





Considere uma execução “B” (diferente da anterior), em que no instante 60, os processos tinham gasto o seguinte cpu:

Processo	CPU gasto
P1	4
P2	8
P3	24
P4	24

Considere agora que possui um sistema de prioridades variáveis, com preempção, em que as prioridades são recalculadas de 60 unidades em 60 unidades de tempo, de acordo com as seguintes fórmulas usadas no Unix original:

Decaimento:  $p\_cpu = p\_cpu / 2$

Prioridade:  $prio = p\_base + p\_cpu / 2 + nice$

A prioridade base de todos os processos é 100 e o nice é 0.

3. [0.5 v] Para a execução “B”, recalcule a prioridade dos processos após o instante 60.

Processo	Prioridade
P1	
P2	
P3	
P4	

Considere agora que o sistema é preemptivo e que atribui o processador ao processo mais prioritário. Considere também que o time-slice é reiniciado sempre que um processo perde o CPU. Considere que o *input* de P1 e P2 está vazio e que P1 recebe 1 caracter no instante 80 e que P2 recebe 4 caracteres no instante 78.



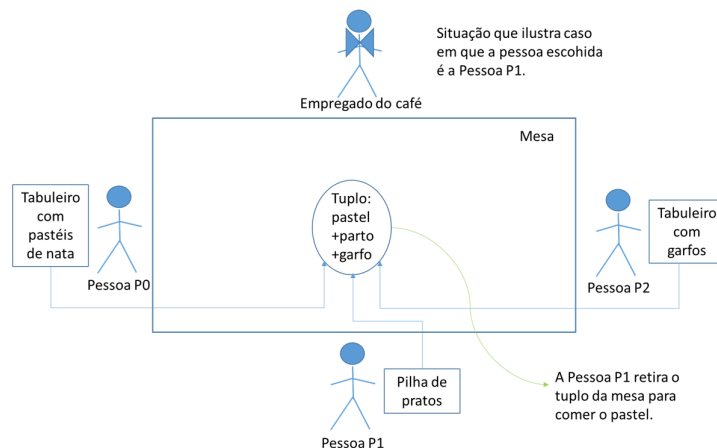
## Grupo II [6 v]

Considere o problema de sincronização seguinte que descreve pessoas que comem pastéis de nata num café, ao lanche, à volta de uma mesa. Para comer um pastel de nata é preciso: pastel, prato e garfo. Assuma que existem 3 pessoas que estão sentadas à mesa; ao lado de cada uma delas, fora da mesa, cada uma delas dispõe de vários itens, em número ilimitado, de um dado componente (que pode ser pastel, prato ou garfo) tal como se indica de seguida:

- P0 tem um tabuleiro com pastéis de nata;
- P1 tem uma pilha de pratos;
- P2 tem tabuleiro com garfos.

Os recursos pastéis, pratos e garfos são manipulados apenas pela pessoa respectiva.

O empregado do café, quando acha que é apropriado, seleciona (segundo algum critério) uma das pessoas e dá-lhe autorização para comer. Depois de uma pessoa ter sido selecionada, cada uma das pessoas (incluindo a que foi escolhida para comer) coloca um (e apenas um) item dos componentes que tem, na mesa, à frente da pessoa escolhida. Assim que esteja na mesa o tuplo formado por “pastel+prato+garfo”, este é retirado da mesa pela pessoa escolhida que come o pastel, demorando um intervalo de tempo indeterminado.



Existem as seguintes limitações:

- o empregado do café não começa a decidir qual a próxima pessoa que irá comer antes do tuplo acima referido ter sido retirado da mesa (pela pessoa escolhida);
- uma pessoa só pode colocar o seu componente (pastel, prato ou garfo) na mesa (em resposta a uma decisão do empregado do café) quando não está a comer.

Considere a seguinte solução, em pseudo-código, na qual o empregado corresponde a um fio de execução que executa a função `empregado` e que cada pessoa é um fio de execução que executa a função `pessoa`. O vector `componenteNaMesa` é um *buffer* circular sendo o valor inicial de cada elemento inicializado a `false` (significando que o respectivo componente não está na mesa). O valor inicial de `pessoaEscolhida` significa que não se encontra escolhida nenhuma pessoa para comer um pastel de nata.

```

1. boolean componenteNaMesa[3] = {false, false, false};
2. int pessoaEscolhida = -1;
3.
4. pessoa(int i) {
5.     int proximoPronto;
6.     int componentesProntos;
7.     while (true) {
8.         proximoPronto = false;
9.         while (proximoPronto == false) {
10            proximoPronto = pessoaEscolhida == -1 || componenteNaMesa[i];
11        }
12        componenteNaMesa[i] = true;
13        if (pessoaEscolhida == i) {
14            componentesProntos = false;
15            while (!componentesProntos) {
16                componentesProntos = componenteNaMesa[(i+1)%3] &&
                    componenteNaMesa[(i+2)%3];
17            }
18            for (k=0 to 2) componenteNaMesa[k] = false;
19            pessoaEscolhida = -1;
20            comePastel();
21        }
22    }
23 }
24
25 empregado() {
26     int f;
27     while(true) {
28         while (pessoaEscolhida != -1);
29         f = pensaEscolherProximaPessoa();
30         pessoaEscolhida = f;
31     }
32 }

```

1. Para as seguintes porções do código, indique um excerto (frase ou parte de frase) do enunciado que essa(s) linha(s) implemente(m):

a. [0.25 v] Linha 9 da função `pessoa`.

b. [0.25 v] Linha 15 da função `pessoa`.

c. [0.25 v] Linha 28 da função `empregado`.

2. [1 v] A solução proposta recorre a espera activa. Indique que consequência tal pode ter no desempenho, ilustrando com um exemplo de uma execução.

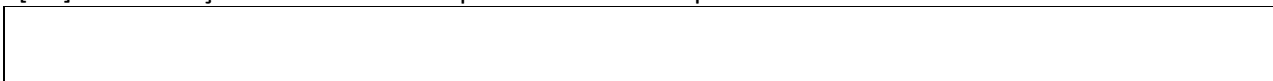
---

3. [1.25 v] Considere que as leituras/escritas em cada palavra são atômicas. Na solução apresentada, diga se o uso de trincos traria alguma vantagem. Caso a resposta seja negativa, justifique. Caso a resposta seja afirmativa, diga quantos trincos utilizaria em que linhas colocaria as chamadas aos trincos, fechar\_trinco(t) e abrir\_trinco (t).

4. [2 v] Proponha uma modificação da solução acima que elimine completamente a espera activa, ou que pelo menos a reduza. Utilize apenas semáforos.



5. [1 v] A sua solução ainda sofre de espera activa? Justifique.





--

**Grupo III [3 v]**

Considere um sistema de memória virtual paginada com 24 bits de endereçamento, em que cada página tem o tamanho de 4Kbytes ( $2^{12}$  bytes). Considere que só possui 8Kbytes de RAM e que a TLB possui 1 única entrada (os dígitos de proteção são R para leitura, W para escrita e X para execução).

Considere o processo com a seguinte tabela de páginas:

Página	Presente	Proteção	Base
0	0	RW	--
1	0	RW	--
2	0	R	--
3	0	R	--
4	0	RW	-

1. [0.5] Qual o papel da TLB no processo de tradução de endereços.



2. [1.5] Considere que o processo não possui páginas virtuais em memória, pode usar qualquer uma das tramas físicas (i.e. páginas físicas) e que a TLB está vazia. Considere que o processo realiza acessos à memória conforme discriminado, exatamente por esta ordem. Complete a seguinte tabela, considerando uma política de substituição FIFO. Note que os endereços virtuais assim como os físicos estão indicados em hexadecimal.

<b>Tipo de acesso</b>	<b>Endereço virtual</b>	<b>Excepção</b>	<b>TBL hit?</b>	<b>Carregou a página?</b>	<b>Endereço físico</b>	<b>TLB após o acesso</b>
Leitura	000002	Page Fault	Não	Sim	0x000002	0 → 0
Leitura	0041FC					
Escrita	000FEE					
Leitura	001123					
Escrita	000FFF					
Leitura	004FFF					
Leitura	002354					
Leitura	002043					
Escrita	003FFA					

3. [1.0] Indique uma razão para a maioria dos sistemas usar aproximações de Least Recently Used ( LRU) em vez de usarem exatamente LRU como políticas de substituição de páginas.


## Grupo IV [3.5 v]

Considere o seguinte programa servidor.

<pre> /*-----   Programa servidor.c (gera binario "servidor") +-----*/  #include &lt;stdio.h&gt; #include &lt;sys/time.h&gt; #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt; #include &lt;sys/un.h&gt;  #define SOCK_PATH "echo_socket" #define SZ_STR 100  int main(void) {     int s, s2, t;     struct sockaddr_un local, remote;     char str[SZ_STR];     char byteread;     int accepted = 0;     fd_set readfds;     int maxfd;      if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {         perror("socket");         exit(1);     }      local.sun_family = AF_UNIX;     strcpy(local.sun_path, SOCK_PATH);     unlink(local.sun_path);     if (bind(s, (struct sockaddr *)&amp;local, sizeof(struct         sockaddr_un)) == -1) {         perror("bind");         exit(1);     }      if (listen(s, 1) == -1) {         perror("listen");         exit(1);     } </pre>	<pre> for(;;) {     FD_ZERO (&amp;readfds);      if (accepted) {         FD_SET(0, &amp;readfds);         FD_SET(s2, &amp;readfds);         maxfd = s2;     }     else {         FD_SET(0, &amp;readfds);         FD_SET(s, &amp;readfds);         maxfd = s;     }      select(maxfd+1, &amp;readfds, NULL, NULL, NULL);      if (FD_ISSET(0, &amp;readfds)){         read (0, &amp;byteread, 1);         if (byteread=='a')             printf ("STDIN\n");     }     if (!accepted &amp;&amp; FD_ISSET(s, &amp;readfds)) {         t = sizeof(remote);         if ((s2 = accept(s, (struct sockaddr *)&amp;remote, &amp;t)) == -             1) {             perror("accept");             exit(1);         }         printf ("Connected to client\n");         accepted = 1;     }     if (accepted &amp;&amp; FD_ISSET(s2, &amp;readfds)) {         read (s2, &amp;byteread, 1);         if (byteread=='a')             printf ("SOCKET\n");     } } </pre>
--	---

1. [0.25] Diga, justificando, que tipo de socket é criado.



2. [0.25] Para que serve a chamada “bind”? Justifique a sua resposta tendo em conta o sistema de ficheiros.


3. [0.5] Explique a diferença entre código quando a flag “accepted” está a true e quando a flag “accepted” está a false.


4. [0.5] Considere que lança o servidor e depois escreve a seguinte *string* no terminal onde lançou o servidor:

- banana<RET>
- 

Diga qual o output do programa servidor.


Considere o seguinte programa cliente.

<pre>/*-----   Programa cliente.c (gera binario "cliente") +-----*/  #include &lt;sys/types.h&gt; #include &lt;sys/socket.h&gt; #include &lt;sys/un.h&gt; #include &lt;stdio.h&gt;  #define SOCK_PATH "echo_socket" #define SZ_BUFF 100  main(int argc, char **argv) {     int sock;     struct sockaddr_un server;     char c;      sock = socket(AF_UNIX, SOCK_STREAM, 0);     if (sock &lt; 0) {         perror("opening stream socket");         exit(1);     } }</pre>	<pre>server.sun_family = AF_UNIX; strcpy(server.sun_path, SOCK_PATH);  if (connect(sock, (struct sockaddr *) &amp;server, sizeof(struct     sockaddr_un)) &lt; 0) {     close(sock);     perror("connecting stream socket");     exit(1); }  while ((c = getc(stdin))!=EOF) {     write(sock, &amp;c, 1); } }</pre>
---	---

5. [0.5] Para que serve a chamada "connect"?


6. [0.5] Considere que lança o servidor num terminal. Considere que, noutro terminal, lança o programa cliente acima (já com o servidor em execução) e depois escreve a seguinte *string* no terminal onde lançou o cliente:

➤ laranja<RET>

Diga qual o output do programa servidor.


Considere o seguinte programa “pai”:

```

/*-----
 | Programa pai.c (gera binario "pai")
+-----*/

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

int main (int argc, char** argv) {
    int fd[2];
    int pid;
    char c;

    if (pipe(fd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    pid = fork ();

    if (pid == 0) {
        if (execl("./servidor", "servidor", NULL) == -1) {
            perror("servidor");
            exit(EXIT_FAILURE);
        }
    }
    else if (pid != -1){

        while ((c = getc(stdin))!=EOF) {
            write (1, &c, 1);
        }
    }
}

```

7. [1.0] Acrescente o código necessário no programa acima, de forma a que o input do processo pai seja processado pelo processo servidor.

### Grupo V [3.5 v]

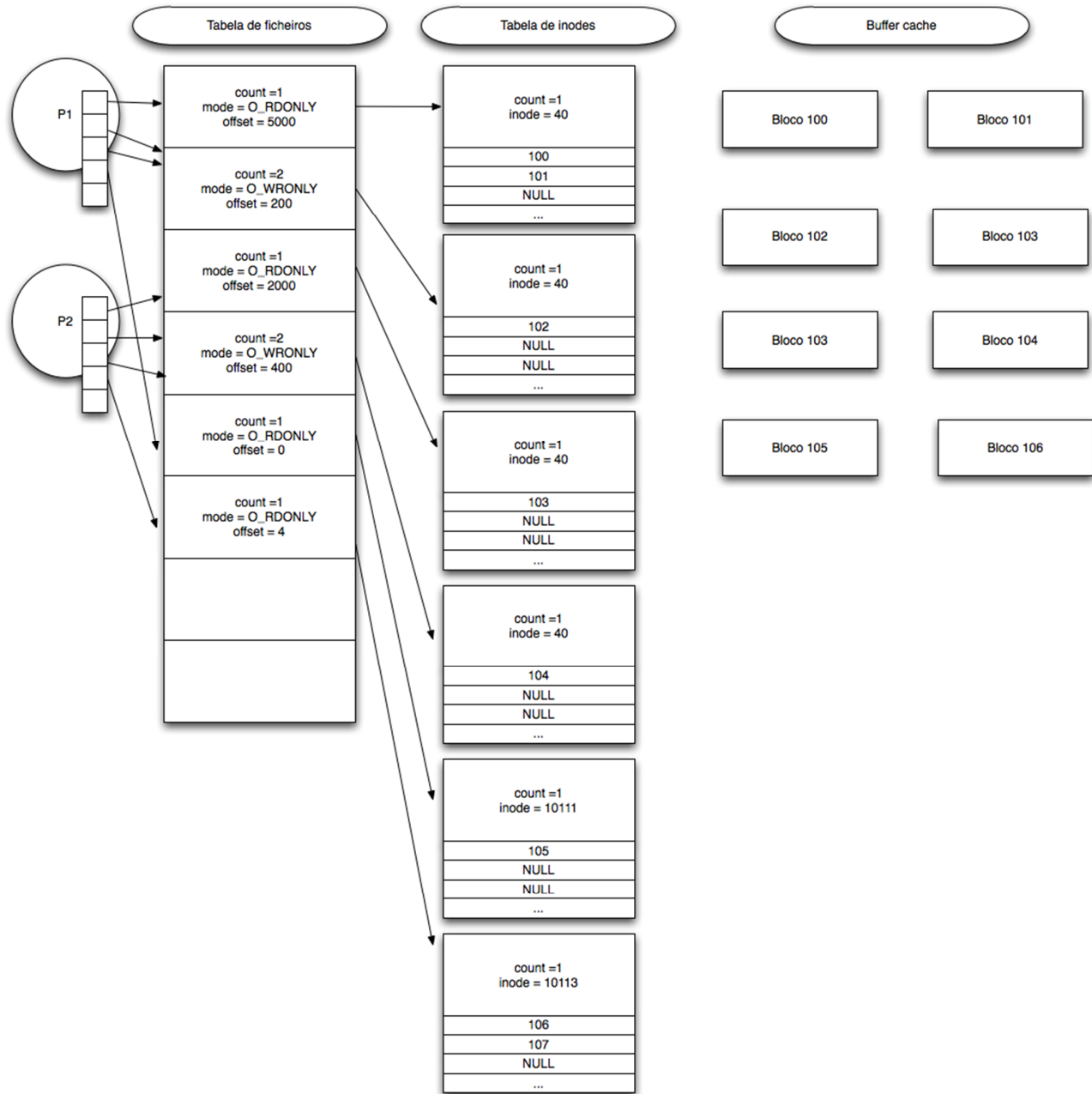
Considere o sistema de ficheiros do Unix e o conteúdo da seguinte diretoria, de nome `/users/so/`:

	<b>Cada entrada na diretoria tem, neste exemplo, exatamente 12 bytes</b>				
	<b>Inode</b>	<b>Tamanho Entrada</b>	<b>Tamanho Nome</b>	<b>Tipo</b>	<b>Nome</b>
<b>Deslocamento</b>	<b>(4 bytes)</b>	<b>(2 bytes)</b>	<b>(1 byte)</b>	<b>(1 byte)</b>	<b>(4 bytes)</b>
0	10111	12	1	2	.\0\0\0
12	10112	12	2	2	..\0\0\0
24	10113	16	8	1	abcd.txt

1. [0.5] Altere o conteúdo da diretoria para reflectir o resultado de executar o seguinte comando para criar um "hard link":

```
link /users/so/abcd.txt /users/so/xxx
```

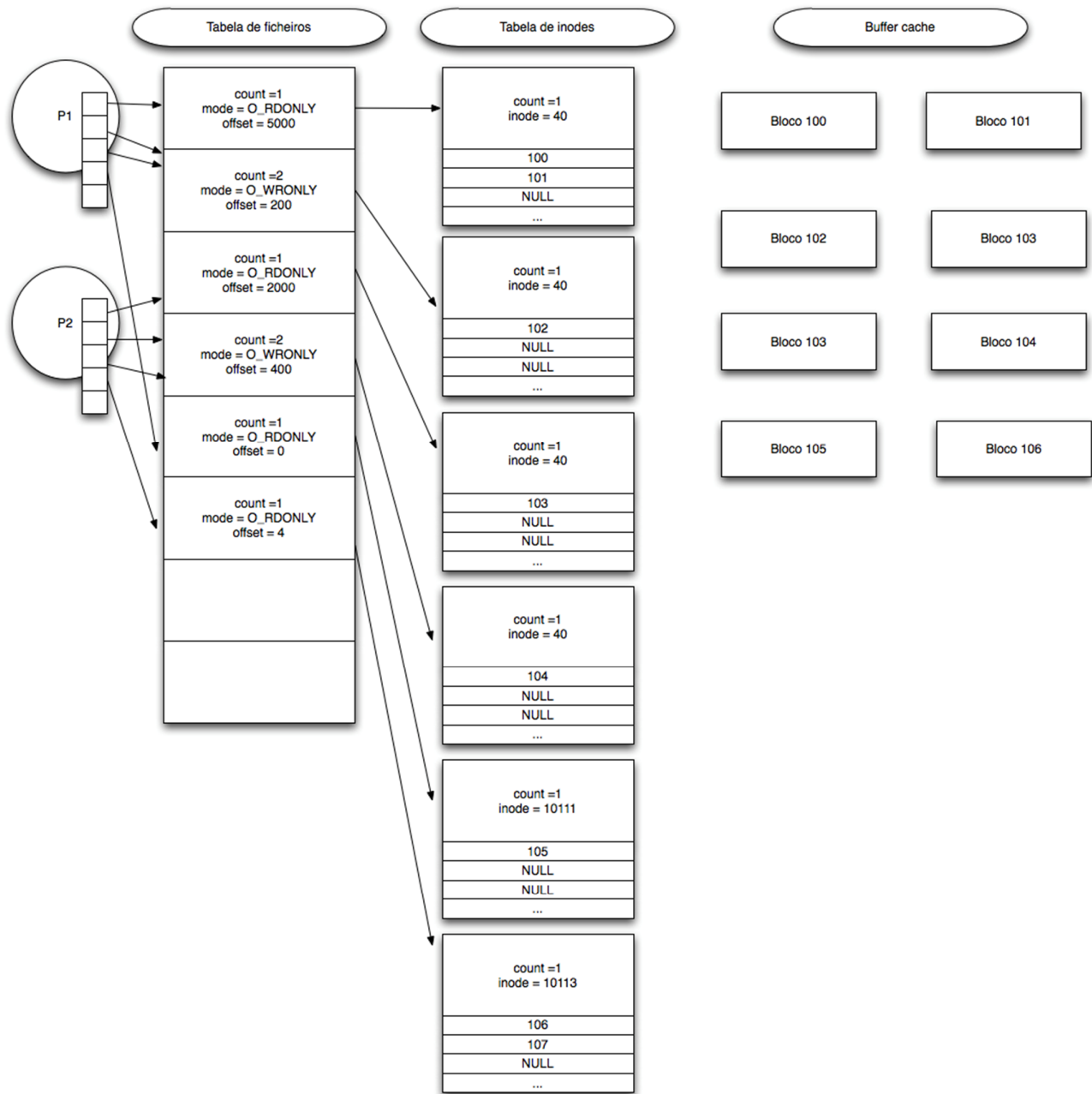
Considere o seguinte estado das tabelas que suportam o acesso aos ficheiros (considere que cada bloco tem 4K de tamanho) num sistema UNIX.



- [1.0] Altere diretamente o esquema acima para representar o estado das mesmas tabelas após o processo P2 fazer "fork", criando desta forma um processo P3 (filho de P2).



Considere de novo o seguinte estado das tabelas que suportam o acesso aos ficheiros:



3. [1.0] Altere diretamente o esquema acima para representar o estado das mesmas tabelas após o processo P1 fazer a chamada sistema

```
open ("/users/so/abcd.txt", O_RDONLY)
```

4. [0.25] Considere que após o `open` acima, o processo P1 executa o seguinte código:

```
nbytes = read(4, buf, 1);
```

será necessário trazer algum bloco para memória? Em caso afirmativo, diga qual. Justifique.


5. [0.25] Considere que após a chama `read` acima, o processo P1 executa o seguinte código:

```
offset lseek(4, 5000, SEEK_SET);
nbytes = read(4, buf, 1);
```

Será necessário trazer algum bloco para memória? Em caso afirmativo, diga qual. Justifique.


6. [0.5] Num sistema de ficheiros Unix, indique para que serve o superbloco: