Número: Nome:

LEIC/LERC – 2009/10
1º Exame de Sistemas Operativos

16 de Janeiro de 2010
Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.
Duração: 2h30m

Grupo I [4,2v]

Considere um Sistema Operativo com um escalonamento com as seguintes características:

- Máquina mono-processador
- Multilista com 4 níveis de prioridade: 1 (menos prioritário) a 4 (mais prioritário)
- Preemptivo
- Cada nível é gerido em FIFO
- Sempre que um processo termina o seu time-slice a sua prioridade é decrementada de uma unidade.
- Sempre que um processo se bloqueia num semáforo ou mutex a sua prioridade é incrementada de 1 unidade (limitada a 4).
- Com semáforos e mutexes com fila de espera gerida em FIFO.

Considere que no sistema existem quatro processos, dois mutexes e um semáforo:

- P1 com prioridade inicial de 4
- P2 prioridade inicial 3
- P3 prioridade inicial 2
- P4 prioridade inicial 1
- Mutex M1, inicialmente Aberto
- Mutex M2, inicialmente Aberto
- Semaforo S1 com o valor 0 na sua variável de controlo

Na tabela seguinte a ordem dos processos representa a sua posição na estrutura de lista FIFO.

Lista escalonador		TS1	TS2	TS3	TS4	TS5	TS6
	Prio 4	P1					
	Prio 3	P2,	P2,P1	P1			
	Prio 2	P3,	P3,	P3,			
	Prio 1	P4	P4	P4			
Mutex M1							
	Variável	Aberto	Fechado	Fechado			
	Lista	Nil	Nil	P2			
Mutex M2							
	Variável	Aberto	Aberto	Fechado			
	Lista	Nil	nil	nil			
Semáforo S1							
	Contador	0	0	0			
	Lista	Nil	nil	nil			

Legenda:
PE é a abreviatura para Processo em Execução.

TSx – indica um timeslice sendo a representação na tabela **a situação no início do timeslice**. Num timeslice os processos podem executar o seu algoritmo mas apenas considerámos as operações de sincronização relevantes. O processo que fica em execução num dado timeslice executa-o até ao fim (caso não se bloqueie entretanto); caso se bloqueie, o timeslice é considerado concluído nesse momento e começa o próximo timeslice (com novo processo em execução).

Considere a evolução SEQUENCIAL do sistema indicada nas alíneas seguintes.

1. [0,7v] O preenchimento inicial da tabela inicial corresponde à seguinte sequência:
   - TS1 – PE executa Fechar(M1)
   - TS2 – PE executa Fechar(M2), seguido de Fechar(M1)
   - TS3 – PE executa Fechar(M2)

Preencha, o estado no início de TS4 (preencha directamente na tabela apresentada acima)**.**

2. [0,7v] Como classificaria esta situação? Justifique.

3. [0,7v] Que sugestão poderia fazer aos programadores destes processos para evitar esta situação. Pode apresentá-la em pseudocódigo. Justifique.

4. [0,7v] Preencha a coluna TS5 da tabela anterior, considerando que o sistema evoluiu da seguinte forma:

- TS4 - PE executa Esperar (S1) (preencha directamente na tabela apresentada acima)

5. [0,7v] Preencha a coluna TS6 da tabela anterior, considerando que o sistema evoluiu da seguinte forma:

- TS5 - PE executa Assinalar (S1) (preencha directamente na tabela apresentada acima)

6. [0,7v] Que processo se irá executar imediatamente depois desta chamada sistema? Justifique com base nas propriedades do escalonador.

Grupo II [4v]

Em 1971, S. Patil enunciou o problema dos "Fumadores de Cigarros", definido de seguida.
Um cigarro precisa de 3 ingredientes para ser fumado: tabaco, papel e fósforo. Sentados numa mesa há 3 fumadores, cada um dos quais tem um fornecimento *ilimitado* de um dos ingredientes: o fumador 0 tem tabaco, o fumador 1 tem papel, o fumador 2 tem fósforos.
Há um 4º elemento, um árbitro não-fumador que, quando acha apropriado, selecciona (segundo algum critério) um fumador e dá-lhe autorização para fumar. Após o árbitro seleccionar um fumador para fumar, cada fumador (incluindo o escolhido) coloca um (e apenas um) item do seu ingrediente na mesa. Assim que todos os ingredientes estejam na mesa, o fumador escolhido retira os ingredientes da mesa, prepara o cigarro e fuma-o por um tempo indeterminado.
Existem as seguintes limitações:

- O árbitro não começa a decidir qual o próximo fumador antes da mesa estar vazia.
- Um fumador só é capaz de colocar o seu ingrediente na mesa (em resposta a uma decisão do árbitro) quando não está a fumar.

```
1. boolean ingredienteNaMesa[3] = {false, false, false};
2. int fumadorEscolhido = -1;

3. fumador(int i) {
4.   while (true) {
5.     while (fumadorEscolhido==-1 || ingredienteNaMesa[i]);
6.     ingredienteNaMesa[i] = true;
7.     if (fumadorEscolhido == i) {
8.       while (!(ingredienteNaMesa[(i+1)%3] && ingredienteNaMesa[(i+2)%3]));
9.       for (k=0 to 2)
10.        ingredienteNaMesa[k] = false;
11.      fumadorEscolhido = -1;
12.      fumaCigarro();
13.    }
14.  }
15.}

16.arbitro() {
17.  int f;
18.  while(true) {
19.    while (fumadorEscolhido != -1);
20.    f = pensaEscolherProximoFumador();
21.    fumadorEscolhido = f;
22.  }
23.}
```

1. [0,8v] Para as seguintes porções do código, indique um excerto (frase ou parte de frase) do enunciado que essa linha implemente:
   a. Linha 6 do Fumador

   b. Linhas 8-10 do Fumador

   c. Linha 19 do Árbitro

2. [0,7v] Indique o que modificaria na solução para proteger as secções críticas. Pode recorrer a semáforos e mutex/trincos lógicos. Seja sucinto, indique apenas as alterações ao código.

3. [0,7v] A solução proposta recorre a espera activa. Indique que consequência tal pode ter no desempenho, ilustrando com um exemplo de uma execução.

4. [1,5v] Proponha uma modificação da solução acima que elimine completamente a espera activa (em modo utilizador), o que pelo menos a reduza. Utilize semáforos

5. [0,3v] A sua solução ainda sofre de espera activa? Justifique.

Grupo III [2,7v]

Considere uma arquitectura de memória virtual paginada com endereços virtuais de 16 bits e com páginas de dimensão 256 B.
Assuma que não há qualquer tipo de cache nem TLB.
Os tempos de acesso são os seguintes:

- Leitura/escrita na memória primária: tmem=10
- Tempo para determinar a localização de uma página no disco e para a ler/escrever: tdisco=1000

1. [0,5v] Qual a dimensão (em bytes) do espaço virtual disponível para cada processo?

2. Num dado momento, apenas dois processos, P1 e P2, se executam. Ambos correm o seguinte programa:

```
0 #DEFINE N 128
1 int array[N]; //Assuma que o vector é inicializado a zeros
2
3  main() {
4    int i;
5    int soma=0;
6
7    for (i = 0; i<N; i++)
8      array[i] = array[i] + 1;
9    for (i = 0; i<N; i++)
10     soma += array[i];
11   printf("Soma = %d\n", soma);
12 }
```

Assuma que:
- o tamanho de um *int* é 4 bytes.
- o endereço da posição array[0] é 0x0200.
- existem 512 B em memória primária (2 páginas apenas) que estão reservados para alojar **exclusivamente páginas da região de dados** de ambos os processos (**ignore o resto da memória primária, usada para outros tipos de páginas**).
- inicialmente, esses 512 B estão livres.
- a política de substituição de páginas é FIFO.

Considere que o escalonamento dos dois processos resulta na seguinte execução:

i. P1 começa em execução e executa completamente as N iterações do primeiro ciclo *for* (linhas 7 e 8).
ii. P2 então ganha o processador, executando completamente as N iterações do primeiro ciclo *for* (linhas 7 e 8).
iii. P1 então retoma o processador, executando completamente as N iterações do segundo ciclo *for* (linhas 7 e 8) e terminando.
iv. Finalmente, P2 volta à execução e também termina o seu programa.

a. [0,5v] Qual o resultado impresso no ecrã por cada processo?

b. [1,2v] Assuma que o passo i foi completado. Indique, para cada passo seguinte (ii a iv) quanto tempo foi dispendido nos acessos à variável array. Justifique os resultados.
Para simplificar, nas alíneas seguintes **considere apenas os tempos de acesso relativos aos acessos à variável array** (isto é, ignore o tempo de acesso necessário para acesso ao código e pilha do processo).

t(Passo ii) =	Justificação
t(Passo iii) =	Justificação
t(Passo iv) =	Justificação

c. [0,5v] Considere a sua resposta à alínea b. Uma duração diferente do time-slice teria permitido um desempenho melhor? Justifique com um exemplo.

Grupo IV [3,1v]

1. [0,9v] Considere o seguinte comando em Linux, que copia um ficheiro:

```
cp  /etc/passwd  /tmp/xpto
```

Na tabela seguinte, indique cada bloco do disco que é lido ou escrito em resultado do comando acima, indicando a razão desse acesso a esse bloco. Apresente os acessos a cada bloco por ordem cronológica.
Considere que: (i) o sistema de ficheiros **não usa qualquer tipo de cache**, (ii) as directorias "/", "/etc" e "/tmp" ocupam um bloco apenas (cada uma); (iii) o ficheiro /etc/passwd ocupa 4 blocos; e (iv) o comando completa com sucesso.

Bloco	**Razão do acesso**
Bloco da tabela de inodes que contém o inode de "/"	Obter inode de "/"

2. Se, em alternativa, tivéssemos executado o comando

```
ln  /etc/passwd  /tmp/xpto
```

teríamos criado um *hard link*; ou seja, "/tmp/xpto" seria descrito pelo mesmo inode que "/etc/passwd".

Assuma que o inode de /etc/passwd tem o número 87 e que a respectiva entrada na directoria /etc é a seguinte:

Extrato do conteúdo da directoria "/etc":

…	
"passwd"	inum=87
…	

[0,6v] Quando se cria o hard link entre /etc/passwd e /tmp/xpto, que estrutura(s) de dados em disco são modificadas? Indique as modificações necessárias na(s) estrutura(s) de dado(s) que indicou.

3. Para cada afirmação seguinte, indique se é verdadeira caso consideremos o comando cp e caso consideremos o comando ln. Cada resposta errada desconta 0,1v.
   a. [0,2v x 2] Se um processo P1 alterar o primeiro bloco de /etc/passwd, e posteriormente um processo P2 ler o primeiro bloco de /tmp/xpto, então P2 lerá as alterações feitas por P1.

cp:	ln:

   b. [0,2v x 2] Se P1 abrir e ler /etc/passwd até à posição 1000, e posteriormente P2 abrir e fizer uma leitura de 100, a leitura iniciar-se-á na posição 1000.

cp:	ln:

   c. [0,2v x 2] Ambos os ficheiros podem existir em volumes com sistemas de ficheiros diferentes.

cp:	ln:

   d. [0,2v x 2] Quanto maior o ficheiro original, mais tempo demora o comando a completar.

cp:	ln:

Grupo V [4v]

1. Considere o seguinte extrato de um programa que usa memória partilhada entre múltiplas tarefas para implementar os produtores-consumidores:

```
int buf[N];
int prodptr=0, consptr=0;
trinco_t trinco_p, trinco_c;
semaforo_t pode_prod = criar_semaforo(N),
        pode_cons = criar_semaforo(0);

produtor()
{
  while(TRUE) {
    int item = produz();
    esperar(pode_prod);
    fechar(trinco_p);
    buf[prodptr] = item;
    prodptr = (prodptr+1) % N;
    abrir(trinco_p);
    assinalar(pode_cons);
  }
}
```

```
consumidor()
{
  while(TRUE) {
    int item;
    esperar(pode_cons);
    fechar(trinco_c);
    item = buf[consptr];
    consptr = (consptr+1) % N;
    abrir(trinco_c);
    assinalar(pode_prod);
    consome(item);
  }
}

main() {
  //Cria tarefas produtores e consumidores
}
```

   a. [1,2v] Resolva o mesmo problema (produtores-consumidores), agora no caso em que o produtor é um processo e o consumidor é outro processo. Baseie a sua implementação em sockets datagram. Assuma que **apenas há um produtor e um consumidor**. Pode usar pseudo-código, desde que o significado das funções seja claro e que os argumentos fundamentais estejam presentes.

Função main do produtor:

Função main do consumidor:

b. [0,5v] Em que modelo de interacção se enquadra a solução baseada em memória partilhada?

c. [0,5v] Em que modelo de interacção se enquadra a solução baseada em sockets datagram?

2. Considere o seguinte programa que permite estabelecer a comunicação entre dois processos pai e filho

```
main() {
    char msg[DIM], tmp[DIM];
    int fds[2], pid_filho;

[...]

    if (pipe (fds) < 0) exit(-1);
    if (fork () == 0) {
       read (fds[0], tmp, sizeof (msg));
       printf ("%s\n", tmp);
       exit (0);
}
  else {
     write (fds[1], msg, sizeof (msg));
     pid_filho = wait();
    }
 }
```

a. [0,6v] Faça uma representação das tabelas ficheiros relevantes (tabela de ficheiros abertos do processo, tabela de ficheiros abertos global do sistema, tabela de inodes) utilizadas pelo processo pai **antes da** execução da chamada sistema pipe.

b. [0,6v] Faça a mesma representação depois do pipe.

c. [0,6v] Quando é efectuado o fork, como se modificam as tabelas?

Grupo VI [2v]

`mem` é um gestor de periférico (device driver) existente desde as primeiras versões de Unix que permite ler qualquer posição de memória indicado pelo argumento de offset na função read que corresponde ao valor do endereço físico na memória.
Considere o seguinte extracto de programa.

```
if((mem_fd = open("/dev/mem",O_RDONLY)) < 0){
       printf("\n\nProblem in opening /dev/mem");
       exit(-1);
       }
ret = 1;
charp = (char *)malloc(sizeof(char) * 1024);

ret = read(mem_fd, charp, sizeof(char) * 1024);
close(mem_fd);
```

1. [0,7v] O que faz este extracto de programa?

2. [0,6v] Com base no exemplo, que funções acha que têm de ser disponibilizadas pelo gestor de periférico?

3. [0,7v] Quando uma função da interface de ficheiros é chamada, como é que o núcleo determina qual a das funções do gestor de periférico que indicou na alínea anterior deve ser executada?
