

Sistemas Operativos

definições

deadlock: Paragem de processos por conflito de recursos.
solução: Reiniciar um processo.

escalonamento: Dá Prioridade, permite alternância.

Sistemas de Ficheiros: blocos de memória / Páginas de mem associados ao disco.

Dual Mode Operative: Consulta hardware e as posições de cada processo → Proteção entre 2 processos.

- Informação dos Processos definida pelo S.O.

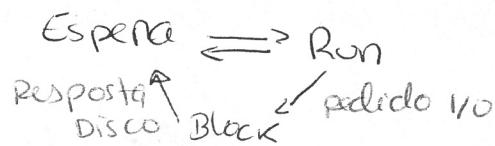
Alternar processos: Kernel Mode.

Para passar de U → K usa-se uma função que gera uma interrupção. Só é possível se existir Dual Mode Operative.

Modelos de Estado

Modelo de 3 estados

Run: Possível aceder à memória



Block: Não está a correr (disco), encontra-se em espera. Quando tem o que precisa passa a wait.

Escalonador: Decide se o processo fica ou não no fim da fila.

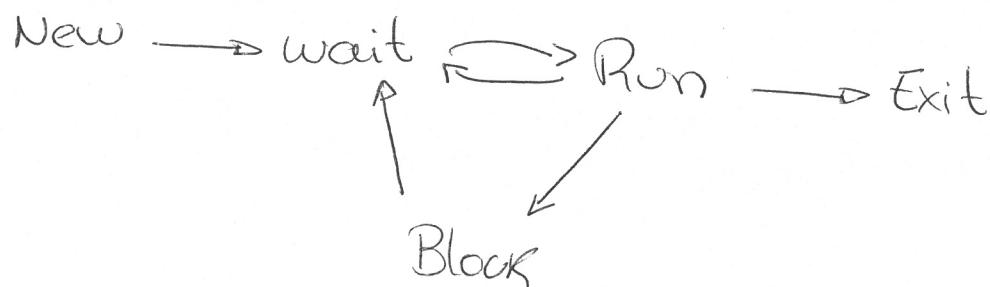
Run → Block: pedido I/O ou interrupção (disco)

Block → Wait: Quando há um evento que tem uma resposta em block. Gera-se uma interrup. → wait

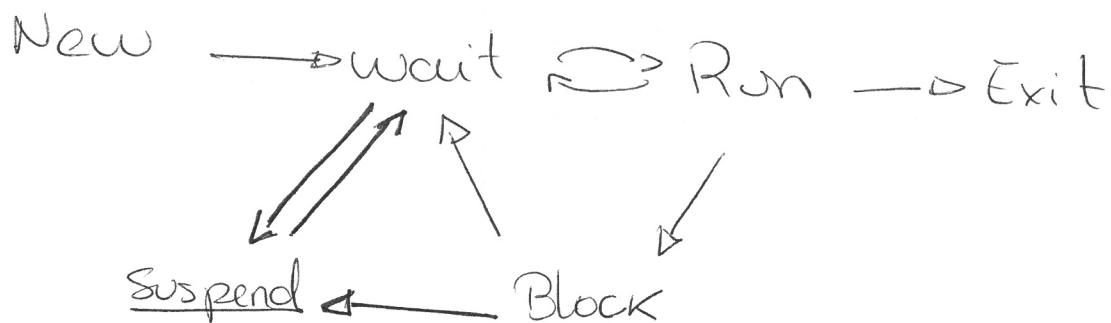
Exit: Terminou. Só apaga o processo após ter sido lida a sua resposta.

* **wait:** Deve estar pronto a correr; Utiliza var. globais / temporárias e código executável, bem como Process control block, exe, dados.

Modelo dos 5 estados

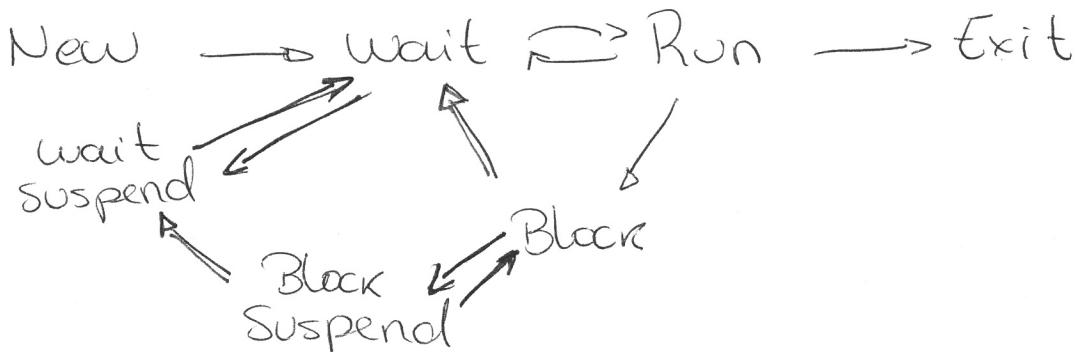


Modelo de 6 estados



- SWAP, em disco ; É usado quando falta memória, pode receber info.

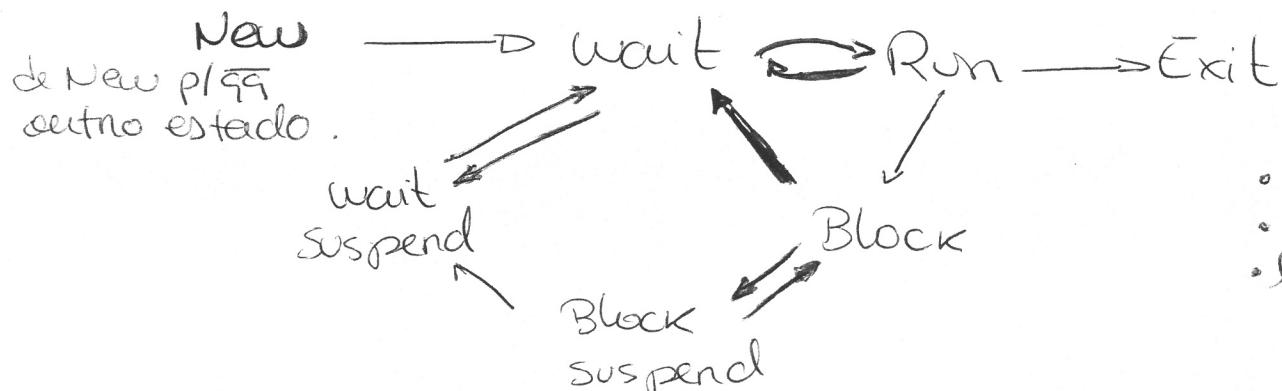
Modelo de 7 estados



Block Suspend: Em espera de inf. I/O.

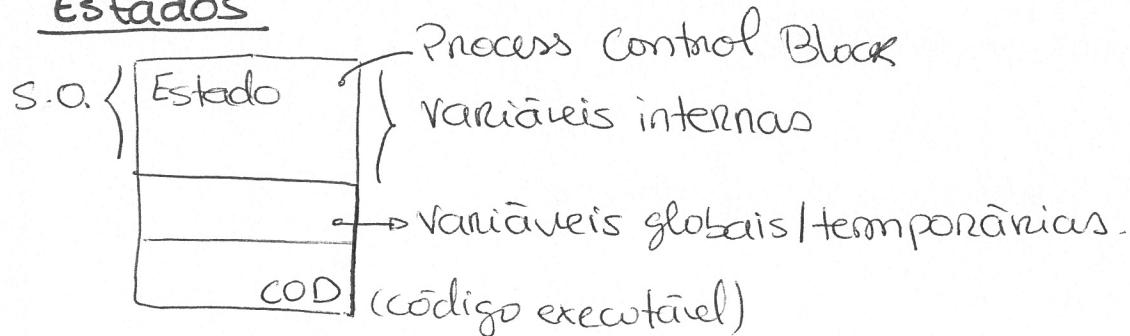
wait Suspend: Já contém inf. I/O.

Escalonamento



Sistemas Operativos

Estados



Estados: CPU ou Espera

Modelo de 3 estados

em Run (CPU) é possível fazer acessos à memória; No disco (Block) não está correndo, em Block considera-se em espera e, quando tem o que precisa vai para o processo wait (se fica ou não no fim da "fila" de espera, depende do escalonador).

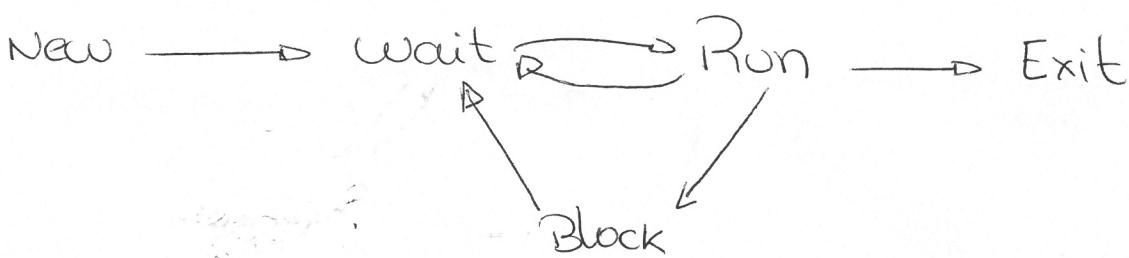
Quando passa de Run para Block significa que houve um pedido I/O ou disco (interrupção).

Quando há um evento a dizer que tem uma resposta em block, dá-se uma nova interrupção para passar ao wait.

* Quando termina, o seu processo não é apagado até que seja lida a sua resposta. → Estado exit é feito parte do Mod. Estados.

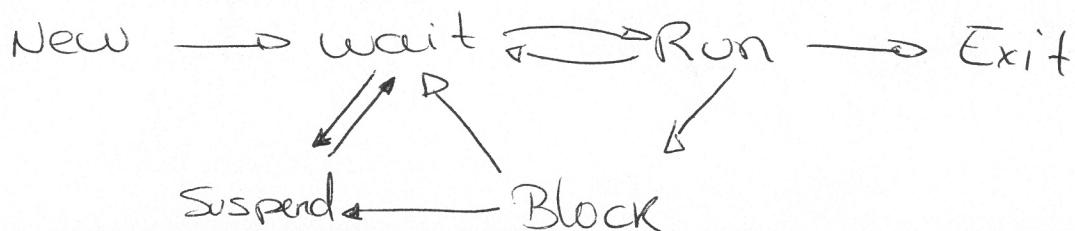
O estado wait tem de estar pronto a correr, utiliza Process control block, Exe, dados.

Modelo de 5 estados



Estados de espera: wait e block.

Modelo de 6 estados



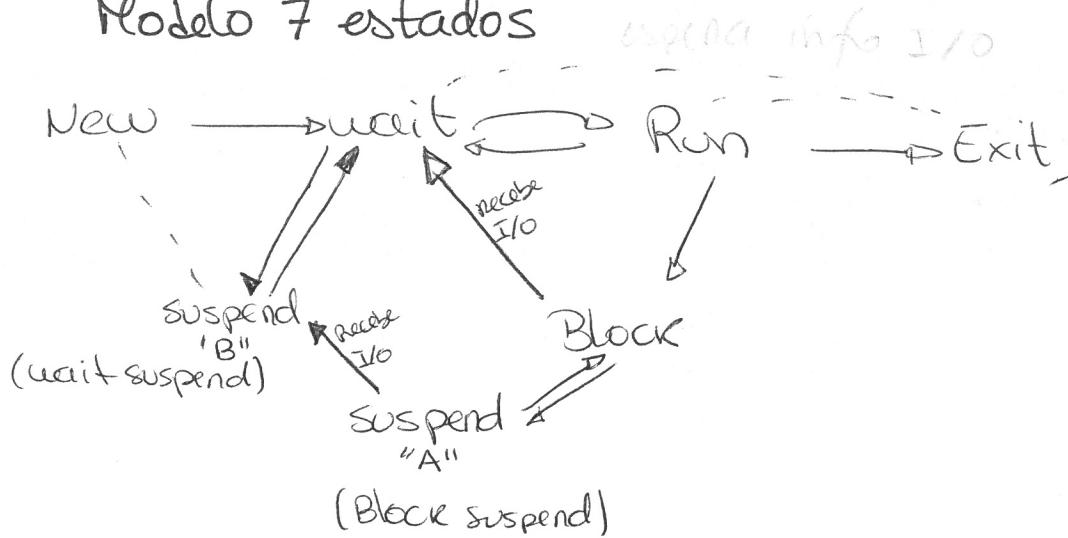
suspende (swap, em disco)

→ Serve para separar?

→ É usado quando existe memória em falta.

→ Ainda pode receber informações.

Modelo 7 estados



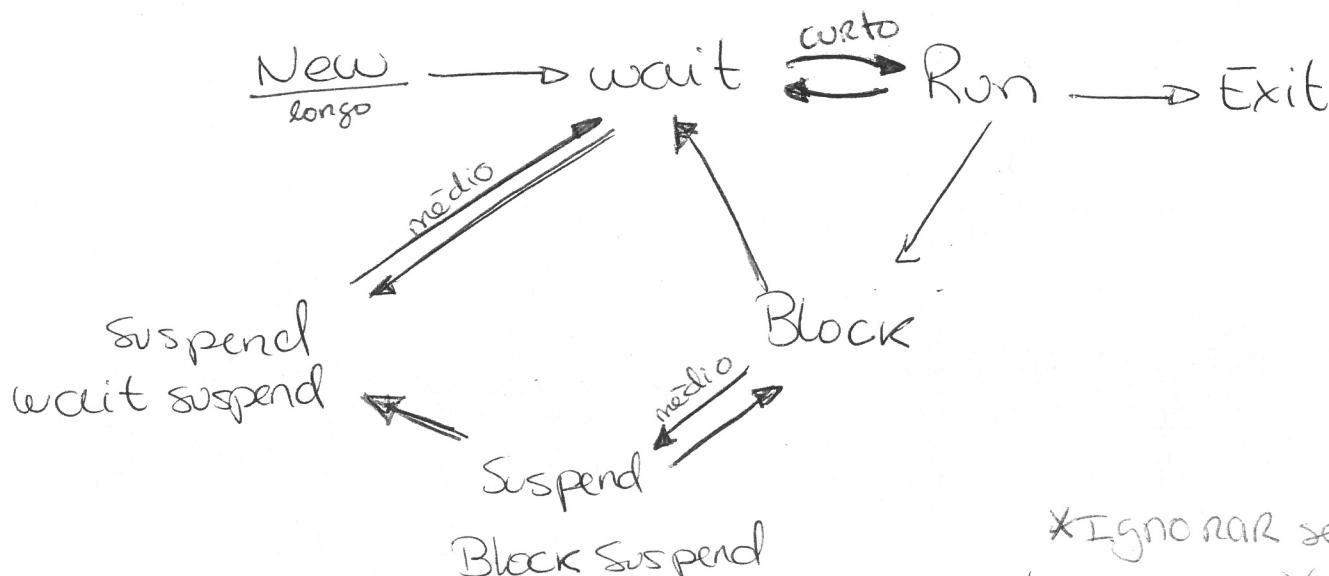
- unidirecional
- saltos (ex. kill)

- Suspend "A" = Block suspend : em espera de informações I/O
- Suspend "B" = Wait Suspend : já contém →

Escalonamento de

- curto
- médio
- longo prazo .

Note:
longo prazo de New para qualquer outro estado .



* Ignorar setas verdes

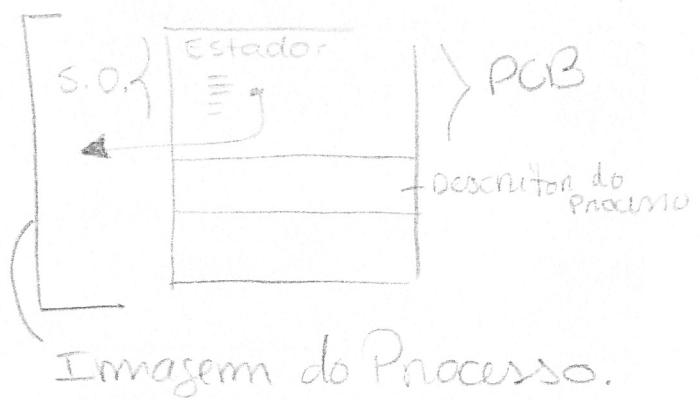
Sistemas Operativos

- Podem ser acrescentados estados, dependendo do S.O.
ex: wait ou block para que se definam prioridades.

→ Tudo o que estiver para trás serve para justificar a criação e necessidade da variável estado.

Variáveis do S.O.

- PID
- Prioridade (escalonamento)
- zonas de memória
- I/O ; Ficheiros.
- PC.
- tempo (escalonamento)
de uso da CPU



FORK

Está a fazer-se uma cópia da Imagem do Processo com um diferente PID.

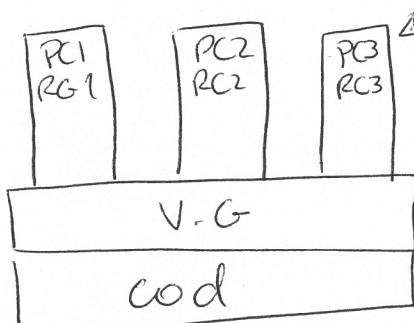
Um dos estados (~~um~~ filho) deve estar em wait enquanto o outro está em Run.

Pai devolve PID filho. Filho devolve PID = 0.

Momento da Cópia

PC1 Reg1	PC2 Reg2	PC3 Reg3
IGual	VG	VG
cod	cod	cod

=>



Há partilha de VG e cod, incluindo mudanças.



(mecanismos:
- semáforos
- locks)

) Programação
concorrente

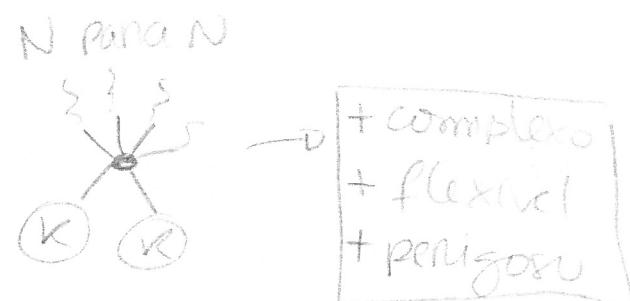
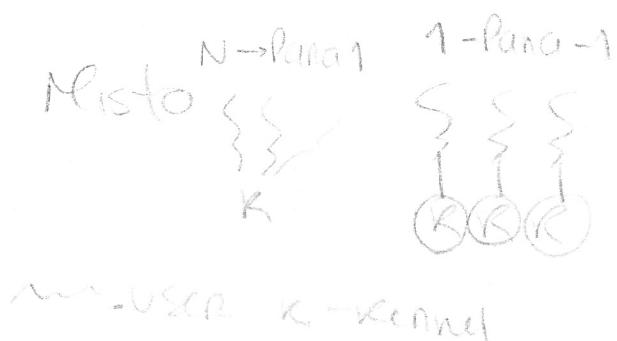


Processo // Thread

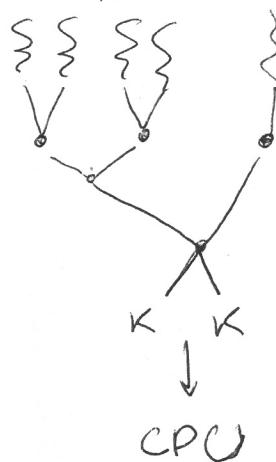
Criar uma thread é muito mais vantajoso do que criar um processo e ter de o pagar. Matar uma thread também é mais fácil que um processo.

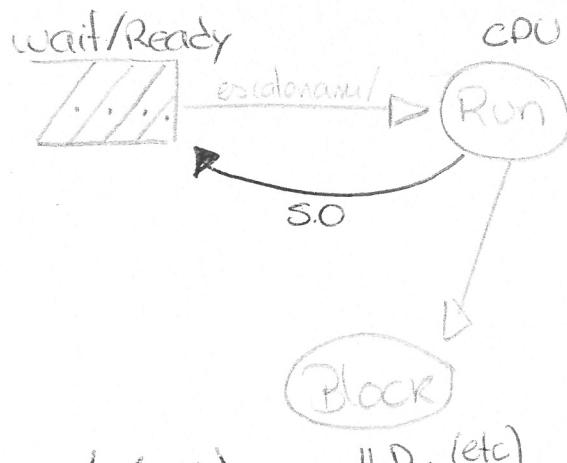
Tipos de threads

	thread Kernel	thread Userlevel
criar	+ lento	• + Rápido
Syscall Block	não bloqueia todas as threads	Bloqueia todas as threads

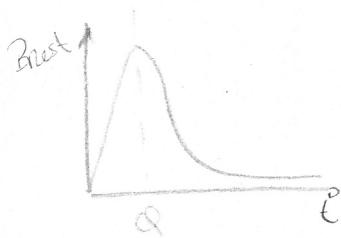
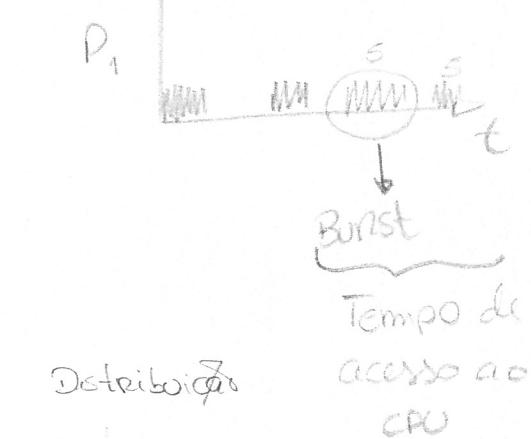


Lightweight Process





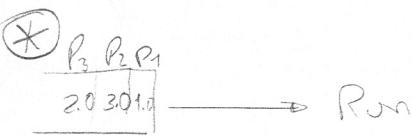
Burst (CPU)



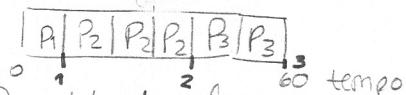
Probabilidade da dimensão de um Burst em função ao tempo.

Algoritmo não preemptivo

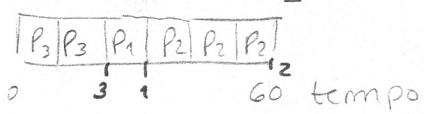
- First come First served.



Resultado do escalonamento:



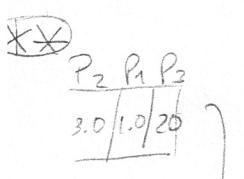
Resultado do escalonamento c/ diferente ordem



Medida de desempenho
Ta Turnaround

Se o critério for o tempo, tanto faz a ordem dos processos

É mais vantajoso, tendo em conta que o tempo total é o mesmo, terminar mais processos.



P1 | P2 | P3 | P2 | P3 | P3 → Melhor tempo turnaround

Note:
A capacidade de se poder ou não interromper um processo é possível através de dois algoritmos de escalonamento:

1. Preemptivos
2. Não preemptivos.

P	T
P1	10 ms
P2	
P3	

Tempo que dura o processo a sair
TMAS = Quantum

função do algoritmo de escalonamento.

(*) FCFS

P ₁	P ₂	P ₂	P ₂	P ₃	P ₃
0	10	40	60		

turnaround médio =

$$= \frac{(10-0) + (40-10) + (60-0)}{3} = 36,66$$

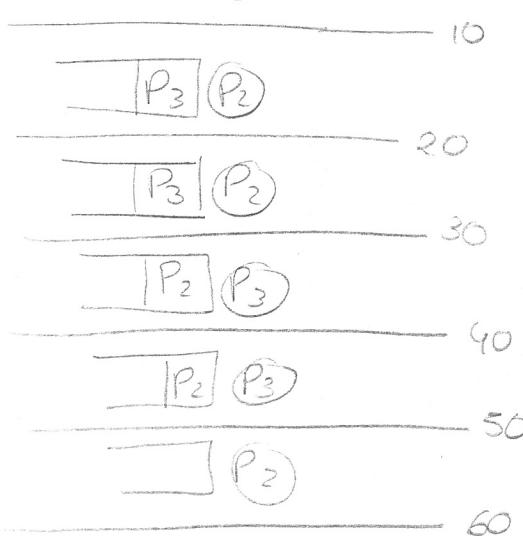
(**) SJF (Precisa dos tempos de serviço).

P ₁	P ₃	P ₃	P ₂	P ₂	P ₂
0	10	30	60		

Não Preemptivo

Algoritmo RR (não precisa de tempo de serviço)

wait CPU



turnaround médio =

$$= \frac{(10-0)(30-0)(60-0)}{3} = 33,3$$

→ Mais otimizado.

com Quantum = 20 ms

P	T
P ₁	10 ms
P ₂	30 ms
P ₃	20 ms

turnaround médio:

$$(10-0)(60-0)+(50-0) = 40 \text{ ms}$$

algoritmo SRT

P	T _s	T _{ch}	TS = tempo de serviço da CPU
P ₁	90	0	
P ₂	10	10	

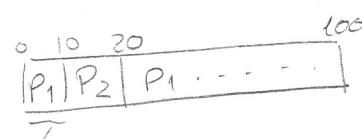
FCFS

P ₁	90	10
----------------	----	----

SJF

90	10
----	----

SJF só é ótimo quando chegam todos os processos chegam no inicio.

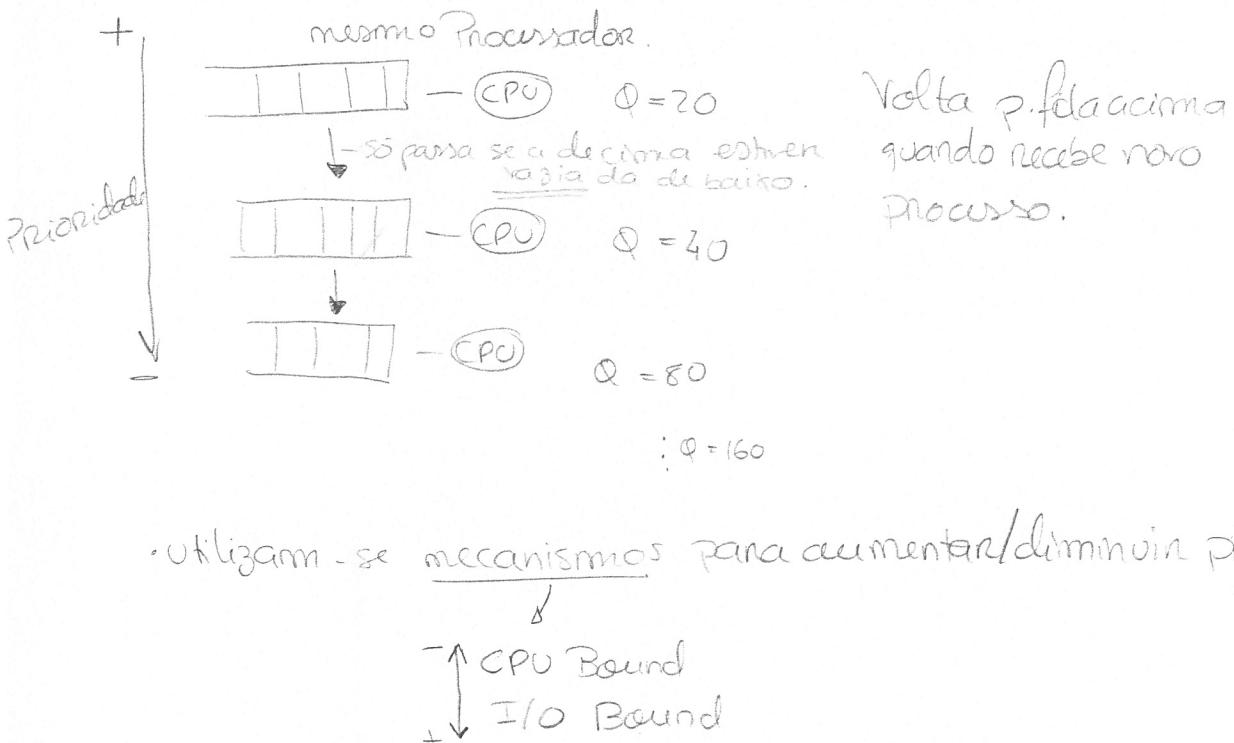


consequência
o P₁ é interrompido
porque chega P₂.

Aqui não há quantum

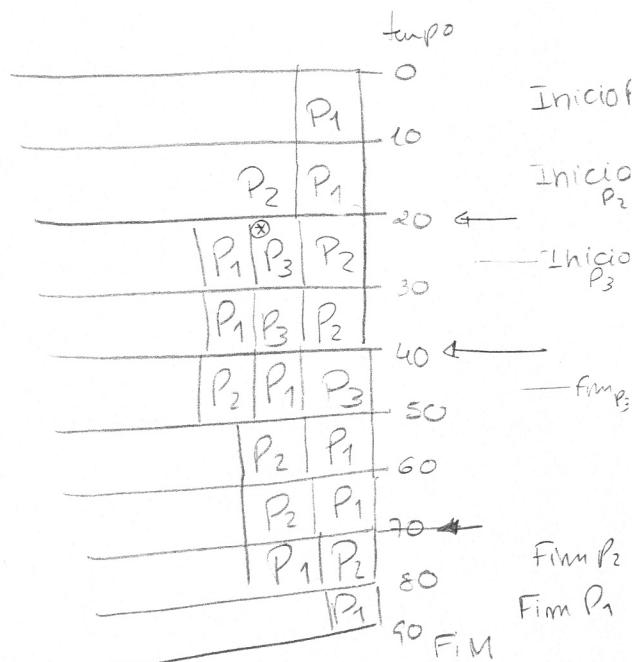
Feedback multi fila - feed back multi Queue (D)

- Vários R.R.
- crescimento do quantum exponencial.



Exercício Round Robin

	CPU	Ts	Tch	$Q = 20$
P ₁	50	0		
P ₂	30	10		
P ₃	10	20		



* A entrada de um novo processo tem prioridade sobre um que entra no mesmo instante para exceder o quantum.
Neste caso, depende do enunciado.

$$T_a = \frac{(90-0)+(80-10)+(50-20)}{3} = \frac{190}{3} = 63,3$$

Nota

Podem ser acrescentados estados, dependendo do S.O.
ex: Para definir prioridades.

Variáveis S.O.

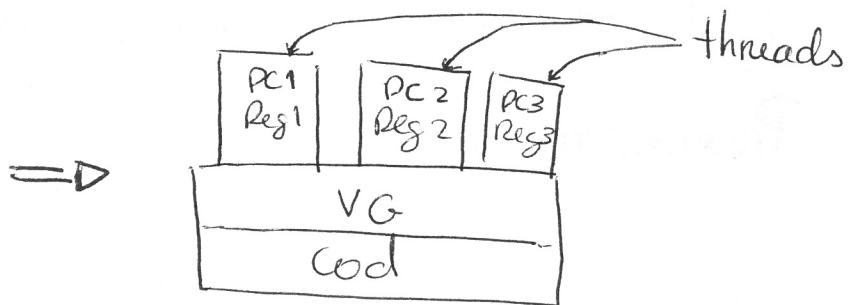
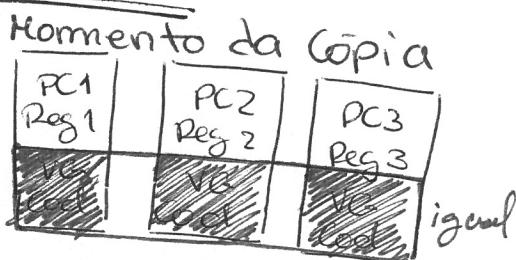
PID, Prioridade, zonas de memória, I/O / ficheiros, P.C., tempo de uso da CPU.

FORK

Faz-se uma cópia da Imagem do Processo comum diferente PID.

O estado filho deve estar em wait enquanto o outro está em R.
Pai devolve PID filho e filho PID = 0.

Threads



há partilha de VG e cod
indivindo mudanças



Denigo:

Programação Concorrente

Criar uma thread é
muito + vantajoso do
que criar um processo
ter de o "matar",
"atirar" uma thread também
mais fácil.

mecanismos:
• semâforos
• locks.

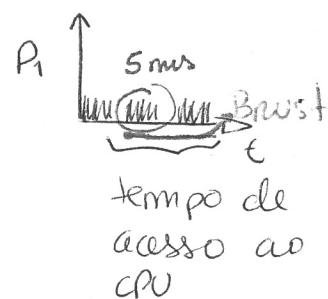
Típos de Threads

	thread Kernel	thread Userlevel
Criar	+ lento	+ rápido
syscall	• não bloqueia todas as threads	• Bloqueia todas as threads
Block		

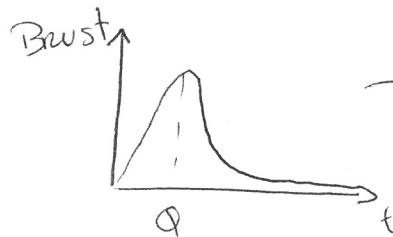
Forma de
conquistar:
lança pedido ao SC
para não bloquear
todas as threads,
quando houver
informações
continua.

Algoritmos

Burst (CPU)



P	T _D
P ₁	10ms
P ₂	
P ₃	



Probabilidade da dimensão de um Burst em relação ao tempo.

Quantum (TMAs) : Tempo que obriga o processo a sair
Função do algoritmo de escalonamento.

A capacidade de se poder ou não interromper um processo é possível através de 2 algoritmos de escalonamento:

1. Preemptivos
2. Não preemptivos.

Preemptivos

Sistemas Operativos

Frequências: 14/3, 11/4, 22/5 (não têm nota mínima)

Nota mínima Global Técnica - $\geq 9,5$
Prática - ≥ 8

Processos

Processos clássicos / Processos leves ou threads.
Escalonamento (CPU)

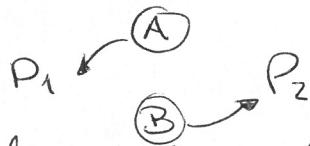
Deadlock

Sistemas de ficheiros

Memória

Input / Output

Deadlock: Os processos param, por conflito de recursos, indefinidamente.

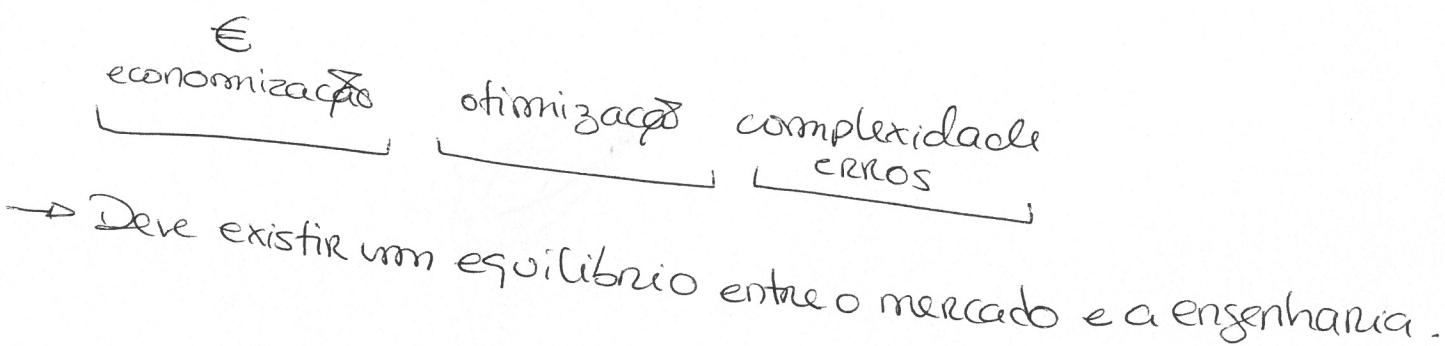


- Esta situação pode resolverse "deitando abaixo" e reiniiciando um processo, de modo a libertar os recursos.

Escalonamento: Dá prioridade; Permite que processos ocorram alternadamente.

sistemas de ficheiros: blocos ou páginas de memória, associados a disco.

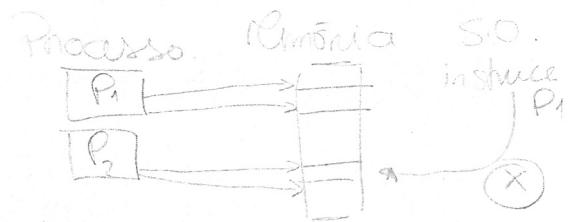
Compromisso:



Personal Computers

S.O. 286 → Dual Mode Operative

- User
- Kernel



Só consulta hardware e as posições de cada processo para que haja proteção entre 2 processos.

A informação dos processos (índice e fim) são divididas pelo S.O.;

Para mexer nos processos (índice inicio e fim) só é possível em Kernel mode, por isso, não é possível guardar inf. referente a P_1 no endereço local da memória a não ser no estabelecido a P_1 .

Existe uma instrução que permite passar de $K \rightarrow U$ mas não de $U \rightarrow K$. Utiliza-se outro mecanismo:

- Uma função que gera interrupção, congelamento do processo, para que se possa passar a outro processo. Exemplo: timers.

∴ Isto é só possível acontecer ($U \rightarrow K$) se existir Dual Mode Operative.



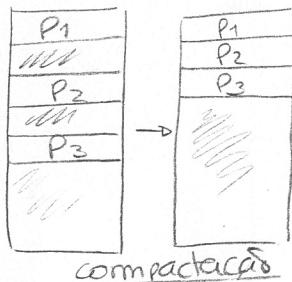
Tempo Real

- Resposta SO

→ Restrições temporais $T_{\text{resp.}} < \underline{T}$?

- controle absoluto de todos os programas.
- garantia de tempo

Exemplo



• demonstrado

- O Sistema tem de garantir que seja possível abortar a qualquer momento.
- Não é uma prioridade.

Algoritmos

→ Não Preemptivos: FCFS; Prioridades; Processos Periódicos e Processos não Periódicos;
RMS

Deadline mais próxima.

Algoritmo RMS

	T_s	Periodicidade	Relação entre T_s e Periodicidade
P ₁	10	40	25% ①
P ₂	20	80	25% ③
P ₃	20	60	33,3% ②
P ₄	20	40	50% *
P _E	20	40	50% *

Periodicidade

Versão

$$\frac{T_{S1}}{T_1} + \frac{T_{S2}}{T_2} + \dots + \frac{T_{SN}}{T_N} \leq 1 \text{ ou } 100\%$$

Para o RMS deve garantir-se também:

n	$n \times (2^{1/n} - 1)$
1	1 (100%)
2	0,82
3	0,77
4	0,75
∞	$\log(2) = 0,693 \approx 70\%$



$$\frac{4}{24} = \frac{1}{6} = 16,66$$

* 100%

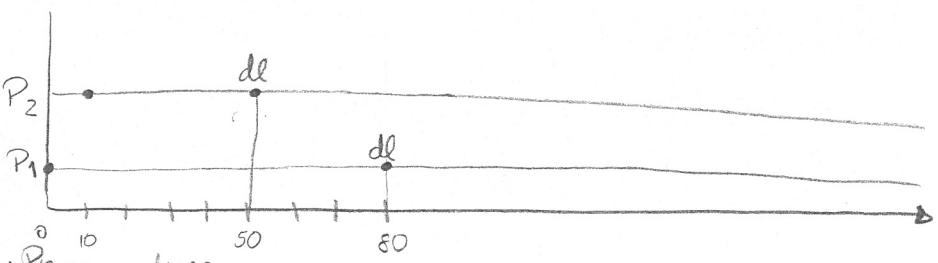
- 83,3% → usado
16,66% → não usado

7,5

Algoritmo Deadline + Próximo

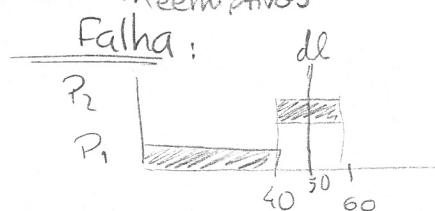
- Não consideramos Processos Periódicos.

	T_s	T_c^{dead}	T_c	T
P_1	40	80	0	80
P_2	20	50	10	40



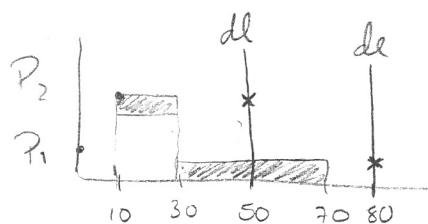
$$T = \frac{40}{80} = 0,5$$

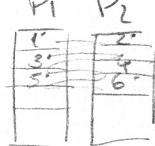
$$T = \frac{20}{40} = 0,5$$



Solução

Algoritmo Deadline + Próximo [c/ tempo morto]



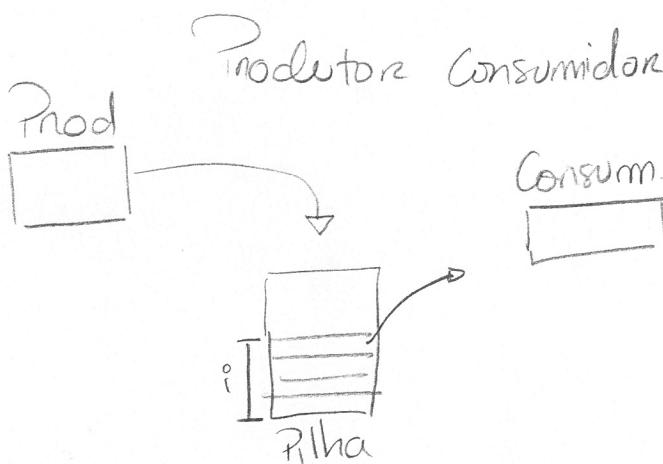
1 CPU $P_1 P_2 \dots P_n$ 

Produtor (insere na pilha)

Consumidor (vai buscar)

Saben quantos elementos na pilha = i

Máximo de elementos da pilha = MAX

• Prod Pilha(1);i++; *

{ term = i;

①

ADD temp + i; → temp ③

{ i = temp;

③

se o processo parar, as variáveis
sao guardadas.Rotina_Pilha();

i--;

② temp₂ = i ;④ temp₂ = temp₂ + i⑥ = temp₂(*) várias instruções.
Deverão ser explícitas
para o CPU.

Alternadamente existem conflitos.

exemplo:

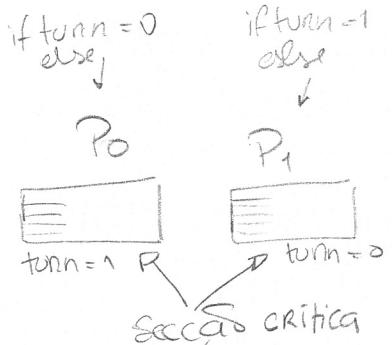
$$t_1 = 5 / t_2 = 5 / t_1 = 6 / t_2 = 4 / i = t_1 = 6 ?? \\ i = t_2 = 4 ??$$

Para Resolver o

Utilizam-se, por exemplo, os semáforos.

Algoritmos de Dijkstra e Peterson.

"Problema da exclusão mútua"



Bloco em que os dois garantem a exclusão mútua, que só lá está um processo de cada vez. Não pode ser partilhado.

E' errado



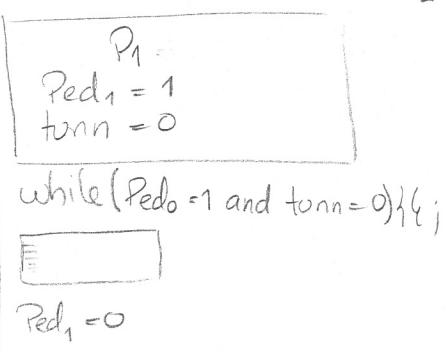
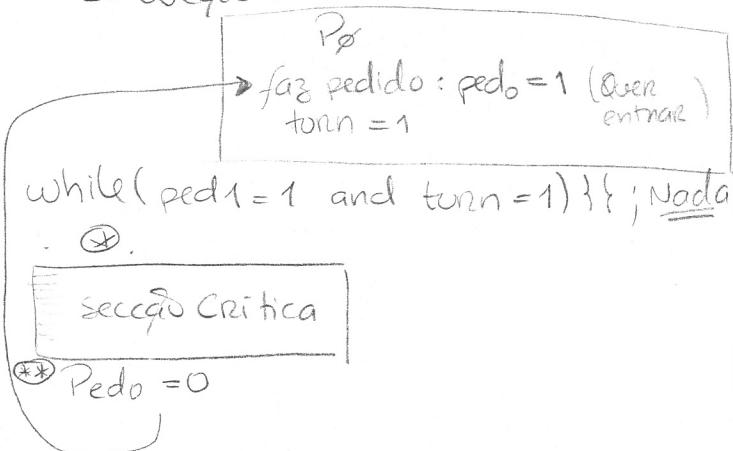
011011...

Se apenas uma quer andar, pône com zero.

Só é bom em alternância absoluta.

Δ Muito confuso

Soluções:



inicializar:
Pedo = 0
Ped1 = 0

Soluções - baixo nível.

→ Dekker: 2 processos
→ Peterson: 2 Processos

→ Test and Set: Verifica variáveis em funções de outras { Hardware.
→ Instruções tipo swap

Soluções - "alto nível"

→ Semáforos: wait / down / P (decrementam) \otimes
 signal / up / V (incrementam) $\otimes\otimes$

wait (semáforos)	signal (semáforos)
{ s--;	} S++;
if (s < 0) Block	if (s ≤ 0) Unblock
{	

E' a inicialização de S que vai definir tudo.

Dekker / Test and set
Peterson / swap

Seção critica: Quando se lê e se altera?

Confinhar com o Pedro que é o único com atenção

Sistemas

Aplicações de Semáforos

Prod.

while (true) {
 wait(E);
 own = 1; // semáforo
 Prod();
 up; // Signal(0);
}

funções ocorr.

Repetidas e
indefinidas vezes

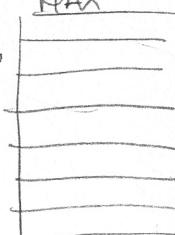
consumidor

while (true) {
 wait(0);
 wait(s);
 Signal(s);
 consum();
 Signal(E);
}

Producir();

MAX

consum



Número de
Processos:

Podem variar o n° Produtores e Consumidores independ.

Restrição: Pilha cheia.

→ Reunso do Produtor: Espaço → Falha

→ Reunso consumidor: Objetos criados.

O consumidor espera quando n existem objetos.

Espesa → starvation.

Signal: função equivalente a incrementar.

Inicialização

No inicio: Espaço = Max → E = Max;
Objetos = 0. → O = 0;

Inicialização dos semáforos

Quando o algoritmo/escalonamento obriga à espera, isto é um starvation.

Podem haver vários produtores/consumidores, mas devem garantir que só vai 1 de cada vez. → wait(s) e signal(s)

Exemplo:

Antes de entrar em Prod(), o wait(s) está a 1. Quando entra em Prod(); passa a 0, bloqueando todos os outros processos, para que +1 não possam aceder à pilha. Quando sai da função, signal(s) passa novamente a 1 permitindo um novo acesso à pilha. Entra q se mexam nas variáveis de controlo da pilha (índice i)

semáforos só operam/funcões wait e signal.

semáforo = critica pode vir da implementação ou Biblioteca (podemos usar)

Várias escritoras e leitores ao mesmo tempo X

Escritoras X Escritor Reader
leitores ✓ / /

wait(s) / /

escritor(i); mem len();

signal(s)

(só fica a última coisa escrita?)

Lector

com $J = 1$ e $i = 0$

Não haveria problema existirem várias escritoras se fosse só uma instrução atómica

wait(J);

Todos os próximos leitores ficam bloqueados aqui.

$i = i + 1$

1º leitor bloqueado; espera pelo escritor.

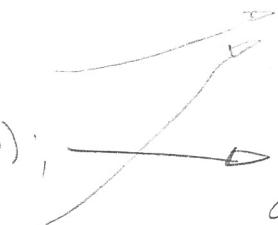
if $i = 1$ then wait(S);

mesma variável, mesmo semáforo

ler();

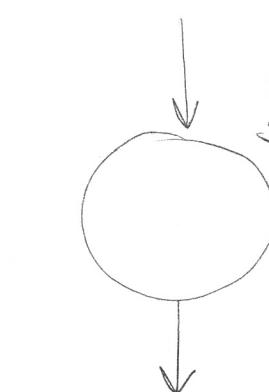
wait(J);

$i = i - 1$



if $i = 0$ then signal(S);

Só o último passa aqui a zero.



Só entra 1 leitura vez
só entra depois do que lá estiver sair.

Sistemas Operativos

• Semaforos

A ordem das atividades no processo de design de software em tempo real depende do tipo de sistema que está a ser desenvolvido, bem como dos requisitos de processo e plataforma.

Pode começar-se com estímulos e processamento associado e decidir sobre as plataformas de hardware ou o contrário.

Processos em tempo real devem ser coordenados e partilhar informações.

Quando um processo que é partilhado é modificado outro que o aceda não deve ser capaz de o alterar em simultâneo.

Para contornar esse problema, deve implementar-se a troca de informação através de um buffer e usar mecanismos de exclusão mútua para controlar o acesso a este buffer.

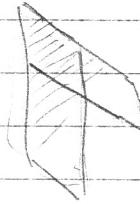
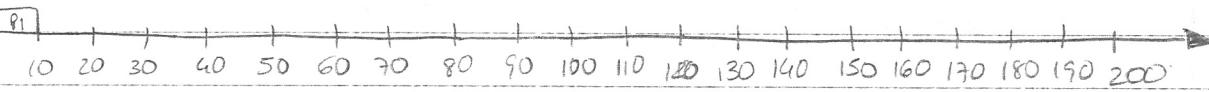
É comum ser implementada uma queue circular.

olmo

RMS

As prioridades são sempre determinadas pelo período.

A periodicidade



$$\text{Cenas} \leq n \times (2^{\binom{n}{2}} - 1)$$

$$3 \times (2^{\binom{3}{2}} - 1)$$

$$\begin{array}{r} 125 - 1 \\ \hline \sqrt{0.25} \end{array}$$

Sistemas Operativos

$x = 1$ semáforo

Aula de
Dúvidas

$$N = N + 1$$

- wait(x)

$$P = P + 1$$

$f(i)$

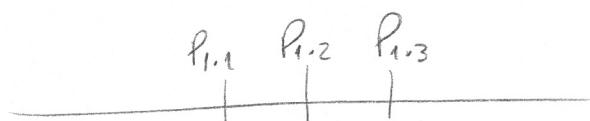
$$P = P - 1$$

signal(x)

$$N = N - 1$$

- P é previsível (entre wait e signal)
- N não pois encontra-se fora de seção crítica e nada protege esta variável. Podem aceder-lhe vários e incrementá-la.

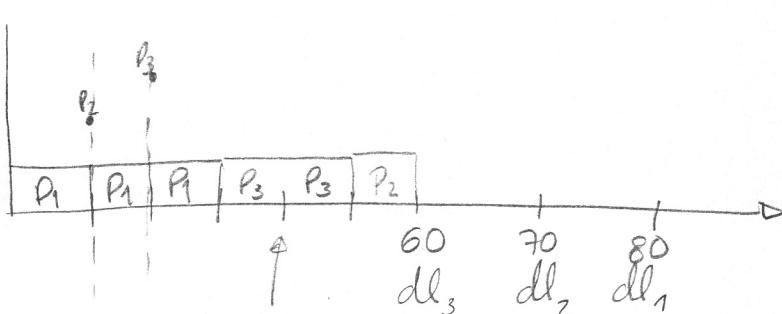
2) RMS: chamadas repetitivas



dead line = inicio da proxima chamada

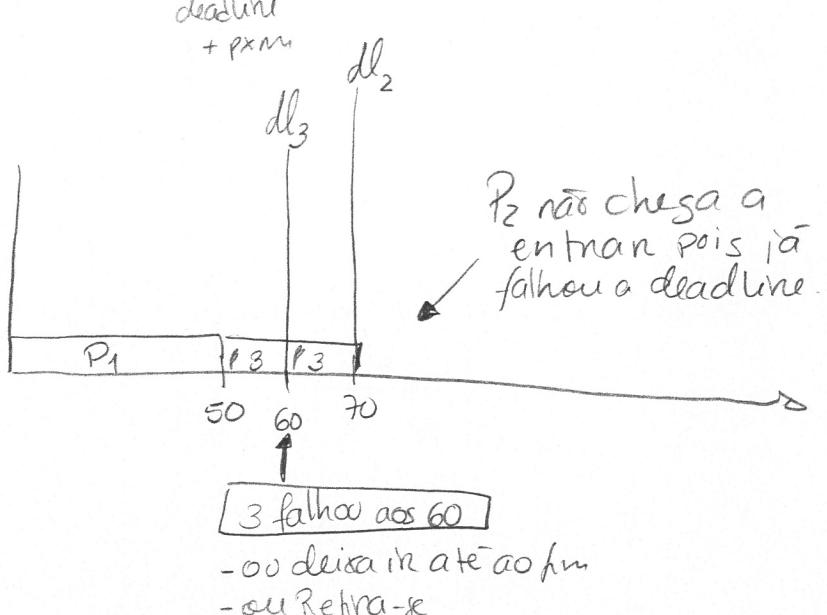
3) Deadline + Próxima (exercício) [sem tempo morto]

	Tc	Ts	Deadline
P ₁	0	30	80
P ₂	10	10	70
P ₃	20	20	60



Variante

	Tc	Ts	deadline
P ₁	0	50	80
P ₂	10	10	70
P ₃	20	20	60

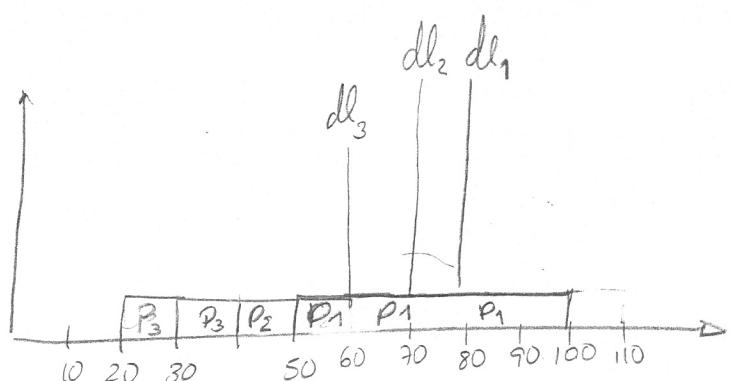


Deadline

Com tempo morto (só sabe deadline. Não conhece o Ts)

espera pelo que tem prioridade

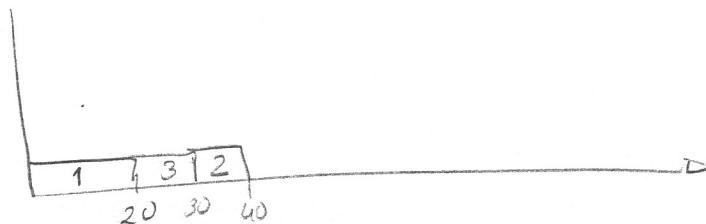
	Tc	Ts	deadline
P1	0	50	80
P2	10	10	70
P3	20	20	60



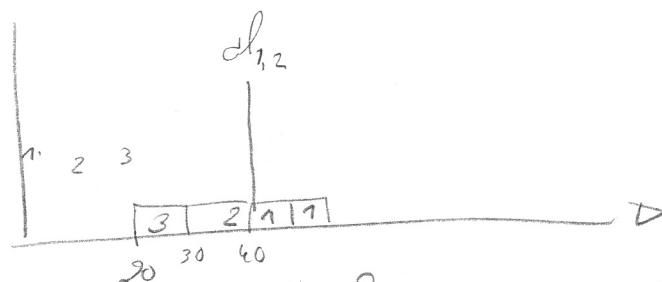
- deadlines iguais = arbitrário.

→ deadline (sem tempo morto)

	Tc	Ts	Deadline
P1	0	20	40
P2	10	10	40
P3	20	10	30



com tempo morto



④ RMS

	Tc	Ts	Período
P1	0	20	80
P2	10	10	40
P3	20	10	30

Ráculos

$$\rightarrow \frac{20}{80} \quad 0,25$$

$$\rightarrow \frac{10}{40} \quad 0,25$$

$$\rightarrow \frac{10}{30} \quad 0,33(3)$$

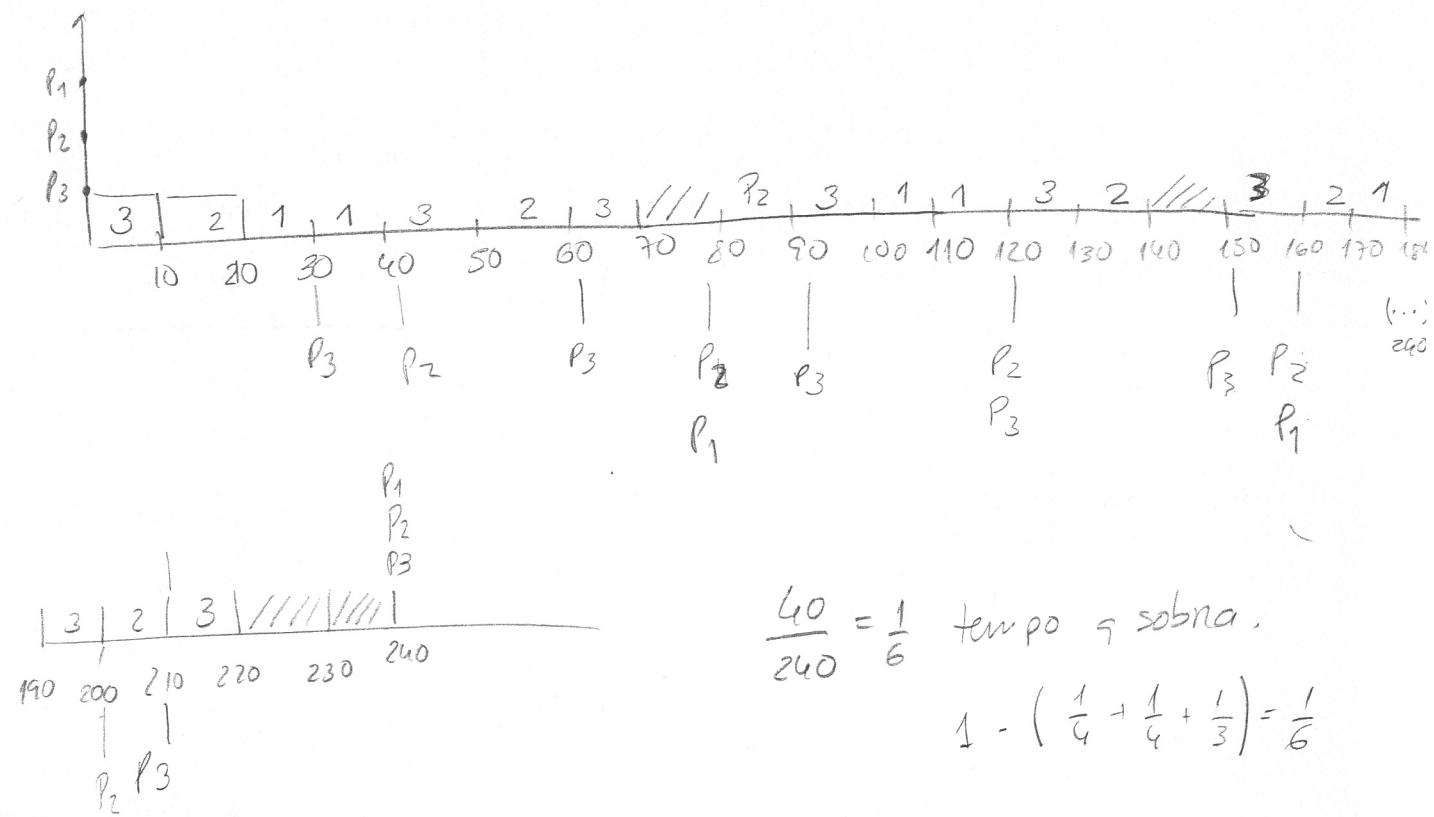
$$n \times \left(2^{\frac{1}{n}} - 1\right)$$

$$3 \times 2^{\frac{1}{3}} - 1 \approx 0,78$$

$0,83(3) \leq 1$
é possível

Aplicação

• continuação do 4)



$$\frac{40}{240} = \frac{1}{6} \text{ tempo é sobra.}$$

$$1 - \left(\frac{1}{4} + \frac{1}{4} + \frac{1}{3} \right) = \frac{1}{6}$$

Deadlock: Nenhum dos processos pode concluir seu aberto.

Starvation: Pelo menos 1 processo muito prejudicado (espera indefinidamente)

deadlock



	P1	P2
wait(A)		wait(B)
bloqueado	wait(B)	→ wait(A)
	;	;
S(A)		
S(B)		

mesmo tempo

Rendez Vous

\textcircled{X}

P₁

signal(y)

wait(x)

thocat();

P₂

Signal(x)

wait(y)

thocat;

Outro Sistema operativo

initializar

4/4/19

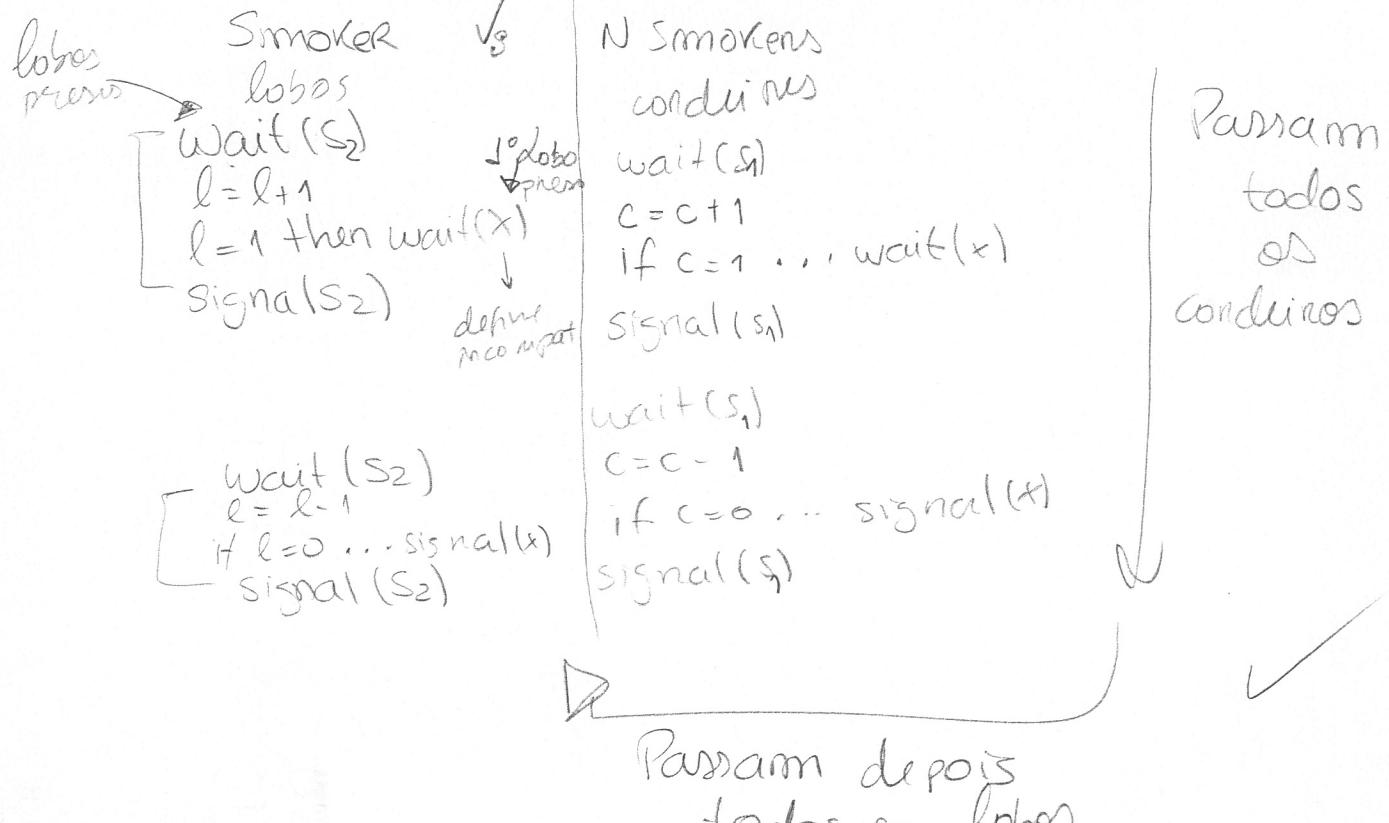
Problema:

Semafóros

$$x = 1 / s_1 = 1 / s_2 = 1$$

Variáveis

$$l = 0 / c = 0$$



Outro Exemplo:



Lobos

W (SL)

$l = l + 1$

if ... wait (SL)

signal (LC)

Condúns

W (SC)

$c = c + 1$

if $c = 1$ then
wait (LC);
wait (CA);

signal (SC)

signal (LC)

signal (CA)

Alfacs

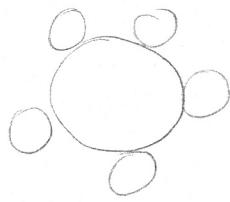
W (SA)

$a = a + 1$

if ... wait (CA)

signal (SA)

Problema dos 5 filósofos.



2 recursos (garfos)

50 existem 5 recursos (garfos)

Proc. 1

garfo-E();

garfo-D();

gena
Deadlock

Proc. 2

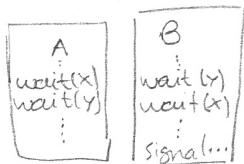
garfo-D();

garfo-E();



AFTER Páscoa
2/5/19

Deadlocks - continuado.



- Kill
 - Round
 - escolher minimizando o impacto
 - Prioridade do Processo
 - Tempo gasto em CPU

- Desalocar Recursos - Real Block

Algoritmo que mata Deadlocks

- Mata processos em deadlock, mas deixa que eles aconteçam primeiro.

- Pode matar das seguintes formas:

- escolhe um processo Random
- escolhe um processo que minimiza o impacto
- por prioridade
- por tempo gasto em CPU

Algoritmo de Detecção de Deadlock

1. Marcar processos (riscar/eliminar) que têm recursos pedidos a zeros, isto é, que se tem a certeza que não estão em deadlock

exemplo: A B C

	A	B	C
P ₁	1	0	2
P ₂	0	0	0
P ₃	0	1	1

← Não está em deadlock

soma das colunas → Recursos Alocados

1 1 3

2. Saber quais são os recursos disponíveis (a partir dos recursos alocados - matriz).

Recursos Totais > Recursos Alocados

Dado no enunciado

exemplo:

Rec.Totais 3 3 3

Rec. Aloc. - $\frac{1 \ 1 \ 3}{2 \ 2 \ 0}$ Rec. Disponíveis

3. Encontrar processos cujos pedidos são satisfeitos com os recursos disponíveis.

exemplo:

Matriz Rec. Pedidos

	A	B	C
P ₁	1	1	0
P ₂	1	1	0
P ₃	1	0	0

Naquele instante

P₁ precisa A, B

P₃ " A

4. Eliminar e atualizar os recursos disponíveis

P₁: 2 2 0

1 0 2

$\frac{3 \ 2 \ 2}{}$ → Recursos

Disponíveis

Atualizados

∴ Como Rec. Disp. = 220 é possível satisfazer ambos os pedidos. Depois de satisfazer os pedidos pendentes até ao fim, os recursos alocados são libertados.

P₃ a seguir, nunca em simultâneo

(falta 1 exemplo)

Recursos Totais 21131

Recursos
Disp.

$$= \frac{21131}{-21130}$$

00001

$$+ 00021$$

00021

$$+ 10110$$

10131

$$+ 11000$$

21131

	Pedidos	Alocados
P ₁	0 0 0 2 1	1 0 1 1 0 X 2°
P ₂	0 0 1 0 1	1 1 0 0 0 X 3°
P ₃	0 0 0 0 1	0 0 0 2 0 X 1°
P ₄	1 0 1 0 1	0 0 0 0 0 X
		2 1 1 3 0

Exemplo 3

Claim (Rec Necessários) Rec Totais 936

Matriz de tudo o que é necessário até ao final do processo		"Faltas"
	3 2 2	2 2 2
1 →	6 1 3	0 0 1
	3 1 4	1 0 3
	4 2 2	4 2 0

Alocados	
1 0 0	◻
6 1 2	X
2 1 1	◻
0 0 2	◻
	925

936
- 925
① 0 1 1 disponível
6 1 2 Aloc R
6 2 3 todos possuem
de satisfazer

- Se consigo chegar ao fim \Rightarrow Estado Seguro.

Algoritmo Banqueiro

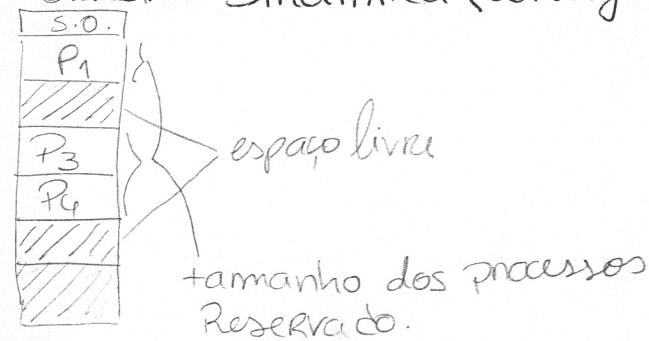
Seguro? Aceita

Não seguro? Recusa

Gestão de memória

- localização
- proteção
- partilha de memória
- organização lógica
- organização física

Memória Dinâmica (contigua)



Na memória fixa todas as divisões estão previamente definidas.

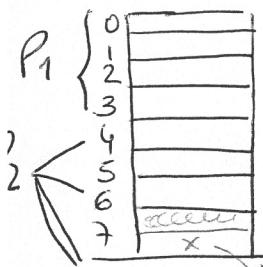
É o sistema operativo que controla os acessos, para que o princípio da proteção seja respeitado.

As zonas de memória dinâmica são livres e podem ser muito distintas.

Fragmentação: Quando o espaço disponível é todo menor que o tamanho do processo.

Desfragmentação: Reorganização do espaço ocupado de modo a aumentar o espaço livre e receber novos ficheiros.

Paginação



Pág.	
0 - 1000	4
1000 - 2000	6
2000 - 3000	8
:	:

Fragmentação interna: há espaço desperdiçado numa página.

Segmentação: Divisão em blocos/segmentos (com características funcionais características).

Tabela de Segmentação

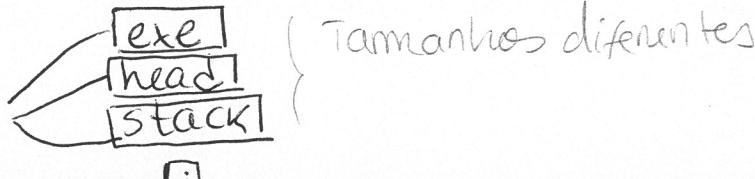
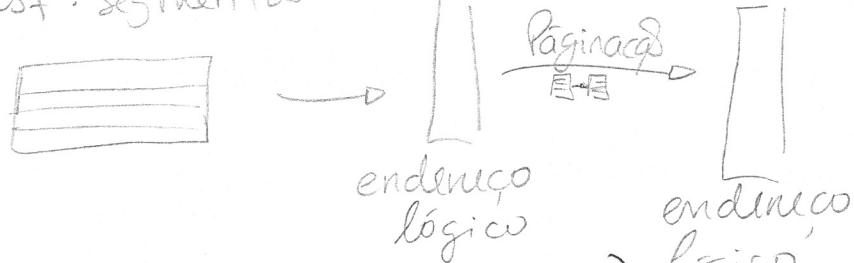


Tabela de Segmentação

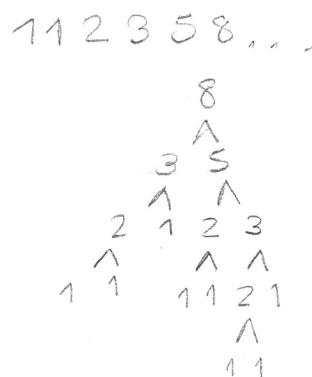
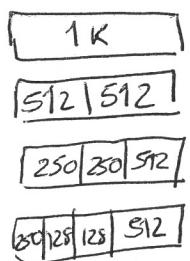
início	dim	
1000	1000	(1000 a 2000)
2000	2000	(2000 a 4000)
4000	3000	(4000 a 7000)
7000		(7000 a ...)

um segmento é uma zona de memória com a mesma característica.

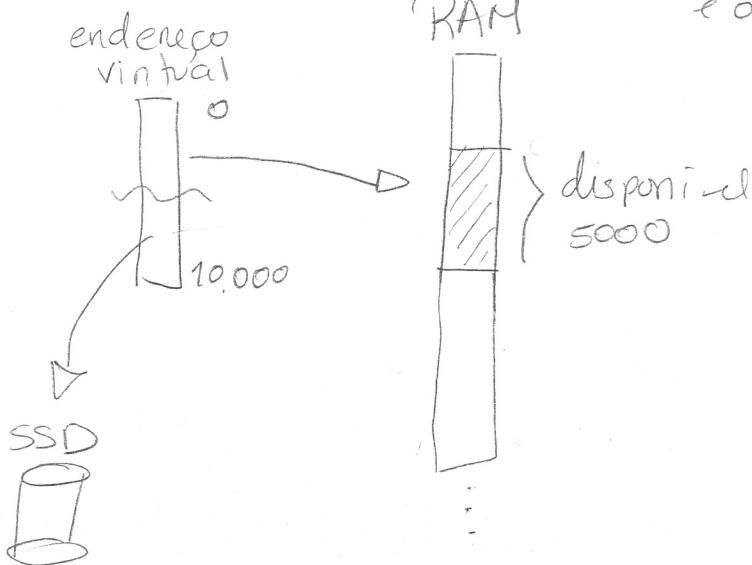
transf. segmentos



Partições de memória (Buddy System)



Memória Virtual guarda o que é possível no espaço disponível e o resto em disco



Exemplo



Algoritmo de
"renovação" LRU

- Next Fit ; • Best Fit ; • Worst Fit ; • First Fit

First Fit

Assim que encontra um bloco livre com tamanho suficiente
ocupá-lo.

Percorre pelo inicio da Mem.

Next Fit

Começa a percorrer a partir de uma posição onde realizou
a última ocupação. Pode começar pelo inicio da Mem.
"circular"

Best Fit

De entre todos os sitios onde o processo cabe, escolhe o com
menos desperdício.

• se desperdiçar muito pouco quase de certeza que não se conseguirá usar

Worst Fit

Desperdiça muito. • como desperdiça muito, o resto ainda está
possivelmente usado.

→ TLB associativa

Tempo de Acesso

está na TLB : TLB + RAM (Hit)

não está TLB : TLB + Níveis RAM + RAM (Miss)

Sistemas Operativos

RAM: 100ms
TLB: 10 ms

Sem TLB
• página 3
 $(3+1) \times 100\text{ms} = 400\text{ms}$
↓
acesso Tab. páginas
Nível da pg

com TLB
(0,98) Hit $10 + 100 = 110\text{ ms}$
(0,02) Miss $10 + (3+1) \times 100 = 410\text{ ms}$

$$0,98 \times 110 + 0,02 \times 410 = T_c / TLB = 116\text{ ms} \quad (\text{Tempo Médio de Acesso})$$

Tempo Ganhado?

Sem TLB = 400 ms

com TLB = 116 ms

Se: TLB₅₁₂ em vez de TLB₂₅₆
99 98

$$\text{Tempo Médio de acesso} = 0,99 \times 110 + 0,01 \times 410$$

FIFO : First in first out

LRU: Least Recently Used

OPT

CLOCK

3 níveis

~~Rotulada~~

	12 14	3 1 5 2 6 7 2	
	FF HF	F F F F F F H	
FIFO	1 1 1 1	3 3 3 2 2 2 2	H = 2
	2 2 2 2	1 1 1 6 6 6	F = 9

$$H = 2$$

F	F	H	F	F	F	F	F	F	H
1	1	1	1	3	3	3	2	2	2
2	2	2	2	1	1	1	1	6	6
.	4	4	4	4	4	5	5	5	7

LRU	1 1 1 1	1 1 1 6 6 6	H = 3
	2 2 2	3 3 3 2 2 2	F = 8
	4	4 4 5 5 5 7 7	

$$F = 8$$

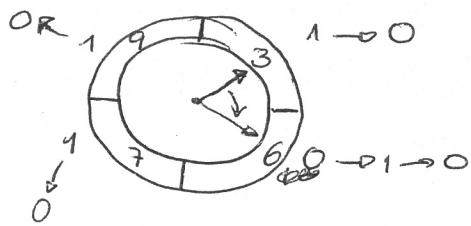
OTIMO	1 1 1 1	1 1 5 6 7 7	
	2 2 2	2 2 2 2 2 2	
	4	3 3 3 3 3 3	

$$H = 3$$

F	F	H	F	F	H	F	H	F	H
①	1	①	1	1	①	5	⑥	⑦	4
②	2	2	2	2	2	2	2	2	2
	④	③	3	3	3	3	3	3	3

1bit clock - Algoritmo

6 páginas



No pior dos casos, todas as páginas estão a 1 e tem de se dar 1 ~~o~~ volta inteira, no entanto vai parar na 1^a posição que é ocupada pois a pos a volta completa é o 1º valor a zero.

2BIT clock - Algoritmo

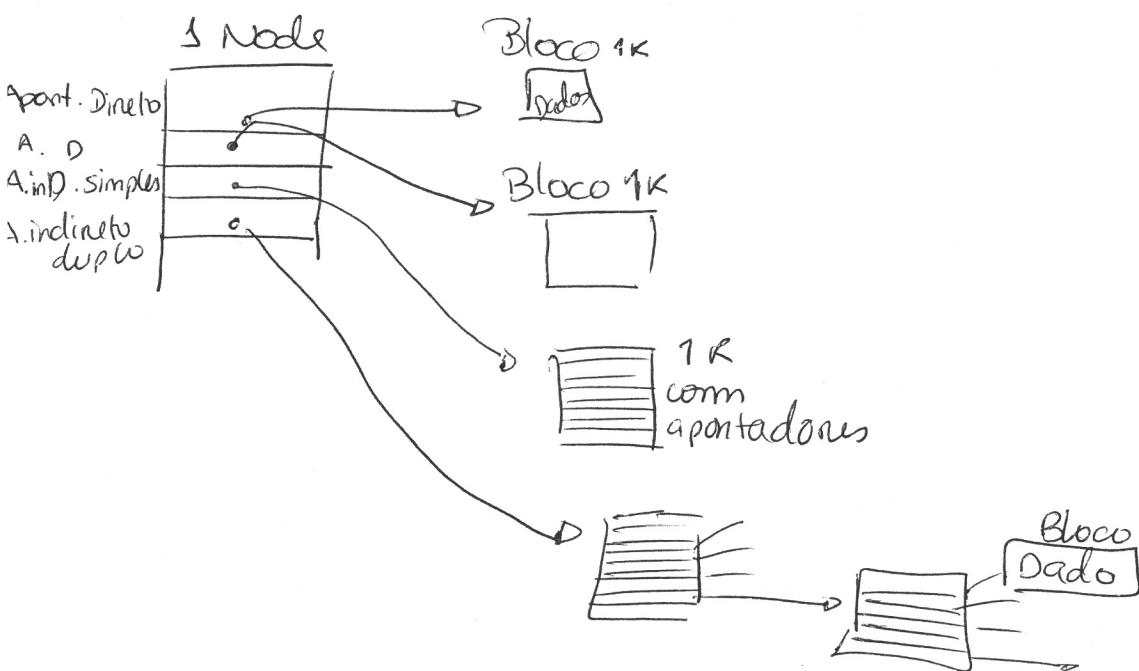
11 10 01 00
escreve

2x volta memória - no pior dos casos

- Sistemas de Ficheiros

- Indexado / Random
- Sequêncial

poupa tempo se a estrutura também for sequêncial.



Bloco 1K

1 Apontador → 2 Bytes

1K + 512 + 256 + 128, 256

Direto → 1K

Indireto → $1K \times 512 = 512K$
simples

Indireto → $1K \times 512 \times 512 = 256M$
Duplo
 $256 \times 1024K$

Indireto

Tripo → 128 G

Diretório

