Grupo A: Gestão de Memória (10 Valores)

1. Especifique quatro objectivos dum sistema de Gestão de Memória dum Sistema Operativo.

Os 5 Topicos
Gestão e Organização Logica
Gestão e Organização Físico
Relocalização de Memoria
Proteção
Partilha

2. Qual são os mecanismos dum sistema operativo que permite que vários utilizadores usem o mesmo programa carregado em memória? Explique.

Usando tabelas de paginas – paginas de diferentes processos podem apontar para a mesma moldura física
Usando tabelas de segmentos – segmentos virtuais de diferentes processos podem apontar para o mesmo segmento de memoria física

3. "Um sistema de memória paginada que disponha duma memória cache para o conjunto de trabalho (*working set*) dum processo é até três vezes mais rápido do que um sistema que utilize apenas processos cujos endereços de instrução e dados são sempre endereços verdadeiros de memória física". Explique porque razão esta frase está incorrecta.

Esta frase é errado. Um sistema de memória paginada devia ser no pior de hipotesis "duas vezes mais lente" do que um sistema que utilize endereços fisicos visto que tem de fazer tradução de endereços virtuais -> fisicas.
Razão : Endereço Virtual /sistema de paginação → aceder a memoria duas vezes (uma vez para fazer lookup na tabela da moldura e uma segunda vez para aceder dados/instrução)
Um sistema cache pode ajudar no sentido que aceder uma tabela em cache é mais rapido do que em RAM – mas gaste tempo e ainda temos de fazer a operação aritmetica de tradução (pelo MMU)

Resposta do aluno : Mem cache está localizado na memeoria secundaria. !!!!!!!

4. Considere um sistema de gestão de memória com locação contígua de memória com partições múltiplas. Dadas partições de memória disponíveis de 50K, 300K, 200K e 100K (nesta ordem) como é que o SO colocará os processos com 170K, 120K, 140K,80K, 50K (nesta ordem) usando os algoritmos de (i) Worst-Fit (Pior Ajuste)e (ii) Best-Fit (Melhor Ajuste).

	50	300	200	100
170		130		
120			80	
140 Tem de Especificar o que acontece nesta situação – aqui Esperar				
80		50		
50				50

	50	300	200	100
170			30	
120		180		
140		40		
80				20
50	0			

5. Um sistema de memória virtual tem um tamanho de página de 16 palavras, 5 páginas virtuais e 4 páginas físicas. A tabela de páginas está no estado apresentado em baixo. Um endereço virtual neste sistema tem 7 bits, sendo que os primeiros 3 bits indicam o número de página.

Página virtual	Página física	*Válido/Inválido*
0	2	*1*
1		*0*
2	1	*1*
3	0	*1*
4	3	*1*

Para os seguintes endereços (i) diga se o resultado é uma page hit/miss (falha/sucesso) ou é uma interrupção/trap para o sistema operativo e (ii) calcule o endereço físico quando apropriado - em valor decimal

(a) 001 0111	→1,7 →miss	Calcular Enderçeo é ERRADO
(b) 100 1001	→ 4,9 → hit	→ 3*16+9=57
(c) 011 0100	→ 3,4 hit	→ 4
(d) 111 0011	→ 7,3 trap	Calcular Enderçeo é ERRADO
(e) 000 1001	→ 0,9 hit	→ 2*16+9 = 41
(f) 010 0001	→2,1 hit	→ 16+1 = 17

6. Num sistema de memória virtual paginada, quantas faltas de página aconteceriam usando os algoritmos de substituição de página "FIFO" (*First in First Out*) e "Optimal" com a seguinte *string* de referência: 1, 2, 3, 4, 1, 2, com apenas três molduras de memória física.

1	**2**	**3**	**4**	**1**	**2**
1	**1**	**1**	**4**	**4**	**4**
	2	**2**	**2**	**1**	**1**
		3	**3**	**3**	**2**

6 falhas/faltas/misses

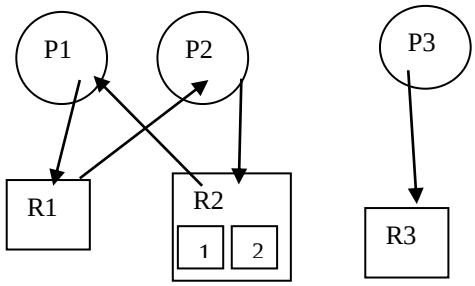
1	**2**	**3**	**4**	**1**	**2**
1	**1**	**1**	**1**	**1**	**1**
	2	**2**	**2**	**2**	**2**
		3	**4**	**4**	**4**

4 falhas/faltas/misses

Grupo B: Threads, Concorrência e Sincronização (10 valores)

7. Considere a seguinte situação com 3 processos (P1,P2 e P3) e 3 recursos (R1,R2 e R3) - os recursos R1 e **R3** tem apenas uma instancia mas o R2 tem duas instâncias do mesmo recurso.
   Ao processo P1 está atribuído uma das instâncias do recurso R2 e está a espera do R1. R1 está atribuído ao P2. O processo P2 está a espera da atribuição duma instância do R2 e o P3 está à espera do recurso R3.

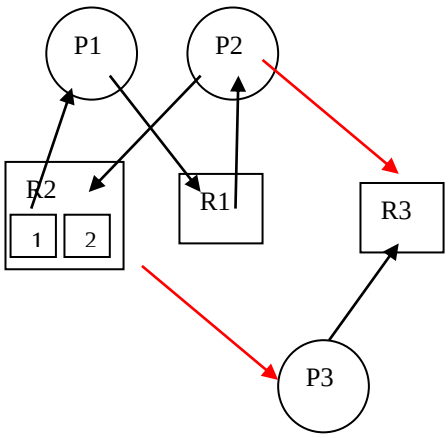
   a) Desenhe o grafo de atribuição/alocações de recursos para esta situação.

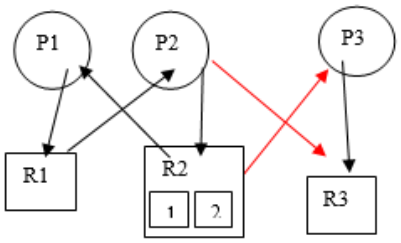


   b) Existe uma situação de bloqueio mútua (*Deadlock*) entre os processos ou não? Justifique.

Existe ciclo no grafo. Mas um ciclo é paneas uma condição necessaria mas não é suficiente para detetar Deadlock (neste caso de instancias multiplas).Analisando o grafo o não há deadlock visto que se o SO atribuir a instância livre de R2 ao processo P2 o P2 poderia assim executar e terminar libertando assim o R1 para o processo P1

   c) Considere o seguinte: o sistema evolve da situação descrita em cima e o P3 pede e é atribuído uma instância do recurso R2 e P2 pede o recurso R3. Desenhe o grafo actualizado de atribuição/alocações de recursos e explicar se agora há ou não bloqueio mútua (*Deadlock*) e/ou em que condição poderá ocorrer.





Repare que o processo de adquirir um recurso é sempre "pedir" e depois "atribuir"

Ainda não há DeadLock. R3 pode atribuido a p3 que assim consegue terminar e libertar R2 para P2 terminar.

Deadlock - depende como o recurso R3 será atribuído. Se for atribuído ao processo P2 haverá deadlock.

Nota que suponhamos que os recursos são do tipo .. ver apontamentos … "**Não-preemptivas"**

8. Considere o seguinte programa.

```
int x=0;
void *maisx(void *args)    { int add = *(int*)args ; x=x+add; }
int main() {
   pthread_t  th[3];
   int ids[3]={1,2,3};
   for (int i=0; i<3; i++)
       pthread_create( &th[i], NULL, maisx, &ids[i]);
   for (int i=0; i<3; i++)
        pthread_join( th[i], NULL );
   printf("x=%d\n",x);
}
```

a) Quais são os outputs possíveis do programa?

x=1, x=2,x=3,x=4,x=5 ou x=6

b) Usando a sintaxe de Posix *Threads* explique como é que pode usar um trinco logico para garantir que o resultado deste programa seja "x=6"

pthread_mutex_t trinco; //Variável Global
pthread_mutex_init(&trinoc,NULL) //Na função main

```
{ int add = *(int*)args ; x=x+add; }
```

→

```
{
int add = *(int*)args ;
pthread_mutex_lock(&trinco);
x=x+add;
pthread_mutex_unlock(&trinco);
}
```

```
- solução mais efeciente
a linha da declaração e atribuição não precisa de estar dentro
do mutex lock
```

9. Segue-se uma possível solução para garantir a exclusão mútua, progressão e espera limitada no acesso a uma secção crítica. Note que a variável int flag[2]={FALSE,FALSE}; é partilhada entre os dois processos.

Processo 0	*Processo 1*

while (TRUE)
{
flag[0]=TRUE;
while (flag[1]); // não faz nada
<seccao critica>
flag[0]=FALSE;
<restante codigo>
}
	while (TRUE)
	{
	flag[1]=TRUE;
	while (flag[0]) ; // não faz nada
	<seccao critica>
	flag[1]=FALSE;
	<restante codigo>
	}

(a) Explique onde pode ocorrer espera activa (*busy-waiting*) na solução apresentada em cima.

Nos ciclos while .. .. a esperar que outro processo efectua a instrução da saída do protocolo.

(b) Mostre que a solução <u>não é</u> satisfatória.

Pode haver deadlock. i.e falha a progressão e espera limitada.
P.ex
flag[0]=TRUE
context switch
flag[1]=TRUE;

E assim os dois processos ficam para sempre no ciclo while

Repare que não há incormidade da condição de exclusão mutua. Este protoclo grante exclusão muta !!!

(c) Escreva o pseudo-código para os dois processos anteriores usando um semáforo por forma a garantir o acesso exclusivo à secção crítica.

SEMAFORO Inicializado a valor 1

```
while (TRUE)
{
   SEM_WAIT
   <seccao critica>
   SEM_POST
   <restante codigo>
}
```

(d) Explique porque não ocorre espera activa (*busy-waiting*) com a solução dum semáforo.

Semaforos : Faz Utilização do mecanismo de <u>escalonamento</u> do sistema operativo– O processo a esperar (sem_wait) é transferido para o estado "Esperar/Blocked" e assim não gaste CPU fazendo o ciclo de estados "Ready"->"Running"→"Ready" etc

10. Considere o código para três *threads* a executar concorrentemente

Variaveis Globais

sem_t s1,s2

Inicialização

sem_init(&s1,0,0);
sem_init(&s2,0,0);

valor inicial zero

void *a(){
puts("um\n");
puts("dois\n");
sem_post(&s1)
}
	void *b(){
	sem_wait(&s1)
	puts("três\n");
	sem_post(&s2)
	puts("quatro"\n");
	}
		void *c(){
		puts("cinco\n");
		sem_wait(&s2)
		puts("seis\n");
		}

Tomando em conta que é necessário imprimir "três" antes de "seis" e "dois" antes de "três" (e isto são as únicas restrições) reescreva as funções usando um máximo de dois semáforos para sincronizar o output. Deverá usar a sintaxe POSIX e explicar como os semáforos são inicializados.