1) Resolução Aula Dúvidas

a)

i) Se todos fossem 0 nenhum processo avançave

Se queremos que possa sair 1 um B então sem3 tem de ser inicializado com 1

Se colocarmos apenas @ sem3 a 1 não é suficiente só que no fim de imprimir B todos os processo estão bloqueados

Assim também temos de inicializar o sem3 com pelo menos 1.

Nota: Não é garantido que a 1ª letra seja o B, pode ser tb o C, mas a pergunta tb não pede que seja obrigatóriamente um B. Pede que haja a possibilidade de ter

Sem 1 → 0

Sem 2 → 1

Sem 3 → 1

ii) Muitas saídas possíveis

BCACACABB

b) "<u>é habitual</u>" não é obrigatório

→ os semaforos podem ser usados na threads

→ as variáveis de condição <u>não</u> podem ser usados nas process

A grande diferença entre semaforos e variaveis de condição é a persistencia

→ UP incrementa a variável e a variável fica incrementada
Se um semaforo ta a 0 → se fizer um down o processo fica bloqueado até haver um up

→ se fizer um up o processo não fica bloqueado quando se faz um down

→ O signal não é persistente não há uma variável de recurso que guarde o seu valor.

O wait bloqueia sempre!

Assim é necessária associar uma condição à variável de condição

(cond, vcond)

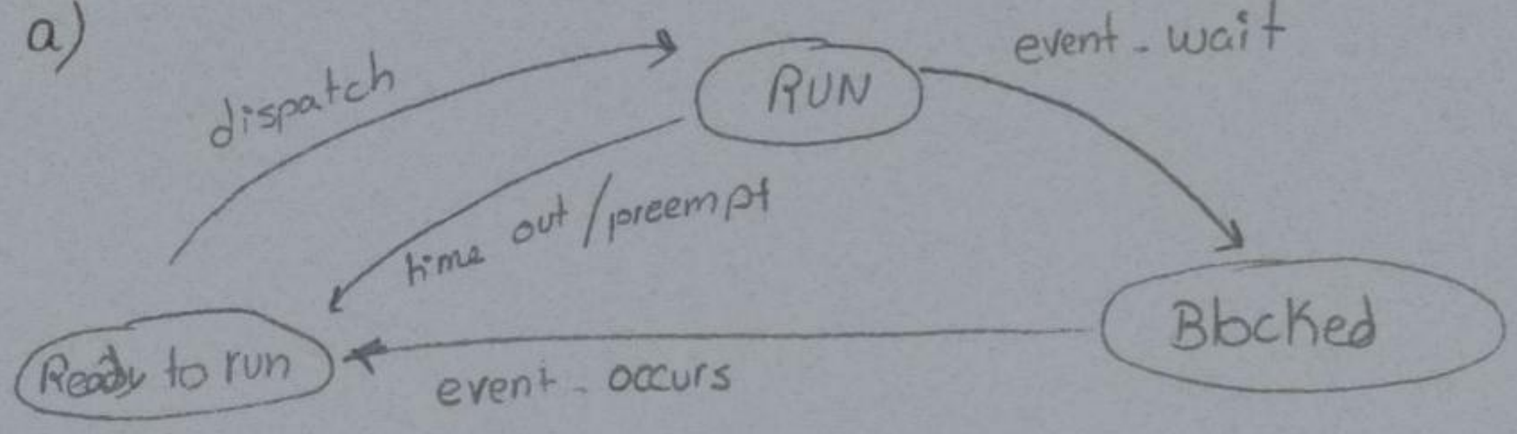
lock (mux) signal (vcond) unlock (mux)	lock (mux) while (!cond) wait (vcond unlock (mux)	⇐ UP	down

c)

```
for (int i=0 ; i<3)
{
    lock (mux)
    while (! cond1)
       wait (vcond1, mux);

    cond1 = false;
    unlock (mux)
    printf ("A") ; fflush (stdout);

    lock (mux)
     cond3 = true;
     signal (vcond3)
    unlock (mux)
}

for (int i=0 ; i<0)
{
    lock (mux)
    while (! cond2)
       wait (vcond2, mux)

    cond2 = false
    unlock (mux)
    printf ("B"); fflush (stdout);
}

for (int i=0, i<3)
{
    lock (mux)
    while (! cond3)
       wait (vcond3, mux);
    cond3 = false;
    unlock (mux)
    print ...
```

2)

a)



event - wait - o processo precisa de aceder a um recurs

event - ocurs - o processo já tem o recurso que necessita e este pronto para executar

dispatch - o anterior processo saiu de run e ~~um~~ o proximo o processo foi selecionado para executer

time - out - o processo em execução esgotou o time quantum e saiu do RUN

preemp - surgiu um processo no ready com mais alto prioridade para ser executado e portanto o que estava a ser executado deixa de ester e vai para o estado ready

b) P_3

P_2

P_1

2 6 10 14 18 22

Ready

P_0, ~~t~~ = 2

P_1, ~~t~~ = 3

P_3, ~~t~~ = 4

P_2, ~~t~~ = 6

P_1 t = 8

P_3, t = 10.5

P_1 t = 15

Run

P_1, t = 0

P_2, t = 3

P_1, t = 6

P_3, t = 7

P_0, t = 10

P_1, t = 12

P_3, t = 15

P_1, t = 18

P_3, t = 19

Blocked

P_1, t = 7

P_3, t = 10

c)

Ready

P_0, t=2
P_3, t=4
P_1, t=4,5
P_3, t=10,2
P_3, t=18,2

RUN

P_1, t=0
P_0, t=4
P_3, t=9
P_1, t=12
P_3, t=16
P_3, t=19,2

Blocked

P_1, t=4
P_3, t=12
P_3, t=19

TERMINATED

P_0, t=9
P_1, t=16
P_3, t=22,2

turnaround

$P_1 = 16$

$P_2 = 9 - 2 = 7$

$P_3 = 22,2 - 4 = 18,2$

3)

a) Numa arquitetura de partições fixas o espaço disponivel na memória principal é dividio num conjunto fixo de partições que podem ter tamanhos diferentes. As grandes vantagem é que são simples de implementar e são eficientes. As desvantagens estão relacionadas com a grande fragmentação interna da memória e são de área muito especifica.

Por outro lado numa arquitetura de partições variáveis é considerada toda a memória disponivel (considerado um único bloco) e vai-se reservando partições de tamanho suficiente para o espaço de endereçamento de cada processo.
As grandes vantagens são a implementação de baixa complexidade e a generalidade na sua aplicação. Como desvantagens a fragmentação externa da memória principal e a pouca eficiencia

Numa arquitetura real há uma relação biunivoca entre o espaço de endereçamento lógico e fisico → ou está todo na memória principal ou todo na área de swapp

b) i) Registo base contem o endereço do inicio do espaço de endereçamento de um processo na memória principal (soma K para obter a posição no espaço de endereçamento na memória)

Registo limite contem o tamanho do espaço de endereçamento de um processo na memória principal (para não ir a situtes fora do espaço de endereçamento do processo

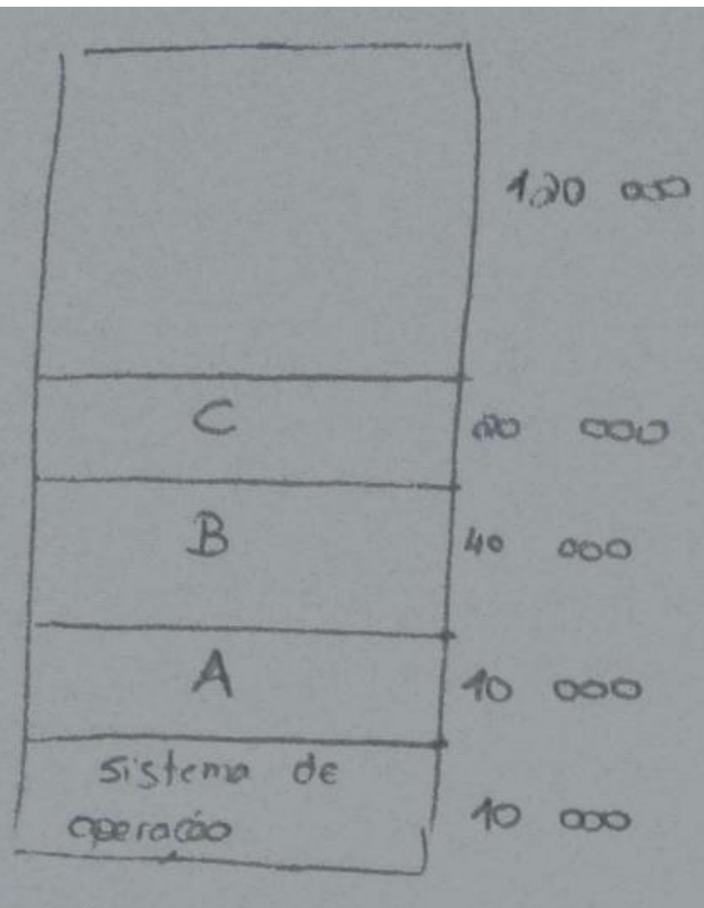
ii) Estes registo são alterados na operação de dispatch

iii) Primeiro o endereço lógico é comparado com o registo limite:

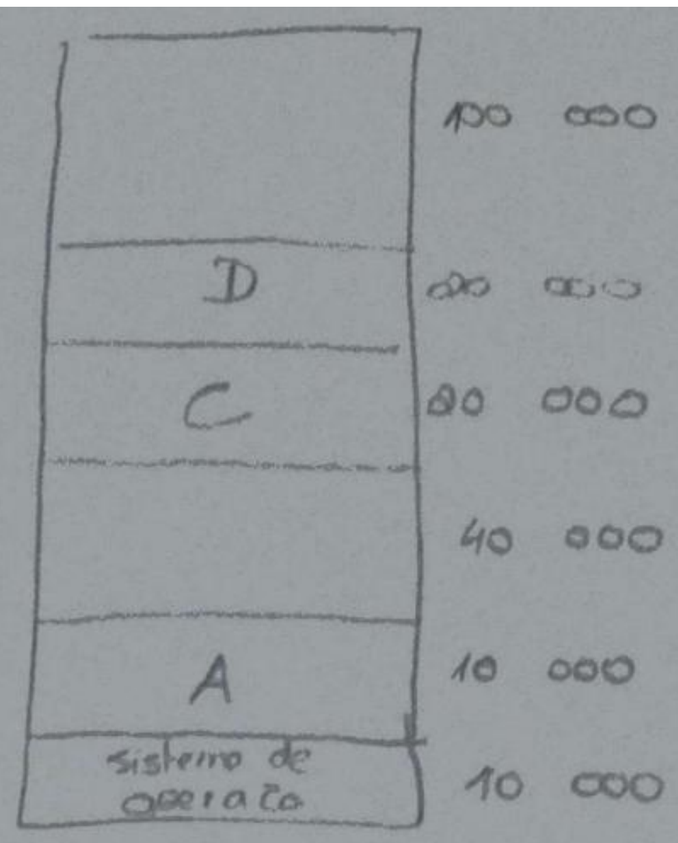
→ Se for menor o endereço corresponde a um endereço do espaço de endereçamento desse processo e de seguida o endereço lógico será somado ao registo base obtendo-se assim o endereço fisico

→ Se for maior ou igual o endereço não corresponde a um endereço do espaço de endereçamento desse processo e é gerada uma execção com erro de endereço

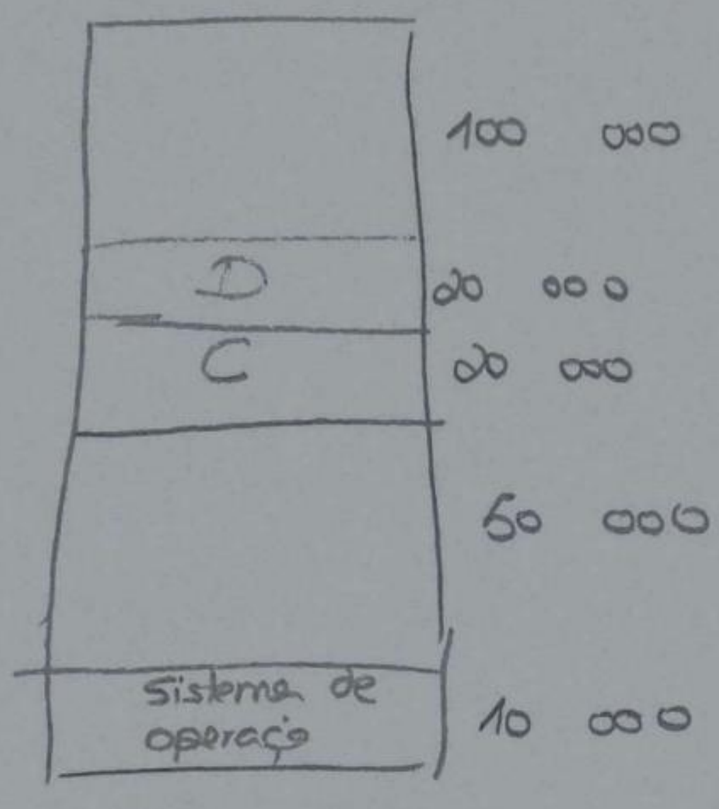
c)



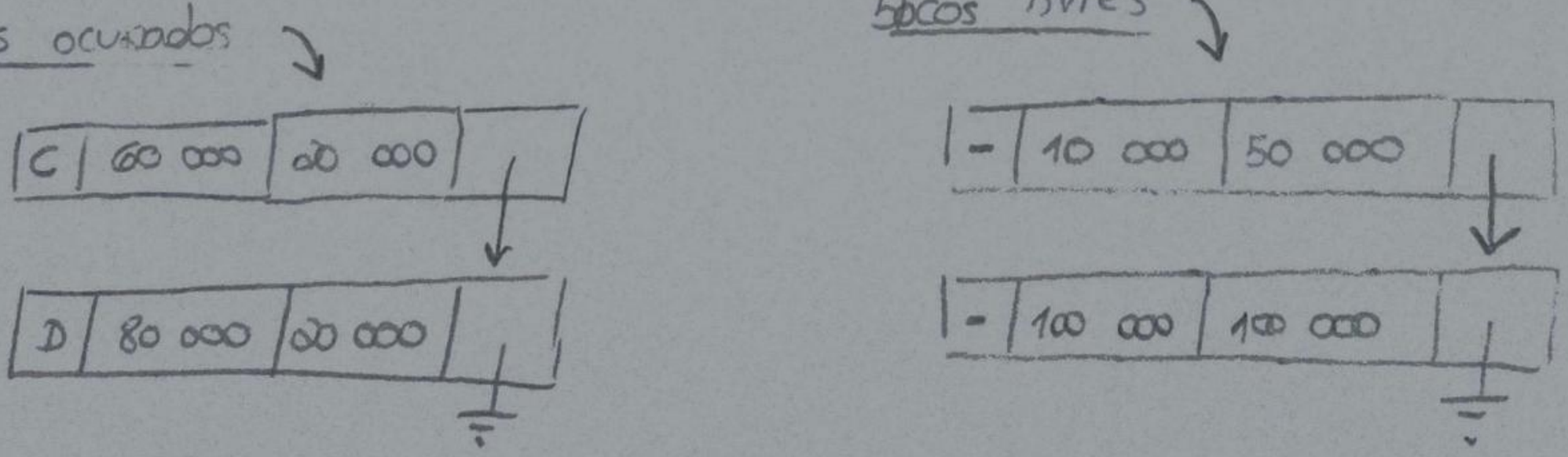
Ponto 1, 2 e 3



Ponto 4 e 5



Se pedisse a estrutura de dados



4)

a) Deadlock avoidance já que o que é feito é uma monitorização constante e o recurso só é atribuido se isso não colocar o sistema num estado unsafe

b)

Se for atribuido o recurso R_1 ao processo P_4 este terá todos os recursos necessário para terminar a sua execução

Quando este terminar os recursos disponiveis são

R_1	R_2	R_3
2	2	1

De seguida podem ser dados 2 recursos R_1 ao processo P_1. Assim este terá todos os recursos necessário para terminar a sua execução

Quando este terminar os recursos disponiveis são

R_1	R_2	R_3
5	3	3

Neste ponto o sistema poderá dar todos os recursos por adquir ao processo P_2 e P_3 ,

c)

Se for atribuido um recurso R_2 a P_3 os recursos disponiveis ficam

R_1	R_2	R_3
1	0	1

De seguida poderá ser atribido a P_4 o recurso R_1 Este processo poderá assim terminar a sua execução. Após isso os recursos disponiveis ficam.

R_1	R_2	R_3
2	1	1

De seguida poderá ser atribuido a P_1 2 recursos R_1.
Este processo poderá assim terminar a sua execução. Após isso os recursos disponiveis ficam

R_1	R_2	R_3
5	2	3

Neste ponto todos os recursos pedidos por P_2 e P_3 poderão ser atribuidos e estes processos terão todos os recursos necessários para terminar.

Como temos uma sequencia de execução o sistema pode dar ao processo P3 um recurso R_2 imediatamente