Número: Nome:

LEIC/LERC – 2009/10
2º Teste de Sistemas Operativos

16 de Janeiro de 2010
Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.
Duração: 1h30m

Grupo I [6,1v]

1. [2,5v] 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. [1,2v] 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
…	

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. [2,4v] Para cada afirmação seguinte, indique se é verdadeira caso consideremos o comando cp e caso consideremos o comando ln. Respostas erradas descontam 0,15v.
   a. 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. 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. Ambos os ficheiros podem existir em volumes com sistemas de ficheiros diferentes.

cp:	ln:

   d. Quanto maior o ficheiro original, mais tempo demora o comando a completar.

cp:	ln:

Grupo II [9,7v]

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. [2,4v]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. [1,2v]Indique uma vantagem de cada uma das duas soluções. Justifique.

c. [1,4v]Estenda a sua solução para permitir que o consumidor termine caso não receba nenhum item do produtor ao fim de 30s. (Apresente apenas as alterações ao código.)

d. [0,8v] Em que modelo de interacção se enquadra a solução baseada em memória partilhada?

e. [0,8v] 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. [1,2v] 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. [1,2v] Faça a mesma representação depois do pipe.

c. [0,7v] Quando é efectuado o fork, como se modificam as tabelas?

Grupo VI [4,2v]

`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. [1,2v]O que faz este extracto de programa?

2. [1v]Como é possível a um gestor de periférico ler de qualquer posição de memória física, sabendo que nelas estarão residentes páginas dos diversos processos e do núcleo? Justifique

3. [1v]Com base no exemplo, que funções acha que têm de ser disponibilizadas pelo gestor de periférico?

4. [1v]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?
