

Du é um mecanismo que controla a execução de programas e atua como uma interface entre esses programas e o hardware do computador.

3 objetivos:

1. conveniência: o SO faz com que o PC programe convenientemente na utilização;

2. eficiência: o SO permite a utilização dos recursos do sistema do computador

3. Abilidade em suportar: SO deve ser estruturado de modo a permitir a eficiência no desenvolvimento, testar e implementação de um novo sistema de funções, nem intervir em o serviço inicial.

servicos do SO:

1. desenvolvimento de programas;

2. execução de programas;

3. acesso a dispositivos I/O;

4. controle de acesso aos ficheiros e processos:

5. acesso ao sistema:

6. detecção de erros e resposta;

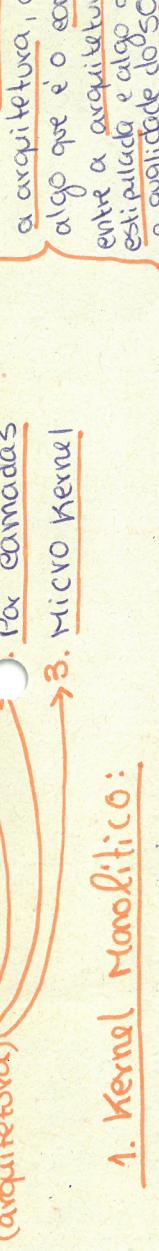
7. contabilidade / estatística de sistema;

8. arquitetura de conjunto de instruções;

9. aplicações binária interface (ABI);

10. interface da aplicação do programa; (API)

Fucionamento:



1. Kernel Monolítico:

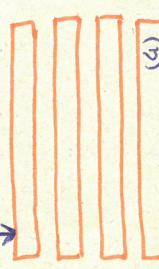
→ implementado como um único processo, em todos os elementos do sistema a partir de o mesmo espaço de endereçamento;

→ só que funcionam pelo kernel monolítico tem no seu interior ficheiros (ficheiros), etc., memória, etc.

→ inicialmente pensado para 1 utilizadora e 1 processo;

→ havia alguns sistemas que inicialmente tinham um de senho bastante estruturado, mas que já não era um sistema operativo monolítico tradicional, apenas partes eram, mas no final acabou por se ter que recorrer ao tradicional mas com uma nova estrutura de sistema;

→ de forma a servir o serviço inicial.



→ problema é quando é necessário ir para q tem de se passar por outras emadas nem quer a que se quer, o que não é eficiente / a melhor solução: no final acaba-se por parar o SO através de um sistema monolítico

2. Par emadas

→ é necessário fazer a gestão dos recursos de forma eficiente (através de regras e controlo da autorização ou não autorização dos recursos.)

→ quando se tem algo funcional em alto nível, a sua execução é feita de forma lenta no nível abaixo; • A emada abaixo tem de aceder ao pedido de serviço da emada acima;



→ melhor para um software complexo

→ é um sistema tem emadas, tutto o que está em SO tem de estar em emadas;

→ tudo o que está em SO tem de estar em emadas;

→ é uma técnica em que um processo, executando um programa ou tarefa, é dividido em threads que correm de forma concorrente / paralela;

→ útil para tarefas que requerem a forma concorrente / paralela;

→ util para tarefas que não necessitam de ser simultâneas;

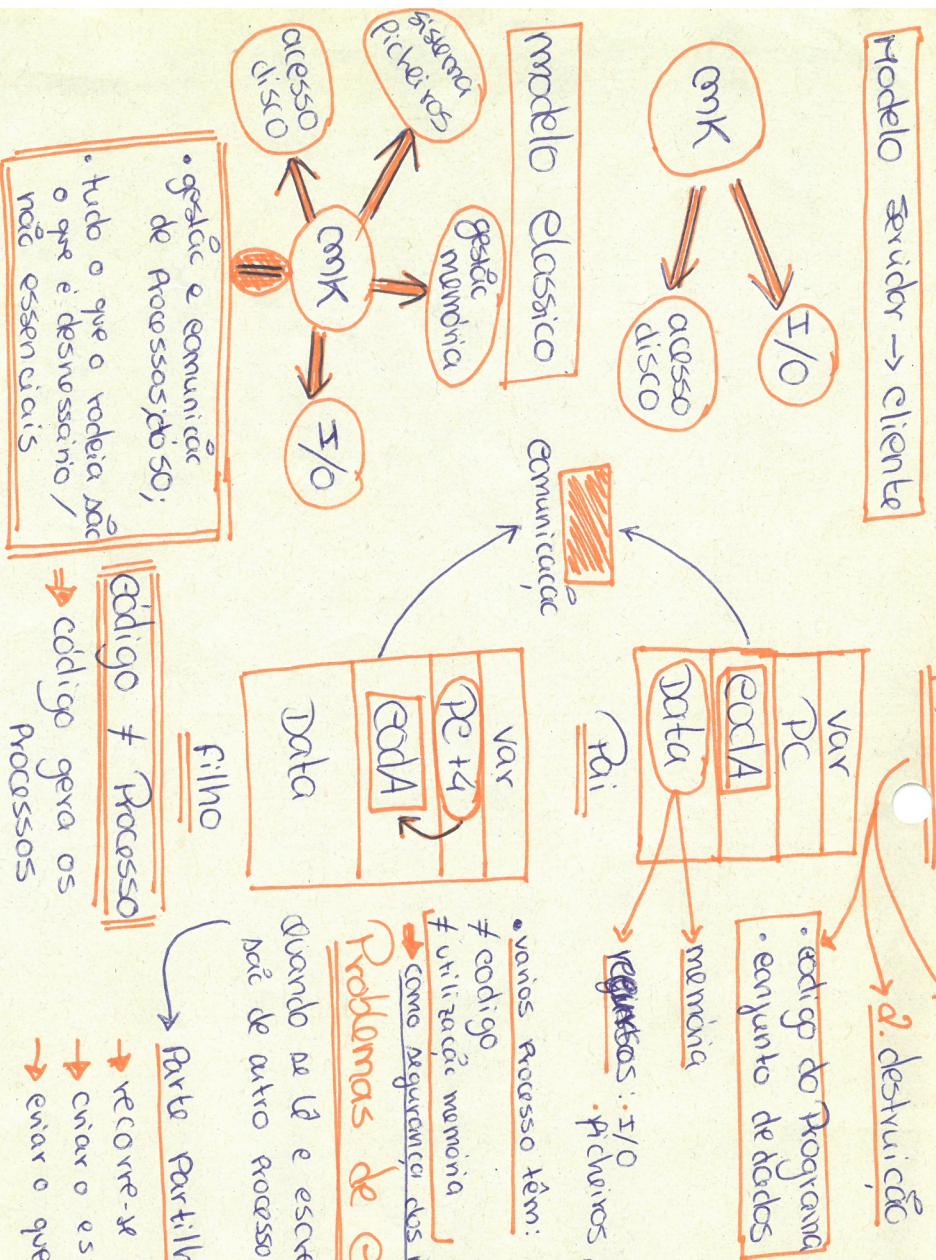
→ independentes que não necessitam de ser simultâneas;

→ conjunto de 1 ou mais threads e sistema de recursos associado;

Processo:

3. Micro kernel

Modelo servidor → cliente

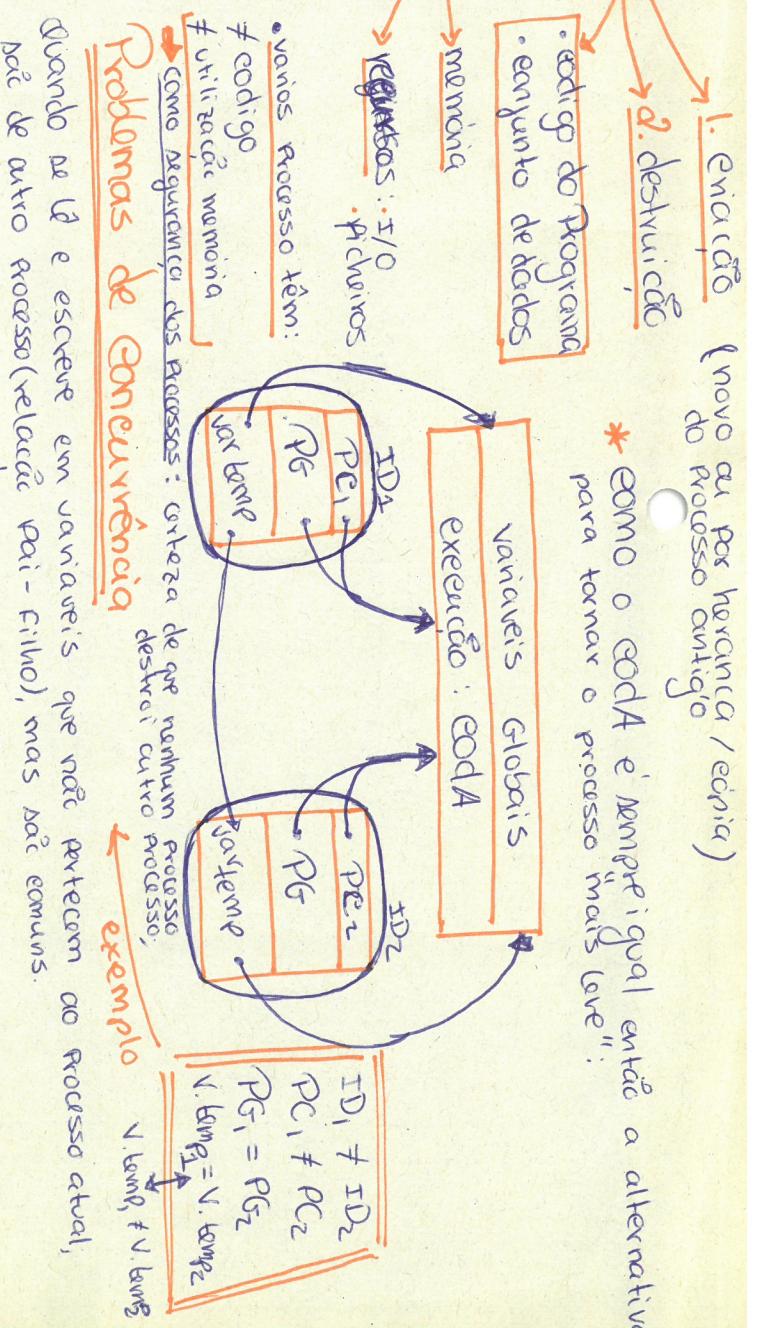


1. enviação

(novo ou por herança / herança)

* como o CODA é sempre igual então a alternativa para tornar o processo "mais leve":

para tornar o processo "mais leve":



2. destruição

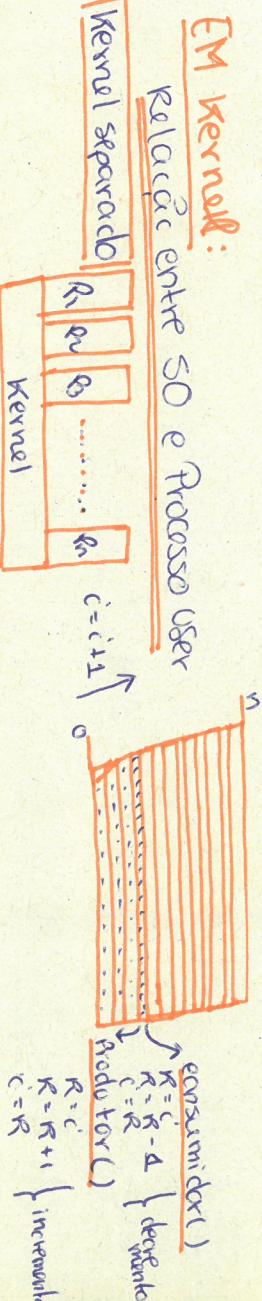
(novo ou por herança / herança)

EM Kernel:

Relação entre SO e Processo User

- recorre-se às threads;
- criar o essencial para cada programa em separado;
- enviar o que é comum em bloco;

varias threads = c'



SO e fúnções
a serem executadas
no user process

P ₁	P ₂	P ₃	...	P _n
S ₀₁	S ₀₂	S ₀₃	...	S _{0n}

troca de processos (função)

* o nº de SO's que se vai
obter depende da capacidade
do hardware

SO e fúnções
a serem executadas
em diferentes
processos

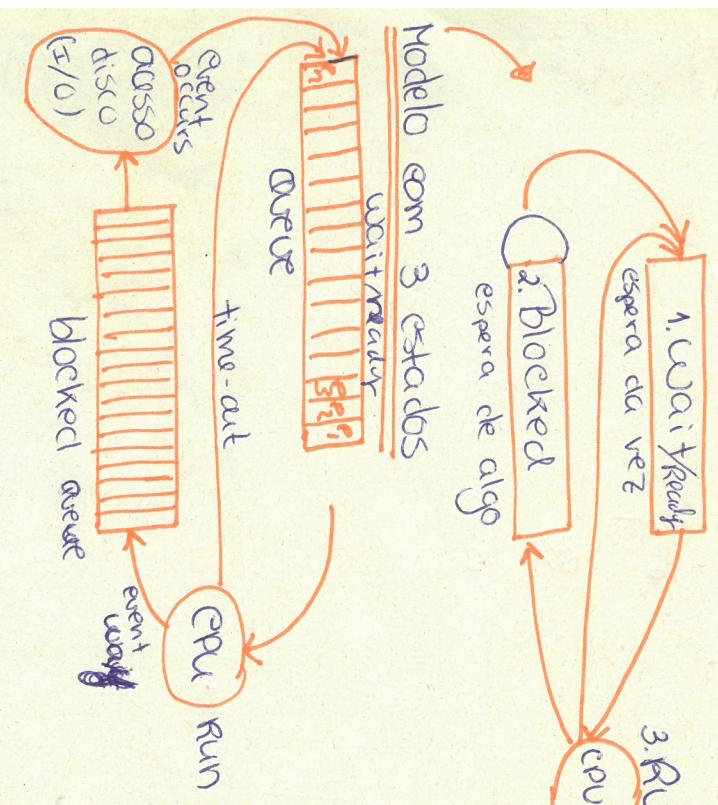
P ₁	P ₂	...	P _n	S ₁	S ₂	...	S _k
R ₁	R ₂	...	R _n	R ₁	R ₂	...	R _k

troca de processos

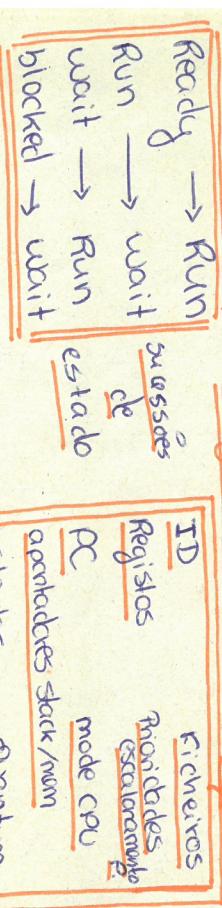
Rúbrica troca de processos

Modelos de estados:

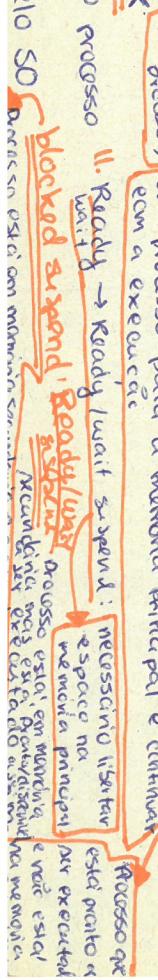
Modelo con 7 estados



nesse existem mais processos que espaço no cau para elesarem executados, os processos a mais não vai poder conter, como os processos não podem equilibrar ao mesmo tempo, estes ou estarão a conter ou estão à espera.



o código e os dados podem ser alterados, até por eles mesmos, dependendo das processos de memória poderem exercitá-los dinamicamente.



- Diagrama de transição entre os estados de um processo:

```

    graph TD
        null((null)) --> new((new))
        new --> ready[Wait/Ready]
        new -- "active, suspend" --> suspended((suspended))
        suspended -- "ready, wait" --> ready
        suspended -- "occurs, blocked" --> blocked((blocked))
        blocked -- "active, suspend" --> suspended
        blocked -- "event, wait" --> ready
        blocked -- "time-out, interrupt" --> run((Run))
        run -- "release" --> ready
        run -- "pedido I/O" --> exit((Exit))
        exit -- "resposta I/O" --> ready
    
```

O diagrama ilustra os estados de um processo e as transições entre eles:

 - Estado null:** O ponto de partida.
 - Estado new:** Criado para ir logo para o estado wait.
 - Estado ready:** Ativo e pronto para executar.
 - Estado suspended:** Ativo mas suspenso.
 - Estado blocked:** Ativo mas bloqueado.
 - Estado Run:** Executando.
 - Estado Exit:** Encerrado.

As transições são controladas por eventos:

 - Entre null e new: null → new
 - Entre new e ready: new → ready
 - Entre new e suspended: new → suspended (ativo, suspenso)
 - Entre suspended e ready: suspended → ready (pronto, suspenso)
 - Entre suspended e blocked: suspended → blocked (ocorre, bloqueado)
 - Entre blocked e suspended: blocked → suspended (ativo, suspenso)
 - Entre blocked e ready: blocked → ready (pronto, suspenso)
 - Entre blocked e Run: blocked → Run (tempo-out, interrupt)
 - Entre Run e ready: Run → ready (liberação)
 - Entre Run e Exit: Run → Exit (pedido I/O)
 - Entre Exit e ready: Exit → ready (resposta I/O)

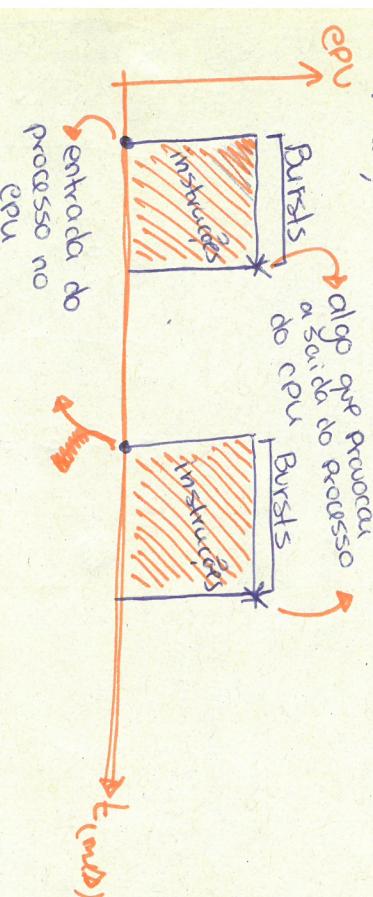
Escalonamento (cont.)

→ faz a transição do wait para o run;

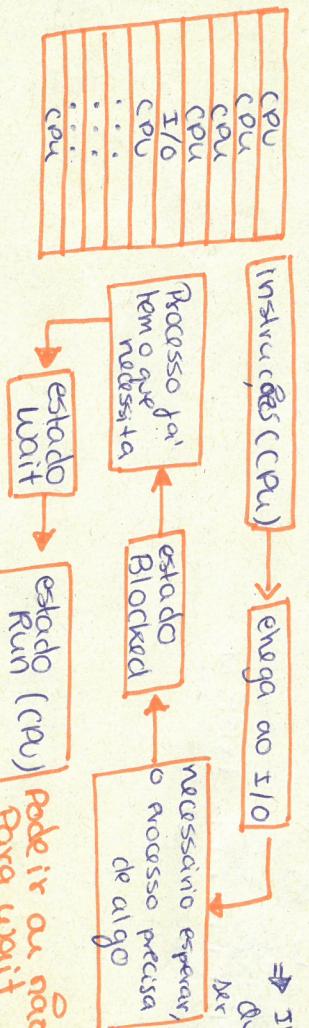
→ seleciona qual dos processos que estão em ready, no wait, vão para o CPU, estado Run.

→ Responsável pela escolha de um processo para sair do CPU para o estado wait ou não, blocked.

A escolha é, normalmente, sempre feita com base num timer.



Bursts → períodos de tempo que o processo se encontra no CPU



? Qual dos processos deve ir ir

1º para o Estado Run (CPU)

R: O processo que devia ir primeiro é aquele que ocupa mais tempo no estado Blocked, deixando assim o CPU disponível.

dois serão o Processo P2.

P1:	90% CPU
P2:	10% CPU 90% I/O

R	Estado Wait
P1:	90% CPU
P2:	10% CPU 90% I/O

cpu Bound	I/O Bound
90% cpu	10% I/O

Algoritmos dos processos

1. First In First Out (FIFO) | First Come First Served (FCFS)

→ Quando o processo que está a executado é terminado, o processo que estiver no estado Ready, i.e na ready queue, à mais tempo é selecionado para ser proximo a ser executado.

FCFS, é abreviado para FST.R.

Processos	Tempo (ms) serviço(s)	Periodicidade (ms)
P1	10	10
P2	50	70
P3	20	10
P4	20	10



$$\text{B} \quad \bar{x} = \frac{10 + 10 + 20 + 30}{4} = \frac{160}{4}$$

$$\text{C} \quad \bar{x} = \frac{70 + 80 + 90 + 100}{4} = \frac{340}{4}$$

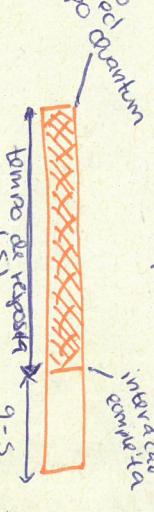
Depois os SIs de tempo real trabalham com o tempo de periodicidade. Logo, neste caso, tperiódica é tratado como o tempo de chegada.

2. ROUND Robin (RR)

→ Interrupções ao nível do clock vão sendo geradas em intervalos periódicos. Quando esta interrupção acontece, o processo que está a ser executado vai ser transferido para a Wait/Ready queue e o processo seguinte da Ready queue é escalonado (gerado FCFSS).

"Cortar o tempo em partes", preemptiva.

processo total disponível x tempo quantum



Quantum (q)

Tarantum = 30 ms

20

[10
P1 P2 P3 P4]

30 [10
P2 P3 P4]

40 [10
P3 P4]

50 [10
P4]

60 [10
P1 P2 P3 P4]

70 [10
P1 P2 P3 P4]

80 [10
P2 P3 P4]

90 [10
P3 P4]

100 [10
P4]

110 [10
P1 P2 P3 P4]

120 [10
P1 P2 P3 P4]

130 [10
P1 P2 P3 P4]

140 [10
P1 P2 P3 P4]

150 [10
P1 P2 P3 P4]

160 [10
P1 P2 P3 P4]

170 [10
P1 P2 P3 P4]

180 [10
P1 P2 P3 P4]

190 [10
P1 P2 P3 P4]

200 [10
P1 P2 P3 P4]

210 [10
P1 P2 P3 P4]

220 [10
P1 P2 P3 P4]

230 [10
P1 P2 P3 P4]

240 [10
P1 P2 P3 P4]

250 [10
P1 P2 P3 P4]

260 [10
P1 P2 P3 P4]

270 [10
P1 P2 P3 P4]

280 [10
P1 P2 P3 P4]

290 [10
P1 P2 P3 P4]

300 [10
P1 P2 P3 P4]

310 [10
P1 P2 P3 P4]

320 [10
P1 P2 P3 P4]

330 [10
P1 P2 P3 P4]

340 [10
P1 P2 P3 P4]

350 [10
P1 P2 P3 P4]

360 [10
P1 P2 P3 P4]

370 [10
P1 P2 P3 P4]

380 [10
P1 P2 P3 P4]

390 [10
P1 P2 P3 P4]

400 [10
P1 P2 P3 P4]

410 [10
P1 P2 P3 P4]

420 [10
P1 P2 P3 P4]

430 [10
P1 P2 P3 P4]

440 [10
P1 P2 P3 P4]

450 [10
P1 P2 P3 P4]

460 [10
P1 P2 P3 P4]

470 [10
P1 P2 P3 P4]

480 [10
P1 P2 P3 P4]

490 [10
P1 P2 P3 P4]

500 [10
P1 P2 P3 P4]

510 [10
P1 P2 P3 P4]

520 [10
P1 P2 P3 P4]

530 [10
P1 P2 P3 P4]

540 [10
P1 P2 P3 P4]

550 [10
P1 P2 P3 P4]

560 [10
P1 P2 P3 P4]

570 [10
P1 P2 P3 P4]

580 [10
P1 P2 P3 P4]

590 [10
P1 P2 P3 P4]

600 [10
P1 P2 P3 P4]

610 [10
P1 P2 P3 P4]

620 [10
P1 P2 P3 P4]

630 [10
P1 P2 P3 P4]

640 [10
P1 P2 P3 P4]

650 [10
P1 P2 P3 P4]

660 [10
P1 P2 P3 P4]

670 [10
P1 P2 P3 P4]

680 [10
P1 P2 P3 P4]

690 [10
P1 P2 P3 P4]

700 [10
P1 P2 P3 P4]

710 [10
P1 P2 P3 P4]

720 [10
P1 P2 P3 P4]

730 [10
P1 P2 P3 P4]

740 [10
P1 P2 P3 P4]

750 [10
P1 P2 P3 P4]

760 [10
P1 P2 P3 P4]

770 [10
P1 P2 P3 P4]

780 [10
P1 P2 P3 P4]

790 [10
P1 P2 P3 P4]

800 [10
P1 P2 P3 P4]

810 [10
P1 P2 P3 P4]

820 [10
P1 P2 P3 P4]

830 [10
P1 P2 P3 P4]

840 [10
P1 P2 P3 P4]

850 [10
P1 P2 P3 P4]

860 [10
P1 P2 P3 P4]

870 [10
P1 P2 P3 P4]

880 [10
P1 P2 P3 P4]

890 [10
P1 P2 P3 P4]

900 [10
P1 P2 P3 P4]

910 [10
P1 P2 P3 P4]

920 [10
P1 P2 P3 P4]

930 [10
P1 P2 P3 P4]

940 [10
P1 P2 P3 P4]

950 [10
P1 P2 P3 P4]

960 [10
P1 P2 P3 P4]

970 [10
P1 P2 P3 P4]

980 [10
P1 P2 P3 P4]

990 [10
P1 P2 P3 P4]

1000 [10
P1 P2 P3 P4]

1010 [10
P1 P2 P3 P4]

1020 [10
P1 P2 P3 P4]

1030 [10
P1 P2 P3 P4]

1040 [10
P1 P2 P3 P4]

1050 [10
P1 P2 P3 P4]

1060 [10
P1 P2 P3 P4]

1070 [10
P1 P2 P3 P4]

1080 [10
P1 P2 P3 P4]

1090 [10
P1 P2 P3 P4]

1100 [10
P1 P2 P3 P4]

1110 [10
P1 P2 P3 P4]

1120 [10
P1 P2 P3 P4]

1130 [10
P1 P2 P3 P4]

1140 [10
P1 P2 P3 P4]

1150 [10
P1 P2 P3 P4]

1160 [10
P1 P2 P3 P4]

1170 [10
P1 P2 P3 P4]

1180 [10
P1 P2 P3 P4]

1190 [10
P1 P2 P3 P4]

1200 [10
P1 P2 P3 P4]

1210 [10
P1 P2 P3 P4]

1220 [10
P1 P2 P3 P4]

1230 [10
P1 P2 P3 P4]

1240 [10
P1 P2 P3 P4]

1250 [10
P1 P2 P3 P4]

1260 [10
P1 P2 P3 P4]

1270 [10
P1 P2 P3 P4]

1280 [10
P1 P2 P3 P4]

1290 [10
P1 P2 P3 P4]

1300 [10
P1 P2 P3 P4]

1310 [10
P1 P2 P3 P4]

1320 [10
P1 P2 P3 P4]

1330 [10
P1 P2 P3 P4]

1340 [10
P1 P2 P3 P4]

1350 [10
P1 P2 P3 P4]

1360 [10
P1 P2 P3 P4]

1370 [10
P1 P2 P3 P4]

1380 [10
P1 P2 P3 P4]

1390 [10
P1 P2 P3 P4]

1400 [10
P1 P2 P3 P4]

1410 [10
P1 P2 P3 P4]

1420 [10
P1 P2 P3 P4]

1430 [10
P1 P2 P3 P4]

1440 [10
P1 P2 P3 P4]

1450 [10
P1 P2 P3 P4]

1460 [10
P1 P2 P3 P4]

1470 [10
P1 P2 P3 P4]

1480 [10
P1 P2 P3 P4]

1490 [10
P1 P2 P3 P4]

1500 [10
P1 P2 P3 P4]

1510 [10
P1 P2 P3 P4]

1520 [10
P1 P2 P3 P4]

1530 [10
P1 P2 P3 P4]

1540 [10
P1 P2 P3 P4]

1550 [10
P1 P2 P3 P4]

D. Shortest Process Next (SPN)

→ não preemptivo

→ O processo com o menor tempo de processamento, tempo de serviço, será o próximo a ser executado/selecionado.

→ desvantagem desse algoritmo é que é necessário saber sempre quais os tempos de serviço de cada processo se o tempo de serviço estimado por inferior / superior ao que de fato o processo torma o SO irá decretar um alerta de erro / interrupção, e acaba por interromper por completo o processo.

4. Shortest Remaining Time (SRT)

→ Preemptivo;

→ Seleciona o processo com menor tempo de serviço e vai executá-lo, no entanto, se aparecer um novo processo mais rápido do que os processos que faltam, inclusive o que está a correr, então o SRT vai interromper esse processo e coloca aquele com menor tempo no seu lugar.

5. Highest Response Ratio Next (HRRN)

$$R = \frac{W + S}{S} \Rightarrow \frac{\text{Rádio de resposta} + \text{tempo serviço}}{\text{tempo serviço}} \geq 1,0$$

→ não preemptivo

→ R = 1,0 : Quando um processo entra no sistema

→ Quando o processo atual termina ou vai transferir para o estado Blocked, reinicia-se um processo no estado Ready / wait com o maior rádio R;

Processos	Tserviço	Tespera
P1	100 ms	+ 20 ms
P2	100 ms	+ 20 ms

→ devia ao aumento da prioridade dos processos,

Algoritmos de tempo real (executamento)

→ Impõe validade do tempo de resposta; se a resposta estiver dentro do tempo certo então é uma resposta válida. caso contrario a resposta é não válida / erro.

→ Resposta \Rightarrow válido \Rightarrow resposta < limite

→ Mesmo que as ações tenham sucesso, negam validades, se o resultado existe um erro.

→ Muitas vezes no linux tem o kernel interrompido por causa que se tem um limite e respostas muito rápidas (kernel em tempo real)

1. Hard real time

→ Tem de cumprir sempre o deadline; cumpre obrigatoriamente o tempo de resposta dentro do limite não existindo um erro.

⇒ Erro fatal no SO;

⇒ sistemas previsíveis;

⇒ sistemas industriais, aquisição de dados, carros, funções periódicas.

2. Soft real time

→ Tem um deadline, no entanto não é obrigatório; flexível.

→ Faz sentido para o escalonador, econome a tarefa independente de ter passado o deadline.

→ não tem quaisquer restrições; tem mais aplicações.

⇒ sistemas mais complexos;

Características de SO Real time (tempo Real)

1. Determinismo (determinista). \Rightarrow os algoritmos mais simples são em deterministas.

2. Capacidade de Resposta. \Rightarrow Prioridades estáticas (+ simples) onde se sabe as prioridades

3. Controlado por parte do User. \Rightarrow Prioridades dinâmicas (SR+T) \Rightarrow podem ser 50 preemptivos ou não preemptivos.

4. Confiabilidade

5. Operações Fail - soft (50 UNIX) \Rightarrow processos periódicos.

O processo P2 é o processo que deverá ser mandado executar

deadlines

Para saber se os processos cumprim o deadline ou não é necessário verificar todos os processos. Se existir um Padrão entre os processos tem-se garantia que nessa parte os processos cumprem o deadline.



L. Nao Periodico

Processos	Serviço	Tela	Garda	Deadline
P ₀	10	0	20	
P ₁	10	10	30	
P ₂	20	20	50	
P ₃	20	30	70	

~~Ro~~ ~~1000~~ ~~1000~~ ~~1000~~ ~~1000~~ ~~1000~~ ~~1000~~

$P_0 > P_1 > P_2$
P₁ e maior
do que

Algoritmos: 1. Readline + Próximo

\Rightarrow Para definir qual é o processo com maior PCTM baseado no deadline dos processos:

⇒ O processo com a deadline mais proxima da ordem de chegada, passa a ser o mais

Service	Responsible	Deadline

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
R1	10	10	30	80	80	80	80	80	80	80
R2	20	20	20	50	50	50	50	50	50	50
R3	20	30	30	70	70	70	70	70	70	70
R4	30	30	30	70	70	70	70	70	70	70
R5	30	30	30	70	70	70	70	70	70	70
R6	30	30	30	70	70	70	70	70	70	70
R7	30	30	30	70	70	70	70	70	70	70
R8	30	30	30	70	70	70	70	70	70	70
R9	30	30	30	70	70	70	70	70	70	70
R10	30	30	30	70	70	70	70	70	70	70

Algoritmo: Deadline + Proximo ej tempo mort

⇒ é dado prioridade ao processo que tem um deadline + próximo
independente de o processo ter chegado aí nac. Utiliza-se quando o padrão e
repetitivo.

line + ~~area~~

Resposta: Os processos têm de ser radicalmente alterados.

processos periódicos em

ocesso a engrenagem é o primeiro a engrenar

Algoritmo: Rate Monotonic scheduling (RMS)

	Tempo	Período	Média dos Processos
P1	10	30	$T_{Período} = \frac{10}{30} = \frac{1}{3}$
P2	10	60	$\frac{T_{Período}}{Período} = \frac{10}{60} = \frac{1}{6}$
P3	20	60	$T_{Período} = \frac{20}{60} = \frac{1}{3}$
P4	20	120	$T_{Período} = \frac{20}{120} = \frac{1}{6}$

$\cdot \bar{X}_{\text{P}} \times 100\% \Rightarrow$ percentagem CRU que é

1/2/34
⇒ 0 Backscattered

021 01 01

$$P_1 < P_2 < P_3 < P_4$$

nos de 6 econsante o numero de processos que de CPU utilizado por todos os processos x RMs remane sempre o mesmo valor. Pode-se ser

⇒ A tarefa / processo com maior prioridade é aquela com menor periodo, maior frequência ou seja, com um maior n.º de repetições. Quando mais do que 1 tarefa está disponível para execução, seleciona-se aquela com período menor.

Ribbons com Threads

- vários recursos partilhados entre vários threads
- espaço de memória que os threads podem mexer

- se as threads podem mexer em algumas zonas da memória, geram alguns problemas de sincronismo e de concorrência
- quando se recorre ao uso de threads, ao mesmo tempo, elas podem aceder / mexer no mesmo sitio mas não. Assim, irão desencadear estes erros.

- geram erros quando existem instruções atómicas
- threads irão ao i para ver qual a última posição ocupada. Se o espaço de memória que os threads podem mexer é grande quando se recorre ao uso de threads, ao mesmo tempo,

Producer - Consumidor

Producer (processo)

```

temp = i;
temp = temp + 1;
i = temp;
  
```

• variável temporária;

• 2 processos que podem partilhar o mesmo espaço da memória

e variáveis temporárias (a variável temp é més i é acedida pelos 2 processos)

Consumidor (processo)

```

temp c = i;
temp c = temp c - 1;
t = temp c;
  
```

• tem com variável temporária,

Producer

```

i → R1
R1 → R1 = Rn + 1
R1 → c
  
```

Consumidor

```

i → R2
R2 → R2 = R2 - 1
R2 → c
  
```

RECORRER A UMA PILHA COM VALOR MÁXIMO

encerramento

retira da pilha

i = max

exemplos

```

i → R1
R1 : 5
R1 → R1 + 1
R1 : 6
R1 → c
  
```

```

i → R2
R2 : 5
R2 → R2 - 1
R2 : 4
  
```

```

i → R2
R2 : 5
R2 → R2 - 1
R2 : 4
  
```

```

c = 6
  
```

```

c = 4
  
```

```

c = 2
  
```

```

c = 0
  
```

```

c = -1
  
```

```

c = -2
  
```

```

c = -3
  
```

```

c = -4
  
```

```

c = -5
  
```

```

c = -6
  
```

```

c = -7
  
```

```

c = -8
  
```

```

c = -9
  
```

```

c = -10
  
```

```

c = -11
  
```

```

c = -12
  
```

```

c = -13
  
```

```

c = -14
  
```

```

c = -15
  
```

```

c = -16
  
```

```

c = -17
  
```

```

c = -18
  
```

```

c = -19
  
```

```

c = -20
  
```

```

c = -21
  
```

```

c = -22
  
```

```

c = -23
  
```

```

c = -24
  
```

```

c = -25
  
```

```

c = -26
  
```

```

c = -27
  
```

```

c = -28
  
```

```

c = -29
  
```

```

c = -30
  
```

```

c = -31
  
```

```

c = -32
  
```

```

c = -33
  
```

```

c = -34
  
```

```

c = -35
  
```

```

c = -36
  
```

```

c = -37
  
```

```

c = -38
  
```

```

c = -39
  
```

```

c = -40
  
```

```

c = -41
  
```

```

c = -42
  
```

```

c = -43
  
```

```

c = -44
  
```

```

c = -45
  
```

```

c = -46
  
```

```

c = -47
  
```

```

c = -48
  
```

```

c = -49
  
```

```

c = -50
  
```

```

c = -51
  
```

```

c = -52
  
```

```

c = -53
  
```

```

c = -54
  
```

```

c = -55
  
```

```

c = -56
  
```

```

c = -57
  
```

```

c = -58
  
```

```

c = -59
  
```

```

c = -60
  
```

```

c = -61
  
```

```

c = -62
  
```

```

c = -63
  
```

```

c = -64
  
```

```

c = -65
  
```

```

c = -66
  
```

```

c = -67
  
```

```

c = -68
  
```

```

c = -69
  
```

```

c = -70
  
```

```

c = -71
  
```

```

c = -72
  
```

```

c = -73
  
```

```

c = -74
  
```

```

c = -75
  
```

```

c = -76
  
```

```

c = -77
  
```

```

c = -78
  
```

```

c = -79
  
```

```

c = -80
  
```

```

c = -81
  
```

```

c = -82
  
```

```

c = -83
  
```

```

c = -84
  
```

```

c = -85
  
```

```

c = -86
  
```

```

c = -87
  
```

```

c = -88
  
```

```

c = -89
  
```

```

c = -90
  
```

```

c = -91
  
```

```

c = -92
  
```

```

c = -93
  
```

```

c = -94
  
```

```

c = -95
  
```

```

c = -96
  
```

```

c = -97
  
```

```

c = -98
  
```

```

c = -99
  
```

```

c = -100
  
```

EM SUMA

- o produtor irá dizer qual última ocupada. Fazendo a incrementação de i, depois i vai ser implementada; no entanto, a última posição ocupada é dita por $\underline{i+1}$;

- consumidor irá dizer i para ver qual a última posição ocupada. Ele o consumidor retira algo, o i irá ficar $i - 1$, e irá voltar ao sitio inicial da pilha;

- problemas surgem quando existem instruções atómicas para garantir que i é incrementado i e simultaneamente, o produtor copia o i, quando isso acontece ocorre problema.

- atualizar a informação no registo temporário, quando tiver o resultado final no registo temporário ele irá adicionar o valor actualizado na i.

- consumidor exerce a dependência da mesma forma mas para a decrementação;

- o consumidor irá dizer qual é quem inicia o processo R0 é o produtor, R0 alternando com o consumidor. Se fosse o contrário, e à mesma alternado, o resultado do produtor daria $i = S$;

- necessario ver variável i que dirá qual o resultado desse problema, diz-se que o bloco das 3 instruções do consumidor é um bloco único, tal como as 3 instruções do produtor.

- quando i instrução tiver a correr quer do consumidor quer do produtor n pode existir interrupção até as instruções terem sido executadas;

- mecanismo de exclusão mutua (mutex) (mais simples)

- lock $\xrightarrow{a \& b}$ semáforos

- se mutaros a monitor (mais complexos)

- 2 processos

- de KF e DIFERENCIAS INIMUTABILIDADES

- se i = 0 não pode retirar (não irá para a retirar)

- 2 processos a correr ao mesmo tempo

- se i = 0 não pode retirar (não irá para a retirar)

Gestão de memória

equações

1. Relocação

2. Proteção / Segurança

3. Partilha

4. Organização

A memória é subdividida de modo a accomodar diferentes processos. A tabela da subdivisão da memória é feita pelo SO, gestor de memória.

- ter múltiplos a correr e garantir que nenhum processo danifica outro processo.



Terminos importantes:

Frame: bloco de dados com tamanho fixo da memória principal.

Page: bloco de dados com tamanho fixo que reside em memória secundária (ex: disco); uma página de dados pode ser temporariamente copiada para uma frame da memória principal.

Segment: bloco de dados com tamanho variável que reside em memória secundária.

segmentação:

Quando um segmento intuito pode ser temporariamente copiado para uma região disponível e acessível da memória principal.

segmentação + Paginação:

Quando o segmento pode ser dividido em páginas que dão para diferentes individualmente para a memória principal.

1. Relocação

Situação ideal: nem poder permutar processos ativos para dentro e para fora da memória principal de forma a maximizar a utilização do processador, tendo uma enorme disponibilidade de processos prontos a ser executados.

Quando um programa é reexecutado para fora do disco, será um pouco limitado e pode ficar quando terá uma nova versão, para dentro do disco e para a mesma região de memória que ocupava.

- não se pode antecipadamente onde o programa será recolocado, mas sabe-se que pode ser permitido na memória principal;
- só necessita de sair a localização da informação do processo de encontro, bem como onde um processo começa e acaba;

2. Proteção / Segurança

- cada processo deve ter proteção contra interferências indesejadas provenientes de outros processos, seja acidental ou intencional;

- programas em outros processos
não devem ter a capacidade de referenciar documentações na memória de processos para a escrita e para a leitura nem permissão

- A satisfação da velocidade requisito aumenta a dificuldade de satisfação do requisito de proteção
 - A localização de um programa na memória principal é imprevisível, logo é impossível identificar o endereço absoluto no tempo de compilar. Desse modo, a segurança protege a satisfação pelo SO.
 - mecanismos que suportam a relocalização também suportam os requisitos de proteção.
- um programa num processo não pode fazer branch para uma instrução dentro de outro processo nem acceder à área reservada aos dados desse processo;

3. Partilha

4. Organização

fragmentação externa: a memória externa a todas as partículas fica incrementalmente fragmentada.

3. Next Fit

A memória vai ficando mais fragmentada e a utilização da memória entra em declínio.

Parceiro ao first fit no entanto em vez de comutar a procura o espaço livre pelo bloco, existe um apelido que indica qual o último espaço utilizado e a procura de espaço livre vai ser desde esse apelido para cima.

compactação:

de X em X sempre o SO procura os processos de modo a serem contínuos e a memoria de espaço livre fica compactada num único bloco.

Algoritmos de alocação:

compactação da memória recorre um grande consumo de tempo necessário alocarmos para atribuir processos à memória, de modo a teremos blocos.

4. Worst Fit

desvantagem: não garante que temos zonas livres de um processo muito grande enegar, porque a memória encontra mais partida existe menos possibilidade de existir uma grande zona livre, o que não é um problema no First Fit.

1. Best - Fit

2. First - Fit

3. Next - Fit

4. Worst - Fit

Algoritmo de substituição:

se se recusar um processo num espaço de memória, então o espaço de sobra na memória tem de ser um grande espaço. A diferença entre o best fit deixe o estado Blocked, o SO vai permutar um dos processos para fora da memória principal para passar a haver espaço para um processo novo da memória do Estado Ready - Se quando um processo novo é o SO que escolhe esse processo que deve ser substituído.

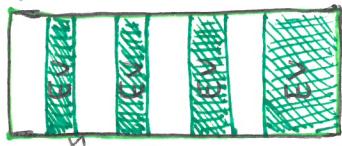
Buddy System

• blocos de memória tem tamanho de 2 palavras
• blocos de memória

• memória possui uma dimensão cacheada e dividida em bocados (árvores) por metade e sub-divisões (árvores)
• sistema da memória continua porque cacheada

• memória é continua porque conseguiu sair

a posição de cada bloco e qual capacidade de cada um.



0,1

0,2

0,3

0,4

0,5



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

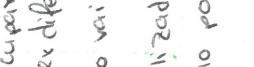
1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

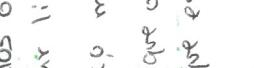
1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4



0,1

0,2

0,3

0,4

0,5

0,6

0,7

0,8

0,9

1,0

1,1

1,2

1,3

1,4

6. Segmentação

Cada processo é dividido num certo número de segmentos. Um processo é carregado, executando todos os segmentos a que foi dividido, para uma partição dinâmica que necessita ser continua.

vantagens: nenhuma fragmentação interna melhoria da memória utilizada e redução do overhead comparado com o particionamento dinâmico;

5. Paginação da Memória Virtual

Paginação normal, apenas não é necessário carregar todas as páginas de um processo. Páginas não residentes que não necessárias não trazidas mais tarde automaticamente.

vantagens: nenhuma fragmentação externa; espaço de endereçamento virtual grande;

desvantagens: overhead de gestão de memória complexa

6. Segmentação da Memória Virtual

segmentação normal, excepto que não necessário carregar todos os segmentos de um processo. Segmentos não residentes não trazidos quando necessários mais tarde de forma automática.

vantagens: nenhuma fragmentação interna; espaço de endereçamento virtual grande;

proteção e partilha de parte (apenas)

desvantagens: overhead de gestão de memória complexa.

Particionamento de partitores com dimensão variada

Single queue

Técnica apenas optimizada em termos da partição, incluindo sempre a vizinhança de forma a minimizar a fragmentação interna.

- Quando não necessário carregar um processo para a memória, a partição mais pequena é acessível

- De todas as partitões existentes que vai ficar com o processo selecionado entao tem de haver permutações.

- Permite para para a memória mais pequena que assegurará o processo.

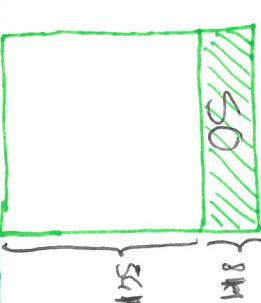
Atribuição de memória por particionamento fixo

Particionamento Dinâmico

- Partições não de comprimento variável e múltiplos;
- Quando um processo entra na memória principal, é aloçado exactamente na quantidade de memória que necessita e não mais;

Situação exemplo:

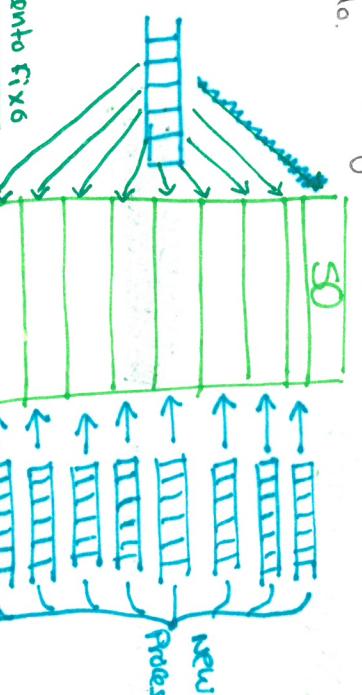
$$MEM = 64 MB$$



1. Inicialmente a memória principal encontra-se vazia excepto o SO.

2. Os 3 primeiros processos são surgidos na RDP, ocupando quando o SO ocupa e ocupando o espaço necessário para cada processo.

Single Queue One Process queue per partition



Vantagem é que os processos estão para para alocados de forma a minimizar a fragmentação interna.

scheduling que

Necessária para cada partição para alocar os processos permitidos para para destinadas a essa partição.

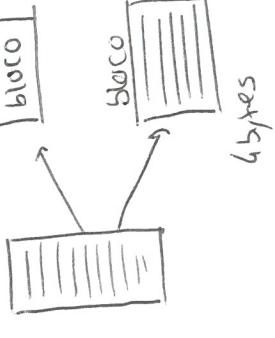
Internal fragmentation: Fenômeno em que existe espaço dentro da partição de dados que é desperdiçado interno de uma partição devido ao bloco de dados que é demasiado pequeno para um processo. A execução muda de não necessitar de todo o espaço de memória. O SO permite processos para para o processo, dando o processo espaço para para o processo.

External fragmentation: Fenômeno em que existe espaço entre partitões que é desperdiçado interno de uma partição devido ao bloco de dados que é demasiado pequeno para um processo. A execução muda de não necessitar de todo o espaço de memória. O SO permite processos para para o processo, dando o processo espaço para para o processo.

Desempenho: nº de vezes per m s em que
buscar informações da RAM

Apartadores

Espaco para 3 paginas = 3 frames



Page Fault (PF): paginais que faltam e tem de
ser buscadas no disco

Hit (H): paginais que já se encontravam na RAM

	1	2	3	4	1	3	1	2	1
FIFO	F	F	F	F	H	H	H	H	3H
CLOCK	1F	1F	1F	1F	2	2	1	1	6F
LRU	1F	1F	1F	1F	3	3	3	3	2
optima	1F	1F	1F	1F	4	4	4	4	4

	1	2	3	4	1	3	1	2	1
FIFO	F	F	F	F	H	H	H	H	3H
CLOCK	1F	1F	1F	1F	2	2	1	1	6F
LRU	1F	1F	1F	1F	3	3	3	3	2
optima	1F	1F	1F	1F	4	4	4	4	4

cada term 1K = 1024
então $\frac{1}{4} = 256$ apartadore disponíveis
por bloco

bloco = 1K
tamanho i-node = 4B
diretório: nome $\Rightarrow 12B$) 16 bytes
A apartador $\Rightarrow 4B$

6 bytes

1 byte \Rightarrow diretorio

4 bytes \Rightarrow apartador direto - 1K
4 bytes \Rightarrow apartador simétrico - $\frac{1K}{4B} = 256$
4 bytes \Rightarrow apartador duplo = $(256)^2$
4 bytes ou apartador duplo

$$AD = bloco = 1024B$$

$$AS = \frac{1024}{2} \times 1024 = 8$$

$$A.I.D = \frac{1024 \times 1024}{2} \times 1024 = 8$$

$$U_1.98(1+RAM) + \frac{1}{2}(1-U_1)(1-\frac{1}{2})X_{RAM}^2 + \frac{1}{2}(1-U_1)(1-\frac{1}{2})X_{RAM}^2 = 100$$

$$U_1.98(1+RAM) + U_1(1-U_1)X_{RAM} + U_1(1-U_1)X_{RAM} = 100$$

$$U_1(1-U_1)X_{RAM} = 100$$

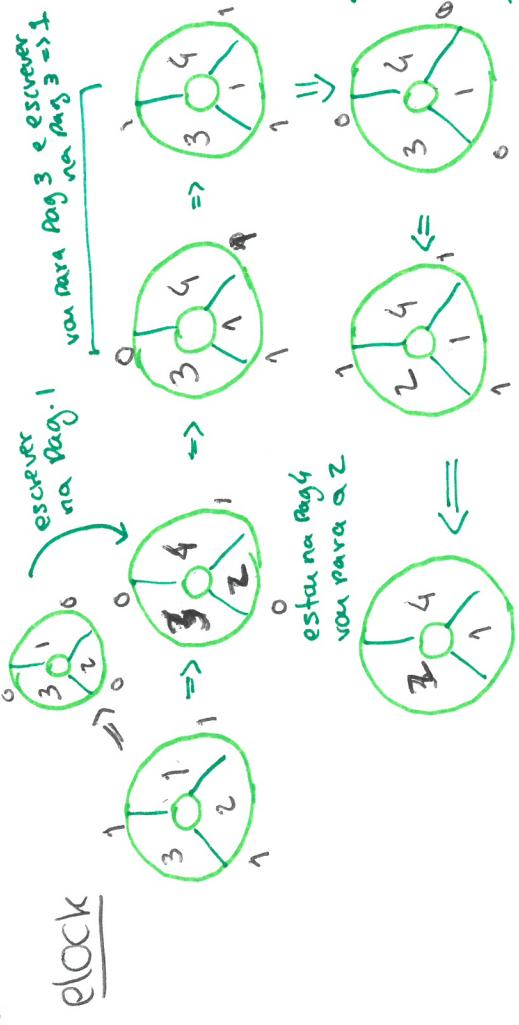
$$X_{RAM} = \frac{100}{U_1(1-U_1)} = 120$$

$$3 \text{ miss} \quad \text{hit} \quad \text{miss}$$

$$TLB = 10n \Delta \quad TLB \quad TLB$$

$$RAM = 100n \Delta \quad RAM \quad RAM$$

$$\text{Hit} = ? = 100n \Delta \quad \text{Hit} \quad \text{Hit}$$



escrever na pag 1

está na pag 2

está na pag 4

está na pag 0

está na pag 3

leitura

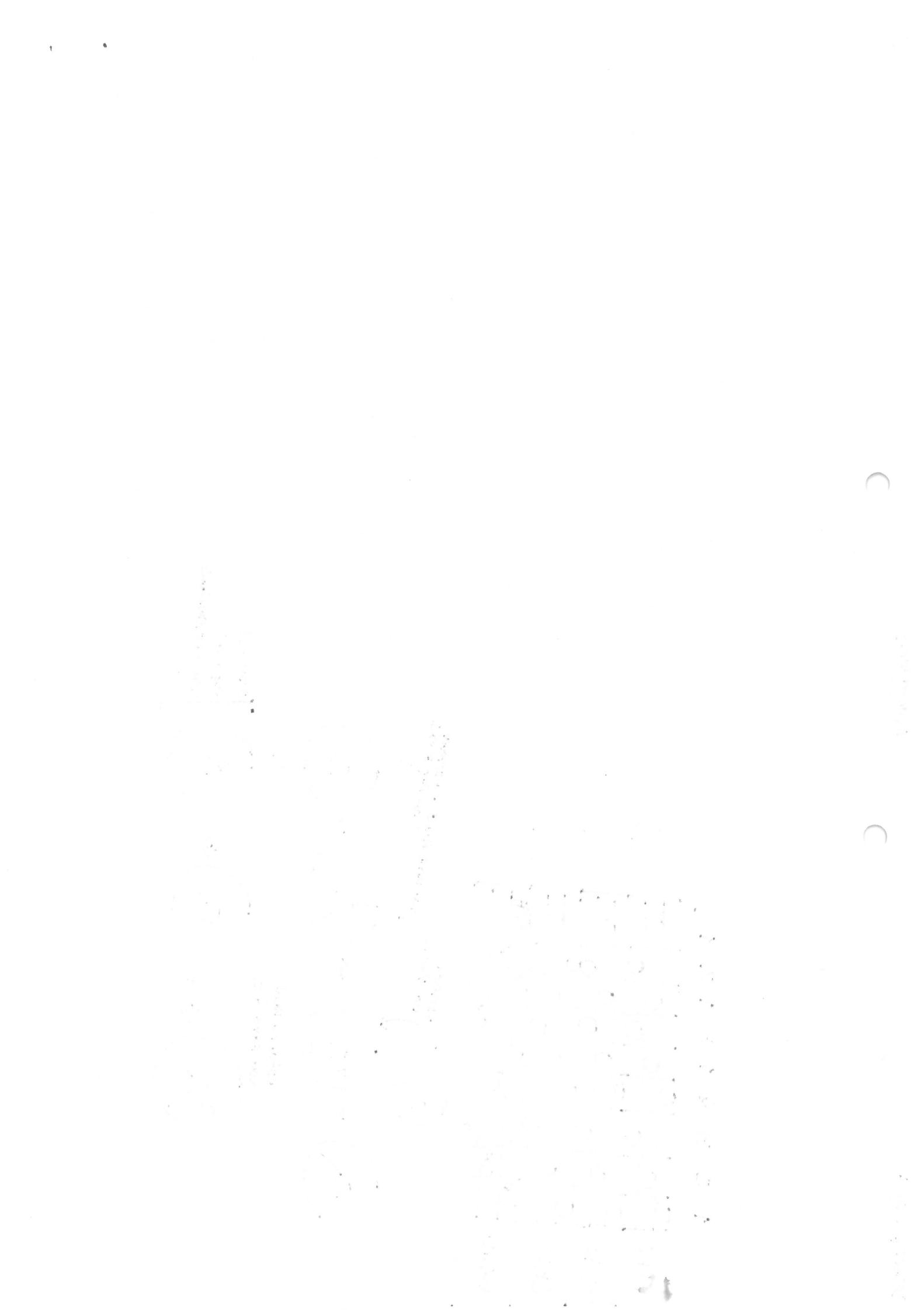


Tabela de Ioginacão

Fazem corresponder onde estão guardadas as coisas, e cada informação tem uma dimensão diferente

Base	dimensão	dimensão
0	1000	100
1	5000	500
2	10 000	300
3	20 000	10 000

Gestão da memória virtual

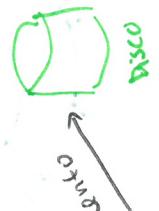
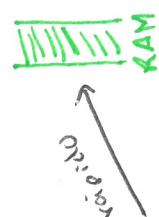
Litura: leitura à memória e sair.

Escrita: Escreve-se na RAM e no disco.
A escrita pode ser feita diretamente na memória e depois fazer uma atualização para o disco.

Escrita > leitura
antes de voltar a escrever a informação já foi atualizada para o disco.

Escrita < leitura

como a escrita é pouca, temos espaço na RAM.
Entendo a certeza que já foi atualizado para o disco.

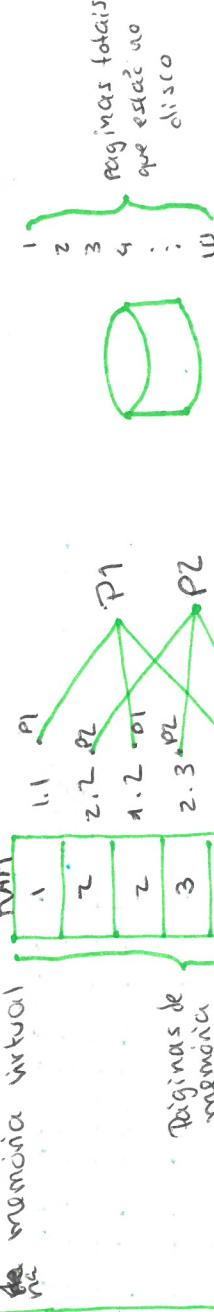


Memória virtual

funciona com tabelas de paginação e segmentação

memória Física - Disco

Existem partes do processo que não vão ficar na memória, mas ficam num dispositivo, nuns



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708<br

2. Óptimo: usado para comparação

dito o futuro e pensa quais os processos que vêm a seguir, e a partir daí se qual o algoritmo que melhor se aplica e compara-o com o algoritmo utilizado.

necessário qual é a página do processo que me vai fazer mais falta no futuro. O algoritmo escuta escrever na página mais longe do presente, aquela que no futuro vai demorar a utilizar. Se n'esse momento pedido para o processo, em nenhum instante significa o endereço das pag.

ex: 3 frames
2 3 2 1 5 2 4 5 3 2 5 2

- 5 páginas
- 3 páginas faltas

FIFO

	2	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3
	1	1	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3

F												
F												
F												
F												
F												

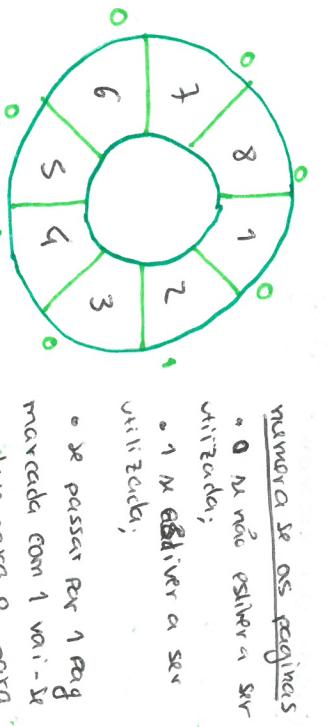
LRU

1	2	3	2	1	2	3	2	3	2	3	2	3
2	3	2	1	2	3	2	3	2	3	2	3	2
3	1	3	2	1	2	3	2	3	2	3	2	3
3	5	3	2	1	2	3	2	3	2	3	2	3
3	5	3	2	1	2	3	2	3	2	3	2	3

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

6. clock

numera-se as páginas
• 0 não estiver a ser utilizada;
• 1 ir dividir a ser utilizada;



3. FIFO: A 1ª página referenciada em 3 é a página referenciada em 2.
ordem em que as pag. são pedidas: 1 2 1 3 4 1 5
exemplo com 3 frames



LRU: escrever por cima da pag. 2

FIFO: escrever por cima da pag. 1, que neste exemplo não é vantoso dado que a pag. seguinte é a 1.

Existem 2 versões deste algoritmo: 1 bit e com 2 bits

• 1 bit, de escrever na pag. tem de estar a 00, incrementa 1 ate a encontrar pagina pronta para 00.
• 2 bits, incrementa 1 ate a encontrar pagina pronta para 00.

Yaginucco

bots que dão as páginas estaticas ou não

215 R

$O \Rightarrow$ Páginas desocupadas

1 => páginas ocupadas

necessário saber onde o dia

para sair de processo

sufficiente una memoria que

Barbara Schenckes-Dietrich

Caso algoritmos de eclocação

DEPARTMENT OF

1
0 0 0

2.1. quanto maiores as páginas, maior a fragmentação interna.
Portanto as tabelas de vecinacão devem manter

TATTOOS WHO MAX OUT MA TATTOO:

- depende do espaço de enderecamento do processo.

$$X = 2 \quad | \quad 9$$

$$= \frac{10 \times 2}{\text{number}}$$

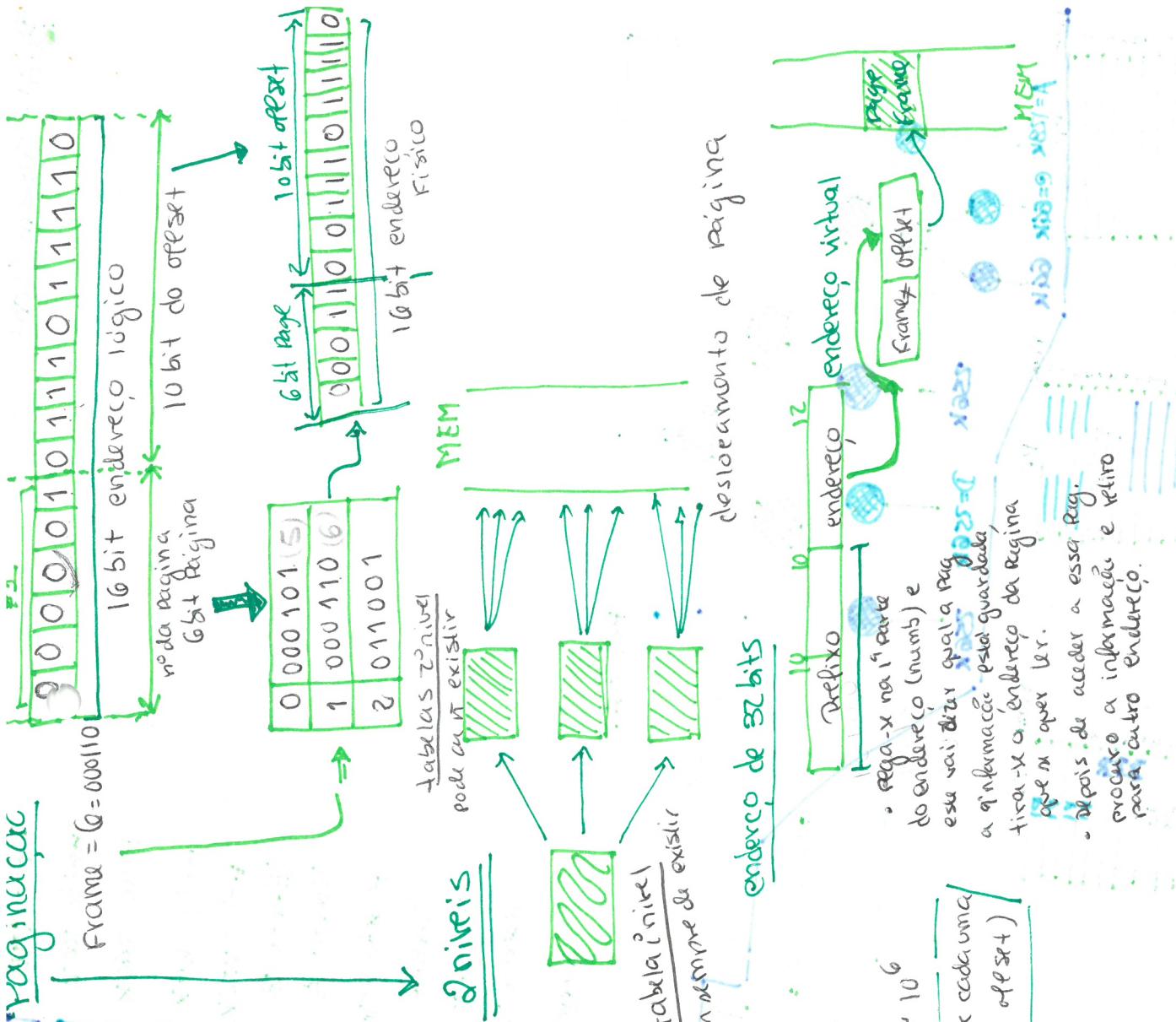
FODE EVIDENCE

Um Taramando da Pagina = $k = \frac{1,024 \text{ bytes}}{10 \text{ bits}} = 0,064 \text{ bytes} = 64 \text{ bytes}$

the following morning.

caiehe:

ag inacar



Buddy System (cont)

bloco inicial de 1MB

A = pedido de 100 KB, necessário bloco 128K

Bloco inicial dividido em 2 blocos iguais 512K (buddies)

1º dos blocos de 512KB é dividido em 2 buddies. blocos 256KB

1º dos blocos de 256KB é dividido em 2 buddies 128KB em que 1º deles é alocado em A.

B = pedido de 256 KB, já se encontra disponível e alocado

Quando E é libertado, 2 buddies 128KB aderem num bloco libertado

de 256K, que obterá imediatamente ao buddy.

⇒ se 2 buddies são folhas, então pelo menos 1 deve ser alocado; As folhas representam a partição presente atual da memória.

As folhas representam a partição presente atual da memória.

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

⇒ quando E é libertado, 2 buddies 128KB aderem num bloco libertado

bloco 1MB

1MB

1MB

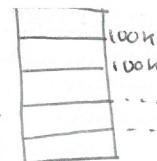
512K

512K

128K

Gestão de Memória

na memória fixa todas as divisões estão previamente definidas. É o SO que controla os acessos, para que o princípio da proteção seja respeitado. Sabe-se sempre onde cada processo começo e acaba. As zonas de memória dinâmica são livres e podem ser muito distintas.

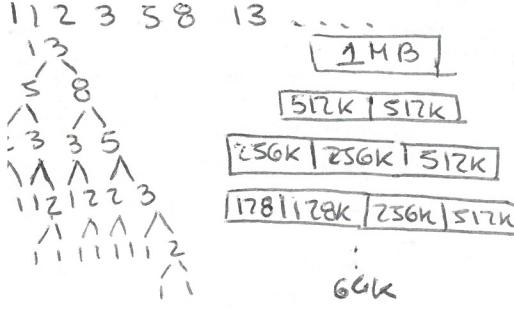


- GM: Requisitos:
1. localização
 2. proteção
 3. partilha de memória
 4. organização lógica
 5. organização física (MP e MS)

Particionamento da mem

- Partici. fixo ①
- Partici. dinâmico ②
- Paginação e Pag da MV
- Segmentação e Seg da MV
- buddy - System ③

blocks tem tamanho 2^k palavras



Deadlocks: de tencāc

→ deadlock => → ciclo

① marcar processos que têm recursos alocados a zero.
Isto é, que não estão em deadlock. Soma das colunas da matriz dos Rec. alocados para saber os recursos totais quantos temos utilizados.

② calcular os recursos disponíveis:

$$\text{Rec. totais} - \text{Rec. alocados} = \text{Rec. disp} \quad (\text{RT} > \text{RA})$$

③ se eu tiver processos que não tem Rec. alocados, ou seja, nenhuma linha só com zeros então eu sei que esse processo pedidos não são satisfeitos com os recursos disponíveis.

④ eliminar e atualizar os recursos disponíveis juntamente com o passo 3 novamente até ao fim

se chegar ao fim => Estado seguro, sem deadlock

	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

Rec. Pedidos

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

Rec. Alocados

$$\text{RT} = 21121$$

$$\textcircled{1} \text{ Recursos alocados} = 21120$$

$$\textcircled{2} \text{ Recursos Disponíveis} = 21121 - 21120 = 00001$$

Estado seguro: Matriz Rec. ~~necessários~~ e Matriz Rec. Alocados depois de um processo correr ficarem com valores que não dão deadlock

③ P4 não vai estar em deadlock. como tenho $\boxed{\text{RD} = 00001}$ e como o valor de Recursos Disponíveis não permite satisfazer nenhum P1 e P2 mas não o consegue fazer com os recursos que tenho disponíveis.

conclusão: os processos que não conseguem satisfazer são os processos que estão em deadlock. Neste caso, os

	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

Rec. Pedidos

	A	B	C	D	E
P1	1	0	1	1	0
P2	1	1	0	0	x
P3	0	0	0	2	0
P4	0	0	1	0	0

Rec. Alocados

$$\text{RT} = 21131$$

$$\textcircled{1} \text{ Recursos alocados} = 21130$$

$$\textcircled{2} \text{ Recursos Disponíveis} = 21131 - 21130 = 00001$$

③ P4 nunca está em deadlock. $\text{RD} = 00001 + 00020 = 00021$

$$\textcircled{4} \text{ RD} = 00021 + 10110 = 10131$$

$$\textcircled{4} \text{ RD} = 10131 + 11000 = 21131$$

conclusão: não há deadlock, já o estado é seguro. Aceite pelo algoritmo do barqueiro. (impede que ocorra deadlock).

	A	B	C
P1	1	1	0
P2	1	1	0
P3	1	0	0

Rec. Pedidos

	A	B	C
P1	1	0	1
P2	1	0	1
P3	0	1	1

Rec. alocados

$$\text{RT} = 333$$

$$\textcircled{1} \text{ Rec. alocados} = 113$$

$$\textcircled{2} \text{ Rec. Disponíveis} = 333 - 113 = 220$$

③ P2 nunca está em deadlock. Nesse instante,

• P1 precisa A, B

• P3 precisa A

conclusão: como $\text{RD} = 220$ é possível satisfazer ambos os processos/pedidos. Depois de satisfazer os pedidos pendentes até ao fim, os recursos alocados são libertados.

Alg. Bang

Matriz Rec. necessários: total que cada processo necessita.

Matriz falta: o que falta a cada processo até ao final para cada um conseguir correr.

↳ M. Rec. nec - M. Rec. alocados

$$\text{D Rec. alocados} = 925$$

$$\text{D RD} = 936 - 925 = 011$$

Comparar o disponível com o que falta de cada processo.

Neste caso, consegue correr P2 e depois de terminado o que está alocado é libertado $\text{RD} = 011 + 612 = 623$

→ todos possíveis de satisfazer. Nesse instante, o estado do sistema é seguro. e garante que a partir daquele momento consiga chegar ao fim. Sem nenhum deadlock.

Deadlock: algoritmo que mata

→ mata processos em deadlock, mas deixa que eles aconteçam primeiro.

→ pode matar:

• escolher um processo random

• escolhe um processo que minimiza o impacto

• por prioridade

• por tempo gasto em CPU

Deadlock: como tomar a decisão qual o processo a matar: nº processos a matar, prioridade dos recursos temp. para terminar: aleatoriamente, tempo gasto em cada processo

→ depende do escalonamento para saber qual é o melhor critério a ser usado. Se o escalonamento usar a prioridade então a prioridade dos processos vai ser o melhor critério. Caso o escalonamento trabalhe com o tempo, então o tempo gasto em cada processo é o melhor critério. A prioridade dos processos é o melhor critério. Aí prioridade vai fazendo a verificação se existem deadlocks ou não, estes critérios vão ser apenas usados quando já não existe deadlock.

$$\textcircled{1} \text{ Recursos alocados} = 21120$$

$$\textcircled{2} \text{ Recursos Disponíveis} = 21121 - 21120 = 00001$$

Estado seguro: Matriz Rec. ~~necessários~~ e Matriz Rec. Alocados depois de um processo correr ficarem com valores que não dão deadlock

apenas posso satisfazer o P3.

desde que os processos que não conseguem satisfazer nem P1 nem P2 mas não conseguem fazer P3.

conclusão: os processos que não conseguem satisfazer são os processos que estão em deadlock. Neste caso, os

	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

Rec. Pedidos

	A	B	C	D	E
P1	1	0	1	1	0
P2	1	1	0	0	x
P3	0	0	0	2	0
P4	0	0	1	0	0

Rec. Alocados

Recursos necessários

A B C

P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Recursos alocados

A	B	C
1	0	0
6	1	2
2	1	1
0	0	2

"Falta"

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

6. Indicar 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 pôs à espera de um evento
- D - Ocorreu um evento enquanto esperava por dados.

7. Indicar a hipótese correta.

- A - o uso de threads só é vantajoso com CPUs múltiplas
- 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 múltiplas
- D - o uso de threads com CPUs múltiplas, torna-se mais lento.

8. dados Partilhados por threads e o que não é partilhado, dum mesmo processador.

Partilhado

- data (dados)
- código
- variáveis globais

Não Partilhado

- ID processos
- Ficheiros (abertos) do CPU
- I/O
- registos (de ativação das funções)
- PC
- Stack/Mem (ptr)
- Quantum
- estados (PC)
- Prop. esca.
- modo CPU

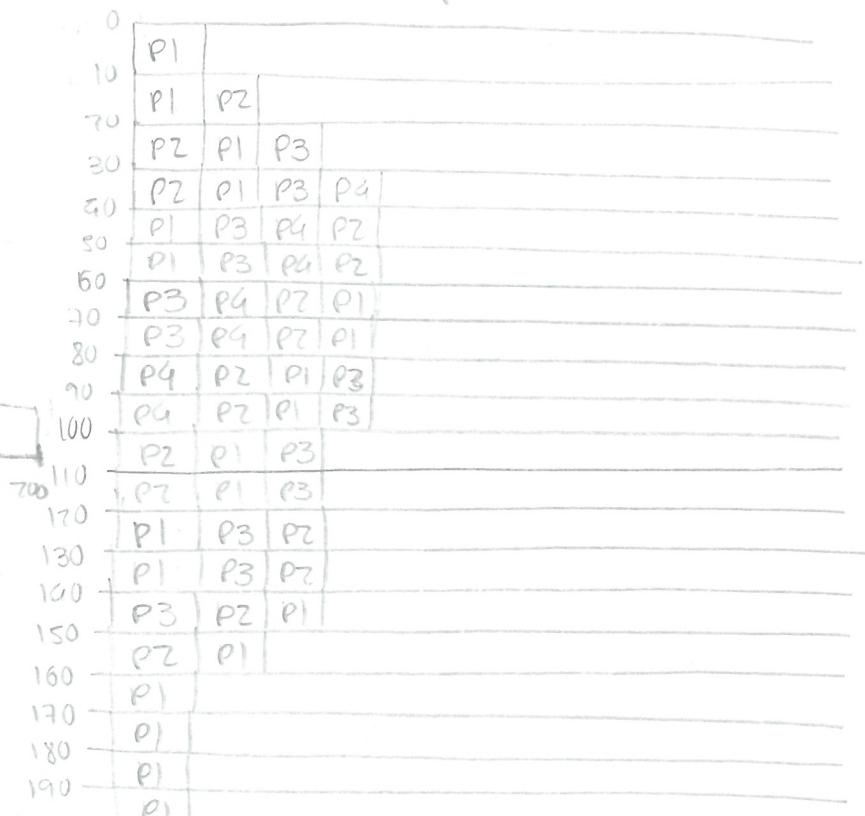
PCB

- variáveis locais / temp

P	tchegada	tserviço
P1	0	100
P2	10	50
P3	20	30
P4	30	20

$$RR : Q = 20 \quad SRT = 200$$

P1	P2	P3	P3	P3	P4	P4	P2	P2	P2	P2	P1
10	20	30	40	50	60	70	80	90	100	110	



$$t.a.t = (200-0) + (110-10) + (50-20) + (70-30)$$

$$= \frac{370}{4} = 92,5 \text{ ms}$$

$$t.a.t = (200-0) + (160-10) + (150-20) + (100-30)$$

$$= \frac{550}{4} = 137,5 \text{ ms}$$

10. Descrever as principais características de um sistema operativo de tempo real.

- Resposta é válida apenas se for inferior ao limite, se o resposta estiver dentro do tempo certo.
- Hard real time: sempre cumprir sempre o deadline exato, para no SO sistemas previsíveis.
- Soft real time: tem deadline, mas não é obrigatório, flexível completa a tarefa, mesmo passando o deadline sistemas mais complexos
- determinista: prioridade aos algoritmos simples
- capacidade de resposta
- controlo por parte do user: propriedades dinâmicas (SRT), podem ser si preemptivos ou não preemptivos

u.

P	tchegada	tservico
P1	0	100
P2	10	50
P3	20	30
P4	30	20
		= 200

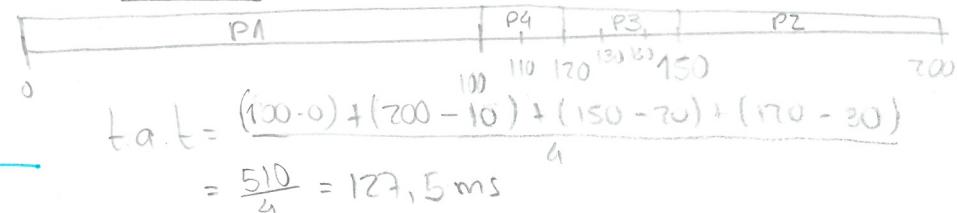
$$RR: Q = 30$$

0	P1
10	P1 P2
20	P1 P2 P3
30	P2 P3 P1 P4
40	P2 P3 P1 P4
50	P2 P3 P1 P4
60	P3 P1 P4 P2
70	P3 P1 P4 P2
80	P3 P1 P4 P2
90	P1 P4 P2
100	P1 P4 P2
110	P1 P4 P2
120	P1 P4 P2
130	P1 P4 P2
140	P2 P1
150	P2 P1
160	P1
170	P1
180	P1
190	P1
200	

$$t.a.t = \frac{200 + (160 - 10) + (90 - 20) + (160 - 30)}{4} = \frac{530}{4} = 132,5 \text{ ms}$$

$$t.a.t = \frac{(160 - 0) + (90 - 10) + (80 - 30) + (100 - 60)}{4} = \frac{350}{4} = 87,5 \text{ ms}$$

SPN



P	tchegada	tservico
P1	0	100
P2	10	30
P3	30	20
P4	40	10
		= 160

$$RR: Q = 20$$

SRT

SPT

P1	P4	P2	P3
100	110	130	160
10	20	30	40
30	50	60	70
50	70	80	90
70	90	100	110
90	110	120	130
110	130	140	150
130	150	160	

$$t.a.t = \frac{(100 - 0) + (130 - 10) + (160 - 30) + (110 - 50)}{4} = \frac{420}{4} = 105 \text{ ms}$$

$$t.a.t = \frac{(160 - 0) + (40 - 10) + (70 - 30) + (50 - 70)}{4} = \frac{260}{4} = 65 \text{ ms}$$

0	P1
10	P1 P2
20	P2 P1
30	P2 P1 P3
40	P1 P3 P2 P4
50	P1 P3 P2 P4
60	P3 P2 P4 P1
70	P3 P2 P4 P1
80	P2 P4 P1 P3
90	P2 P4 P1 P3
100	P4 P1 P2
110	P1
120	P1
130	P1
140	P1
150	P1

5 CONCLUIA o tempo médio para terminar um processo (turnaround time) para o escalonamento RR de Q=20.

5: RR : Q = 20

P	tchegada	tservico
P1	0	60
P2	10	40
P3	20	30
P4	30	10
P5	40	10

P	tchegada	tservico
P1	0	50
P2	10	40
P3	20	30
P4	30	20
P5	40	10

P	tchegada	tservico
P1	0	60
P2	10	40
P3	20	30
P4	30	10
P5	40	10

P	tchegada	tservico
P1	0	50
P2	10	40
P3	20	30
P4	30	20
P5	40	10

$$t.o.t = (160 - 0) + (110 - 10) + (150 - 20) + (90 - 30) + (120 - 40) \quad taf = \frac{(160 - 0) + (120 - 10) + (150 - 20) + (100 - 30) + (130 - 40)}{5} = \frac{840}{5} = 168 \text{ ms}$$

$$= \frac{840}{5} = 168 \text{ ms}$$

5

5

5

5

5

5

5

1: Indicar a hipótese correta. O uso de threads é vantajoso...

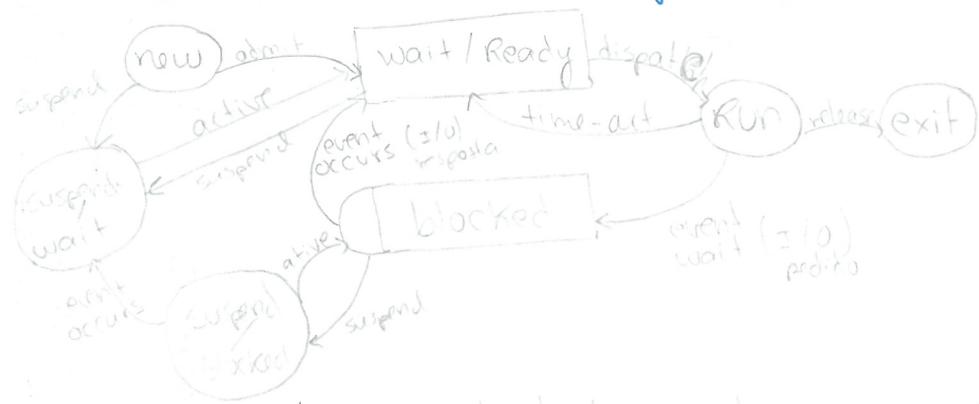
A - arenas de existe mais do que um com variaias tarefas.

B - N existirem N tarefas do mesmo tipo com uso de CPU e uso de I/O.

C - apenas N existirem N tarefas, todas diferentes, com varios CPU's.

D - N existirem N tarefas, todas diferentes, com varios CPU's.

1. Descrever graficamente o modelo de 7 estados e explicar as transições de um processo dos estados "Suspend" para outros estados.



new - wait/ready: SO admite/ preparado para um novo processo;

new - suspend/wait: Processo é criado mas, não se encontra pronto para ser corrido;

wait/ready - Run: Seleção de um processo com maior nível de prioridade para correr, esse que está na base do estado Ready;

wait/ready - suspend/wait: necessário libertar espaço na memória principal;

Run - exit: Processo foi terminado. Ocorreu acabado ou interrompido pelo SO.

Run - wait/ready: Runtimes do processo, time-out,

Run - blocked: Processo necessita de algo que ainda não está disponível, pedido de I/O;

Blocked - wait/Ready: Processo que está em espera recebe o que lhe falta, pode ser retomado.

blocked - suspend/blocked: como nenhuma processo pronto, então um lugar de processos é ocupado em suspensão. logo, é trocado para fora de modo a haver lugar para outro processo que não seja blocked.

suspend/blocked - blocked: Processo que está pronto a ser executado e não está em memória principal

suspend/blocked - suspend/wait: Processo está em memória secundária e recebe algo que necessitava e fica à espera que haja espaço na memória principal, dado que o processo está pronto a ser executado.

suspend/wait - wait/ready: processo está pronto a ser executado localizado em memória secundária, como há espaço em memória principal, o processo passa para MP para o estado wait/ready.

2. Indicar diferenças entre Threads User Level e Threads de Kernel.

Threads User Level:

- só consideradas processos pelo SO;
- operações podem correr em 1 CPU;
- só bloqueantes; quando necessita de info o processo transfere para estado blocked;
- funciona em apenas 1 CPU;
- troca de processo mais rápida, pois não possui intervenção do SO;

Threads de Kernel:

- reconhecem as threads pelo SO;
- existência depende do SO;
- não bloqueante;
- funciona em N CPU;
- paralelismo;
- mudança de processos necessita da intervenção do SO;

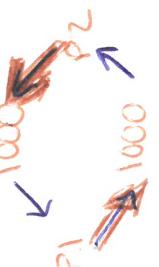
Algoritmo do Banqueiro

como ver se há deadlock

⇒ algoritmo de banqueiro { impede que deadlock avoidance } acorra deadlock

Exemplo:

Existem 2 clientes, cada cliente necessita de 10.000€ em primeiramente para abrir um negócio. No entanto no total os 2 homens vão necessitar de 2000€ para abrirem o negócio. O dinheiro total que o banco tem para emprestar é 2000€, então no inicialmente o P1 pedir 1000€ ao banco e este o deve e o P2 pede 1000€ também, nesse caso depois vai existir um deadlock porque o P1 passado que só existem recursos utilizados, os processos não geram um tempo vai necessitar de mais 1000€ tal como o P2 e o banco vai não ter dinheiro para dar.



P1
wait()
wait()
1000
1000
1000
signal()
signal()

P2
wait()
wait()
1000
1000
1000

⇒ neste algoritmo o que está em causa é a existência de um pedido e a aceitação ou não desse pedido. Causa o que o algoritmo para é caso o P2 pedisse 1000€ esse iria dizer não esse pedido porque sabia que existe a possibilidade de deadlock desse processo. Assim, o algoritmo diz não a todos os pedidos que podem levantar deadlock. logo tem sempre que perguntar e responder (sim ou não)

Passo: soma dos recursos alocados a coluna, para saber quantos totais recursos disponíveis

Passo: calcular os recursos disponiveis: [recursos totais - recursos utilizados] = [recursos disponiveis] → total de recursos disponíveis

recursos pedidos (request)

o que precisa pedidos =

A B C

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

0 1 2

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

1 1 1

recursos alocados

o que temos

A B C

1 1 1

1 1 1

1 1 1

5º PASSO: dar para os recursos disponíveis e dizer para a matriz dos pedidos. Repetir até se conseguir satisfazer algum pedido com os recursos que temos disponíveis.

neste caso, como temos 00001 de recursos disponíveis apenas conseguia satisfazer o processo 3.

como o processo 3 vai querer emprestar os recursos e esse vai querer quando ele acabar os recursos que tinha alocados vai ficar com recursos disponíveis.

recursos disponíveis = [00011]

5º PASSO: temos recursos disponíveis são diferentes dos anteriores volta-se a ver a matriz dos pedidos para ver se conseguia satisfazer algum pedido que falta satisfazer.

neste caso, falta satisfazer P1 e P2 mas não é possível dado que 1º P3 vai querer \Rightarrow recursos disponíveis = 21131 - 21130 = 00001 não disponibiliza recursos suficientes disponíveis!

Conclusões: 1. Os processos que não são possíveis satisfazer entram estão em deadlock (P1 e P2)

2. Finance está em deadlock

acessa do algoritmo:

- nunca existe deadlock

- Estado Seguro: quando a matriz de recursos necessários é a matriz de alocação depois de um processo conter picarem com valores que não dão deadlock

Exemplo

recursos pedidos

	A	B	C	D	E
P1	3	2	2		
P2	1	1	3		
P3	3	1	4		
P4	1	2	2		

$$3^{\text{º}} \text{ Pedido P2} \rightarrow R.D = 013 + 612 = 625$$

(i) P4 não está em deadlock e não consegue

possuir nenhum processo logo não haver deadlock

Outro exemplo

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

</

DEADLOCK E CICLO

ambos os processos precisam de 2 recursos A e B. se de escalar um semiparo de modo a garantir que um recurso só pode ser utilizado por um recurso de cada vez.



`P1`
 wait(A)
 wait(B)
 wait(A)
 P1()
 signal(A)
 signal(B)

Como eliminar um ciclo

ao eliminar o ciclo, eliminar o deadlock. ou seja, o deadlock existe para isso é necessário eliminar o ciclo, eliminando um nó (um processo) desse problema fica resolvido. o que o SO vai fazer é matar um processo e dessa forma o deadlock desaparece.

se P1 está a utilizar o recurso A e o P2 está a utilizar o recurso B, então neste caso os 2 recursos estão a ser utilizados, então vai existir deadlock porque P1 precisa de B recursos no entanto só possui 1. já que o outro encontra-se a ser executado por o P2.



`P1`
 P1 → P2 → B → recurso a ser usado
`P2`
 P2 → P1 → A → recurso a ser pedido

P1: Pica A espera que o recurso B seja liberado e o P2 pica à espera que o recurso A seja também libertado.
 Eternamente à espera em deadlock.

como forma de evitá-lo é tentar o problema, se se tiver 2 recursos, este adiciona mais cronologia.
 se temos os recursos A e B nos 2 processos, este adiciona mais cronologia.

Deadlock → Ciclo

Qual quer nº de recursos

⇒ se existe um ciclo então existe um deadlock, mas se apenas existir uma unidade de recursos;

⇒ se vai fazer verificações para ver se existe deadlock, isso delete vai tentar quebrá-lo.

Como eliminar um ciclo

ao eliminar o ciclo, eliminar o deadlock. ou seja, o deadlock existe para isso é necessário eliminar o ciclo, eliminando um nó (um processo) desse problema fica resolvido. o que o SO vai fazer é matar um processo e dessa forma o deadlock desaparece.

Algoritmo de detecção do deadlock

como tomar a decisão sobre qual o processo que é morto

critérios

- nº de processos a matar
- Prioridade dos processos
- revisão optimizada

tempo exec. terminado

- aleatoriamente

- tempo gasto em cada processo

o algoritmo de escalonamento para saber qual é o melhor critério a ser usado. se o escalonamento usava a prioridade então a prioridade dos processos vai ser o melhor critério, caso o escalonamento trasalhe com o tempo, então o Tgusto em cada processo é o melhor critério.

- 40 pessoas podem entrar numa sala, apenas.

init S = 1

P

wait(S)

apenas 40 processos podem estar na seção crítica,

se eu tenho um nº limite de processos que podem

estar em simultâneo

coloca a inicialização = numero;

aluno. ler

aluno. programar

signals(s)

instruções

problema que é resolvido

quando iniciámos o

semaforo para a 1

exclusão mutua

signals(s)

para que isso aconteça,

usar os semáforos

para que isso aconteça,

p. mas quando o 1º vai

entrar, todos os outros

tem de ficar à espera

Produtor

edocar - na - pilha()

para poder colocar na pilha

preciso de recursos (espaço)

wait(e) espaço

edocar - na - pilha()

para retirar da pilha
preciso de algo selocado
para conseguir tirar

retirar - da - pilha()

signal(θ)

objeto

Restrições

leitores (L)

1º: necessário saber as restrições que têm de cumprir em n processos;



1. Leitores - Escritores

escritores (escreve)

Problemas

if N = ...

como só 1 processo

acessa de cada vez

seção crítica

processos

possuir cópias d

processos

Exemplo

- 40 pessoas podem entrar numa sala, apenas.

init S = 1

P

wait(S)

apenas 40 processos podem estar na seção crítica,

se eu tenho um nº limite de processos que podem

estar em simultâneo

coloca a inicialização = numero;

aluno. ler

aluno. programar

signals(s)

instruções

problema que é resolvido

quando iniciámos o

semaforo para a 1

exclusão mutua

signals(s)

para que isso aconteça,

usar os semáforos

para que isso aconteça,

p. mas quando o 1º vai

entrar, todos os outros

tem de ficar à espera

tem de ficar à espera