Exame 2016/2017

1)

a)

→ Exclusão mútua - apenas um processo pode usar o recurso de cada vez

→ Circular wait - existe um conjunto de processos em espera tal que cada um está à espera de recursos que estão detidos por outro processo desse conjunto

→ Hold and wait - o processo fica em posse de pelo menos um recurso enquanto espera por outro recurso

→ No preemption - apenas o processo que tem o recurso o pode libertar

b)

→ Exclusão mútua é satisfeita já que temos um mecanismo de sincronização (os semaforos) que garantem que apenas um filosofo tem o garfo i de cada vez

→ circular wait - os filosofos em espera por um garfo estão à espera que outro filósofo o liberte

→ Hold and wait - o filosofo agarra o garfo da esquerda e depois o da direita. Se o da direita não estiver disponivel ele vai manter o garfo esquerdo em sua posse até obter o direito

→ No preemption - apenas o filosofo que agarrou o garfo o pode liberta.

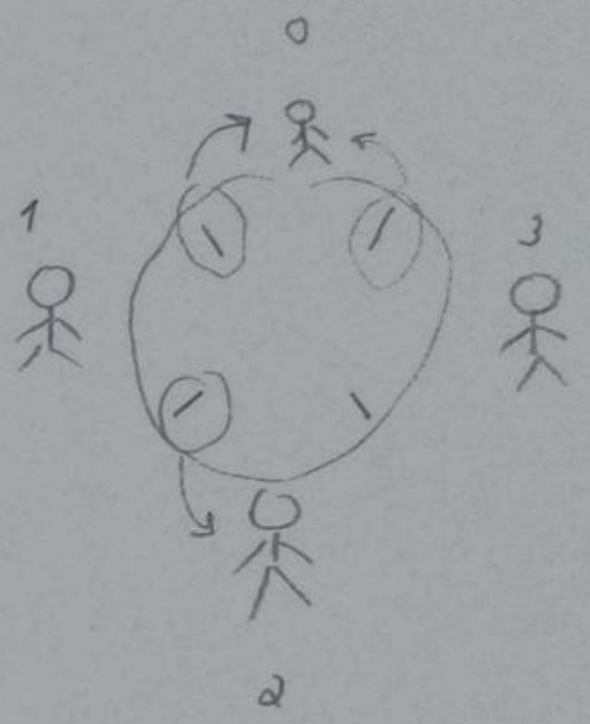
c)

```
While (true)
{
    think ();
    if (id = 0)
        down ( fork [right (id)] );
        down ( fork [left  (id)] );

    else
        dow ( fork [left (id)] );
        down ( fork [right (id)] );

    eat ().
    up ( fork [ left (id)] );
    up ( fork [ right (id)] ) ;
}
```

Anula assim o Circular wait



2)

a) I/O intensivos utilizam o CPU várias mas curtas vezes

CPU intensivos utilizam o CPU poucas vezes mas em períodos longos

I/O intensivos ⇒ P_1 e P_3

CPU intensivos ⇒ P_2

b) A grande diferença entre as duas é que uma é preemptive e a outra não

FCFS é non preemptive ou seja o processo utiliza o CPU durante o tempo que necessiter

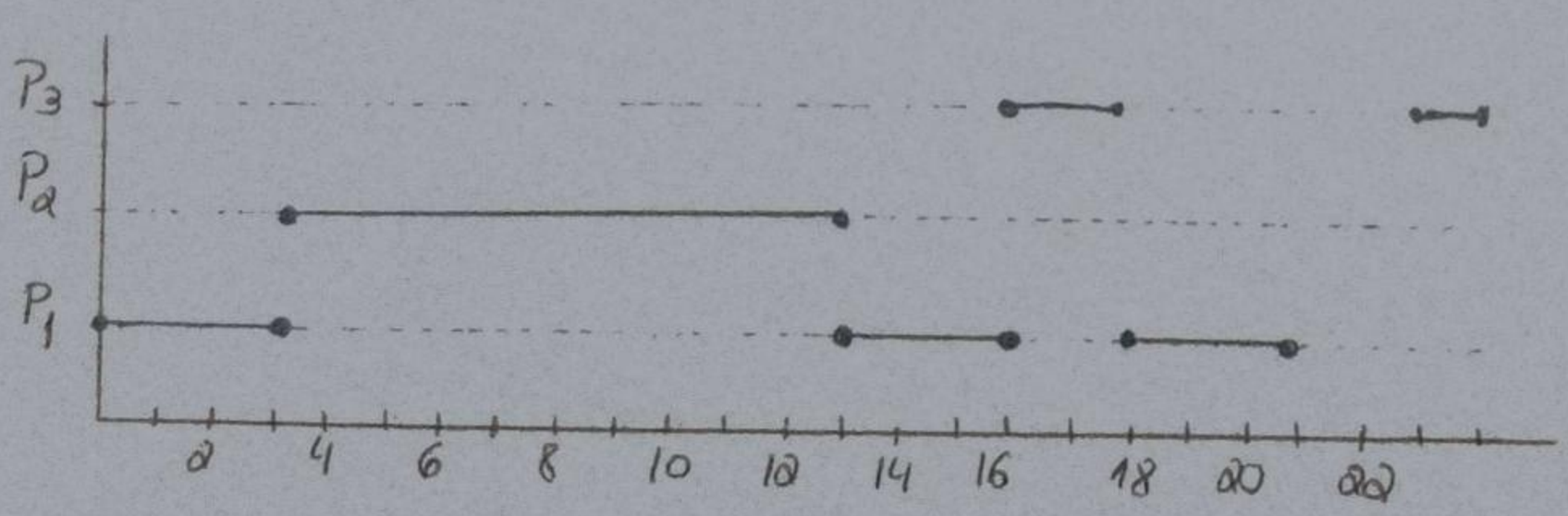
Round Robin é preemptive baseado no clock, ou seja existe um limite máximo de tempo q.e o processo pode estar no CPU se houverem outros em espera

c) Para haver multiprocessamento tem de haver mais de que um processador para permitir que o sistema computacional consegue executar dois ou mais programas ao mesmo tempo (um processador por cada programa em execução simultanea)

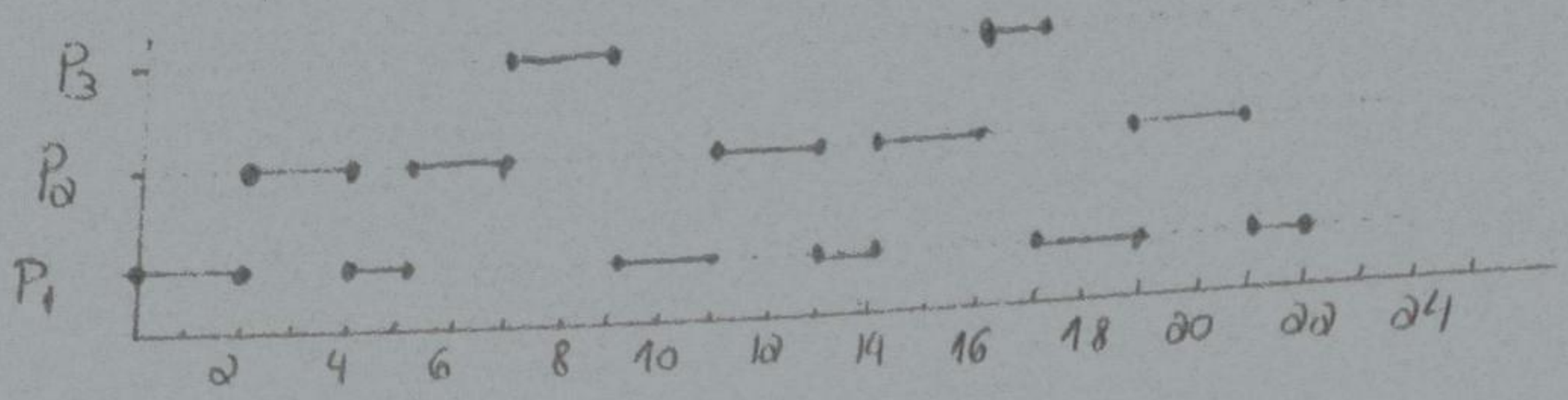
Para haver multiprogramação o sistema computacional deve criar a ilusão que é capaz de executar mais programas, simultaneamente, que o número de processador. Neste conceito sempre que um programa espera por um recurso é colocado outro programa a executar no CPU

d)

FCFS



Round Robin



Ready

✓ P_0, t=2
✓ P_1, t=2
✓ P_2, t=4
✓ P_3, t=5
✓ P_1, t=6
✓ P_2, t=7
✓ P_1, t=11
✓ P_2, t=13
✓ P_3, t=14
✓ P_1, t=15
✓ P_2, t=16
P_1, t=19

Run

P_1, t=0
→ P_0, t=2
P_1, t=4
→ P_2, t=5
P_3, t=7
P_1, t=9
→ P_0, t=11
P_1, t=13
→ P_2, t=14
P_3, t=16
P_1, t=17
P_2, t=19
P_1, t=21

Blocked

P_1, t=5
P_3, t=9
P_1, t=14

3)

a) O processo filho executa o case 0 e o pai o default. Isto acontece já que o valor retornado pelo fork() é 0 se for um processo filho e um valor diferente de 0 se for o processo pai

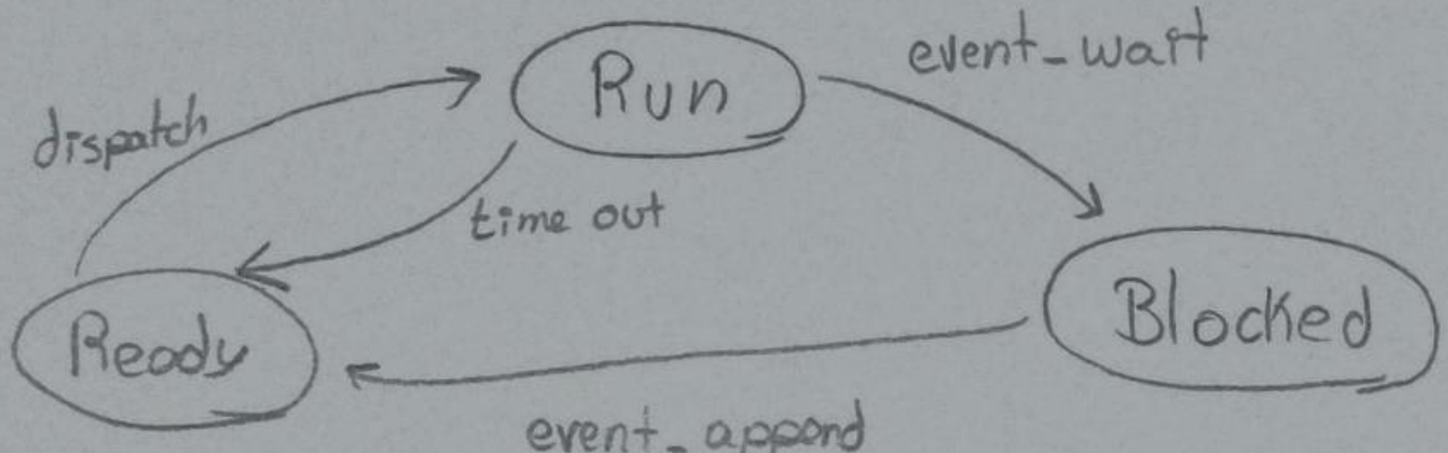
b) A mensagem 0 é sempre a primeira e a mensagem 4 é sempre a ultima já que esse print só é executado quando o processo filho acaba a sua execução. As restantes 3 mensagens podem aparecer por qualquer ordem. Tudo depende do time quantum

(a 0 aparece depois do 1 sempre!)

msg 0
msg 1
msg 3
msg 2
msg 4

msg 0
msg 3
msg 1
msg 2
msg 4

c)



→ Dispatch : o processo que estava a executar saiu do estado run e o dispatch é responsável por colocar um processo da lista dos ready a executar

→ Time_out : o processo estava em execução e esgotou o time quantum. Deixa de estar a executar e passa para a fila dos Ready

→ Event_wait : o processo em execução necessita de um recurso. A sua execução é interrompida e é colocado no estado blocked à espera desse recurso

→ Event_append : o processo já tem o recurso pelo qual estava à espera. Vai ser colocado na lista de prontos a executar

d) Considerando a saida apresentado na alinea 5 o processo pai não passa pelo estado blocked.

A única hipotese de o processo pai ficar no estado blocked era se a sua execução fosse bloqueada pelo wait(NULL), ou seja ele já tá feito o printf mas o processo filho ainda não tinha terminado

Isso não é o que acontece na alinea b. Pelo saida é possivel concluir que o processo filho terminou antes do processo pai faz o print

4)

c) Neste algoritmo são usados o bit Ref e Mod
O ref é atualizado em cada acesso à pagina e o mod é atualizado em cada escrita na pagina.

Periodicamentos o sistema de operação coloca o Ref a 0.

Quando ocorre uma falta de página a pagina que é retirada é a que tiver

Ref=0 Mod=0

Se não houver nenhuma é retirada a que tem

Ref=0 Mod=1

Se não houver nenhuma é retirada a que tem

Ref=1 Mod=0

Se não houver nehuma é retirada a que tem

Ref=1 Mod=1

4. Considere um sistema de memória virtual paginada, onde a cada processo é atribuído um máximo de 10 páginas e em que a memória principal do sistema tem 5 *frames*. Considere ainda que um processo (único) executou a seguinte sequência de referencias, em termos de páginas de memória acedidas: 1, 2, 3, 4, 5, 3, 4, 9, 6, 7, 1, 7, 8, 1, 7, 8, 9, 5, 4, 5, 2.

(a) A tabela seguinte representa, ao longo do tempo, as páginas residentes nas *frames* de memória. Complete-a considerando que o algoritmo de substituiçao de páginas utilizado é o FIFO. Preencha apenas as células da tabela quando há mudança de página.

	1	2	3	4	5	3	4	9	6	7	1	7	8	1	7	8	9	5	4	5	2
F5					5								8								
F4				4							1										
F3			3							7											2
F2		2							6										4		
F1	1							9										5			

(b) A tabela seguinte representa, ao longo do tempo, as páginas residentes nas *frames* de memória. Complete-a considerando que o algoritmo de substituição de páginas utilizado é o LRU (Least Recently Used). Preencha apenas as células da tabela quando há mudança de página.

	1	2	3	4	5	3	4	9	6	7	1	7	8	1	7	8	9	5	4	5	2
F5					5					7		7			7						2
F4				4			4						8			8					
F3			3			3					1			1					4		
F2		2							6									5		5	
F1	1							9									9				

(c) O algoritmo LRU tem um custo de implementação elevado e é pouco eficiente. Uma aproximação menos exigente e relativamente eficiente é o algoritmo NRU (Not Recently Used). Descreva o princípio de funcionamento deste algoritmo.

.....