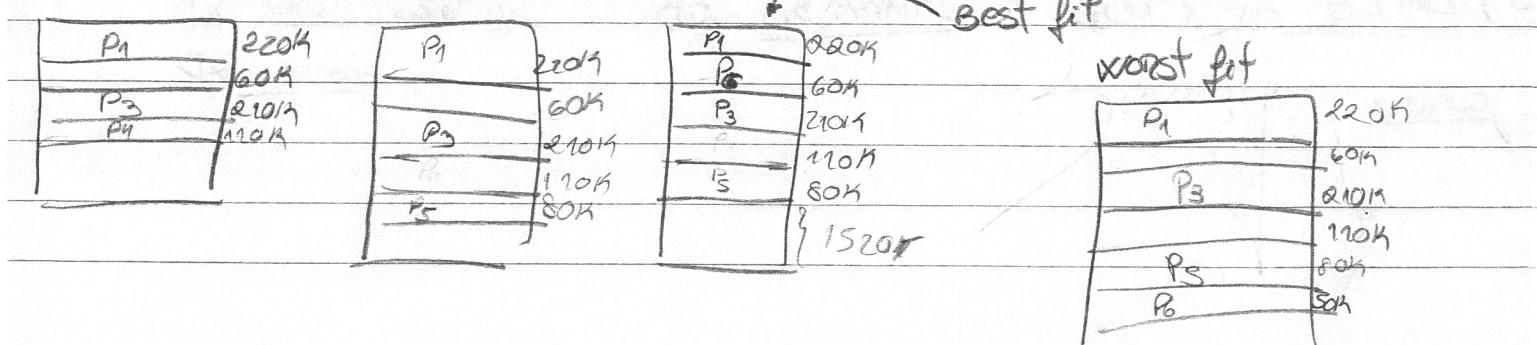
220	000
000	-680
010	2320
000	110
000	600
000	20
000	620

Como consigo correr todos os processos não existe deadlock

c) ? *?



best fit



worst fit

③ 3 níveis

TLB: 5ns

Hit 98%

$$0,98(5 + RAM) + 0,02(5 + 4RAM) = 100$$

Hit	Miss	$4,9 + 0,98RAM + 0,1 + 0,08RAM = 100$
TLB	TLB	$1,06RAM = 95$
RAM	RAM	$RAM = 89,6 \text{ ns}$
RAM	RAM	
RAM	RAM	
RAM	RAM	

6 Um sistema de memória Página:

④ LRU

4 2 1 6 4 2 6 4 2 3 1 4 6 2 4 1 3 6 9 6

④	4	4	4	④	4	4	④	4	4	4	4	④	4	4	④	4
②	2	2	2	②	2	2	②	2	2	2	2	⑥	6	6	6	③
①	1	1	1	1	1	1	③	3	3	3	②	2	2	2	⑥	6

5 Blocos de 1000B Dimensão 2B Diretório Nome 8B

a) estrutura

Endereço 4B

⑥ 2 níveis com TLB

RAIP 50ns

Hit = ?

TLB 5ns

tempo < 60ns

Hit

Miss

$\alpha = \text{Hit}$

TLB

TLB

$\gamma = \text{Miss}$

RAIP

RAIP 7
RAIP / 2 níveis

$$\alpha(\text{TLB} + \text{RAIP}) + \gamma(\text{TLB} + 3\text{RAIP}) = 60 \quad \text{---}$$

$$\Rightarrow \alpha(5 + 50) + \gamma(5 + 3 \times 50) = 60$$

$$\Rightarrow 55\alpha + 155\gamma = 60$$

$$\Rightarrow 55\alpha + 155 - 155\alpha = 60$$

$$\Rightarrow -100\alpha = -95$$

$$\Rightarrow \alpha = 0.95$$

⑦ Piada

⑧ 1 2 4 0 3 5 1 4 5 3 5 1 2 4 5 3 1 6 2 6 4

F F F F F H H H H H N F H H H H F F H H H

①	1	1	1	1	①	1	1	1	①	②	2	2	2	2	②	2	2
②	2	2	5	5	5	5	5	5	5	5	5	5	5	5	1	1	1
④	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
③	3	3	3	3	3	3	3	3	3	3	3	3	3	3	6	6	6

⑨ 4KB

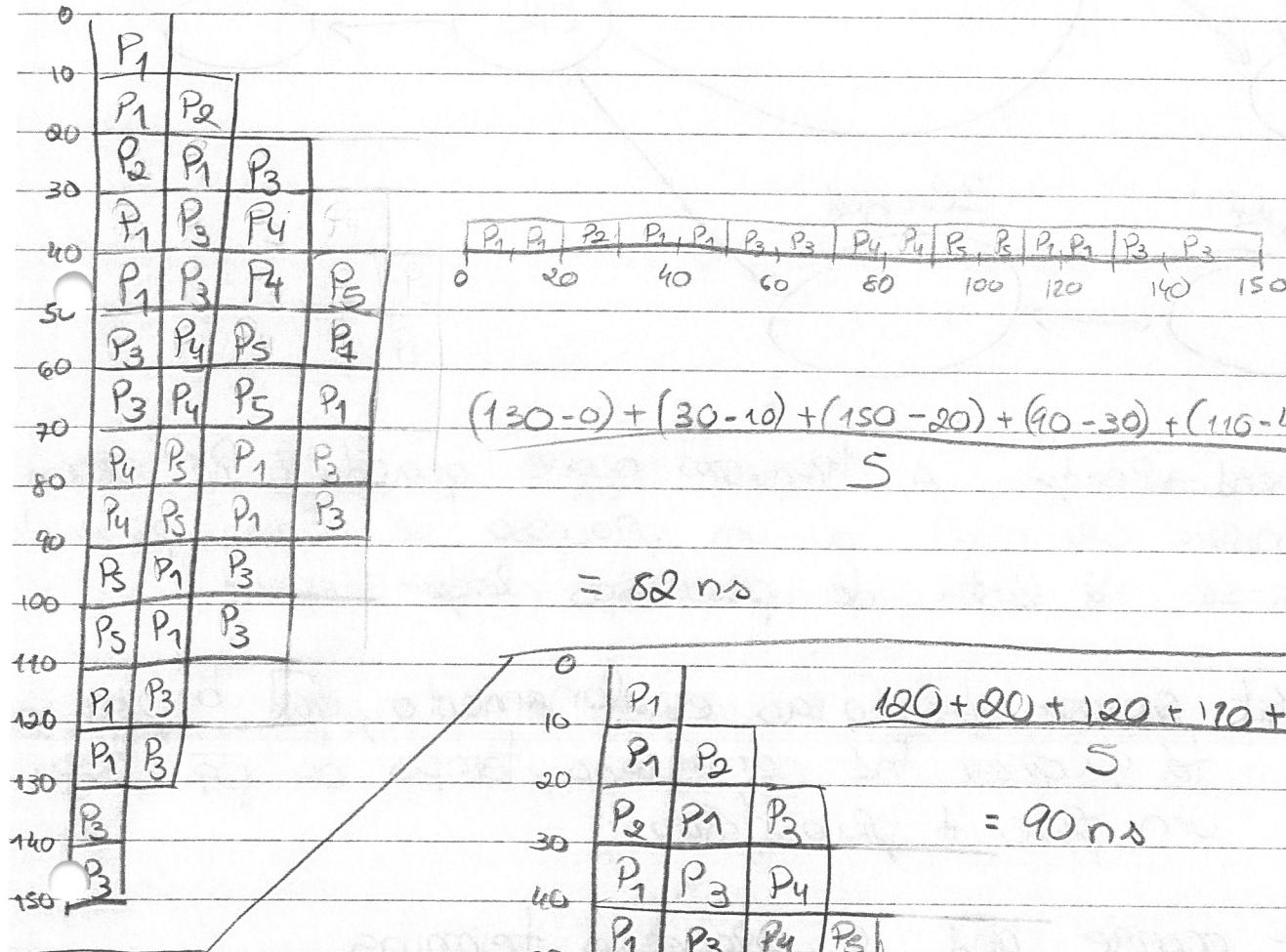
diretório 12B + 4B = 16B

8A.D + 1 A.I.S.

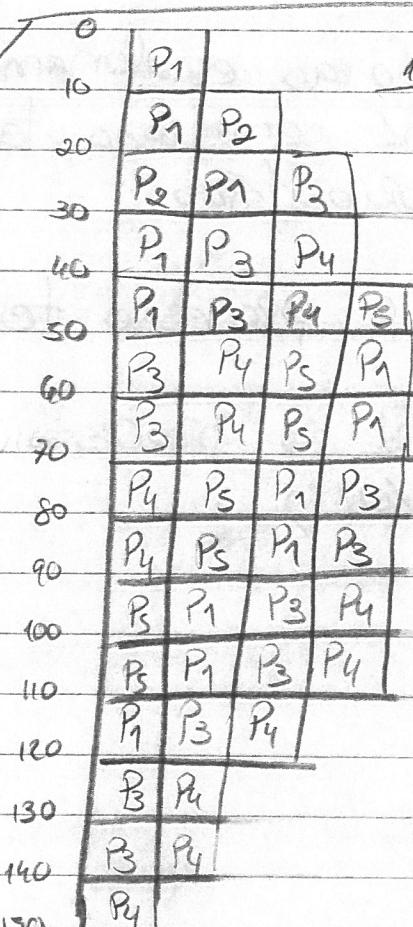
2B

a)

Proc.	t chega de	t servico
1	0	60 1620
2	10	10
3	20	40 20
4	30	20 /
5	40	20 /

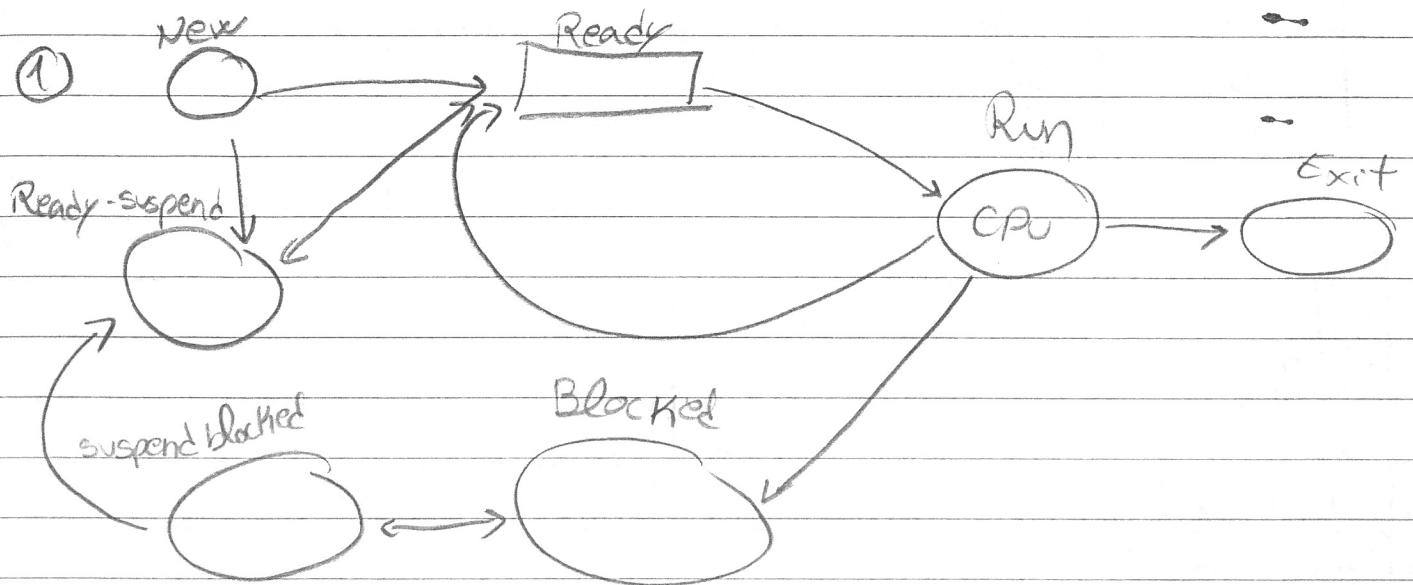


Proc	chrg.	serv.
1	0	50 10
2	10	10
3	20	40 20
4	30	30 10
5	40	20



$P_1 P_4$	P_2	P_3	P_5	P_6	P_7	P_8	P_9	P_{10}
20	40	60	80	100	120	140	150	

Exame deste ano



Ready - suspend → Ready : A transição ocorre quando n̄ há espaço na memória principal ou um processo de maior prioridade encontra-se na lista de processos Ready - suspend

Run - Wait: ocorre devido ao escalonamento, qnd o processo tá a correr há demasiado tempo ou pô chegar um proc. + prioritário.

Run - Exit: ocorre qnd o processo termina

Run - Blocked: ocorre qnd o processo necessita de alguma informação do disco, I/O(...)

Column Matrix

④	A	B	C	D
P ₁	4	3	2	2
P ₂	1	2	4	1
P ₃	1	2	1	1
P ₄	0	5	0	2

Alloc Matrix

	A	B	C	D
P ₁	3	1	1	0
P ₂	1	1	2	1
P ₃	0	1	0	0
P ₄	0	4	0	0

Req

16 - 8 = 42

P₃ (0,0,1,1)

Alloc Matrix

	A	B	C	D
P ₁	3	1	1	0
P ₂	1	1	2	1
P ₃	0	1	1	1
P ₄	0	4	0	0

6842
4742

2100

Falta

	A	B	C	D
P ₁	1	2	1	2
P ₂	0	1	2	0
P ₃	1	1	0	0
P ₄	0	1	0	2

4 7 4 2

Como não consigo
conseguir nenhum proc.
ent. caso aceitasse o

pedido ia haver a
possibilidade de existir
deadlock logo o pedido
seria negado.

⑤ 1 2 1 4 3 1 5 4 1 6 4 1 7 1 4 3 2 1

FF H F F H F H H F H H F H H F H P

①	1	①	1	1	①	1	1	①	1	1	1	1	①	
LRU	②	2	2	2	⑤	5	5	5	5	5	⑦	7	7	⑦
	④	4	4	4	④	4	4	4	4	4	④	4	4	4
	③	3	3	3	③	3	3	3	3	3	③	3	3	3
	⑥	6	6	6	6	6	6	6	6	6	④	4	4	4

⑤ 1 2 1 3 4 1 5 3 1 6 3 1 7 1 3 4 7 1

F F H F F H F H H F H H F H H F H H

①	1	①	1	1	①	1	1	①	1	1	1	1	1	①	
LRU	②	2	2	2	⑤	5	5	5	5	5	⑦	7	7	7	⑦
	③	3	3	3	③	3	3	3	3	3	③	3	3	3	
	④	4	4	4	4	⑥	6	6	6	6	6	④	4	4	
	⑥	6	6	6	6	6	6	6	6	6	④	4	4	4	

④	A	B	C	D
P ₁	4	3	2	2
P ₂	1	2	4	1
P ₃	0	5	0	2
P ₄	1	2	1	1

(0,0,1,1)

	A	B	C	D
P ₁	3	1	1	0
P ₂	1	1	2	1
P ₃	0	4	0	0
P ₄	0	1	0	0

Rec 16842

Como não consigo correr nenhum processo existe a possibilidade de haver deadlock

Alloc

	A	B	C	D
P ₁	3	1	1	0
P ₂	1	1	2	1
P ₃	0	4	0	0
P ₄	0	1	1	1
	4	7	4	2

Falta

	A	B	C	D
P ₁	1	2	1	2
P ₂	0	1	2	0
P ₃	0	1	0	2
P ₄	1	1	0	0

Logo recusa-se o pedido

$$\textcircled{6} \quad \begin{array}{l} \text{Blocos: } 1\text{KB} \\ \text{Endereço: } 2\text{B} \end{array} \quad \begin{array}{l} \text{Diretório: } 32\text{B} \\ \text{Endereço: } 64.000 \end{array} \quad \frac{\alpha = 64.000 \text{ e } 2 = 0.048000}{32}$$

A.D: 1024B

$$A.I.S: \frac{1024}{2} \times \frac{1024}{2} = 524288\text{B}$$

$$A.I.D: \frac{1024}{2} \times \frac{1024}{2} \times \frac{1024}{2} = 268435456\text{B}$$

- Escrever código (implementação de testes), como testes (pseudo código) a gestão de configurações;
- Quais os comandos git para executar determinadas tarefas,
- Quais as metódos logias
- descrição de problema → implementar etapas

ppf41

- git checkout, git merge, git push [sincronizar o branch de feature com o branch master)

origin - Repositório atual, "New game", é o projeto q se está trabalhando
Master - É o branch master principal deste projeto

git tag · 1.0.0 1b2e1d63ff
 O " " representa os 10 primeiros caracteres
~~do id de commit~~

git log Usar menos caracteres do id de commit, ele só precisa de ser

- status
- pull
- branch
- add
- commit
- reset
- git fetch origin

gestão de memória: subdivisão da memória de modo a aceder/acomodar múltiplos processos. Esta tarefa é entregue ao sistema operativo e chama-se gestão de memória.

• Frame

bloco fixo (tamanho) da memória principal

• Página

bloco de dados e/ tamanho fixo que reside na memória secundária (disco).

Uma página de dados pode ser temporariamente espalhada para uma frame localizada na memória principal.

• segmento: bloco de dados

de tamanho variável que

reside na MS. Um segmento

é copiado para uma região

acessível e disponível na MP

(segmentação) ou o segmento

pode ser dividido em páginas

que podem ser copiadas individualmente para MP (segmentação + paginação)

Requerimentos / Requisitos para a Gestão de Memória:

1. Relocation (relocalização)

2. Proteção (segurança)

3. Partilha

4. Organização: lógica

5. Organização: física

1. A memória principal

é geralmente partilhada por todos os processos.

Quando um programa é trocado para disco, seria limitado especificar quando passaria a ir para dentro do disco que tem que recolocar na mesma região da MP que antes. Em vez disso, será necessário redorar o processo numa diferente área da MP.

strlen Retorna o Comprimento da string fornecida (não conta)

strcpy Copia a string origem para a string destino

strncpy

strcmp

strcat String origem permanece inalterada e seu anexado ao fim da " de destino

strcmp Compara a string1 com a string2

- se forem idênticas retorna 0

- caso contrário retorna não zero

Dw t₁, 0(t₀)

t₁ = "Hello"

3	00
2	0 H
1	e l
0	l o

Sb t₁, 0(t₀)

srcl t₁, t₁, 8

Sb t₁, 1(t₀)

srh t₁, t₁, 8

Sb t₁, 2(t₀)

srcl t₁, t₁, 8

Sb t₁, 3(t₀)

Capítulo 7: 305 - 336 → 307 - 326

Capítulo 8: 340 - 391 → 341 -

do ex 4.

$$d) (\phi \wedge \psi) \vee \theta \vdash (\phi \vee \theta) \wedge (\psi \vee \theta)$$

1	$(\phi \wedge \psi) \vee \theta$	H	(Hipótese)
2	θ	[H \rightarrow]	(Hipótese temporária)
3	$\phi \vee \theta$	$\alpha(V_{\phi}^t)$	
4	$\psi \vee \theta$	$\alpha(V_{\psi}^t)$	
5	$(\phi \vee \theta) \wedge (\psi \vee \theta)$	3, 4 (V_{θ}^t)	
6	$\phi \vee \theta$	[H \rightarrow]	
7	ϕ	6 (\neg_{ϕ}^t)	
8	ψ	6 (\neg_{ψ}^t)	
9	$\phi \vee \theta$	7 (V_{θ}^t)	
10	$\psi \vee \theta$	8 (V_{θ}^t)	
11	$(\phi \vee \theta) \wedge (\psi \vee \theta)$	9, 10 (\wedge^t)	
12	$(\phi \vee \theta) \wedge (\psi \vee \theta)$	1, 2-5, 6-11 (\vee^t)	eliminação disjunção

Portanto, $(\phi \wedge \psi) \vee \theta \vdash (\phi \vee \theta) \wedge (\psi \vee \theta)$

$$e) \phi, \phi \rightarrow \psi \vdash \psi$$

$$f) \phi, \phi \rightarrow \psi, \psi \rightarrow \theta \vdash \theta$$

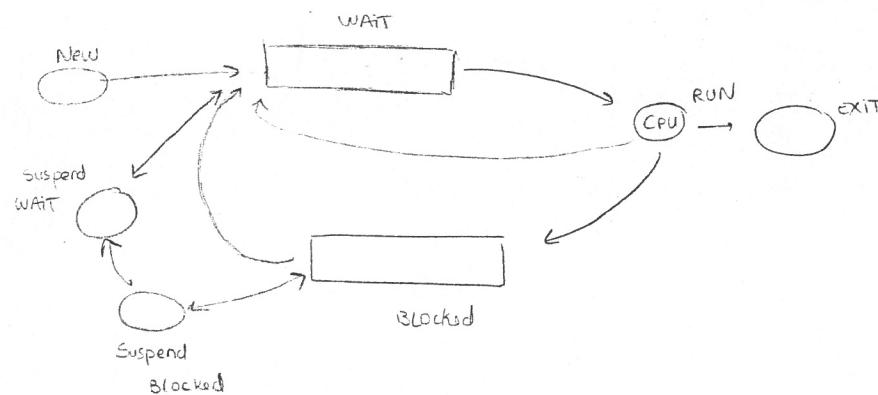
1	ϕ	H	1	ϕ	H
2	$\phi \rightarrow \psi$	H	2	$\phi \rightarrow \psi$	H
3	ψ	1, 2 (\rightarrow)	3	$\psi \rightarrow \theta$	H
			4	θ	
			5		

$$g) \phi \wedge \phi \vdash \phi$$

1	$\phi \wedge \phi$	H
2	ϕ	

• 1º frequência

2.



RUN → EXIT : quando um processo termina

RUN → Blocked : quando o processo precisa de informações

RUN → WAIT : Devido ao escalonamento

3. A informação que existe num processo são os dados, o Código e o PCB. O PCB é composto pelos Estados, o Program Counter, o ID, os registos, I/O, os ficheiros, o Reban e o Tempmark, este último é que são do tipo de escalonamento

4. As Threads de kernel tem vantagem em relação às Threads de user level, porque as Threads de user level são consideradas processos pelo sistema operativo, então as Threads de user level apenas podem correr em 1 CPU com n CPUs e são bloqueantes, ou seja, quando precisam de informação supostamente o processo irá ser obrigado a parar e a ir para o Blocked, o que faz com que a principal vantagem das Threads deixe de existir em Kernel conseguindo aproveitar o Parallelismo. As Threads de user level tem vantagem em relação as threads de user level na troca dos processos, a troca de processos em user level é mais rápida que em kernel porque não possui a interrupção do sistema operativo enquanto que a ação da flutuação de processos em kernel necessita da intervenção do sistema operativo.

5. Partilhado:

Dinâmicos (variáveis globais)

Código

Dados

Não partilhado

Ficheiros Arquivos

o que está no PCB

ID

Estado

o PC

Variáveis locais ...

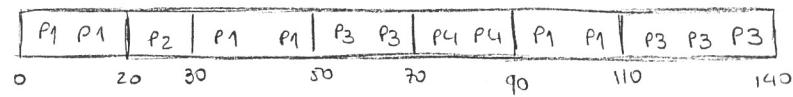
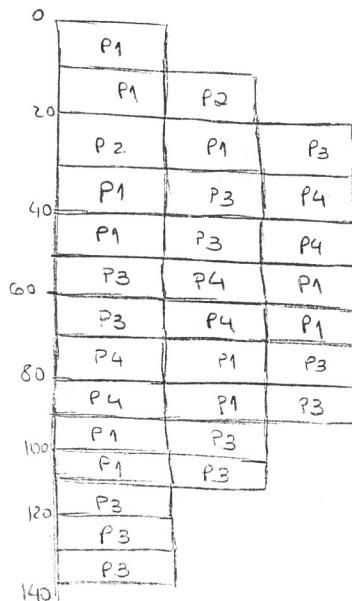
STACK

registos do cpu

5.

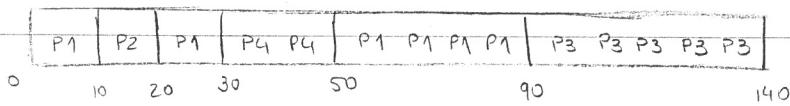
Processo	t chegada	t serviço
1	0	60
2	10	10
3	20	50
4	30	20

5.a) RR, T Quantum = 20



$$\frac{110 + 20 + 120 + 60}{4} = 77,5 \text{ ms}$$

5.b) SRT



$$\frac{90 + 10 + 20 + 120}{4} = 60 \text{ ms}$$

6.

Processo	T serviço	Período	Período de 8 ms
A	30	120	$\frac{3}{12} = \frac{1}{4}$
B	10	30	$\frac{1}{3}$
C	20	60	$\frac{1}{3}$

$$n \times (2^{\lfloor n \rfloor} - 1)$$

$$3 \times (2^{\lfloor 3 \rfloor} - 1) = 0,78$$

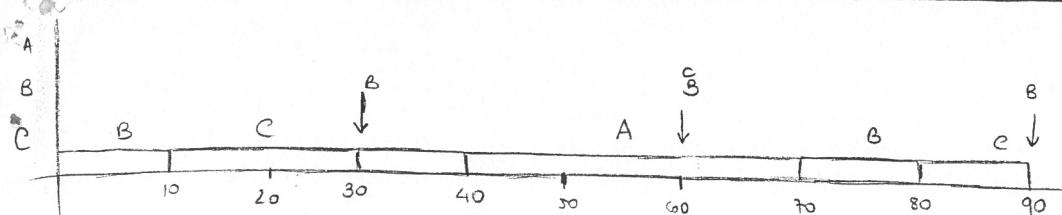
↓

Para que o Algo

ritmo res. fique garantida
mente o espaço que os processos
ocupam no cpu tem de ser igual

Como o espaço que os processos ocupam no CPU é 1 não sabemos se o Algoritmo

é ou menor a 0,78.



O Processo que possui um período mais curto é o mais prioritário.

Neste caso o Algoritmo de escalonamento RTES funciona porque todos os Processos terminam antes da sua deadline.

Portas

Partilhados

código
Dados
Variáveis globais

N Partilhadas

Ficheiros Abertos
O que está no PCB
Registo da CPU
Registos de Ativação de funções
STACK
Variáveis locais

PCB

PC
I/O
ID
Registros
Ficheiros
Estado
Tipo de Estacionamento

$$lotação = 100$$

$$Guarda = 0$$

$$Porta = 1$$

to signal (Guarda)

wait (Guarda)
if n=0 then wait (Porta)
wait (lotação)
n=n+1

entra ()

estar - no - jardim();

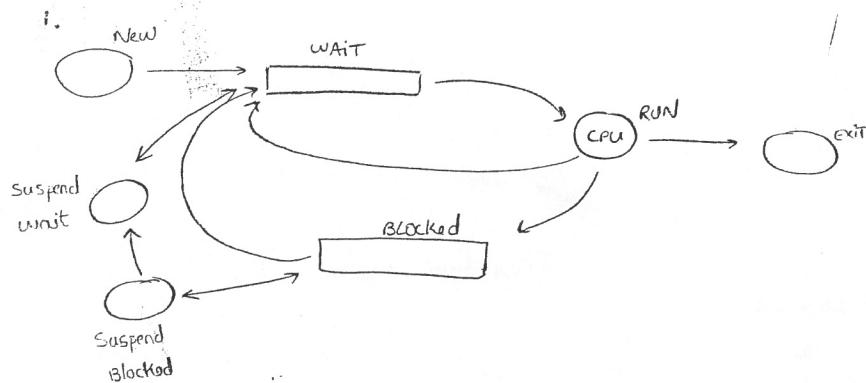
n=n-1
sair();
signal (lotação)
if n=0 then signal (Porta)

Signal (Guarda)

wait (Guarda)

1.2. frequência

(2014)



RUN → EXIT : ocorre quando um processo termina

RUN → blocked : quando o processo precisa de uma informação, que pode ser do disco, I/O, etc..

RUN → WAIT : ocorre devido ao escalonamento, quando o processo está a correr à demasiado tempo e porque chegou um processo mais prioritário

2. A informação que é guardada no PCB é os estados, o Program Counter, o ID, os registos, I/O, os ficheiros e Reiben e o Temporeh, sendo que o Reiben e o Temporeh são dois tipos de escalonamento. Um Processo Possui também Código e Dados.

3. As vantagens das Threads em relação aos processos são a utilização de menos RECURSOS, porque as Threads são processos que partilham uma parte do Código e por isso à mudança de contexto e a criação das Threads é mais rápida que a criação e a mudança de contexto são mais lentas. As Threads são processos que possuem uma parte do mesmo Código e que foram pensados para trabalhar em Paralelo e em n CPU's, o que é uma vantagem em relação aos processos clássicos. Outra vantagem no uso de Threads é quando é necessário uma informação se o processo precisa de uma informação este para de correr e tinha de ir para o blocked, mas Thread isso não acontece este não para nem vai para blocked mas continua a correr enquanto espera a informação.

4. Dados Partilhados:

Código

Dinâmicos (variáveis globais)

Dados

Dados não partilhados:

Ficheiros abertos

ID
Estado

O PC

Variáveis locais

Registo de Ativação das funções

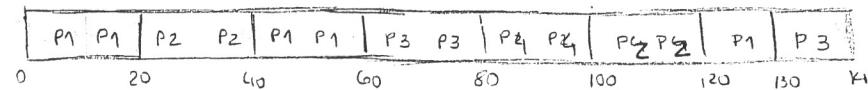
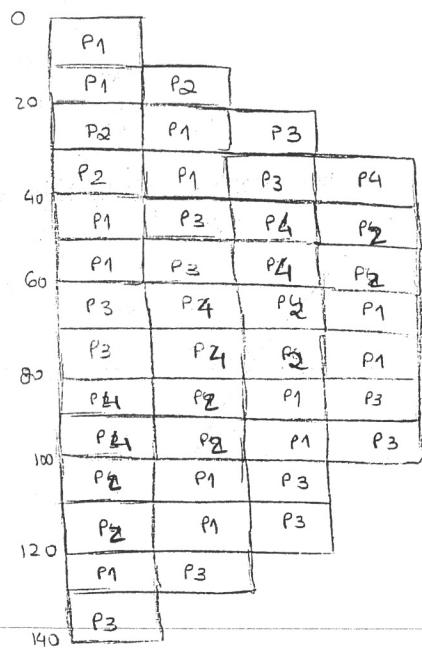
STACK

Registros do CPU

5.

Processo	T chegada	t serviço
1	0	50
2	10	40
3	20	30
4	30	20

$$T_{Quantum} = 20$$



$$\frac{130 + 110 + 120 + 70}{4} = \frac{430}{4} = 107,5 \text{ ms}$$

6. Período = 80 ms

Processos	T serviço
A	20
B	10
C	10

Período

$$80 \quad 2/8 = 1/4$$

$$20 \quad 1/2$$

$$40 \quad 1/4$$

—

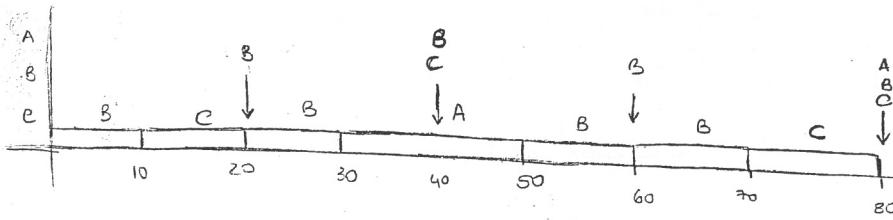
$$n \times (2^{1/n} - 1)$$

$$3 \times (2^{1/3} - 1) = 0,78$$

, ou seja, para escalonamento ser de certeza possível apenas se puder utilizar 0,78 da CPU.

Como a Percentagem de CPU que os processos vão utilizar é 1 e como apenas temos ter a garantia que o momento RTOS funciona se a Percentagem de CPU utilizada for 0,78 então neste caso o escalonamento pode ser não garantido tempo utliza a garantia se o escalonamento fununna

Caso cheguem todas ao mesmo tempo.



Neste caso o escalonamento RPS funciona porque todos os processos terminam antes da sua deadline.

PCB

~~ficheiros Abr~~

ficheiros

PC

I/O

ID

Registros

Tipo de escalonamento

Estado S

Pachitham

Código

Dados

Variáveis globais

N Pachitham

ficheiros Abrertos

PCB

registro da CPU

registro da Atividade de funções

STACK

Variáveis locais

Licenciatura em Engenharia Informática
 Sistemas Operativos 1- 1^a frequência – 7 de Abril de 2016
 Departamento de Informática - Universidade de Évora

Justifique as suas respostas apresentando os cálculos, quando aplicável.

- 1 ✓ 1. Descreva graficamente o modelo de 7 estados.
- 2 ✓ 2. Indique a hipótese correta. Um processo transita do estado RUN para o estado BLOCKED porque...
 A - Terminou o tempo que estava reservado para correr no CPU e por isso o processo é interrompido.
 B - O Processo precisa de esperar na fila de WAIT.
 C - O processo executou uma instrução de I/O e fica à espera de um evento.
 D - Ocorreu um evento enquanto esperava por dados.
- 1 3. Indique a hipótese correta.
 A - O uso de threads só é vantajoso com CPUs múltiplos.
 B - Com apenas um CPU o uso de threads permite aumentar a velocidade de resposta usando hardware de modo paralelo.
 C - O uso de threads não é aplicável com CPUs múltiplos.
 D - O uso de threads com CPUs múltiplos, torna-se mais lento.

- ✓ 4. Assinale quais dos seguintes são dados partilhados entre *threads* dum mesmo processo: Program counter, Registos temporários do CPU, Variáveis globais, Código, Process ID, Estado, Ficheiros Abertos.

- ✓ 5. Considere a seguinte tabela com o instante de chegada de cada processo à fila ready e com a duração do tempo de serviço no CPU:

processos	T chegada	T serviço
1	0	100 60
2	10	50 10
3	20	30 50
4	30	20 20



- 2 6. Calcule o tempo médio para terminar um processo (*turnaround time*) para os algoritmos:

- ✓ 5.1 - RR - round robin, quantum Q=20. *fazer conta*

- ✓ 5.2 - SRT short remaining time *fazer conta*

Nota: admite (se necessário) que num instante em que se interrompe um processo (se o algoritmo de escalonamento o impuser), primeiro passa-se o processo do CPU (RUN) para a fila de READY e só depois se testa se há processos novos para entrar na fila de ready (de NEW para READY).

6. Considere a seguinte tabela com o período e o tempo de serviço de três tarefas de tempo-real.

processos	T serviço	Período
A	30	120
B	10	30
C	20	60

- ✓ 6.1 Defina o escalonamento dos processos num período 120 ms, com o algoritmo de escalonamento "RMS-rate monotonic scheduling"

- ✓ 6.2 Indique Justificando se é garantido que existe escalonamento RMS sem violação de deadlines.

7. Considere o semáforo x, com as funções usuais *wait(x)* e *signal(x)* inicializado a 1.

Considere o seguinte programa que é lançado por vários processos em paralelo

While(true) do {

N=N+1
wait(x) → - no x
P=P+1
função()
P=P-1
signal(x) → + no x
N=N-1

Indique justificando se:

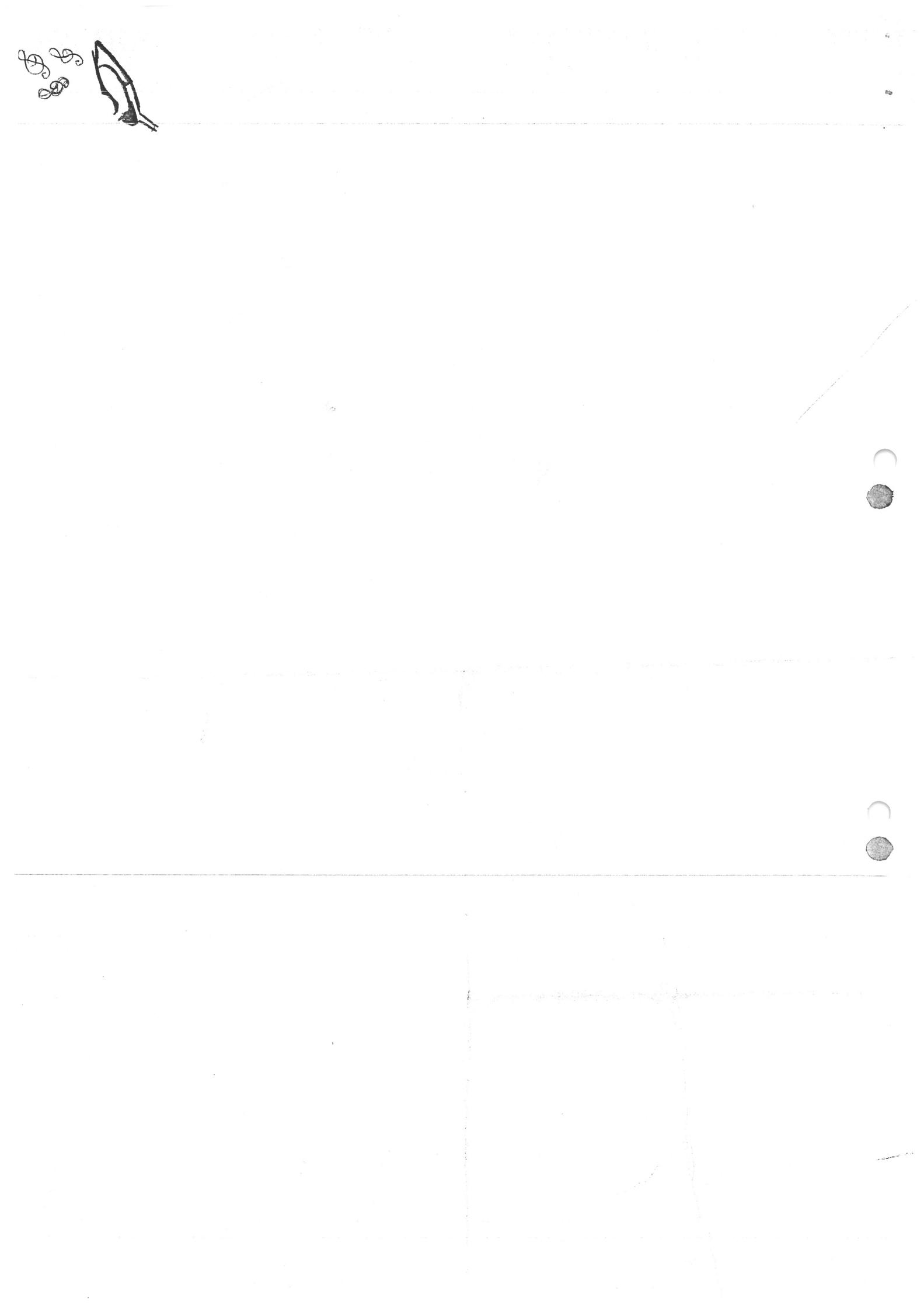
- 7.1 se o valor de N é sempre previsível

- 7.2 se o valor de P é sempre previsível

imprevisível

por estarem dentro de uma função

= 13
=



Licenciatura em Engenharia Informática
Sistemas Operativos 1- Exame – 16 de Março de 2017
Departamento de Informática - Universidade de Évora

1. Descreva graficamente o modelo de 7 estados e explique as transições de um processo do estado "RUN" para outros estados.

2. Indique a hipótese correta. O uso de threads é vantajoso...

- A - apenas se existe mais do que um CPU e várias tarefas.
 B - se existirem N tarefas, todas diferentes, com vários CPUs.
 C - apenas se só existe uma tarefa e apenas um CPU.
 D - se existirem N tarefas do mesmo tipo com uso de CPU e uso de I/O.

3. Considere a seguinte tabela com o instante de chegada de cada processo à fila ready e com a duração do tempo de serviço no CPU:

Proc	t de chegada	t de serviço
1	0	50
2	20	40
3	30	30
4	40	20
5	50	10

Calcule o tempo médio para terminar um processo (*turnaround time*) para o algoritmo RR – round robin, quantum Q=20. Admita (se necessário) que num instante em que se interrompe um processo (se o algoritmo de escalonamento o impuser), primeiro passa-se o processo do CPU (RUN) para a fila de READY e só depois se testa se há processos novos para entrar na fila de ready (de NEW para READY).

4. Considere a seguinte tabela com o período e o tempo de serviço de três tarefas de tempo-real. Defina o escalonamento dos processos num período 80 ms, com o algoritmo de escalonamento "RMS-rate monotonic scheduling"

processos	T serviço	Período
A	10	30
B	40	120
C	10	40

Indique também, justificando, se à priori é garantida, ou não, a existência de um escalonamento para o exemplo acima, usando RMS.

5. Usando semáforos, e indicando a sua inicialização, implemente um solução para o seguinte problema: considere um museu com uma lotação total de 300, mas que tem uma lotação máxima de 200 pessoas, na sala 1, e uma lotação de 200 na sala 2. Cada visitante entra no museu, ficando um tempo na sala 1, passando à depois à sala 2, saindo depois do museu. Implemente em pseudo-código os processos "visitante", cumprindo as restrições enunciadas. Considere os seguintes procedimentos que pode usar: `entrar_museu()`, `sair_museu()`, `estar_sala1()`, `estar_sala2()`.



Sistemas Operativos

1

Os sistemas operativos podem funcionar pelo Kernel monolítico ou por Camadas.

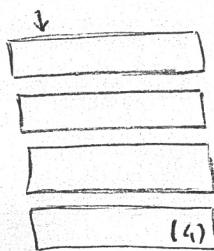
Kernel monolítico

os sistemas operativos que funcionam pelo Kernel monolítico tem no seu interior funções (ficheiros, memória, mo), nos portátiles.

Inicialmente o sistema monolítico foi pensado para 1 utilizador e 1 processo, sendo que o primeiro sistema dispositivo com este sistema o MS-DOS

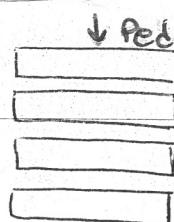
Exemplo: Linux (n utilizadores, n processos)

Existiram algum sistemas que inicialmente tinham um desenho bastante estruturado, mas que mais tarde esse tipo de desenho ~~deixou~~ acabou por ser um problema e o que inicialmente não iria ter um sistema operativo monolítico no final acabou por o ter de utilizar, utilizando-o com uma certa estrutura, um exemplo deste tipo é o windows nt4, inicialmente era:



- o problema é que quando eu tenho ir para o 4, tenho de passar por outra Camada, o que não é ~~é~~ a melhor solução, por isso no final acabaram por fazer o sistema operativo através de um sistema monolítico

Por Camadas



↓ Pedido de serviço é preciso fazer a gestão dos recursos de forma eficiente (atendendo às regras de autorização ou de uma não autorização, competição pelas recursos)

A Camada abaixo tem de aceder ao pedido da Camada acima.

O melhor para Software Complexo é um sistema em Camadas.

Por exemplo o Unix.

UK (processos, microkernel)

A questão dos ficheiros, gestão de memória virtual, dispositivos I/O (teclado, ecrã, rede, ...), são coisas ultra elementares de funcionalidades que tem a ver com a gestão de processos,

Não são um sistema operativo para o desktop mas sim para tempo real ou embedded.

ex: QNX

A principal função são a gestão dos processos (do S.O.)

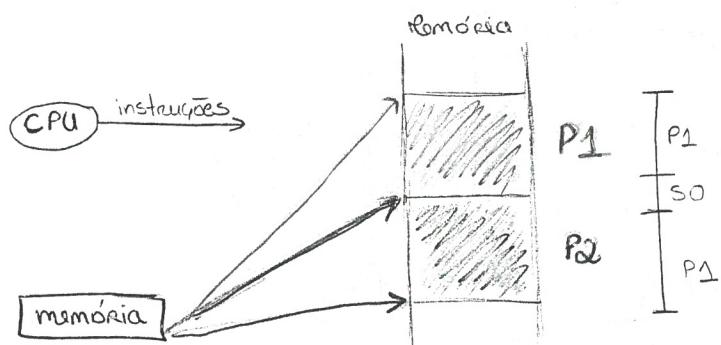
Processos

Existem 2 tipos de actos nos Processos:

- Criação (pode ser feito novo ou Cópia do Processo antigo)
- Destrução

Vários processos num Computador têm um Código diferente e utilizam memória diferente, para seguirmos os processos temos de ter a certeza que nenhum processo destrói outros processos.

Segmentation fault → Caso eu tentar ir a uma zona da memória que não seja a zona do processo que está a ocorrer aparece esta mensagem.



Nem o Processo 1 nem o Processo 2 podem meter na memória apenas o sistema operativo o pode fazer.

Para cada processo é necessário existirem limites na memória para cada processo, existem instruções que apenas o sistema operativo pode fazer e que os processos não podem, porque o sistema operativo tem Código privilegiado vai ser o sistema operativo que irá mudar os apontadores que delimitam os limites de cada processo na memória, consoante o processo que está a ser executado.

Para se diferenciar o sistema operativo dos processos e das operações que cada um pode fazer, criou-se o User mode e o System mode (CM).

* Como fazer a Passagem do User mode para System mode e de System mode para User mode

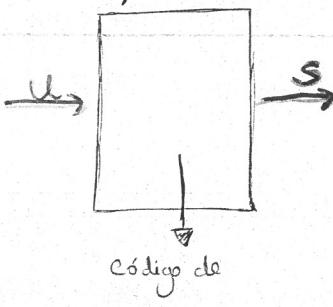
Para se passar do System mode para User mode basta ter uma instrução que possa de System mode para User mode, como estamos no System mode podemos fazer esta passagem, mas para passar do User mode para System mode já não o podemos fazer através de uma instrução porque estamos em User mode e que não nos permite fazer isso porque o User mode não tem todas as funções. Para conseguirmos então passar do User mode para System mode temos de utilizar funções

que já foram criadas que se chamam as System Call.

S.O.

②

System Call

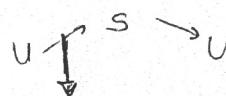


quando não te nós estamos em User mode não temos todas as funções que existem porque não temos permisão.

As system call são uma espécie de interfaces que tem um Congunto de funções que são permitidas em User mode e que vão permitir que algo aconteça em System mode e que quando acaba de fazer o que é necessário volta para o User mode, porque de System mode para User mode é fácil passar.

Existem funções de escalonamento (Quando estudamos 1 processo para outro, para a execução)

o Hardware, o timmer ou a resposta do disco.



Apenas possível por System Code, escrito pelo sistema operativo ~~por~~ por um Congunto de funções específicas

Processos

o que possui?

Cada processo está associado a uma tarefa, caso a tarefa seja Complexa existem vários processos para a mesma tarefa.

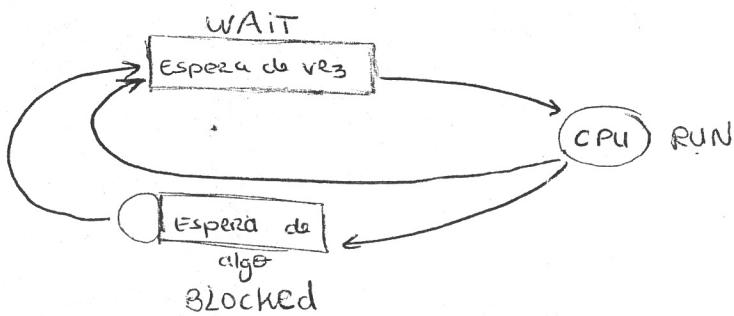
Cada processo tem associado um Congunto de instruções e código. Um código pode ser utilizado em mais que um processo e o código em dois processos pode ser igual ou não

Código ≠ Processo

Cada processo tem também associado um Congunto de dados que pode ser alterados.

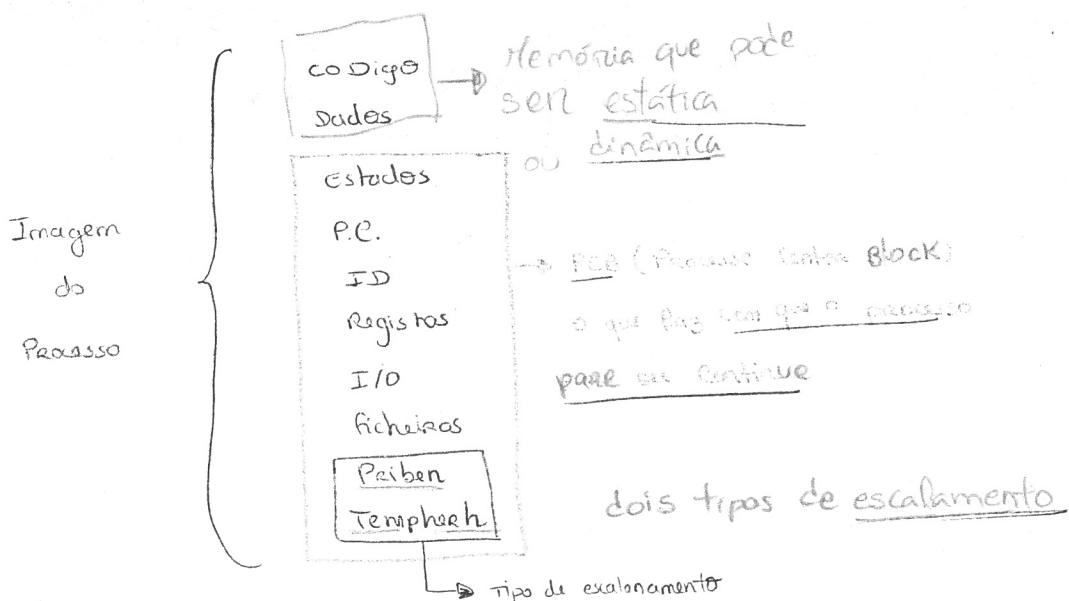
O código e os dados exigem espaço em memória.

Caso existam mais processos que espaço no CPU para eles serem executados, os processos alguns deles não vão poder correr, porque os processos não podem correr todos ao mesmo tempo os processos ou estão a correr ou estão em espera, estes são dois dos muitos estados que podem existir. Quando os processos estão à espera, podem estar à espera que o CPU tenha espaço para o processo correr ou então pode estar à espera de alguma coisa, (do disco), logo a espera que cada processo pode vir a ter pode ser diferente dependendo da razão.



os Processos tem de ter também alguma informação auxiliar, por exemplo quais os estados dos Processos, qual o sítio do Código em que o Processo parou (onde está o Program Counter), têm de ter um ID (Identificador Único, que identifica o processo).

registos variáveis que correspondem a informação acessória, ficheiros que podem estar ^{abertos} no momento da execução do Processo e que tem de estar abertos depois da sua execução e I/O (input e output).



Escalonador obriga os Processos a saírem do CPU para que outros Processos Comecem a Correr, isto pode acontecer quando um Processo tem um nível de Prioridade, ou quando um processo já foi criado a muito tempo e é necessário existir rotatividade dos Processos, esta informação é dada pelo sistema operativo e dependendo do mesmo o que irá influenciar o escalonamento será diferente.

PC gerido pelo Sistema Operativo Apesar

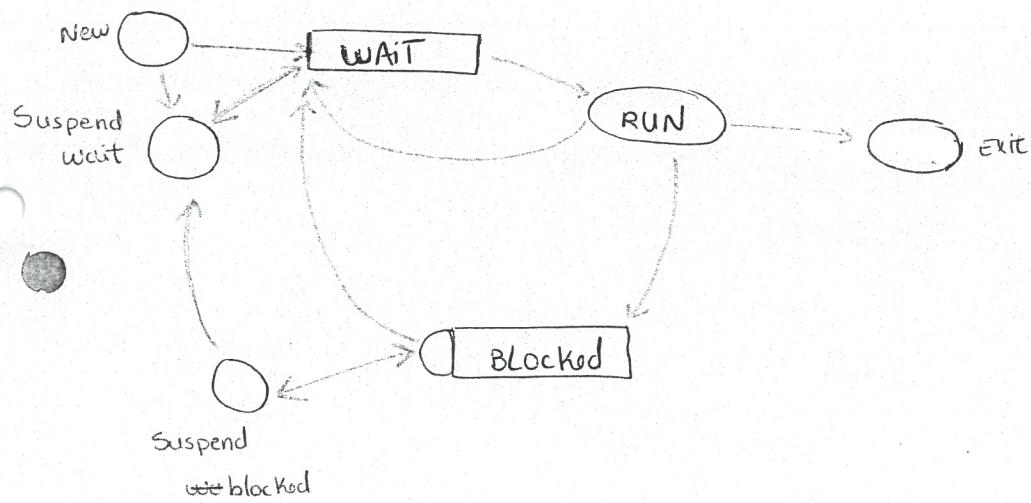
o código e os dados podem ser alterados, até por eles próprios dependendo dos Processos

Succesão de Dados

um processo quando é criado pode logo não ir para o estado WAIT e só irá para esse estado se tiver pronto a correr. os processos para serem criados precisam de ocupar espaço na memória

- então se a memória tiver sobrecarregado o processo pode ir para stand by dessa maneira
- o processo não irá ocupar memória por estar no estado WAIT e só quando a memória estiver livre
- Sobrecarregado é que os processos vão para o estado WAIT. Quando os Processos acabam de ser executados, seja de correr o processo pode desaparecer ou pode ir para o estado exit, onde irão ficar um balde de tempo até podarem desaparecer.

Modelo de 7 estados



Quando um processo está no estado Suspend blocked por vezes a informação que está em memória do processo pode ser passada para o disco de modo a que a memória fique com mais espaço, para que não misturarmos processos que precisam de algo com aquela que estão a esperar do "espaço" no CPM.

Ligações a estados Suspend.

Poderíamos construir um modelo de 9 estados com o mesmo esquema o que mudaria iria ser a prioridade dos processos.

(a) antes de os processos irem para o wait, considerante a prioridade dos processos, sendo que os processos prioritários iriam primeiros que os restantes para o estado wait.

Transição de Estados

RUN → BLOCKED (quando o processo precisa de uma informação qualquer)

RUN → EXIT (quando o processo termina)

RUN → WAIT (quando está a correr à demasiado tempo e o escalonamento decide interromper o processo)

Kiloprocesso quando transita de um estado qualquer para o exit

mil quando passa de sleep para suspend

any state → exit

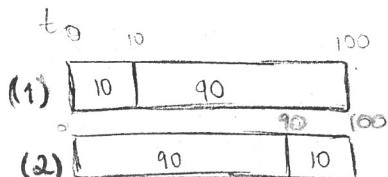
New → escalamento do Longo Prazo

Suspend → escalonamento de medio prazo

Block, wait, run \rightarrow escalonamento do curto prazo (é o escalonamento mais utilizado)

Processos

se existirem 2 processos, 1 com 10 ms e outro com 40 ms, como fica mais eficiente a execução dos processos



os processos independentemente de qual o 1º a ser executado vai demorar o mesmo tempo a correr os dois processos, mas qual será o mais eficiente.

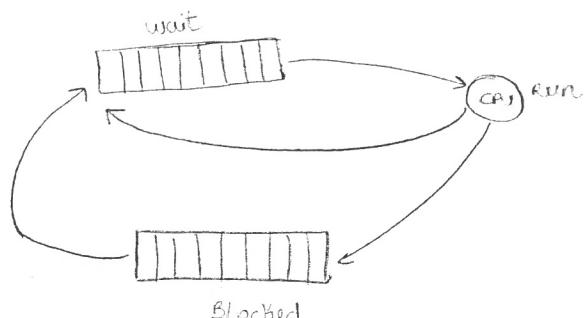
$$(1) \quad \text{média} = \frac{110}{2} = 55$$

$$\text{média} = \frac{990}{6} = 95$$

Através da média de execução dos processos percebermos que era mais eficiente os processos com um tempo mais curto fossem os primeiros a serem executados era mais eficiente.

Poem algum Processos podem ser partidos dos bocados e dessa maneira já não iríam os pensar o m^{esmo}
que os mais curtos se fossem os primeiros a serem executados era mais eficaz. Logo os Processos podem estar
divididos em bocados (Preemptivos) ou podem estar inteiros (não Preemptivos), ou seja, quando os Processos
não são interrompidos.

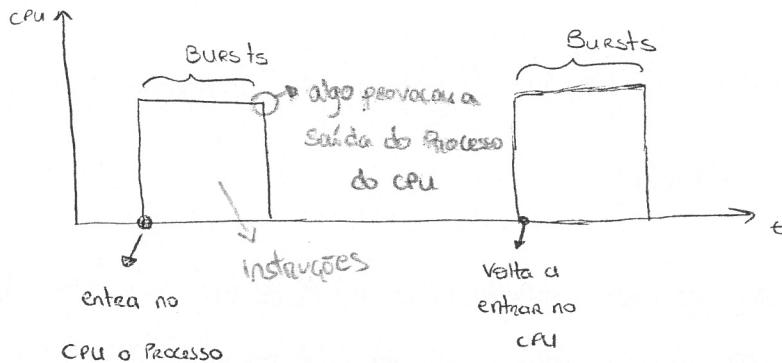
Escalamento de Processos (Cap. 9 do livro)



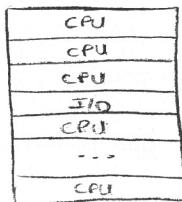
O que o escalonador faz é a transição do wait para o run, o escalonador escolhe qual dos processos que estão em fila, no wait, vai para o CPU, caso a escolha seja pela ordem da fila (FCFS) ou se estiver em Fila de wait, não.

- * FIFO ou FCFS. O escalonador também é responsável pela escolha de um processo sair da CPU para o estado wait ou não, esta escolha quase sempre é o resultado de um timer.

Em termos práticos o que acontece tipicamente:



Bursts → Períodos de Tempo que o Processo se encontra na CPU



Em termos de Programação vamos ter instruções (CPU) e quando chega ao I/O temos de passar para o estado Blocked, porque o processo está a esperar por algo, quando o processo tiver o que necessitava este vai para o wait e só depois vai para o CPU, entre as instruções, ou seja, quando o processo tiver no CPU o processo pode ter que ir para o wait dependendo do escalonamento na CPU.

Eu posso ter um Processo que ocupa CPU 10% e 10% do I/O e outro que ocupa CPU 10% e I/O 90%, os dois são muitos diferentes. Um Processo tem a maior parte do tempo na CPU, já o outro tem a maior parte

do tempo em I/O. Se eu tiver os dois processos em wait qual dos dois deveria ir primeiro para o CPU?

O que deveria ir primeiro é o que ocupa mais tempo no Blocked, porque dessa maneira deixa o CPU disponível

CPU Bound

CPU 90%.

I/O 10%.

I/O Bound

CPU 10%.

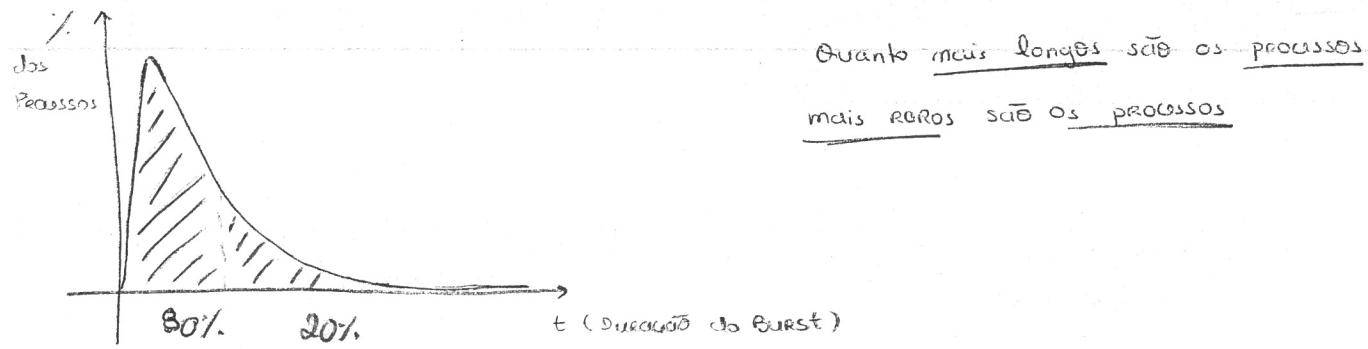
I/O 90%.

O escalonamento dará prioridade ao I/O Bound

Existem vários algoritmos para os processos.

um dos Algoritmos é o Round Robin (RR) este tipo de escalonamento serve tudo pela ordem de chegada mas se tiver mais do que um x de tempo no CPU, este obriga o processo a sair da CPU e a ir para o estado wait, se for um processo curto este não irá ser partido porque demora menos tempo que o tempo max a cada processo tam na CPU, este algoritmo privilegia os processos curtos.

Tempo Quantum → tempo que os processos podem ter no CPU no algoritmo RR (Round Robin)



o estado de Paragem de um processo (saída do CPU para o estado wait, ou Blocked) vai gastar tempo para guardar dados e ir buscar os dados então devemos tentar o mínimo de Paragens Possíveis.

o TQuantum deve que 80% dos processos não são interrompidos apenas 20% dos processos são

Algoritmos de Escalonamento (Comparação)

HPO / FCFS

Não à interrupção dos processos

(Não preemptivos)

Round Robin
RR (TQuantum)

Itá à interrupção dos processos
(preemptives)

um escalonamento Não preemptivo tem um problema, significa que quando um processo está em ciclo infinito nunca vai sair do CPU.

os Algoritmos não preemptivos são utilizados muitas vezes em sistemas de Tempo Real, porque pode haver já que em sistemas de tempo real é tudo muito controlado então acaba existindo um ganho de tempo.

Para sabermos qual o tipo de escalonamento que devemos utilizar tem a ver com o tempo médio para despachar um processo. Começamos a medir o tempo dos processos pelo seu tempo de chegada (o seu pedido) se por acaso chegou tudo ao mesmo tempo, só temos de perceber o tempo que cada um irá demorar

Algoritmo de Escalonamento

SPN ou OPT ou Sjf → quando o processo mais curto é escolhido, também é conhecido como

algoritmo ótimo, privilegia os processos mais curtos estes são os 1ºs a serem despachados.

Tempo de serviço → quanto tempo o processo precisa de rodar.

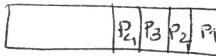
Imaginando que temos 4 processos P_1, P_2, P_3 e P_4 e que tem tempos de serviço:

Tempo de serviço

P1	10
P2	40
P3	20
P4	60

Todos os Processos chegam ao wait no instante zero.

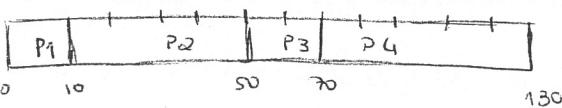
wait



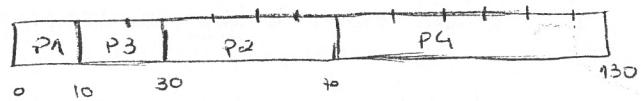
$$P_1 > P_2 > P_3 > P_4$$

Consoante o Algoritmo de escalonamento usado o tempo que cada processo demora é igual mas o tempo total é diferente:

diferente:

FCFS

$$\frac{10 + 30 + 70 + 130}{4} = 65 \text{ ms}$$

SPN (não preemptive)

$$\frac{10 + 30 + 70 + 130}{4} = 60 \text{ ms}$$

RR (Round Robin)

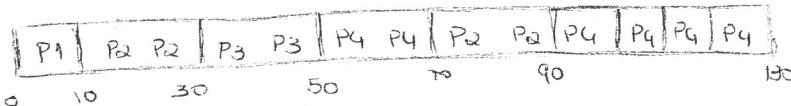
o TQuantum = 20 ms



P1	P2	P3	P4
P2	P3	P4	
P3	P4	P2	
P3	P4	P2	
P4	P2		
P2	P4		
P2	P4		
P4			

abriga o processo a ser interrompido porque o TQuantum é 20 ms

Como agora só existe um processo → independentemente do processo ultrapassar os 20 ms (TQuantum) o processo irá ficar no CPU até acabar



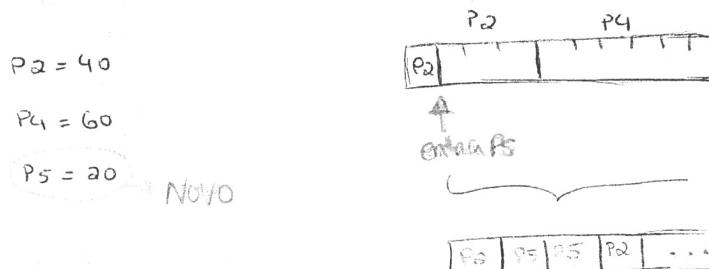
$$\frac{10 + 90 + 130 + 50}{4} = 70 \text{ ms}$$

Um bom motivo para não usar SPN é porque temos de saber o tempo de serviço o que nem sempre sabemos, se não soubermos esse tempo não sabemos a ordenação dos processos. Porém o SPN é bom para comparar com outros algoritmos de escalonamento porque este é sempre e mais rápido. Isto quem ^{tem} para adivinar o tempo de serviço, através da medida dos Bursts, fazendo uma estimativa e assim saber-se mais menos o tempo de serviço, esta é uma forma de se saber o tempo de serviço.

RR - Em termos de representação não precisa de termos nenhum

outros Algoritmos

SRT - É uma espécie de SPN mas do tipo preemptivo. faz o mesmo que SPN mas se aparecer um novo processo mais rápido do que os processos que faltam, inclusive o que está a correr, então o SRT vai interromper o processo que está a correr e coloca o processo mais rápido (o que tem menor tempo) no lugar do corte, a correr.



HRRN

Se eu tiver um processo de 20ms a faltar 20ms e se eu tiver um processo de 100ms e demora 20ms que é o primeiro que eu doveria mandar correr?

20/20

100/20

Se o 20/20 for o primeiro então 100/20 tem 20% de espera, caso seja o contrário o processo 20/20 tem 100% de espera.

Este coloca os processos à espera consoante o tempo de espera de cada processo e consoante o tempo de serviço.

A conta que o utilizador faz é $\frac{\text{Tempo Espera} + \text{Tempo Serviço}}{\text{Tempo Serviço}}$

este coloca os processos na fila consoante o resultado dessa conta, é colocado em Primo na fila da espera o que tem um maior resultado nesta conta, logo o que tem um menor resultado nesta conta é o que irá ter que ficar mais tempo a espera.

50 ⑥

P1	P3	P3
----	----	----

$$P_2 = \frac{10 + 40}{40} = 1,2$$

$$P_3 = \frac{10 + 20}{20} = 1,5$$

$$P_4 = \frac{10 + 60}{60} = 1,2$$

Depois de P3 selecionado e da ele ter corrido por 20s o que tem um tempo maior, temos de voltar a refazer as Co para P2 e para P4.

$$P_2 = \frac{30 + 40}{40} = 1,75$$

$$P_4 = \frac{30 + 60}{60} = 1,33$$

Neste Caso o seguinte na fila seria P2

P1	P3	P3	P2	P2	P2	P2
0	10	20	30	40	50	

E se chega-se um PG no instante 50 e que tem 50ms?

$$P_1 = \frac{30 + 60}{60} = 2,1666$$

$$P_6 = \frac{20 + 50}{50} = 1,4$$

entre os 2 o P4 e o P6, o P4 é o que tem a maior perda, neste caso não é o mais curto que é o favorecido.

o fator da simplicidade ganha quando trabalhamos em baixo nível com o escalonamento.

De todos os algoritmos de escalonamento em termos de implementação o que está mais perto da realidade é o RR. um dos inconvenientes do RR é se eu tiver um Processo muito grande e o TQuantum for muito pequeno, o Processo irá ser

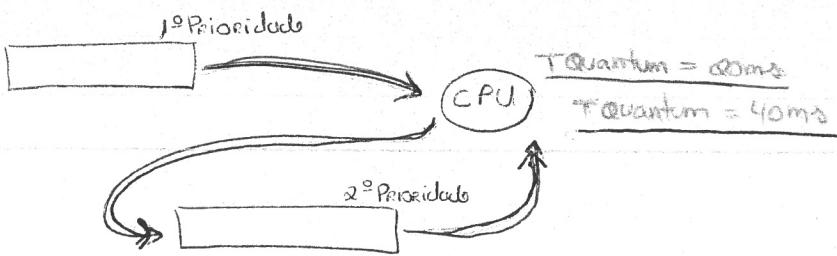
partido em vários bocados, logo existe perda de tempo na troca de contexto, caso os blocos sejam grandes, os

processos mais pequenos não são favorecidos. Se tiver processos grandes posso modificar o TQuantum, alteran-

tando-o de maneira a que cada processo tenha menos "partes".

Round
Robin

Para isto faz-se:



Ao fazer filas com diferentes prioridades posso aumentar o T quantum poupar tempo ao não mudar o contexto.

A este modelo chama-se fed Back Multifila

O T quantum pode ir aumentando ou diminuindo, se o tempo de espera de um processo tiver muito tempo para acabar passa para uma fila de maior Prioridade.

O tempo que o escalonamento demora, ou seja, o tempo que se demora a escolher qual o processo que entra para o CPU é sempre o mesmo independentemente do numero de filas e da sua Prioridade, por isso existe um tempo para o escalonamento que é finito e que nunca cresce.

Algoritmos de Escalonamento em tempo real

funcionamento \rightarrow importância do tempo de resposta

Nos Algoritmos de Escalonamento em tempo real existe a importância do tempo de resposta, porque se a resposta vier dentro desse tempo, então a resposta é válida, se a resposta não vier dentro desse tempo então a resposta não é válida.

$$T_{resposta} \Rightarrow T_{resposta} < \text{limite}$$

\rightarrow Válido vs. ERRO

mesmo que as ações (respostas) têm sucesso, ou seja, que sejam válidas se o $T_{resposta} > \text{limite}$ existe um ERRO.

HRT

Hard real Time:

é o que cumpre estritamente o tempo de resposta dentro do limite não existindo um erro. É utilizado em sistemas Industriais, Aquisições de dados, em carros, etc..., por ser utilizado em coisas do tipo repetitivas, em chamadas de funções periódicas, funções simples e ações previsíveis (funções).

Soft real Time:

o bom funcionamento do sistema não está só ligado ao $T_{resposta} < \text{limite}$ mas também a um fator estatístico (% tempo), por exemplo se cumprir 99% então é válido mesmo que $T_{resposta}$ não seja menor que o limite, porque através do fator estatístico dá algum "espaço" de tempo para se o $T_{resposta}$ não for dentro do

S.0.7
Limite pode ser válido na mesma. É mais complexo a nível de Projeto e Análise mas também é mais flexível porque não tem restrições e por essa razão tem mais aplicações. Utilizado em Processamento de vídeo, quando eu quero passar um vídeo ou áudio, se faltar uma frame não é grave porque provavelmente nem nos vemos perceber.

As construções do sistema operativo por vezes tem algumas especificidades. Por exemplo:



se o sistema operativo só permite acções durante 10ms, e se estiver existir uma ação com um tempo superior pod-

e constar um sistema operativo que seja interrompido, podendo-se ser interrompido até por ações que são mais prioritárias existindo assim um tempo de resposta menor. Muitas vezes no Linux ter o Kernel interrompido faz com que se tenha tempos de respostas muito rápidos (Kernel em tempo real).

Soft Real Time

Principais diferenças

Hard Real Time

só sistemas previsíveis

Sistemas mais complexos

(Posso ter um sistema tipo desktop)

os algoritmos em tempo real não são muito complexos. os mais simples são em prioridades e dentro destes prioridades dinâmicas que costumam ficar no soft real time.

Também podem ser sistemas preemptivos ou sistemas não preemptivos (cada processo consegue não é interrompido).

que é ideal era um sistema preemptivo com prioridades estáticas.

se eu tiver só processos periódicos, normalmente o que se tem é um processo que é repetido (P1), os processos têm que terminar antes que chegue o pedido do novo processo



Quando o P1 faz o processo

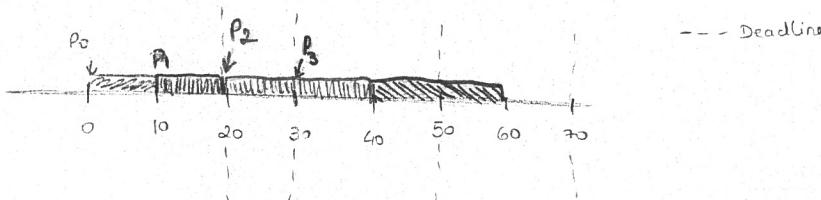
para saber se os processos cumpriram o deadline ou não tenho de verificar todos os processos, por isso se existe um padrão entre os processos então se eu garantir que nessa parte os processos cumpriram o deadline não preciso de verificar mais.

Não Periódico

o que se faz é que independentemente de os processos cumprarem ou não o deadline, se os processos chegar ao final com certas características então os processos estão "bom", mas este método num tempo é possível.

	T serviço	T chegada	Deadline
P0	10	0	30
P1	10	10	30
P2	20	20	50
P3	20	30	70

Através desta tabela temos as enregressos temporais para cada um dos processos



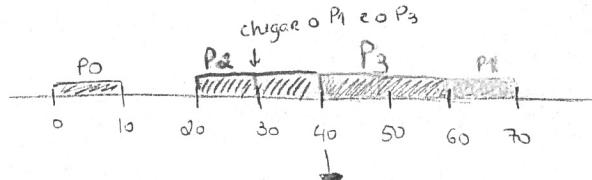
entre os processos podem existir relações entre os processos, por exemplo, $P_0 > P_1 > P_2 > P_3$, isto quer dizer que o processo P_1 é mais prioritário que o processo P_0 . Para termos em conta a prioridade dos processos podemos utilizar um algoritmo que é utilizado em sistemas periódicos e não periódicos, sendo considerado um sistema de tempo real.

Deadline + Próximo

Para definir qual é o processo mais prioritário com base no deadline dos processos. O processo que tem o deadline mais próximo independentemente de ter sido o primeiro processo a chegar ou não, o que tem o deadline mais próximo passa a ser o mais prioritário.

Exemplo:

	T serviço	T chegada	Deadline
P0	10	0	50
P1	10	30	80
P2	20	20	50
P3	20	30	70

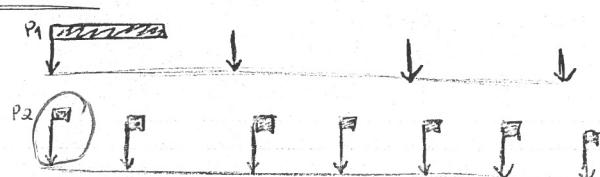


Chega aqui e é só os processos que têm a espera, que é o P1 e o P3 e vai olhar para os seus deadlines e como neste caso o P3 tem um deadline mais próximo é ele o mais prioritário.

Só pode completar e correr os que estão em fila no instante zero, só existe o processo P0 então vai ser esse que vai executar.

foram cumpridos todos os deadlines dos processos.

O algoritmo não é ótimo em todas as situações, existe uma variante que apenas é aplicada quando o sistema é periódico.



■ Tempo de serviço
■ Tempo de chegada

O que acontece é que quando o Processo P₁ acaba de correr existem 2 processos P₂ que estão à espera de correr, porém o Primeiro P₂ já tem o deadline ultrapassado e por esse motivo já não pode ir para o escalonamento, por essa razão e como o sistema é periódico e não preemptivo (os processos não são interrompidos) um dos processos P₂ não entra no escalonamento, por este motivo o Algoritmo não é ótimo, nesta situação o algoritmo não funciona.

Para ter em conta estas situações podemos usar um algoritmo ≠ em que é dado prioridade ao que tem um deadline mais próximo independentemente de o Processo não ter chegado ainda. Apenas o utilizar quando sei que o pedido é repetitivo. Independentemente do P₁ ter chegado ele não vai correr o Processo P₁ até chegar o Processo P₂, quando o Processo P₂ chega ele vai correr-o primeiro, repetindo assim o Processo, conseguindo assim correr todos os processos sem que nenhum ultrapasse o deadline.



A este algoritmo chama-se deadline + Próxima com Tempo Isto.

Este Algoritmo apenas pode ser utilizado se os processos forem periódicos e não preemptivos, porque se fossem preemptivos não haveria necessidade do algoritmo porque quando o Processo P₂ chega-se (o pedido) o Processo P₁ não se é interrompido.

Deadline + Próximo

Podem ser processos periódicos e não

periódicos:

o primeiro processo a chegar se for o único processo, é o primeiro a correr sem ser interrompido.

Principais diferenças

Deadline mais Próximo com

Tempo Isto

Tem um certo espaço de tempo "Isto", em que

nenhum processo corre

RQS (rupte monotonic scheduling)

↓
frequência

Aplicável a processos periódicos, aquela com mais periodicidade aos processos que têm frequência menor e logicamente tem período mais curto, ou seja, o que tiver menor Repetição (processos) tem uma periodicidade menor

Exemplo:

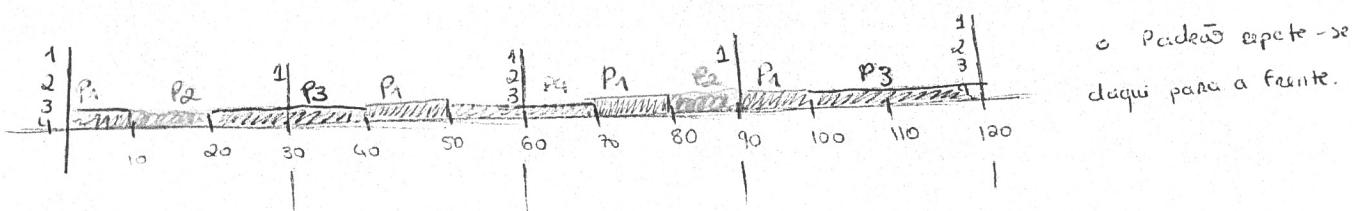
Tarefa	Período	Média	A primeira coisa que temos de fazer é calcular a média
P ₁	10	20	cada processo de forma a sabermos qual a <u>percentagem de CPU</u> que <u>cada processo</u> ocupa. De seguida temos de calcular a
P ₂	10	60	que <u>cada processo</u> ocupa. De seguida temos de calcular a
P ₃	20	60	Soma da média dos processos se essa soma for superior a ou igual então é possível, não consigo faz escalonamento.
D.	90	120	

$$\frac{T_{S1}}{T_1} + \frac{T_{S2}}{T_2} + \dots + \frac{T_{Sn}}{T_n} \leq 1 \text{ ou } 100\%. \rightarrow \text{Se esta condição não for verificada então é impossível fazer o escalonamento dos processos.}$$

Se a soma das médias dos processos for menor que 1 ou 100%. não quer dizer que este algoritmo funcione sempre, apenas quer dizer que existe uma possibilidade de ele ser possível.

Visto que a média dos processos ≤ 1 ou 100% , então de seguida vamos ver quais são os processos mais prioritários, neste caso, o m é o processo mais prioritário por ter um período curto, entre o P2 e o P3 é igual porque têm o mesmo período e o processo P4 é o menos prioritário.

Este algoritmo também depende do tempo de chegada, mas se os 4 processos chegarem ao mesmo tempo.



O pacote espera-se daqui para a frente.

Todos os processos terminam antes da sua deadline (respeitam as restrições de tempo real). Isto não significa que por eu usar o algoritmo RTTS que tenha sempre uma solução e que os processos irão sempre respeitar as restrições de tempo real.

Porém existe uma fórmula que nos diz quanto o número de processos que nos diz que de o tempo de CPU utilizado por todos os processos for x, então de forma que o Algoritmo RTTS resulta sempre, esse valor costuma ser menor que 100.

$$n \times (2^{1/n} - 1)$$

$n \rightarrow n^{\text{o}}$ de processos

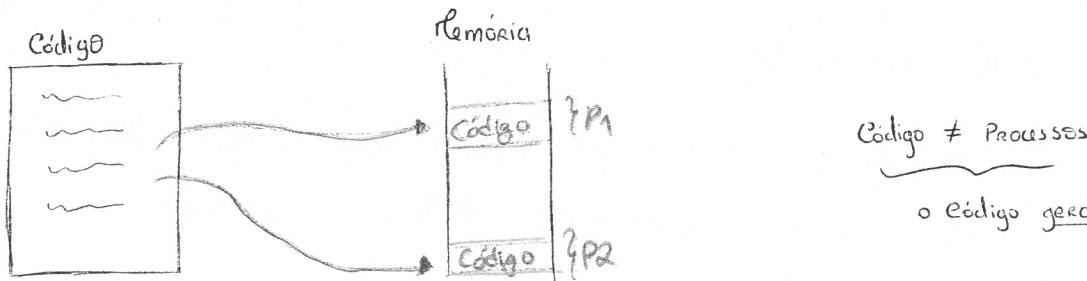
<u>n</u>	<u>100%</u>	<u>Número de CPU que eu posso utilizar no CPU dependente do nº dos processos que assegura o sucesso do RTTS.</u>
1	100%	
2	82%	
3	77,9%	
4		
∞	$0,693 = \ln 2 \approx 70\%$	\rightarrow se eu tiver um número de processos por muito grande eu tiver de utilizar 30% de CPU de sorte que o RTTS funcione sempre.

É fácil garantir com este algoritmo que existe escalonamento em tempo real sabendo que os processos tem de ser periódicos.

Processos e Threads

S.O.

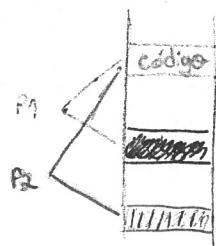
9



Os Processos P₁ e P₂ têm o mesmo Código mas são processos separados um do outro, porque os inputs de um não são iguais aos inputs de outro, logo o fluxo de Código é diferente. Porém os 2 processos também tem le c o s a s.

Enves te ter tudo o Código igual em vários processos eu posso ter uma parte do Código partilhada por pro

sses



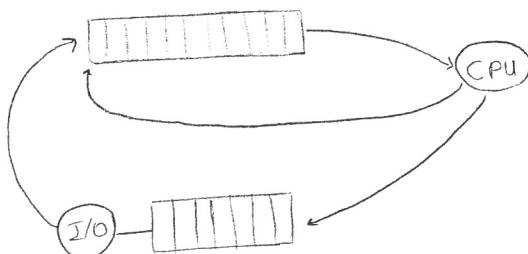
Ao existir uma partilha de uma parte do Código existe uma poupança de espaço, assim não vai ser necessário duplicar essa parte do Código.

Se existissem dados iguais entre os Processos, estes também podiam ser partilhados, se esses dados foram dinâmicos não podem ser partilhados para todos os processos, podem ser partilhados com as versões da Processos.

Se forem dados dinâmicos os que são partilhados, ficam a existir problemas entre os Processos

Threads → Processos que usam o mesmo Código e que são pensados a Trabalhar em Paralelo mesmo que não existam

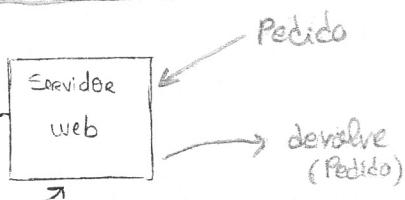
n Cpu's.



Existem Hardware's que funcionam em Paralelo em todos os PC's. Por exemplo enquanto existem Processos que estão a Correr no CPU estão outros Processos a pedir informações ao I/O, quando isto acontece existe Hardware em Paralelo

Utilidade do Paralelismo

Base de dados
ou algo que
não quer
envolver



Existem n pedidos ao mesmo tempo. Podem existir em simultâneo processos a chegar o pedido, outros a construir o que se devolve e ainda a consultar dados.

O paralelismo do sistema existe entre construir output, entrada de input e consulta dos dados, este sistema tem de sempre garantir que para pedidos de diferentes dimensões consigo ir dando sempre uma resposta.

A construção das threads parte do pressuposto que existem coisas partilhadas pelos processos.

Entre os processos:

O que é Partilhado

- Códigos
- Dinâmicos (variáveis globais)
- Dados

O que não é Partilhado

- ficheiros Abertos
- o que está no PCB
- ID
- Estado
- o PC (Program Counter)
- variáveis locais
- Registo Ativação das funções
- STACK
- registos do CPU

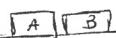
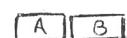
Cada thread tem a sua própria parte do que não é partilhado.

A parte das threads que é mais complexa é a parte que é partilhada.

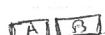
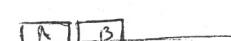
As threads não podem estar a fazer o mesmo, uma thread podem fazer estar a fazer uma coisa outra

Thread a fazer outra coisa.

Caso eu tenha uma função que requer vários recursos, imaginando A e B e caso existam n pedidos.



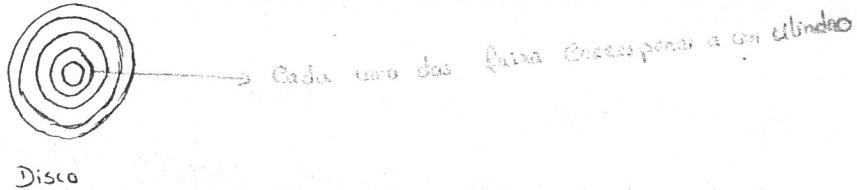
Pode fazer um pipeline e quando o recurso B está a correr podemos começar a correr o A também



↓ com Paralelismo

Num disco a informação está organizada num cilindro, pode existir um atraso por a cabeça do disco ter que ir até ao cilindro que contém a informação necessária, quando existem n pedidos. Se a informação necessária tiver todos no mesmo cilindro vai ser mais rápido do que se a informação tiver espalhada em diferentes cilindros.

O tempo necessário em média é muito menor se eu tiver a informação toda no mesmo cilindro.



existem vantagens se existirem recursos partilhados, ao criar uma Thread never é criar apenas alguns recursos necessários, o número de recursos é mais pequeno do que os recursos que são utilizados ao criar um processo, logo criar um processo é mais lento.

Parcialização "verdadeira"

Se existirem n CPU's é melhor utilizar Threads em relação aos Processos.

As threads partilham algo em relação a um ou mais processos e são mais baixos os recursos que é preciso na mudança de contexto logo essa mudança também é mais rápida.

Como as Thread's se metem a funcionar

Os sistemas operativos sabem da existência das threads. Existem os processos leves e os processos pesados em alguns sistemas operativos



Nos existem outros sistemas operativos que em vez de estruturarem em 2 tipos, estruturam em 3 tipos, com uma hierarquia consistente o que os processos partilham



Estas classes também têm a ver com a forma como só implementadas as Threads

O Linux não reconhece as threads, no Linux chama-se Tasks.

Todos os Sistemas Operativos conseguem chegar à mesma funcionalidade mesmo que utilizem caminhos diferentes para lá chegarem.

utilizo fork para Criar Processos no Linux.

Kernell Thread's → Recorrem as Thread's pelo Sistema Operativo

User Level Thread → Posso criar Thread's com software disponibilizado pelo utilizador.

Posso ter um Sistema Operativo que tem Kernell Threads e correr no sistema operativo um User Level Thread.

Todos os Threads que estão em User Level são "conhecidas" pelo Sistema Operativo como Processos, isto

traz uma desvantagem porque se a Thread pedir algo do disco então vai tudo para o Blocked e a vantagem

das Threads deixa de existir. Mas este problema pode-se contornar, pode existir algo que preve no sistema

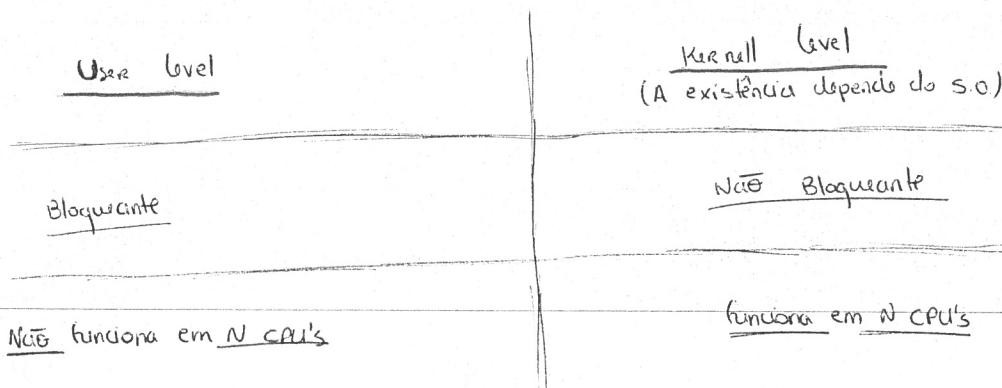
operativo quando alguma Thread é Bloqueante ou não, quando existe um pedido que seria bloqueante a Thread

irá para para Blocked, o processo irá continuar ativo na CPU, porém o Sistema Operativo irá fazer as coisas

a Thread que está a correr vai para o Blocked e outra Thread começa a Correr (contexto interno), a Thread

que precisa da informação continua a Correr no same CPU mas vai ver periodicamente se a informação necessária

já chegou, a própria Thread é responsável por isso, assim as Thread's são mais complexas.



Quando tenho n CPU's posso ter uma Thread Blocked e outra a correr, para eu ter várias Thread's a

Correr o User Level Thread não funciona com n CPU's

o Kernell Thread Level não é uma solução muito melhor que o User Level, porque como no User Level as Thread's

são todas no mesmo nível não é preciso de fazer mudança de contexto para fazer a Troca de Thread's, no User Level é mais rápida essa mudança.

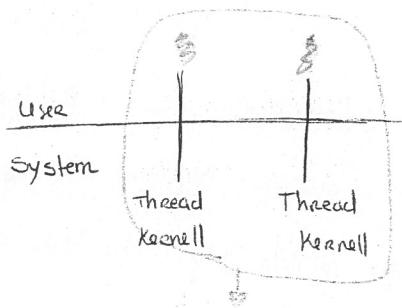
Em Kernell Level consigo aproveitar o Paralelismo mesmo com N CPU's diferentes.

A Troca de Thread's do mesmo processo é mais rápida no User Level porque o Sistema Operativo não intervém essa ação de Kernell Level o sistema operativo intervém nessa ação.

Thread's Programadores e Thread's do sistema Operativo

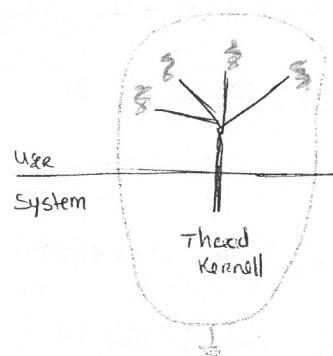
S.O. 11

→ Modelos mais comuns



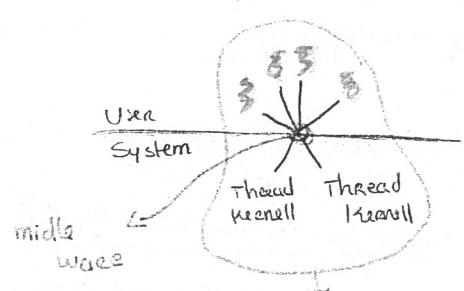
Modelo 1:1 (User:System)

Versões do Windows (soluções)



Modelo n:n (User:System)

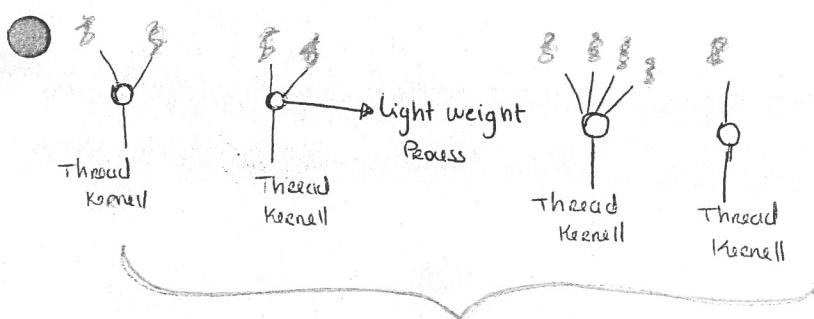
Soluções para Unix



Modelo middle ware (User:System)

Mais complexo, solução para Unix

Solução



Permite que os Kernel ~~executar~~ em CPU's diferentes aproveitando o Paralelismo. Tenho flexibilidade posso ter os 2 Thread's a correr ou só 1 e nenhuma.

As Thread's do Kernel vão "disputar" os CPU's

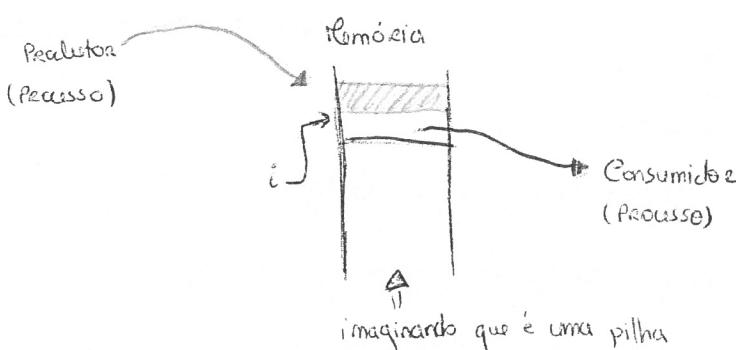
Existe uma gestão, (O), que irá escolher qual das Thread's é que terá o privilégio de correr.

O Solaris também pode ser no Unix.

Se os Thread's podem mexer em algumas zonas de memória, as ferramentas que são utilizadas

estão ligadas a alguns problemas de sincronismo e concorrência (Prioridades), quando tenho 2 thread's que ao mesmo tempo querem mexer no mesmo sitio, ocorre esses problemas.

Problemas Produtor Consumidor



Tenho de saber qual é o sitio em que posso escrever e o sitio em que leio, para isso tenho de ter uma regrinha, que me diga qual o 1º que está livre ou a última ocupada.

- c) Produtor irá dizer qual a ultima ocupada fazendo $i+1$, depois esse $i+1$ irá ser implementado e o $i = i+1$, o Consumidor irá usar i para ver qual a ultima ocupada se o consumidor estiver algo no i irá fixar $i = i-1$, e irá voltar ao sítio inicial da Pilha.
- c) Problema que surge é quando não existem instruções atómicas (1 única instrução para o Produtor incrementar o i e uma única instrução para incrementar o i), quando isto não acontece irá ocorrer um problema. O Produtor copia o i para um registo Temporário, depois irá atualizar a informação no registo Temporário, quando tiver o resultado final no registo Temporário ele irá colocar o valor novamente no i, o consumidor irá fazer algo semelhante.

$i \rightarrow R_1$

$R_1 = R_1 + 1$

$R_1 \rightarrow i$

A ordem pelas quais as funções são executadas são variadas e pode não acontecer tudo de sequência, ou seja, ocorre

alternadamente entre o Produtor e o Consumidor, assim irá ocorrer o problema porque as instruções não são atómicas

Produtor

$i \rightarrow R_1$

$R_1 = 5$

$R_1 = R_1 + 1$

$R_1 = 6$

$R_1 \rightarrow i$

$i = 6$

Consumidor

$i \rightarrow R_2$ $R_2 = 5$

$R_2 = R_2 - 1$ $R_2 = 4$

$R_2 \rightarrow i$ $i = 4$

No caso acima começou o Produtor e depois foi alternando com o Consumidor, mas se fosse o Consumidor

começar e fosse a mesma alterada o resultado seria diferente o do Produtor dava $i = 5$.

Para resolver este problema digir que o bloco das 3 instruções do Produtor é um bloco único tal como

para resolver este problema digir que o bloco das 3 instruções do Consumidor (existem instruções atómicas), Assim enquanto 1 instrução tiver a posse que

as 3 instruções do Consumidor não podem existir interrupções até as 3 instruções termos sido executadas.

seja do Produtor ou Consumidor.

Mais simples é o mecanismo exclusão mútua ou mutex, mas também existe o lock, semafóros ou monitoras,

em linguagens diferentes e diferentes sistemas operativos.

os monitores são os mais complexos. O Deikos e o Petrópolis conseguem implementar o mutex.

O semáforo também utiliza o Dekos e o Petrópolis e os monitores utilizam os semafóros.

Semáforos

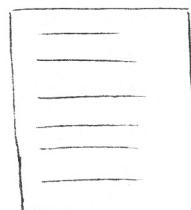
S.O. 12

Conjunto de instruções que são executadas de forma Atómica (ou seja, são instruções que funcionam como um bloco de instruções e que não pode ocorrer uma interrupção dessas instruções até todas serem executadas)

Secção Crítica (soluções)

- Algoritmo Dekker
- Algoritmo Petersen

{ Reconhece de
modo átomo



Conjunto de
Instruções

↓
só pode existir
um processo

2 funções (normalmente standard)

P() = wait() = down() = testa()

→ Mais utilizadas

V() = signal() = up() = decrementa()

O semáforo tem sempre associado uma variável, neste caso o semáforo é o s.

wait

(diminuir)
decrementa o valor de s

wait(s) } S -- ; (s tem de ter sempre um valor inicial)

if (s <= 0) Processo → fluBlocked; (bloqueado)

{

Espuma Ativa → se tiver num ciclo infinito, não faz nada
Espuma

da {{ ~

Se eu quiser que espere eficientemente vou para o estado Blocked até o timmer dizer algo, aí o processo tem que fazer alguma verificação de uma certa condição.

Signal

incrementa o valor de s

Signal(s) } S + + ,

if (s <= 0) tira um processo de Blocked; wakeup(p)

→ Processo P qualquer
que está em espera

{



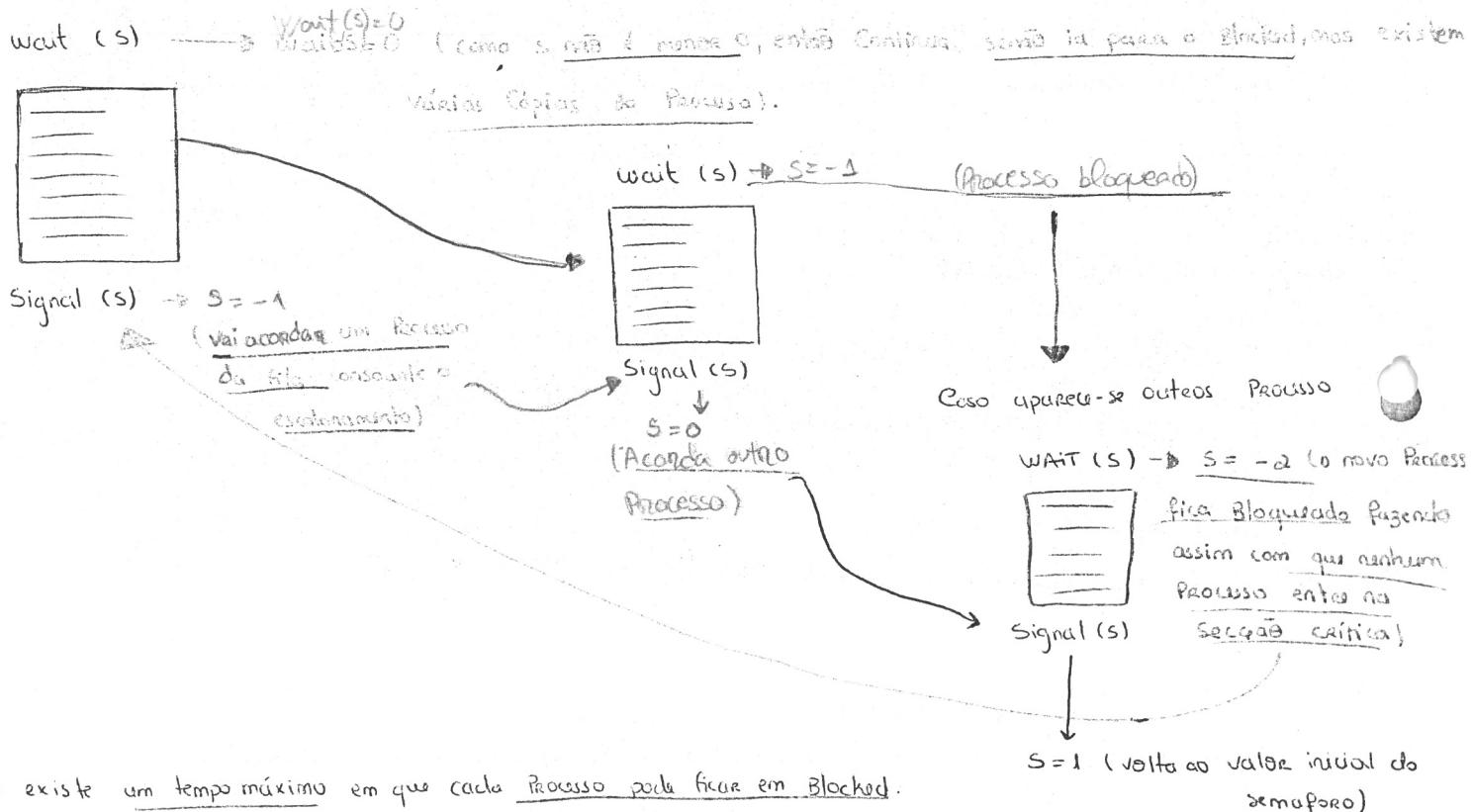
→ Instruções que quero que sejam executadas na secção crítica

Os semáforos aparecem aos Pares

Quase sempre o Processo é algo cíclico que se vai repetindo n vezes.

Inicialização do semáforo (o valor de S)

init S = 1 → semáforo inicializado a um



Não existe um tempo máximo em que cada processo pode ficar em blocked.

O Processo é sempre o mesmo.

No máximo existe um processo lá dentro. [] (na secção crítica)

E se'

init S = 2

wait (s) = 1

wait (s) = 0

Neste caso, existem dois processos a correr dentro da secção crítica.

- o que não pode acontecer

Então, init S = N° de recursos que posso ter

Como só posso ter 1 processo a correr na secção crítica, então init s=1, assim garantir que no máximo só tem

1 processo a correr na secção crítica.

Em relação ao Código e a secção crítica tipicamente usamos 1 na inicialização se por outro tipo usarmos out número.

Exemplo

40 Pessoas Podem entrar numa sala, apenas.

init s = 40 → Apenas 40 Processos podem estar na seção crítica, se em tempo de limite de processos que
wait (s) podem estar em simultâneo coloca a inicialização = número

aluno . Perguntar
aluno . ler } instruções
aluno . Programar
Signal (s)

Lusitâo Retira

Problema que é resolvido quando inicializamos o semáforo a 1

Produtor

Colocar - na - pilha ()

Para poder colocar ne pilha
preciso de espaço (de espaço)

Consumidor

Retirar - da - Pilha ()

para poder retirar da
pilha preciso do algo colocado
para conseguir tirar

Produtor

Wait (e) → espaço

Colocar - na - pilha ()

Signal (e)
↓ Objeto

Consumidor

Wait (e) → Objeto

Retirar - da - Pilha ()

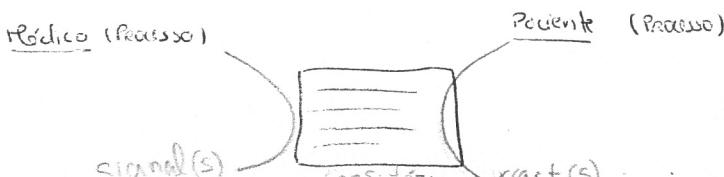
Signal (e)
↓ espaço

O wait (e) e o signal(e) não estão no mesmo Processo, estão em 2 Processos diferentes

Wait → função que verifica se as condições dos processos existem e se está tudo bem ou não

Quando tem de existir um encontro entre 2 processos e os 2 fazem sincronização.

Exemplo

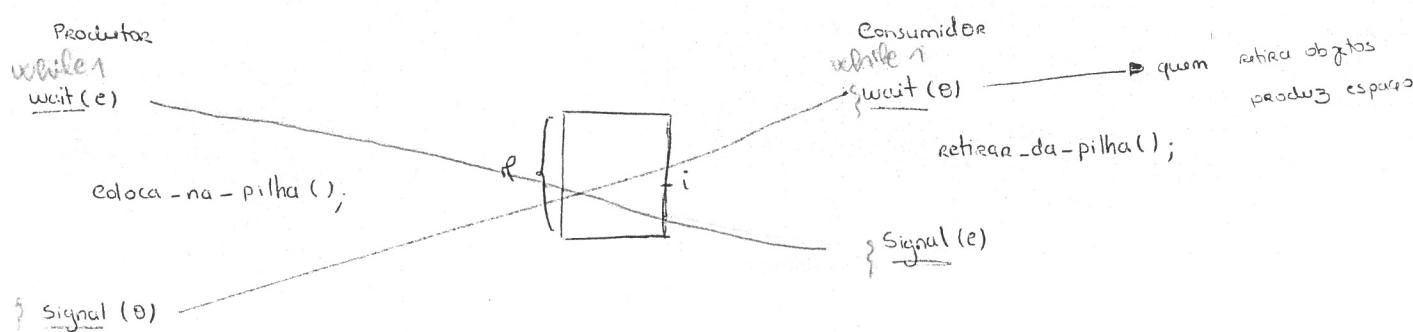


O Consultório é onde os 2 vão passar, só que tenho que garantir que 1 espera pelo outro.

Coloca-se um Signal (s) do que não quer que espere, o s é inicializado a 0 e colocar o wait(s) no que & que que espere, assim quando o Paciente chegar ao Consultório, já o médico passou por ele 1 vez e coloca-se o $s = -1$, assim sendo quando o Paciente chegar ao consultório terá de esperar pelo médico.

Utilização de semáforos

init e = 0 } Initialização positiva → desde que o init de init e em 0 sempre temos inicialização possível.
 init e = m



Como as funções são finitas, por vezes existe um while

os semáforos estão associados a recursos

o recurso para cada um dos Provedores é o Espaço, para os Consumidores seria a parte ocupada.

os semáforos aparecem aos pares, mas nesse caso trocados entre o Consumidor e o Provedor

Existe uma seção crítica & tenho um P processo com instruções.

init s = 1

P

wait(s)

Quando é 1 se if x = ...
 Pode ser feita x = x + 1

vez de vez → Signal (s)

Semáforo

Para que não ocorra:

SEÇÃO CRÍTICA

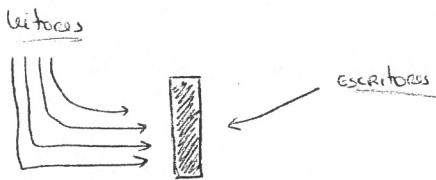
Possui ter n cópias de P, mas quando o Primeiro Processo P entra, todos

os outros tem de ficar à espera

os 2 problemas mais básicos dos semáforos ↑

OUTRO PROBLEMA

S.O. (AU)



1º termos de saber as restrições que temos de cumprir em n processos.

Neste Case

Se eu tiver escretores a querer escrever apenas 1 podia escrever de cada vez

Já leitores eu posso ter vários a ler ao mesmo tempo.

Não posso ter o mesmo tem escritores e leitores, eu tenho uma coisa ou a outra

→ Caso existam vitóres e um escritor queira entrar é Bloqueado, o que também acontece ao Contrário e se um escritor já se esteja lá os restos dos escritores também vão ser Bloqueados.

Variável i -> Conta números de botões que você passar

$$\text{int}_R = 1$$

Lutde

init s=1 ~~→~~ semiforse ~~+~~

Escritor

west (R)

if ($i = 1$) then $\text{wait}(s) \rightarrow$ Imbeds
asynchronous de

wait (s)

escrever());

→ seção crítica: que apenas permite 1 escritor
lá dentro

Signal (R)

$\ln()$.

wait (2)

$i = i - 1$ —> conta o número de saídas

if ($i = 1$) then signal (s)

sign w (R)

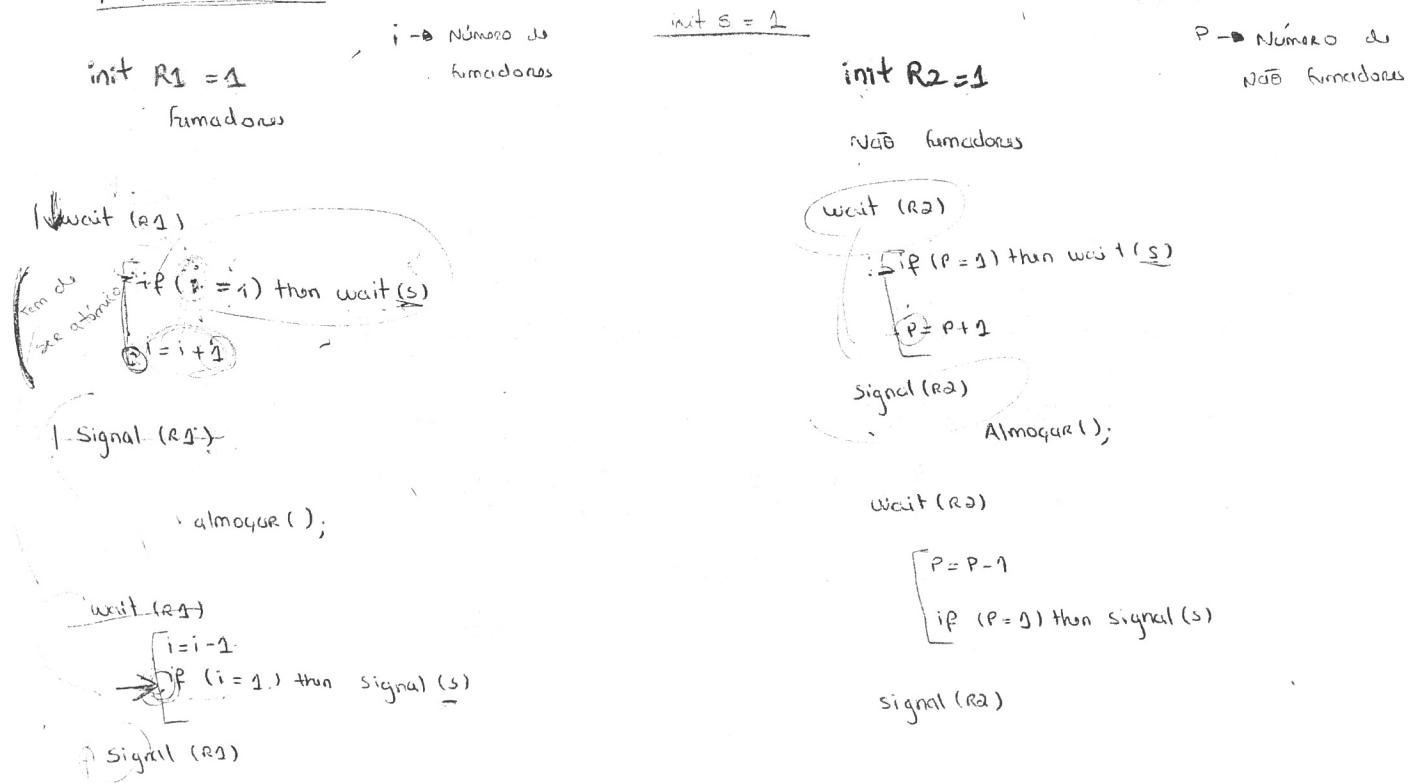
Eu fui um rei sacrificado para que desse o rei que é leito no meu tempo, Eu sou o leitor para o

wait(2) para que o escritor seja impedido de escrever, mas não para no sinal (6) quando sai os outros escritores.

Já que não posso rodar (\rightarrow) com "Costurando" e o último tijolo é que necessitava o sinal (\rightarrow).

Para que este problema seja resolvido temos de garantir que qualquer é atómico, para isso é que eu vou usar o semáforo R, para o deixar atómico. Tais são usos de semáforos as instâncias seguem atómicas]

Nós temos de pensar se existem 2 tipos de restrições, e no que cada um tem de ter em conta. os escritores / leitores é um problema semelhante ao dos fumadores e não fumadores, em que tenho um espaço que só se tem fumador só admite n fumadores mas nenhum não fumador e vice-versa, sendo que quando a sala está desocupada pode entrar qualquer 2 dos 2.



Temos obrigatoriamente de mudar as variáveis (I, P) entre os fumadores e os não fumadores, sempre. Neste caso o colocar os semáforos R_1 e R_2 ≠ é apenas porque não existe problema em querer, porque não iria existir problema, neste caso, caso eu utilizar-se apenas 1 semáforo R , comum ao 2 processos.

OUTRO PROBLEMA

Lobos e Corderos, a solução é a mesma, sendo que ~~todos~~ podem existir n lobos e n corderos, mas não podem existir corderos e lobos ao mesmo tempo.

Porém se eu tiver lobos, corderos e alfares e os corderos não podem estar com os alfares ou com os lobos, mas os lobos podem estar com os alfares. Assim neste problema na secção crítica podemos ter 2 processos no mesmo tempo, pois isso usamos 1 contador comum aos lobos e às alfares.

$init\ R_1 = 1$
 $init\ s = 1$
 $init\ R_2 = 1$

$i \rightarrow$ Número de corderos
 $p \rightarrow$ Número de lobos e de alfares.

Cordino

• wait (R1)

if ($i=1$) then wait (s)

$i = i + 1$

Signal (R1)

Almocar () ;

wait (R1)

$i = i - 1$

if ($i=1$) then Signal (s)

Signal (R1)

Lobo

wait (R2)

if ($P=1$) then wait (s)

$P = P + 1$

signal (R2)

Almocar () ;

wait (R2)

$P = P - 1$

if ($P=1$) then Signal (s)

Signal (R2)

Alfau

wait (R2)

if ($P=1$) then wait (s)

$P = P + 1$

signal (R2)

Almocar () ;

wait (R2)

$P = P - 1$

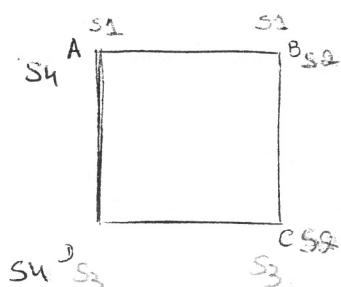
if ($P=1$) then signal (s)

Signal (R2)

S.O. (15)

OUTRO PROBLEMA

 eu tenha 4 professores todos incompatíveis
caso



Uma solução são ter varíos semáforos que vão dizer quais as incompatibilidades dos processos. Para saber quais os semáforos que tenho de chamar, coloco outro do if uma Contradição, então se a condição for verdade fago 1 semáforo por exemplo se é verdade então fago S2.

O que é uma Solução quando eu não quero misturas entre processos, mas que posso ter n elementos do mesmo processo ao mesmo tempo.

existem n mecanismos quando usamos semáforos que são mais ou menos iguais, existem pessoas que não utilizam o

wait e o signal e usam outras construções. As vezes existem instruções instalados no hardware, uma instrução de

'Baixo nível' que pode ser utilizada no Controlo do Sincronismo como o test & set, o que ele faz?

if ($i=0$), $\infty \{ i=1 ; \text{return } 0 \} \rightarrow 2^{\text{a}} \text{ Modifica esse valor}$

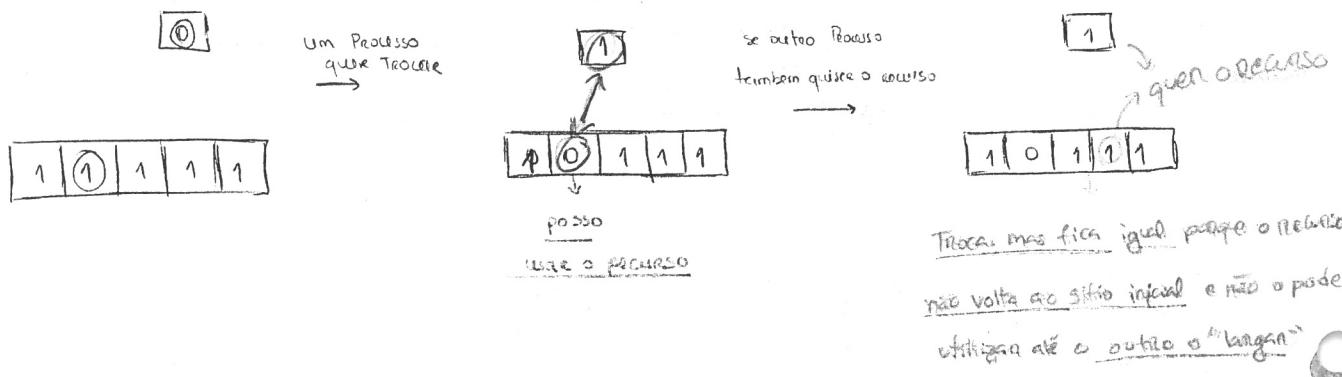
↓

1º Confirma o valor de i

else return (1);

Existe outra opção:

Visualmente é mais "apelativo" utilizar algo semelhante que é o exchange. Se eu tiver n processos cada um deles tem acesso a uma certa variável e existe uma outra variável que é interlocked. Para isso Copio o que tenho para uma variável temporária e só depois volta a ser Copiado, ~~Assim garante-se~~ se eu conseguir garantir que em baixo nível consigo mudar as variáveis numa instrução.



OUTRO PROBLEMA

Problema dos filósofos, que não tem uma solução ótima, existem 5 filósofos que estão à mesa cada um precisa de 2 recursos para comer (que podem ser os garfos) existem uma série de incompatibilidades.



1º Solução

Vamos associar semáforos aos garfos, primeiros vão agarrar o da direita (garfo) com o wait e depois agarrar o da esquerda, porém se isto acontecer ao mesmo tempo eles ficam em deadlock, ou seja, nenhum deles consegue agarrar o da esquerda, porque se isto acontecer ao mesmo tempo eles ficam em deadlock, ou seja, nenhum deles

consegue porque todos ficam com 1 garfo na mão

2º Solução

Se um deles agarrar no garfo, os outros já não podem agarrar em nenhum garfo, assim só 1 deles pode

comer de cada vez. (Existem restrições a mais)

3º Solução

O filósofo vê se os seus vizinhos não estão a agarrar no garfo, só quando os 2 garfos forem disponíveis

é que este pega nulos, mas se 1 vizinha tiver a comer e o outro não, o filósofo tem de esperar, mas se os vizinhos estiverem ocupados, este filósofo não vai comer, fica infinitamente

• à esquerda, fica em starvation (falta de equidade). Alguém que foi inicialmente cooperativo, esperou pela • comer, fica infinitamente assim sem comer.

À máxima equidade não coincide com a máxima eficiência, os programas assim não são os mais justos mas também não são o mais eficiente porque uma coisa impede a outra.

Deadlocks

Starvation → No problema dos filósofos, nunca comer porque nunca tinha os 2 garfos (é demais Cooperat)

Deadlock → No problema dos filósofos, diger que todos 1º agarraram o garfo esquerdo e depois direito, ao mesmo tempo, e assim eles ficam eternamente à espera do 1º garfo.

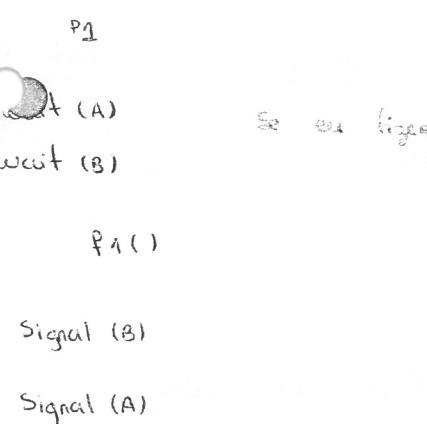
Deadlock Com 2 Processos

Ambos os processos precisam de 2 recursos A e B, se eu colocar um semáforo de modo a garantir que um recurso só pode ser utilizado por um processo de cada vez.

A

P₁ . P₂

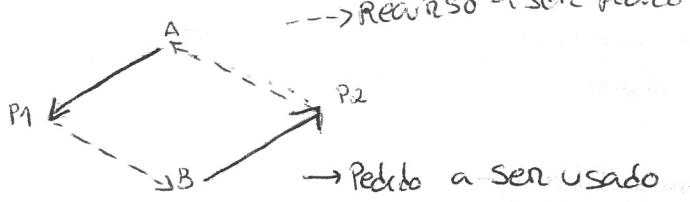
B



* Pensarmos que o P₁ está a utilizar o recurso

A e o P₂ está a utilizar o recurso B, caso isto ocorra tudo ao mesmo tempo, então neste caso os 2 recursos estão a ser utilizados, então vai existir deadlock porque o P₁ precisa dos 2 recursos mas só possui um deles já que o outro está a ser utilizado, o mesmo ocorre com o P₂.

Gráfico



Sempre que existe um deadlock, existe um ciclo que é feito pelas setas, no gráfico.

P₁ → P₂ (Recurso a ser usado)

P₂ → P₁ (Recurso a ser pedido)

O P1 fica a espera do recurso B e o P2 fica à espera do recurso A, eternamente à espera em deadlock. Eu posso tentar contornar este problema, neste caso caso tiver-se chamado pela mesma ordem os recursos A e B nos 2 processos, este problema não teria ocorrido.

Deadlock \Rightarrow 3 ciclo

Qualquer número de recursos

Deadlock \Leftrightarrow 3 ciclo

Quando apenas existe 1 unidade de recursos

} Se ocorre uma equivalência se cada processo apenas tiver uma unidade

Caso exista um ciclo podemos afirmar que existe um deadlock se apenas existir uma unidade de recursos,

senão não sei se existe deadlock.

O sistema operativo vai fazendo várias verificações para ver se existe deadlock, caso este detecte um deadlock então este vai tentar "quebrá-lo".

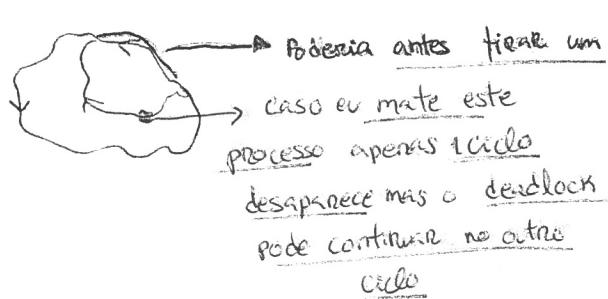
Como se elimina um ciclo

Ao se eliminar o ciclo, eliminar-se o deadlock, ou seja, o deadlock deixa de existir para isso é

necessário eliminar o ciclo, eliminando-se um nó (um processo) esse problema fica resolvido. O que o sistema opera-

tivo vai fazer é matar um processo e dessa forma o deadlock desaparece.

Se eu tiver:



Algoritmo de Detecção de Deadlock

como tomar a decisão sobre qual o processo que é morto.

Alguns critérios que podem ser usados:

S.O. (17)

- Nº Processos a tratar
- Prioridade dos Processos
- olhar para a quantidade de trabalho investida e o que falta para trabalhar (devisão estimada para terminar)
- Tempo gasto em cada processo
- Aleatoriamente

Estes critérios são pausados com os de escalonamento.

Depende do escalonador para saber qual é o melhor critério a ser usado, se o escalonador usar a Prioridade, então a Prioridade dos Processos vai ser o melhor critério, porque é uma informação classificável, caso de escalonar o trabalho com o tempo então o Tgasto em cada processo é o critério melhor.

A Prioridade dos Processos e o Tempo gasto são os mais usados, mas caso não haja estes 2 critérios sej possivel utilizar então é utilizado feita a escolha do processo aleatoriamente, já que a tendência é se escolher o mais simples possível.

O sistema operativo vai fazendo a verificação se existem deadlocks ou não, estes critérios vão ser apenas usados quando já existe um deadlock.

Algoritmo do Banqueiro (Algoritmo de Deadlock Avoidance)

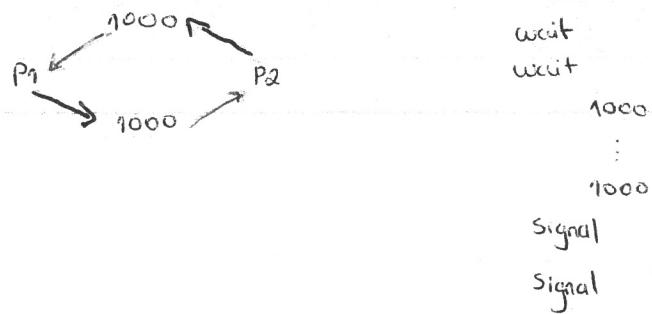
Impede que ocorra deadlocks

Exemplo:

existem 2 clientes, cada cliente necessita de 1000 euros primeiramente para abrir um negócio, no entanto no total os 2 clientes vão necessitar de 2000 euros para abrir o negócio, o dinheiro total que o banco tem para emprestar é 2000 euros, então se inicialmente o P1 pedir 1000 euros ao banco e este lhe o deu e o P2 pedir 1000 euros também ao banco, se este lhe deu depois vai existir um deadlock porque o P1 passado um tempo vai necessitar de mais 1000 € tal como o P2 e o banco já não tem dinheiro para dar.

o banco já não tem dinheiro para dar.

lhes dar.



No Algoritmo do Banquiro o que está em causa é a existência de um Pedido e a detectão se não desse pedido, ou seja, o que o Algoritmo iria fazer neste caso é quando o P2 pedisse 1000 euros este iria dizer que não a esse pedido porque sabia que existia a possibilidade de este ficar em deadlock, o Algoritmo do Banquiro diz que não aos pedidos desde que existe uma possibilidade de existir deadlock Algoritmo que tem uma pergunta e que consequentemente uma resposta sim ou não.

Como ver se existe ou não um deadlock

Recursos Pedidos (Request) e Recursos Alocados

o que PRECISO			o que tenho				
Pedidos			Alocados				
	A	B	C	A	B	C	
P1	0	1	2		2	1	0
⋮							
Pn							

↓ ↓

matriz dos Pedidos matriz dos Recursos Alocados

Consegui saber se existe ou não um deadlock através das matrizes.

Detectão de deadlocks através das matrizes

Para existir um deadlock \Rightarrow existe ciclo



Nestes 2 casos em que só à pedido de recursos ou só existem recursos utilizados, os Processos

--- deadlock ocorre nunca irá poder fazer parte de um ciclo.

Para conseguir saber se existe ou não deadlock preciso de ter a matriz dos Pedidos, a matriz dos recursos Alocados e o número total de recursos, já que se esse número for infinito ou muito grande não irá existir deadlock \Leftrightarrow porque não existem sempre recursos disponíveis.

exemplo:

Pedidos

	A	B	C	D	E
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Alocados

	A	B	C	D	E
	1	0	1	1	0
	1	1	0	0	0
	0	0	0	1	0
	0	0	0	0	0

(total de recursos)

2 1 1 2 0

vetor dos recursos totais em

vetor dos recursos disponíveis:

1 2 1 2 1

→ 1º Passo:

fazer a sema coluna a coluna da matriz dos Alocados, para saber os recursos totais que já estão a ser utilizados. 21120

→ 2º Passo:

Calcular os recursos disponíveis, $\boxed{\text{recursos totais} - \text{recursos utilizados} = \text{recursos disponíveis}}$.

$$21121 - 21120 = \underline{\underline{00001}}$$

→ 3º Passo:

se eu tiver processos que não tem recursos Alocados, ou seja, uma linha só com zeros, então eu

sei que esse processo não vai estar em deadlock, porque é do tipo $\leftrightarrow 0 \leftrightarrow$.

Neste caso é o Processo 4

→ 4º Passo:

Olho para os recursos disponíveis e olho para a matriz dos Pedidos e veja se consigo satisfazer algum pedido com os recursos que tenho disponíveis.

Neste caso como tenho 00001 de recursos disponíveis consigo satisfazer o Processo 3

Como consigo satisfazer o processo que lhe empreste os recursos e este vai correr e quando ele acaba

os recursos que tinha aloccados vai ficar recursos disponíveis.

$$\text{recursos disponíveis} = 00011$$

→ 5º Passo:

Como os recursos disponíveis são diferentes dos anteriores vai voltar a olhar para os pedidos a matriz dos pedidos de para ver se consigo satisfazer algum pedido que falta satisfazer.

Neste caso falta satisfazer o P1 e o P2 mas não o consigo satisfazer com os recursos que tenho disponível.

Conclusão:

os processos que eu não consigo satisfazer são os processos que estão em deadlock caso eu conseguisse que todos os processos conseguissem correr não existiria deadlock.

Neste caso os processos P1 e P2 estão em deadlock

OUTRO EXEMPLO:

Pedido

Alocados

A B C D E

A B C D E

P1	0	0	0	2	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

1	0	1	1	0
1	1	0	0	0
0	0	0	2	0
0	0	0	0	0

$$1^2 + 2^2 1^2 1^2 3^2 0$$

Vetor de Recursos Totais =

$$= [21131]$$

$$2^{\text{d}} \text{ recursos disponíveis} = 00001$$

3º O P4 nunca está em deadlock

S.O. (19)

4º O P3 consegue-mos correr com o vetor de recursos disponíveis. Os recursos disponíveis após o P3 consegue ficar 000021, agora já consigo correr o P1 e depois de este correr os recursos disponíveis são 2031 e agora já consegui correr o P2. sendo que neste caso não vai existir deadlock

Algoritmo do Banqueiro

Nunca existe deadlock

Total que cada Processo precisa

Preciso da matriz de Alocados, Preciso de saber o total dos recursos, A matriz dos recursos necessário

e também Preciso de saber o que falta a cada processo até ao final para cada um conseguir correr.

diferença entre recursos necessários e a matriz de Alocados
diferença entre recursos necessários e a matriz de alocados

Neste caso os pedidos são feitos Processo a Processo.

Estado Seguro \Rightarrow quando a matriz de recursos Necessários e a matriz de Alocados depois de um Processo correr ficarem com valores que não dão deadlock.

Exemplo:

	Recursos Necessários de Cada Processo			Alocados	A	B	C
	A	B	C	A	B	C	
P1	3	2	2	1	0	0	
P2	6	1	3	6	1	2	
P3	3	1	4	2	1	1	
P4	4	2	2	0	0	2	

falta \rightarrow diferença entre Matriz Alocada e os Recursos Necessários

A	B	C
2	2	2
0	0	1
1	0	3
4	2	0

Vetor dos recursos totais = 936

\rightarrow 1º Passo:

fazer total de recursos Alocados 923

\rightarrow 2º Passo

fazer a conta para saber os recursos disponíveis = totais - Alocados = 013

\rightarrow 3º Passo:

Comparar o disponível com o que falta a cada processo
este caso consegue correr o P4 e depois de este correr o que está alocado é libertado

recursos disponíveis = 625, como agora já consigo correr qualquer 1 dos processos que falta correr então significa que o estado do sistema Noquele instante é um estado seguro, garantindo que a partir daquele momento consigo chegar ao final sem nenhum deadlock.

Estado seguro \rightarrow quando recursos disponíveis for maior que todas as entradas a matriz falta de todos os processos que ainda não correram, garantindo que a partir daquele momento consigo chegar ao final sem nenhum deadlock.

Banquinho

O Processo P1 faz como pedido 001, se eu acita-o o Pedido o matriz Alocamento aumentaria os recursos disponíveis diminuam, a questão é se eu aceitar o Pedido o estado é seguro ou não?

Retomo todo o raciocínio que fiz anteriormente mas acrescento os 001 nas ~~000~~ a matriz Alocamento e diminuo os recursos disponíveis, se eu no momento final ficar no estado seguro então eu aceito o Pedido, senão não aceito o Pedido.

se o estado não for seguro não quer dizer que se quita-se o Pedido ele entra-se em deadlock podendo eu não entrar, assim elimina-se a possibilidade de entrar.

Gestão de memória

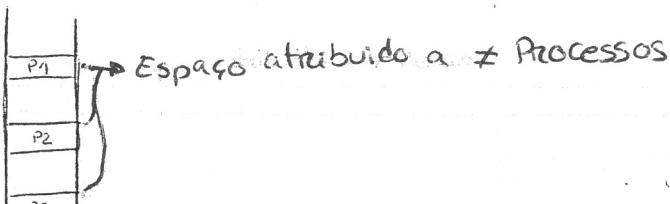
Descrição sistemas ao longo do tempo

As já, mais antigas são quase simulações e são bastante mais simples que as atuais

Segurança

Tenho vários processos a rodar por isso tenho de garantir que nenhum processo esteja outros processos

mem



existem estruturas especiais no CPU que indicam qual a zona de memória autorizada para cada processo.

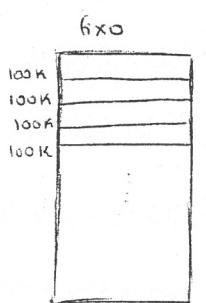
Como determinar em que zona os processos são alocados

A forma mais simples de fazer a alocação, ou seja, de decidir onde o processo vai residir é:

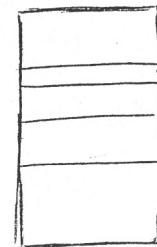
• fixo

Significa que uma zona de memória está pré-determinada, para um processo, ou seja, esta pré-determinada a zona onde um processo começa e Acaba, cada zona para um processo é igual, tem o mesmo comprimento, isto implica que todos os processos tem de ser menores que a capacidade de memória utilizada, já que os processos ^{fazem} parte muito menor que essa capacidade vai existir muito espaço inutilizado.

Sabe-se exatamente onde está o slot, que contém cada processo



fixo

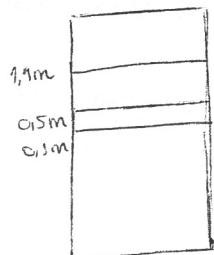


→ Sabe-se à mesma onde cada processo termina e acaba mas cada slot tem uma dimensão.

lógicas

Dentro destas existem algumas que são mais elementares que outras.

1º Gestão memória Contígua



Pode atribuir o espaço de memória que depende da dimensão que o processo necessita se o processo precisa de 0,1M, então será um slot com esse exato tamanho que irá corresponder a esse processo.

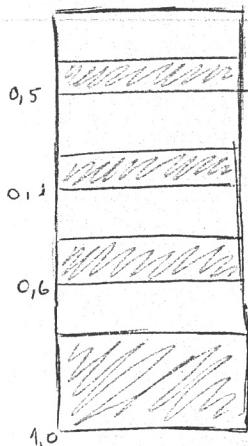
Quando um processo termina, ele liberta o espaço que estava a ocupar e quando um novo processo "pedir" memória o que faz?

Pode ter 4 coisas diferentes dependendo do algoritmo que eu irrei utilizar.

Ao longo do tempo vai existir n zonas livres com n tamanhos

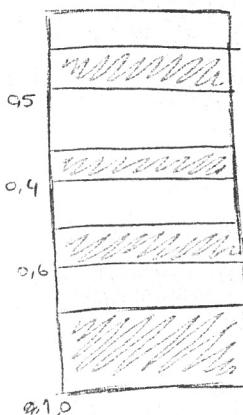
first fit (-Complexo)

Este Algoritmo vai procurar as zonas livres e quando o Processo couber numa dessas zonas, então vai ser alocado ali, convocar-se a Procurar o espaço livre desde o Princípio da memória.



Não espace livre

Algoritmo Best fit (+Complexo)



Se eu quiser alocar um processo que necessita 0,3, neste caso o processo couberia em todos os espacos; mas o espaço que ele vai ocupar é o espaço que vai ficar livre em cada uma dessas zonas vai ser diferente. Este algoritmo tenta achar um espaço mais próximo do espaço que o processo vai ocupar. Este Algoritmo vai fazer com que o espaço que sobra entre o espaço que é utilizado pelo processo e o espaço que este pode utilizar seja o mais pequeno possível.

Problema: se o espaço que é fica sem ser utilizado for demasiado pequeno deixa de servir para qualquer processo.

Next fit (-Complexo)

É parecido ao first fit, porém invés de Procurar o espaço livre pelo índice, existe um apontador que indica qual o último espaço utilizado e a procura do espaço livre vai ser desde esse apontador para

Cima.

Vantagem: os processos vão ficar mais espalhados, assim existe um uso mais uniforme da memória.

Existe vantagem no ponto de vista do Hardware.

Desvantagem: não garante que tenho zonas livres se um processo muito grande chegar, porque se a memória

esta mais partida existe menos

S.O 6.1

Não é fácil perceber do ponto de vista do cálculo matemático qual dos dois, Next fit ou first fit, apresenta sente mais vantagem apenas se consegue perceber pela simulação.

Worst fit (+ Complexo)

Se eu colocar um processo num grande espaço da memória, então o espaço que ele ocupa será um grande espaço.

entre o worst fit e o best fit apenas se sabe qual o melhor por simulação.

→ os Complexos são os mais usados

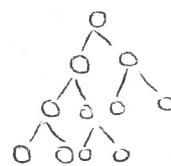
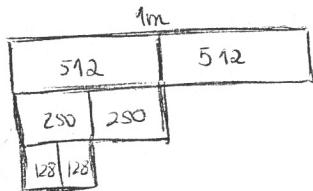
Compactação ou desfragmentação (fragmentação externa)

Se eu tiver muita fragmentação na memória e já tiver muitos espaços livros pequenos que já não tem capacidade por serem demasiado pequenos, vai-se alterar o sítio onde estão alojados os processos na memória de forma a que o espaço que está a ser utilizado fique todo junto, seguido deendo que assim o espaço que está livre será muito maior.

Só ocorre este processo ou passado algum tempo ou quando a memória já está muito cheia.

OUTRO SISTEMA

Bottom System



A memória possui uma dimensão que é Continua e que possui divisões e sub-divisões sempre por metade, isto é que

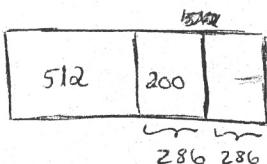
é uma árvore.

é um sistema de memória Continua porque consegue dividir a memória em bloco com diferentes dimensões, porém também é não continua porque consegue sempre saber a posição de cada bloco e qual a capacidade de cada um deles.

Imagineando que Possuo estes Processos: esta fragmentação:

32 64 128 256 512 1024

Consegui fazer a gestão mais rápida e eficiente com pedaços que não são utilizados.



fragmentação Interna.

Espaço livre, que sobra do processo com a capacidade total do espaço.

O que irá antes acontecer é que eu irei "partir" a memória até que o valor da memória que eu consigo alocar o que quero seja o mais próximo. Possível portanto se eu quero alocar 200 tenho de fragmentar a memória até os 286 porque se fragmentar mais o processo já não irá caber e se fragmentar menos o espaço livre vai ser maior.

Quando o processo terminar ver se o vizinho neste caso 286 também está livre porque se tiver o lado está livre, se tiver este juntam-se.

É um Algoritmo muito rápido.

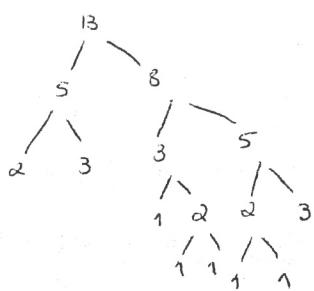
Tem alguma fragmentação interna, o disspendício de espaço não tem um Grande Impacto.

Variantes do Body System

Em vez da dimensão ser dividida por 2, as slots são divididas de outra maneira, utilizando a sequência Fibonacci.

Fibonacci, em que se somam os últimos 2 números 1, 1, 2, 3, 5, 8, 13, ...

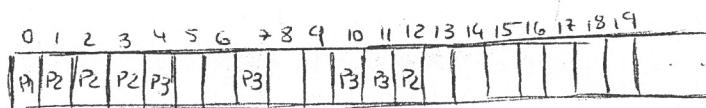
O mais importante é que o Algoritmo seja rápido na divisão das slots.



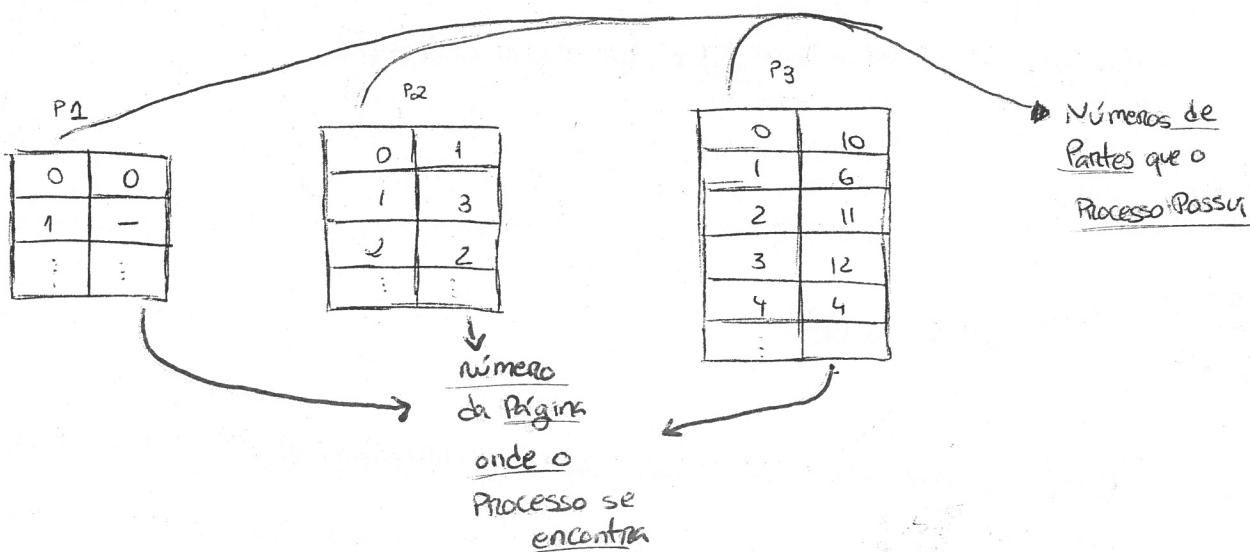
é um Algoritmo mais Complexo mas também é mais eficiente porque o comprimento das slots cresce de uma maneira mais rápida, então existem mais medidas de slots com diferenças mais pequenas.

Além de mais tempo de alocação fixa e dinâmica com algumas restrições

Pega na memória e divide-a em pedaços pequenos. Um Processo não tem de caber num bocado. Cada espace cada divisão corresponde a 1 Página. Cada Processo pode estar em mais de 1 Página Consoante o tamanho do Processo, as páginas referentes a 1 Processo não precisam de estarem seguidas, mas tenho de saber quais as Páginas referentes a cada Processo.



PAGE TABLE → Tabela de Páginação, diz fisicamente onde estão a reunidas as Páginas de Cada Processo.



Nesta tabela existe a representação física de cada Processo

Para gerir as Páginas que estão livres existem diferentes Algoritmos:

Bit RAP

Tenho Bits que me dizem se as páginas estão ou não ocupadas ou não, existe a denotação o caso estejam desocupadas e 1 caso estejam ocupadas • Tenho de saber qual o número de páginas que estão livres, para saber se o Processo que eu quero alojar tem espaço suficiente na memória que está livre.



Para saber que páginas permanecem ocupadas posso usar o next fit, o best fit, o first fit ou o

worst fit, a não ser que existam restrições ou preferências de Hardware posso utilizar Algoritmos que Preferir.

Não existe fragmentação externa, pode só existir fragmentação interna porque se eu tiver um processo que que só irá utilizar um bocado de uma página o que sobra é fragmentação interna.

Quanto mais pequenas as páginas são mais pequena será a fragmentação interna, porém os peges tables serão maiores.

Qual é o tamanho máximo de uma page table?

Depende das tabelas de Paginação e depende do Espaço de endereçamento do Processo.

Se o espaço de endereçamento for de 32 bits

$$2^{32} = 2^{10} \times 2^{10} \times 2^{10} \times 2^2 = (10^3)^3 \times 4 = 10^9 \times 4 \rightarrow \text{que Poderia entregar}$$

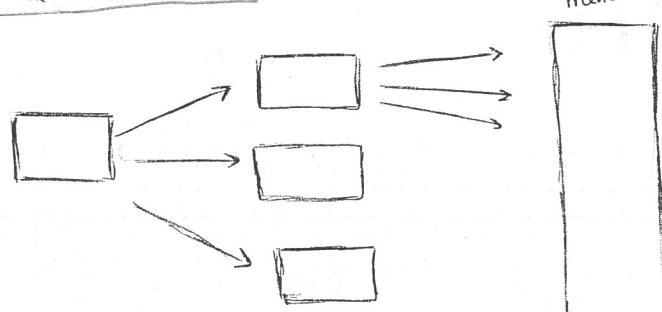
se tiver Páginas de 4k

$$\frac{2^{32}}{2^{32} \cdot 2^{10}} = 2^{20} \approx 10^6 \rightarrow 1 \text{ million}$$

Significa que 1 tabela de Páginação de um sistema com espaço de endereçamento de 32 bits e que ocupa x o espaço todo, a tabela de Páginação teria $\approx 10^6$. Era uma tabela muito grande e como é que eu guardo uma tabela de endereçamento, se eu sei o número de páginas sei quantos bits ou megabits, neste caso 3 megabits, são necessários para guardar a tabela de Páginação, como tenho de guardá-la na memória e esta não cabe numa única página iria utilizar muitas páginas para guardar a tabela de Páginação. Eu vou ter 1 página que me aponta onde vou ter as outras páginas e nessas páginas vou ter a tabela de Páginação que tem os apontamentos que me mostram onde está a informação que a tabela de Páginação contém em memória, as páginas dos processos.

Paginacão de Niveis

Paginacão de 2 níveis

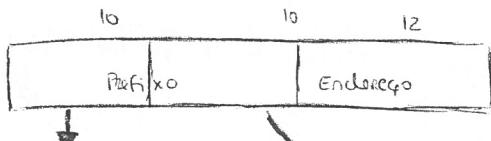


de $4K$ cubos dentro da página logo apontadores que iam apontar para páginas diferentes que por sua

vez vão apontar para outra parte da memória.

É mais fácil fazer pelo Endereço

Tenho endereço de 32 bits



depois na página a que

fui pelo endereço

do número (do endereço)

que tem um número

desde o até 1023,

este diz-me qual a

Página em que tenho a

informação então

tira o endereço da

Página que quero ler

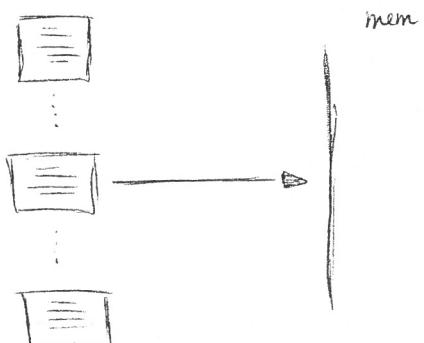
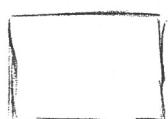


Tabela de

1º Nível

(1ª parte pesquisa na primeira tabela)

... resultado

Pesquisa pela
tabela de 2º Nível
(várias tabelas)

deslocamento de

Página de shift
da Página

a 2ª tabela tem sempre de existir a tabela do 2º nível porque se não existe

Consequências:

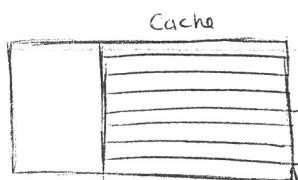
Nos sistemas de Paginação com 2 e 3 níveis, para que eu saiba uma posição de memória, tenho de ter

uma consulta à RARF para ver a tabela do 1º e 2º nível e também para ir ler e escrever o que quero

à memória.

Quando tenho um endereço tenho de utilizar a RARF o que torna o processo lento.

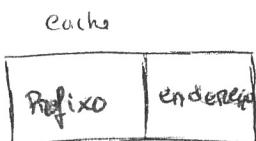
Para isso existe uma solução



TLB (Table Lookaside Buffer)

é uma mémoria associativa, ou seja, é uma tabela que para cada entrada temos 1 endereço significa que sempre que tiver algo que tenha um Prefixo que eu já tinha tido acesso a informação existe um endereço \downarrow $10,10$

relacionado que é o endereço da informação que se necessita.



Nos proximos aulas é só diger o Prefixo caso esteja já existir, ou seja, o Prefixo vai entrar em todos os entradas e caso existam Prefixos iguais, então já se possui o endereço à informação, senão, caso não existam Prefixos iguais teremos de ir a 1^o e 2^o tabela de Paginação até achar o endereço.

Mémoria Associativa é mais páspida que o acesso à Ram.

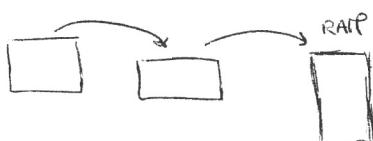
Impacto da TLB

TLB - 10 ms

RAM - 100 ms

• Sem TLB

Paginação de 2 níveis



$$\text{Tempo médio de acesso} = 3 \times 100 \text{ ms} = 300 \text{ ms}$$

HIT Ratio $\rightarrow 95\%$. \rightarrow quanto estiver no cache depende das entradas

Quando o Prefixo $\overset{\text{1}}{\underset{\text{2}}{\text{não}}} \text{existe}$

TLB
|
RAM

Quando o Prefixo ainda não existe na cache

TLB
|
RAM

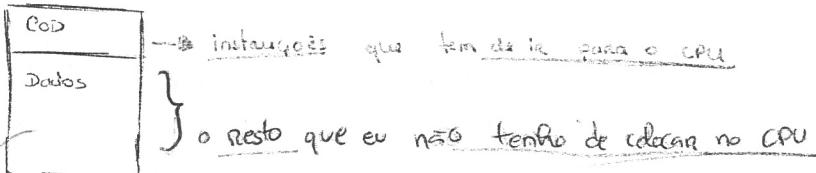
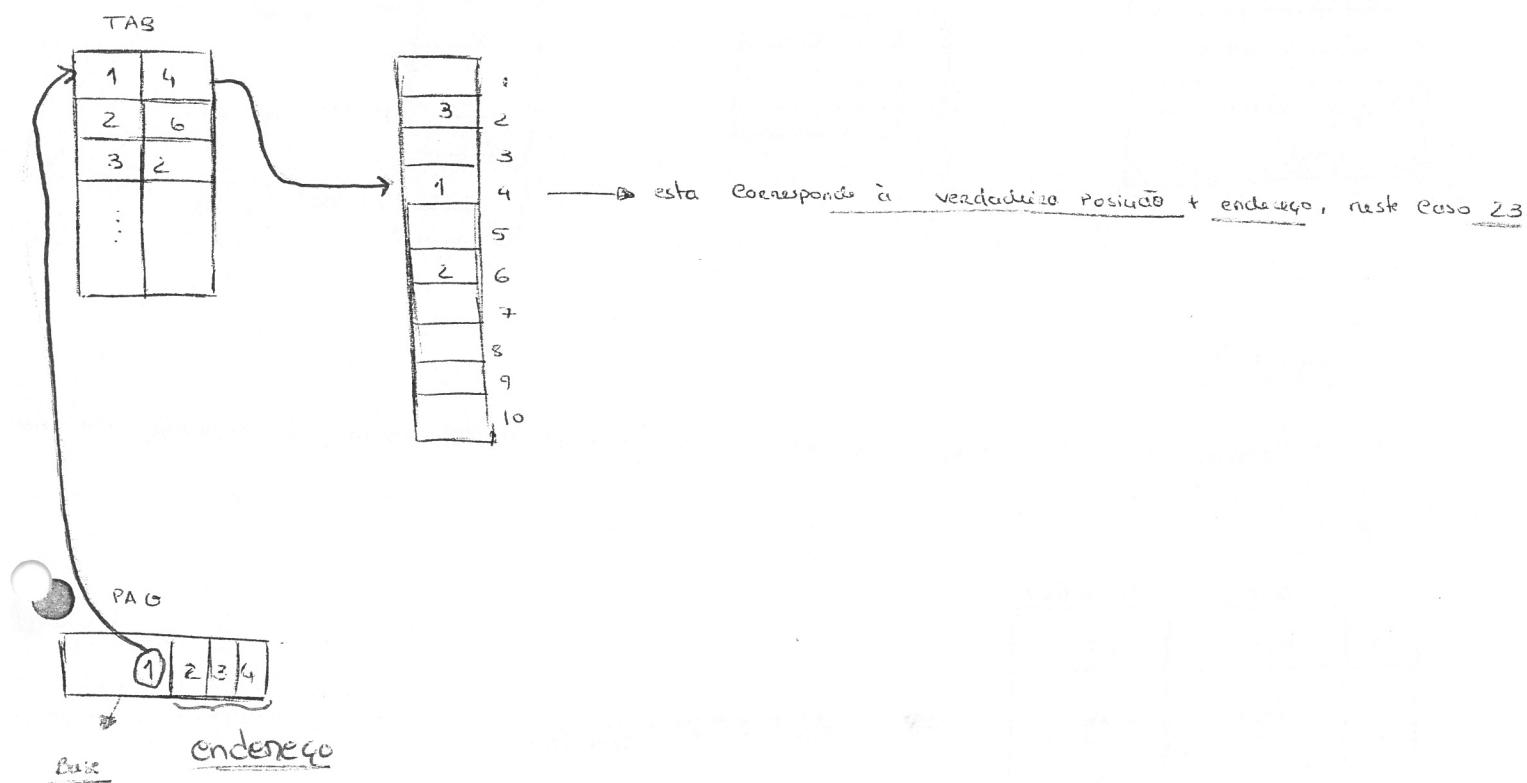
$$3 \times 100 + 10 = 310 \text{ ms}$$

$$10 + 100 = 110 \text{ ms} \quad \text{tempo médio} = 0,95 \times 110 + 0,05 \times 310 = 120 \text{ ms}$$

Gestão de memória

S.O. (24)

- Correspondência entre a TAB (distribuição da memória do ponto de vista lógico) e onde está fisicamente



Dados → Algo que o input é suposto escrever

forma mais comum de stocar tramas no CPU:

o que tenho de garantir é que não estou a utilizar (bits) Bytes, de forma excedidas, eu não quero ba-

nhave o que é código com dados ou estocar outros processos,

variáveis globais
variáveis locais
chamadas de funções

} Dados

Armazenar código fisicamente

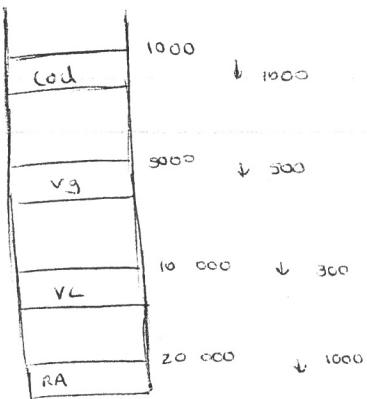
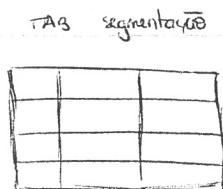
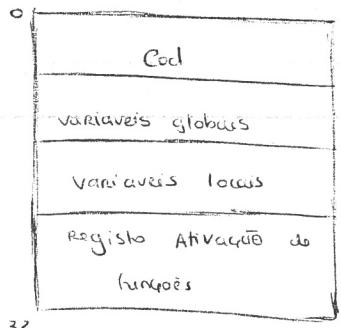


Tabela de segmentação

(as corespondem onde estão guardados os códigos, mas tem de se ter cuidado porque só o informaçao tem uma dimensão diferente).

	Base	dimensão
0	1000	100
1	5000	900
2	10 000	300
3	20 000	1000

↓ ↓ ↓

Posição onde onde Acaba

Comeca

⇒ Representação de Memória

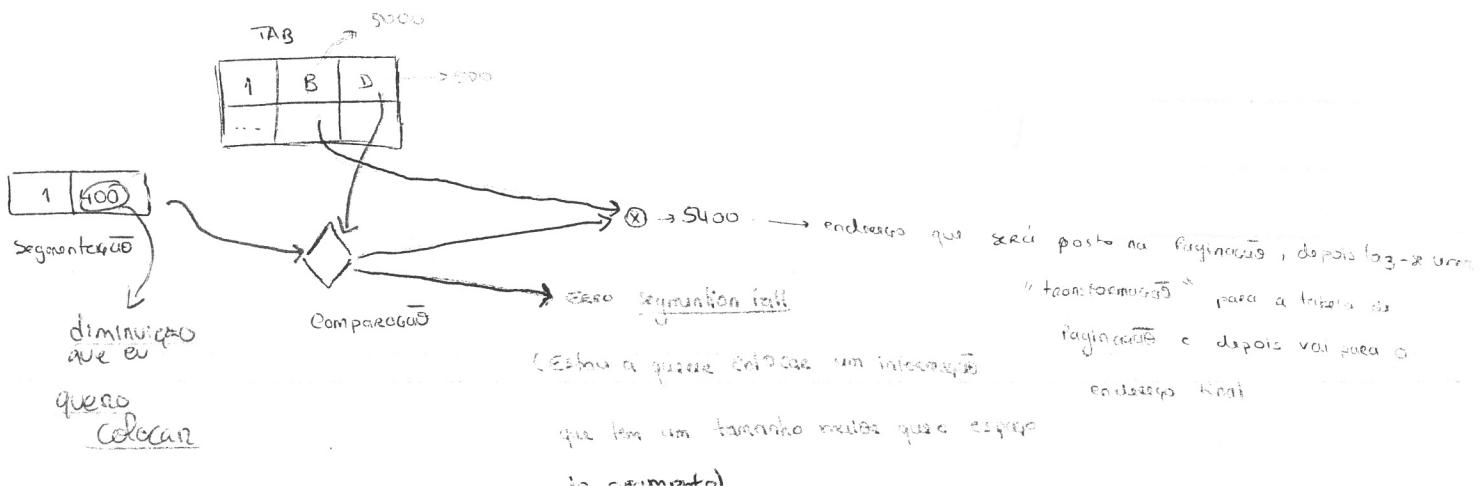
Sem ser física

1º utilizamos uma tabela de segmentação e depois um tabela de Páginação, que irá mostrar em que parte física da memória se encontra a informação

física da memória se encontra a informação

No Páginação tem tudo o mesmo termo (dimensão do segmento, porém na tabela de segmentação isso já não

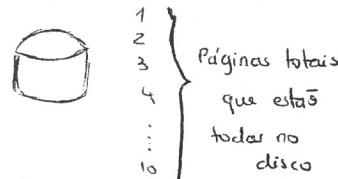
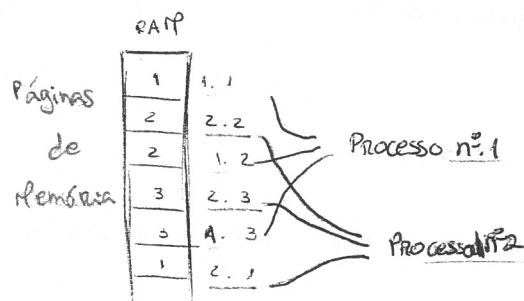
acontece então tem de ter uma verificação a partir da dimensão do segmento)



Costuma funcionar em cima das tabelas de segmentação e paginação

Existem partes do processo que não vão ficar na memória, mas fica num dispositivo, no disco, memória

Virtual → Relações entre a memória física e o disco



Tanto o Processo 1 como o

Processo 2 não estão na memória todo o processo, mas cada um delas tem 10 páginas que estão todas no disco, mas nem todos estarão na memória.

estão mv

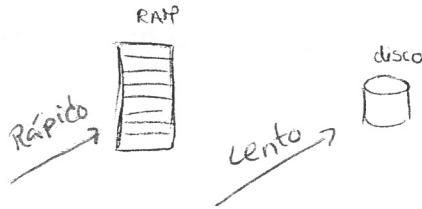
leitura, vou fazer leitura à memória e volto

e escrever:

Vou escrever na Ram e no disco, mas com ele tenho a certeza x são iguais ou diferentes, a escrita pode

ser feita diretamente na memória e depois faz-se uma atualização para o disco, quando o numero de escritas é pequeno, porque se eu quiser ler, como a escrita é para terho o que quero na Ram, e o que eu tiver escrito já atualizou para o disco.

caso a escrita seja encoste que a leitura, antes de voltar a escrever que a informação já foi atualizada para o disco.



O que acontece quando 1 página não está lá?

se tiver espaço livre, significa que tenho de ir buscar informação ao disco e colocar na Ram, mas a maior parte das vezes o espaço pode não ser suficiente, então preciso de escrever em cima da outra página, mas tenho de garantir que apenas sustente páginas que já foram atualizadas.

para que digramente de atualizações possíveis, a informação que é armazenada no disco é mais atrasada,

do que a que se encontra em memória.

Quanto menos vezes é necessário ir buscar páginas ao disco quer dizer que o Processo é mais Eficiente e mais Rápido.

Algoritmos de Sucessões

Quando temos de escrever em Cima de outra Página, Como fazemos a escolha.

LRU

Uma página que foi usada a pouco tempo tem maior Probabilidade de ser usada então vai escrever na que foi usada a mais tempo

FIFO

A 1º Página a entrar é a 1º Página a sair, é um algoritmo simples mas per Vezes pode ser desastroso.

Exemplo:

As páginas que a memória pode guardar são apenas 3.

Ordem em que as Páginas vão ser Pedidas: 1, 2, 1, 3, 4, 1, 5
Como a Página 1 já está no RAM não preciso de ir buscar ao disco.

RAM

1
2
3

A página 4 não se encontra na memória, então tenho de a ir buscar ao disco, mas como não tenho um espaço livre vou ter que escrever em cima da outra Página, mas qual?

Sé utilizar:

LRU → Vou escrever em Cima da Página 2

FIFO → Vou escrever em Cima da Página 1, o que neste caso não ia ser vantajoso porque depois da

Página 4, volta a ser pedida a Página 4.

Algoritmo ótimo

S.O. 36

Este Algoritmo é usado para Compreensão, ele olha para o futuro e pensa quais os processos que vem a seguir, e a partir daí vê qual é o melhor Algoritmo a utilizar, fazendo a comparação com o algoritmo que utiliza.

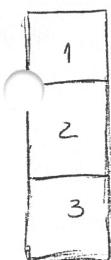
Tenho de ver qual a Página do Processo que me vai faltar no futuro.

Ele vai escolher escrever naquela que é mais longe do presente, aquela que no futuro vai demorar a utilizar.

Se não existir nenhum pedido para aquele processo, em nenhum instante, significa que é 0.

Ex:

1, 2, 3, 1, 4, 2, 3, 4 → Tempo que falta para pedido



3

1

00

→ Tempo onde vai faltar o Processo, quando ele volta a ser pedido

LRU → é um algoritmo mais complexo porque tenho de saber qual o instante em que cada Página do Processo foi utilizada.

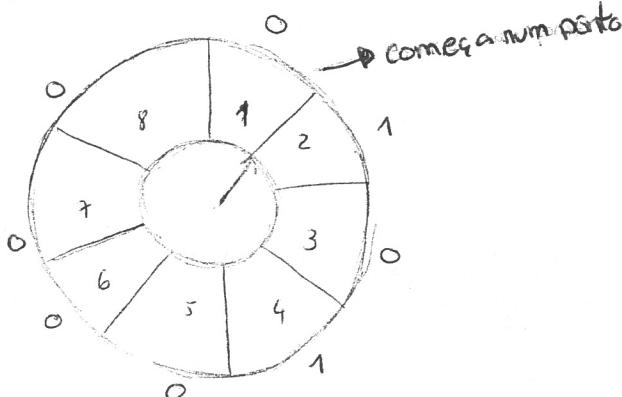
O Algoritmo procura no momento de decidir qual é o valor ~~máximo~~ menor do instante quando as páginas foram utilizadas.

Desvantagem: Implementação deste Algoritmo

Algoritmo clock (second chance)

Algoritmo que diferencia as páginas que foram mais recentemente utilizadas, para dar lugar a mais menos.

Processo.



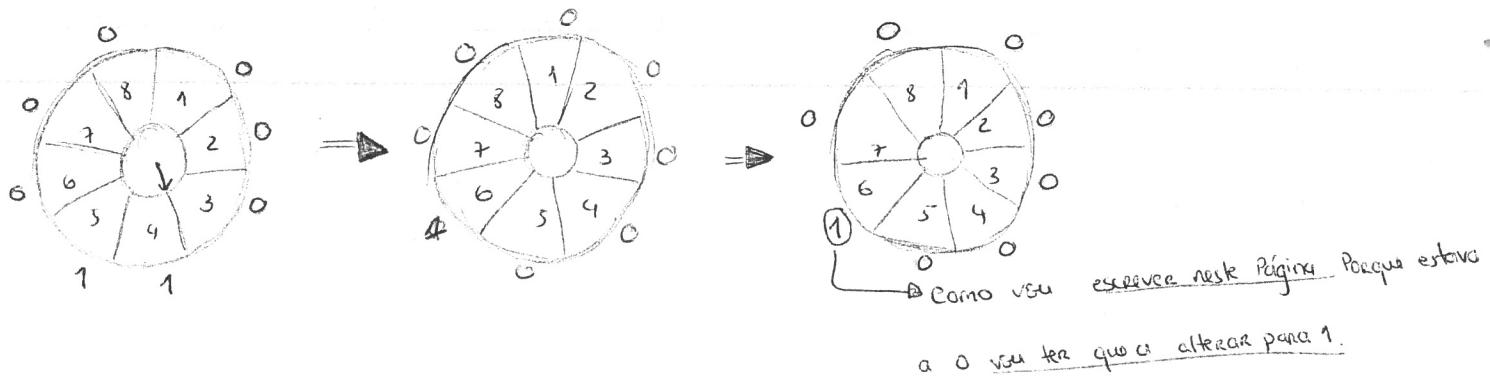
Vou numerar as Páginas com 0 e 1, o se não estiver

sido utilizada a Página e 1 caso a Página tenha sido utilizada.

Se eu passar por 1 página que esteja marcada com 1 vou mudar.

1 para 0, para que se todas as páginas tivessem marcadas a 1 na 2^a visita pelo menos 1 já está a 0.

Neste Caso continuo a provar no Página 4



uma página que está sempre a ser utilizada nunca seria escrita, porque quando passasse pelo Página ela tirava a 0, mas como depois dela logo utilizada voltava a 1.

Este Algoritmo não garante que a Página que é escrita seja a mais antiga, mas garante que a Página mais utilizada não é escrita.

É um Algoritmo mais Grosseiro do que o LRU, mas o Princípio é mais ou menos o mesmo.

Existem 2 versões do Algoritmo do clock uma com 1 bit e outra com 2 bits.

A versão com 2 bits é muito semelhante à da com 1 bit a diferença é que em vez de esta a 0 ou 1 tem 2 bits e para se escrever na Página a página tem de estar a 00, sendo incrementada até que se encontre a Página que esteja a 00, se escrever na Página a página tem de estar a 00, sendo incrementada até que se encontre a Página que esteja a 00.

caso não tenha espaço livre, alguma Página a 00, perca de pelo menos de 2 voltas para ter espaço livre.

Vantagem: Tem mais uma classe, logo vai existir mais uma diferenciação entre os Páginas, desempenho mais rápido.

Desempenho: Número de vezes que necessitamos de ir buscar informação ao disco.

Exemplo:

se eu apenas tiver espaço apenas para 3 Páginas.

Página falt -> Páginas que faltam e tem de se ir buscar ao disco $\rightarrow F$

Hit -> Página que já se encontra na RAM $\rightarrow H$

1 2 3 4 1 3 1 2 1

O -> Página que se está a utilizar

$$H = 3 \quad F = 6$$

Último sítio onde se tinha escrito

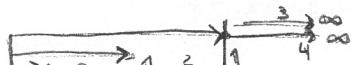
FIFO	1	1	4	4	4	4	4	H	H	F	H
	④	1	1	④	4	4	4	4	1	1	③
	①	2	2	①	1	①	1	1	2	2	③
	③	3	3	③	3	③	3	2	3	3	③

	F	F	F	F	H	H	F	H
LRU	① ②	1 2	1 2	④ 3	4 ①	4 ③	② 3	② 3
	③	③	3	3	③	3	3	3

123413121

S.O (37)

$$H = 3 \quad f = 6$$

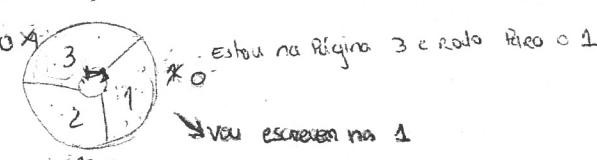


	F	F	F	F	H	H	H	H	F	H
OPT	④	1	1	1	①	1	①	1	①	1
	②	2	④	4	4	4	4	②	2	2
	③	3	3	3	③	3	3	3	3	3

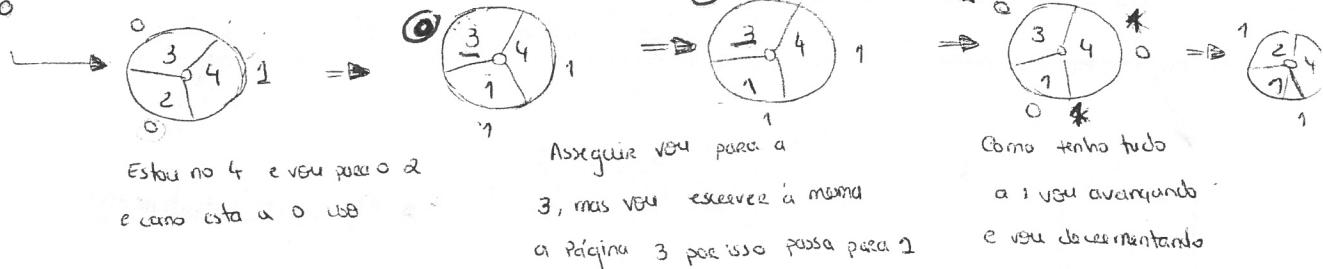
$H = 4$ $f = 5$ \Rightarrow Mithat que isto era impossível

um litit é sempre mais rápido que um ração tatt.

	F	F	F	F	F	H	H	F	H
clock	①	②	③	④	⑤	⑥	⑦	⑧	⑨
	1	2	3	4	5	6	7	8	9

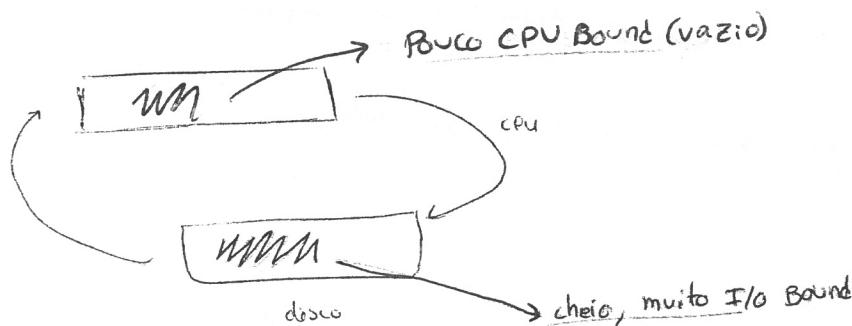


Estou na Página 3 e Rodo Páro o 1



România mult chiac.

se existe pouco espaço para cada processo com memória virtual que diga que existem poucas páginas em memória e é mais frequente se iram buscar páginas do disco.



Se o disco for mais usado a CPU pode decidir se quer a criação de novos processos ou então colocar os processos que estão em suspend, mas assim o espaço em memória fica cada vez mais pequeno, então este fica

mais cheia e existe mais probabilidade de existirem Page falls e se isso acontece o CPU vai ficar ainda mais cheia e isso peca ainda mais cheio, porque vão mais processos buscarem páginas ao disco.

Thashine → gerado pelo mecanismo de gestão da memória e escalonamento do Processo que quando funcionam juntos

deixam porcaria (isto é o disco estar muito cheio, excesso de Page faults)

Podemos dizer que quando o espaço entre cada Processo fica muito pequeno, ^{devendo} evitá-lo e assim não existem tantos Page faults.

Sistema de ficheiros



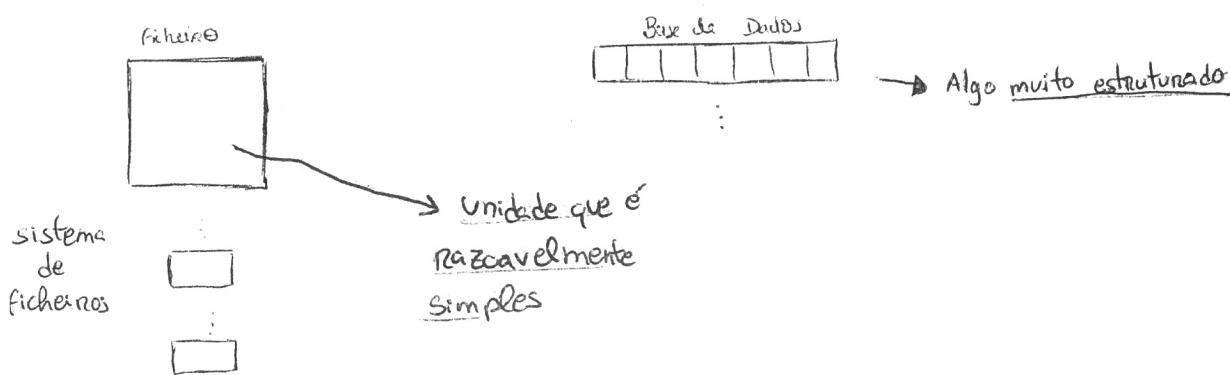
Serve para guardar informações mas é diferente da Base de dados.

Se existirem erros tenho de saber

Só algumas pessoas podem manter na informação

Quais as diferenças entre Base de Dados e Sistema de ficheiros

Os sistemas de ficheiros são mais simples e menos estruturados



Existem formas mais exigentes e mais estruturadas para guardar ficheiros

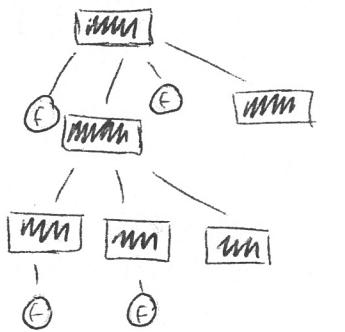
Sistema de ficheiros → equilíbrio entre a exigência (a estrutura) e a sua complexidade

As unidades (os ficheiros) têm um comportamento que é variável, a sua dimensão

O que existe numa Unidade

Meta-dados → são dados sobre os dados. Tem informação que nos ajuda a trabalhar os dados (a data de acesso, quem pode ter acesso, tipo de ficheiro, ...)

Nome → Identificador

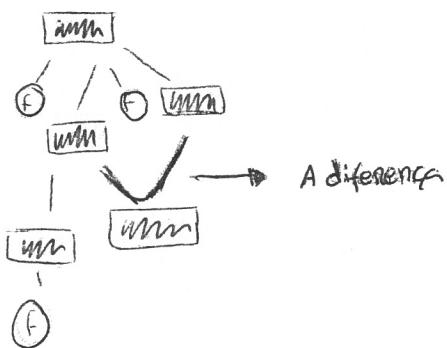


$m \rightarrow$ diretórias
 $O \rightarrow$ ficheiros

versões mais simples

Se eu tiver uma diretoria (um diretório) que seja dada através de 2 símbolos, então nesse caso é uma árvore mais complexa e que tem o nome de Grafo.

Grafo



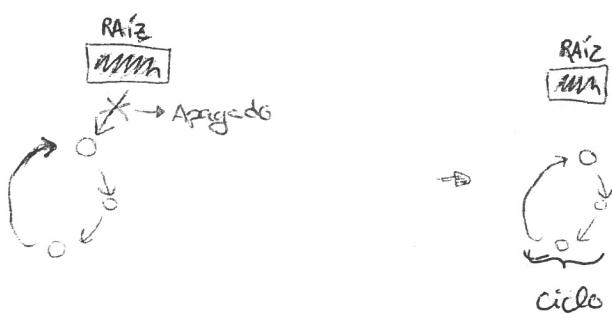
Neste caso o Grafo é acíclico porque não possuem ciclos no Grafo, é um dos Grafos mais Simples.

Mas porque é que não posso voltar até lá para Cima? Será que eu devo dizer permitir ou não?

Caso dê jeito voltar para Cima, posso-o permitir porque isto terá consequências:

Se apagase-mos a raiz da diretoria, mas não podesse apagar a diretoria porque outra diretoria também a estaria a utilizar, então nesse caso ficar com uma parte em que existem diretórias que não estão ligadas à raiz e que estão lá num ciclo.

Neste caso podia utilizar um Algoritmo que caso existam ciclos que não estão ligados à raiz (diretórios), eu poderei apagar esses diretórios



Onde vai Arrumar a memória em Baixo Nível?

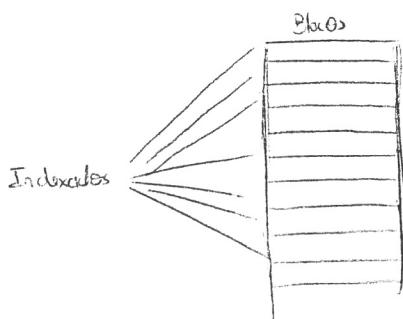
Memória → Disco

Blocos

Arrumar os ficheiros no disco vai ser parecido como se arruma na memória (páginas) mas neste caso chamam-se ~~Blocos~~

que em termos de hardware vão ser semelhante às Páginas.

Os Blocos Como vão ser arrumados?



TARE → Algo muito lento e que isso tenta que aceder a algo que está no fim da TARE é um processo muito lento.

Acesso sequencial → Quando o acesso é feito pela sua ordem, Princípio → fim

Acesso sequencial → Quando o acesso é feito pela sua ordem, Princípio → fim

Acesso Aleatório → Quando vou buscar a informação sem ter que a procurar desde o princípio.

Mecanismos Indexados são muito usuais Hoje em dia (Acesso Aleatório)

Mecanismos Associados ao Acesso

Mecanismos Indexados. ÁRVORES

Listas Ligadas

Como Guardo fisicamente num dispositivo, Como vou CARRAPAGAR?

Pode ser algo que estiver associado a uma árvore, em que cada corresponde a 1 ficheiro, sem caractér

sequencial



, ou listas ligadas

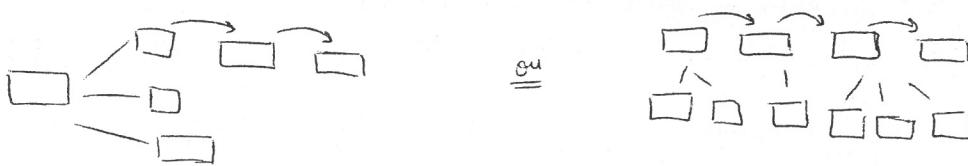
Listas Ligadas

S.O (39)

Tem um endereço sequencial



os 2 tipos de estruturas mais comuns para armazenar o sistema de ficheiros são os Arvores e as Listas ligadas,
porém os 2 tipos também podem existir em conjunt



O que se utiliza no Unix

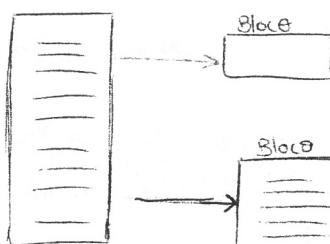
O Unix usa sistema Indexado

Existem Arvores

é um sistema que seja eficiente para todos os tipos de ficheiros

Como Arranjo uma estrutura de dados que tenha suporte para todos os ficheiros?

Vou ter que arranjar apontadores que não apontam para um bloco que não possui informação sobre os ficheiros mas sim apontadores, e esses apontadores é que ainda apontam para o ficheiro.



Apontador direto

Apontador Indireto

cada um

tem 1K então $\frac{1}{4} = 256$ apontadores, neste caso cada bloco pode ter 256 apontadores

4 Bytes

Apontador indireto duplo

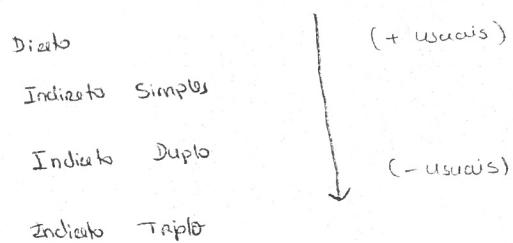
65. 536 apontadores no total dos 2



quando utilizo um apontador Indireto duplo apenas é eficiente quando a quantidade que quero guardar é grande.

Quando utilizo um apontador Indireto duplo é para ter a garantia que consigo ter ficheiros muito grandes, se apenas possuir ficheiros pequenos ^{use} um apontador indireto simples e se forem mesmo muito pequenos então só um apontador directo.

Tem-se apontadores diretos, mas o número é dependente do sistema, nestes sistemas existe uma Eficiência muito grande.
Para os ficheiros Pequenos, existe uma grande desvantagem no tipo de apontadores, existem mais apontadores diretos.



Unix

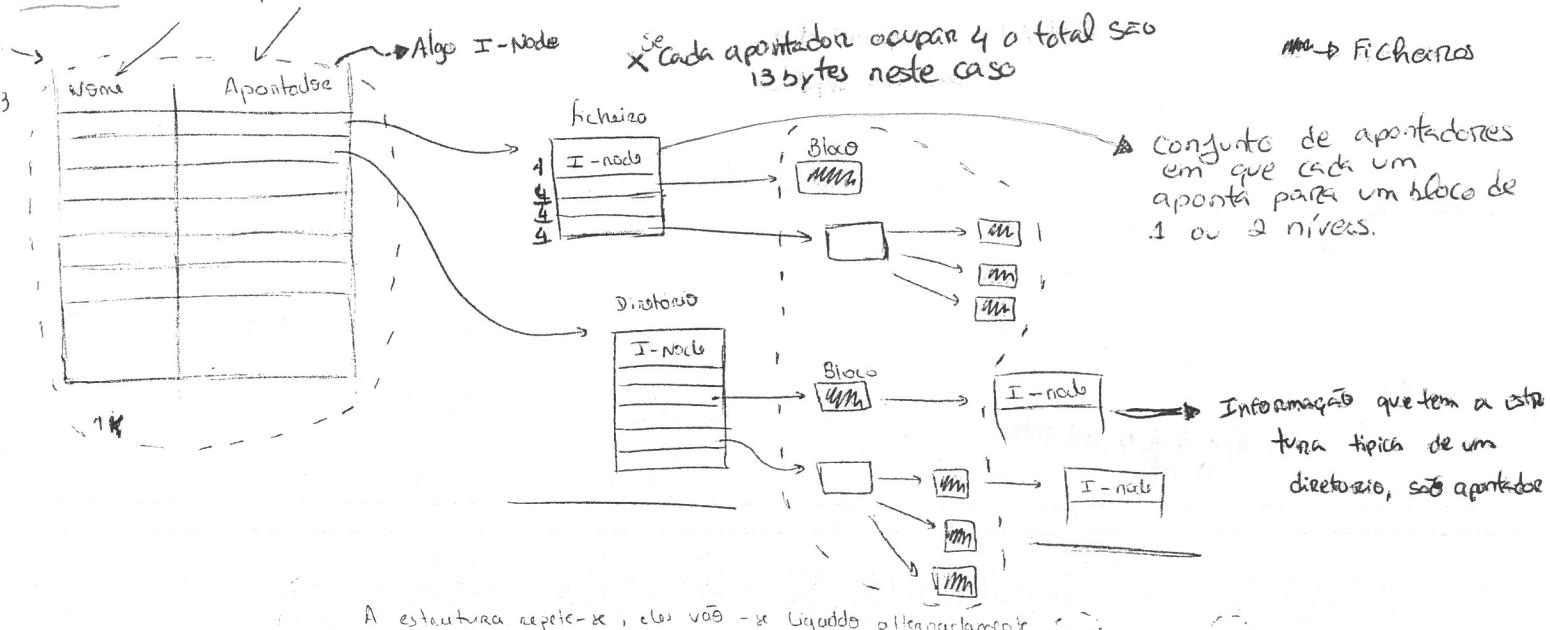
Vai definir o I-nóculo, a unica Roça que possui sus Indices (Díctio, Indíctio Simples, Indíctio Duplo, Indíctio Triple).

tipo do fichero, informaçāo sobre o tipo. (se é um fichero ou um diretorio)).

Estrutura mínima de um Dicionário

ficheiros equivalentes a diretórios \Rightarrow quase uma base de dados.

Dicionário (minimo que se tem de TER)



I-nodes → onde está guardada a informação que guarda os ficheiros

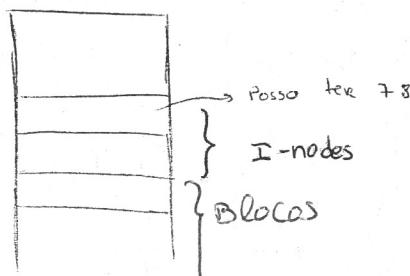
S.O. (40)

os Bloco possuem todos o mesmo tamanho

Têm-se Bloco que estão reservados para serem usados para i-node

Se só tiver 13 bytes reservados para a informação que tenho e um bloco for de 1024, então $\frac{1024}{13} = 78,7$, assim cada bloco

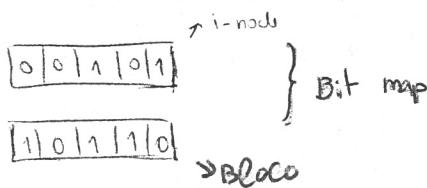
consigo guardar 78 i-nodes.



Preciso de subir o tempo esperado para guardar os Bloco
em os I-nodes

existem várias formas de gerir espaço livre

Pode-se pensar na estrutura como tendo 2 vetores em que tenho uma notação para limpar e 1 para o ocupado



Apagar a informação é deixar de apontar a informação, não preciso de fazer nada, não preciso de apagar basta apontar.

Se eu perder as listas (os vetores)?

O que nos ~~sabemos~~ é o espaço livre mas sabemos que o espaço ocupado porque se conhecemos o diretorio da raiz, saberíamos o ocupado, iria marcar o que sabia que estava ocupado como ocupado e resto como livre

Exemplo:

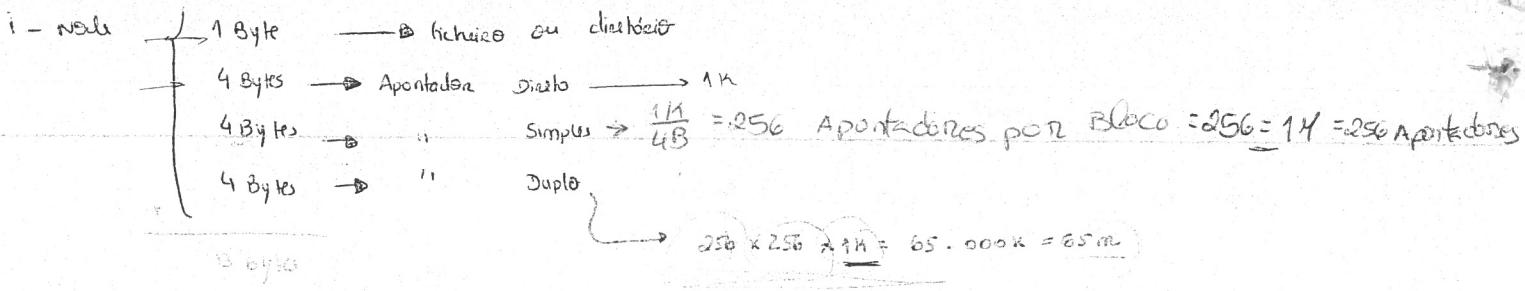
Sinto ter um bloco → 1K

Tamanho I-node = Bloco → 4B

Diretório:

Nóis → 12B) 16 Bytes
A ponteiras → 4B

In**for**mação que é
necessária
(mínima)



O maior arquivo que eu consigo guardar é de 65 megas.

Porque:

$$\begin{array}{r}
 65\ 536 \\
 + 256 \\
 \hline
 65\ 787
 \end{array}$$

Com esta estrutura quantos diretórios e sub estruturas do diretório consigo?

65m → dimensão máxima dos arquivos

$$\frac{65m}{16} = 4m \rightarrow \text{arquivos e sub diretórios}$$

1 diretório tem o nome e o apontador base logo utiliza 16 Bytes

Dentro de 1 bloco se o diretório tiver 1K quantos inteiros podem estar lá dentro $\frac{1024}{16} = 64$ inteiros por blocos.