Número: Nome:

LEIC/LETI – 2014/15 - 2º Teste de Sistemas Operativos
9/Janeiro/2015

Identifique todas as folhas. Responda no enunciado no espaço fornecido.
Justifique todas as respostas. Duração: 1h30m

Grupo I [6 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	Protecção	Base
0	0	RW	--
1	0	RW	--
2	0	R	--
3	0	R	--
4	0	RW	-

1. [1.0 v] Qual o papel da TLB no processo de tradução de endereços.

Trata-se de uma memória associativa, de acesso rápido, que mantém uma cache das traduções de endereço virtual para endereço real realizadas recentemente. Permite evitar o acesso frequente à tabela de páginas.

2. [3.0 v] 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.

Tipo de acesso	**Endereço virtual**	**Excepção**	**TLB hit?**	**Carregou a página?**	**Endereço físico**	**TLB após o acesso**
Leitura	000002	Page Fault	Não	Sim	0x000002	0 → 0
Leitura	0041FC	**Page Fault**	**Não**	**Sim**	**0x0011FC**	**4 → 1**
Escrita	000FEE	**--**	**Não**	**Não**	**0x000FEE**	**0 → 0**
Leitura	001123	**Page Fault**	**Não**	**Sim**	**0x000123**	**1 → 0,**
Escrita	000FFF	**Page Fault**	**Não**	**Sim**	**0x001FFF**	**0 → 1**
Leitura	004FFF	**Page Fault**	**Não**	**Sim**	**0x000FFF**	**4 → 0**
Leitura	002354	**Page Fault**	**Não**	**Sim**	**0x001345**	**2 → 1**
Leitura	002043	**--**	**Sim**	**Não**	**0x001043**	**2 → 1**
Escrita	003FFA	**Seg Fault**	**Não**	**Não**		**2 → 1**

3. [2.0 v] 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.

Uma vez que a execução de qualquer instrução obriga o acesso a uma ou mais páginas, o micro-código usado para registar os acessos deve ser muito eficiente, de preferência alterar apenas o valor de alguns bits. Manter o LRU perfeito obrigaria a executar muitas micro-instruções.

Grupo II [7 v]

Considere o seguinte programa servidor.

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

#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

#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 *)&local, sizeof(struct
      sockaddr_un)) == -1) {
    perror("bind");
    exit(1);
  }

  if (listen(s, 1) == -1) {
    perror("listen");
    exit(1);
  }
```

```
  for(;;) {
    FD_ZERO (&readfds);

    if (accepted) {
      FD_SET(0, &readfds);
      FD_SET(s2, &readfds);
      maxfd = s2;
    }
    else {
      FD_SET(0, &readfds);
      FD_SET(s, &readfds);
      maxfd = s;
    }

    select(maxfd+1, &readfds, NULL, NULL, NULL);

    if (FD_ISSET(0, &readfds)){
      read (0, &byteread, 1);
      if (byteread=='a')
        printf ("STDIN\n");
    }
    if (!accepted && FD_ISSET(s, &readfds)) {
      t = sizeof(remote);
      if ((s2 = accept(s, (struct sockaddr *)&remote, &t)) == -
        1) {
        perror("accept");
        exit(1);
      }
      printf ("Connected to client\n");
      accepted = 1;
    }
    if (accepted && FD_ISSET(s2, &readfds)) {
      read (s2, &byteread, 1);
      if (byteread=='a')
        printf ("SOCKET\n");
    }
  }
}
```

1. [0.5 v] Diga, justificando, que tipo de socket é criado.

É criado um socket da família UNIX orientado à ligação (do tipo stream).

2. [0.5 v] Para que serve a chamada “bind”? Justifique a sua resposta tendo em conta o sistema de ficheiros.

Permite associar um nome, global ao sistema, ao socket. No caso dos sockets da família UNIX, esse nome é registado no sistema de ficheiros.

3. [1.0 v] Explique a diferença de funcionamento quando a flag “accepted” está a true e quando a flag “accepted” está a false.

No início “accepted” está a falso, o que indica que ainda não foi aceite nenhuma ligação no socket. Nesse caso, o servidor espera por pedidos de ligação no socket “s”. Quando chega esse pedido, e este é aceite, é devolvido um socket “s2” para interagir com o cliente e a flag passa a true. A partir deste momento o servidor deixa de esperar por pedidos de ligação e espera por caracteres enviados pelo cliente. Em qualquer dos casos, o servidor espera também por caracteres a partir do stdin (daí a necessidade de fazer select).

4. [1.0 v] 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.

STDIN
STDIN
STDIN

Considere o seguinte programa cliente.

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

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>

#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 < 0) {
  perror("opening stream socket");
  exit(1);
 }
```

```
server.sun_family = AF_UNIX;
 strcpy(server.sun_path, SOCK_PATH);

 if (connect(sock, (struct sockaddr *) &server, sizeof(struct
      sockaddr_un)) < 0) {
  close(sock);
  perror("connecting stream socket");
  exit(1);
 }

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

5. [1.0 v] Para que serve a chamada “connect”?

Envia um pedido de estabelecimento de ligação ao servidor e espera que este o aceite. Caso o pedido seja aceite, a ligação fica estabelecida e o socket pode ser usado para trocar informação com o servidor.

6. [1.0 v] 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.

SOCKET
SOCKET
SOCKET

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) {
      close (0);
      dup (fd[0]);
      close (fd[1]);
      close (fd[0]);

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

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

7. [2.0 v] Acrescente o código necessário no programa acima, de forma a que o input do processo pai seja processado pelo processo servidor.

Grupo III [7 v]

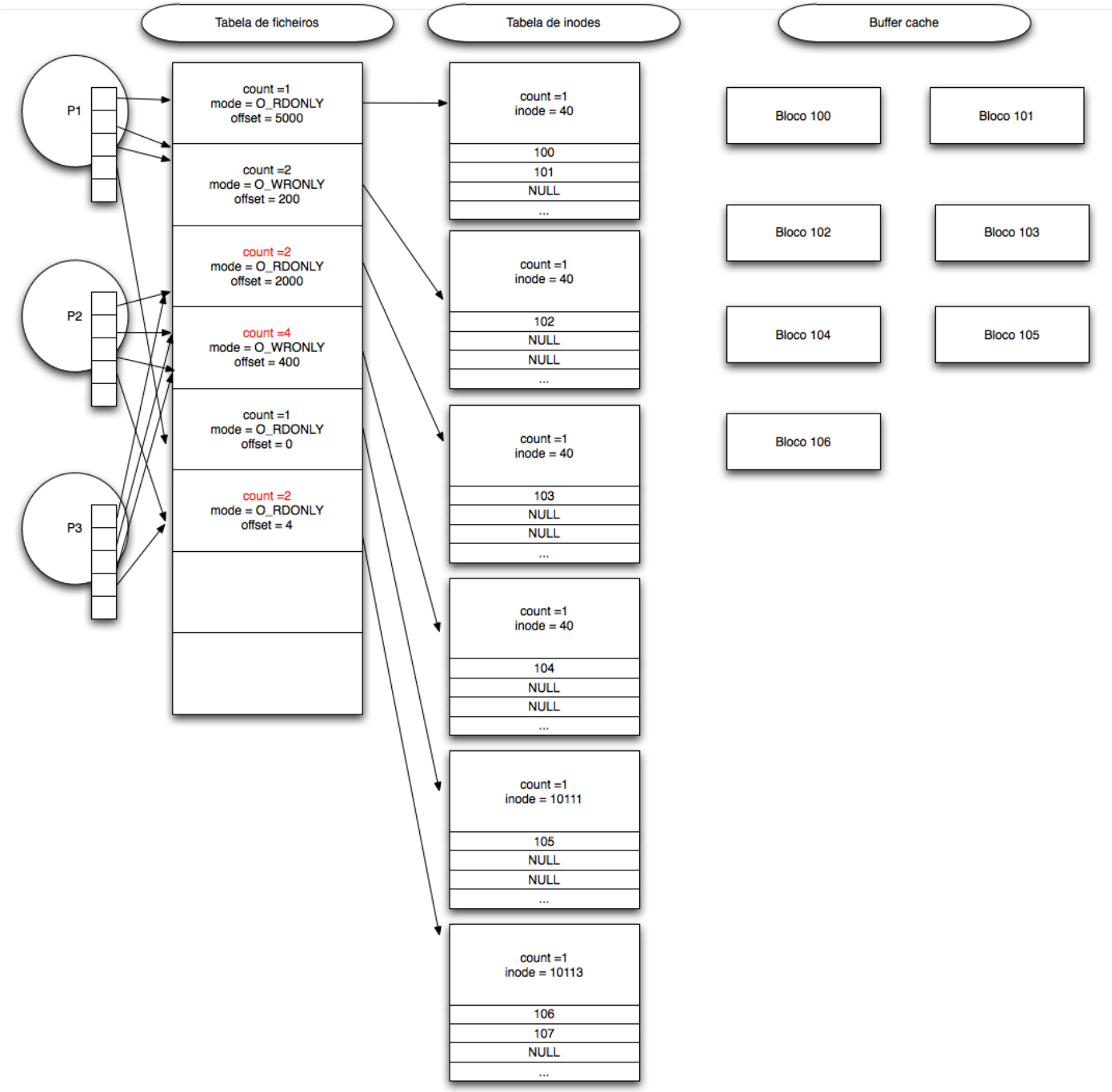
Considere o sistema de ficheiros do Unix e o conteúdo da seguinte diretoria, de nome /users/so/:

	Cada entrada na diretoria tem, neste exemplo, 12 ou 16 bytes				
	Inode	**Tamanho Entrada**	**Tamanho Nome**	**Tipo**	**Nome**
Deslocamento	**(4 bytes)**	**(2 bytes)**	**(1 byte)**	**(1 byte)**	**(n bytes)**
0	10111	12	1	2	.\0\0\0
12	10112	12	2	2	..\0\0
24	10113	16	8	1	abcd.txt
40	**10113**	**12**	**3**	**1**	**xxx\0**

1. [1.0 v] Altere o conteúdo da directoria para reflectir o resultado de executar o seguinte comando para criar um "hard link":

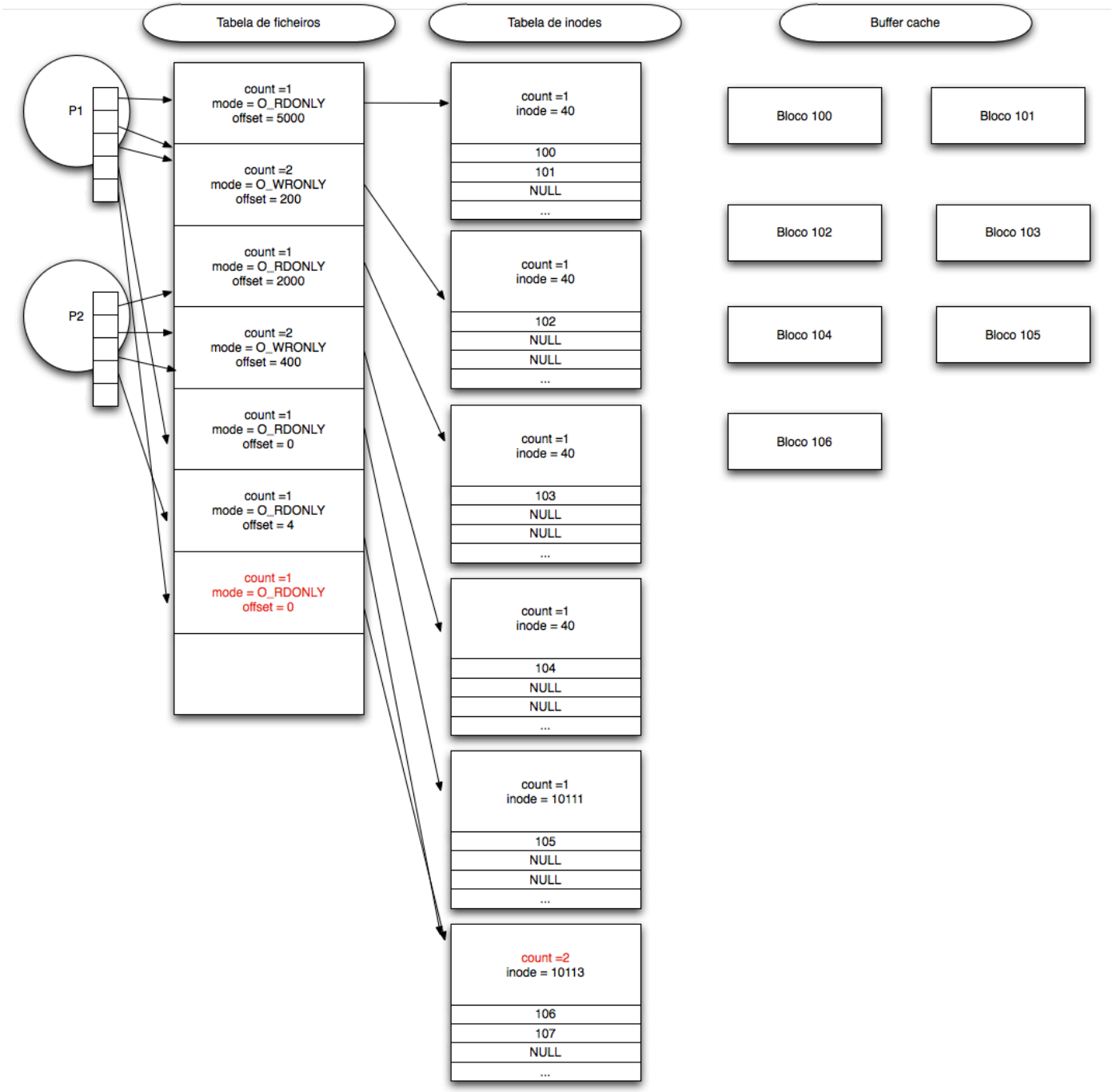
```
link /users/so/abcd.txt /users/so/xxx
```

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



2. [2.0 v] 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. [2.0 v] 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. [0.5 v] 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.

Não. O primeiro bloco do ficheiro é o 106 (veja-se o índice do inode 10113) e já esta na cache.

5. [0.5 v] 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.

Sim. O byte 5000 já está no 2º bloco do ficheiro, o bloco 107, e este não está em cache.

6. [1.0 v] Num sistema de ficheiros Unix, indique para que serve o superbloco:

Mantém uma descrição da meta-informação do sistema de ficheiros, tal como o tipo de sistema de ficheiros, o tamanho, tamanho dos blocos, o número de blocos, número de inodes, o inode da directoria raiz, inicio da lista de blocos e da lista de inodes livres, etc.