**Versão: v0** (deve corresponder ao algarismo menos significativo do número de aluno)

# LEIC/LETI – 2020/21 – Repescagem do 2º Teste de Sistemas Operativos

**8 de fevereiro de 2021, Duração: 1h00m**

- Confira as instruções fornecidas no site da disciplina sobre a realização online desta prova.
- Por omissão, os excertos de código **omitem o tratamento de erros**; nas suas respostas com código, **pode também omitir o tratamento de erros**.
- Nas perguntas de escolha múltipla **existe apenas uma resposta certa**. Em caso de dúvida, **pode selecionar uma ou mais alíneas**. A nota é calculada pelas alíneas que escolheu na sua resposta, da seguinte forma: a alínea correta conta com a cotação completa; cada alínea incorreta desconta 1/3 da cotação da pergunta.

---

## Grupo I [Comunicação entre processos, 6v]

Considere um processo P que, inicialmente, tem dois *sockets*, um *stream* (variável global *socketStream*) e outro *datagram* (variável global *socketDatagram*).

Estes *sockets* são usados da seguinte forma:

- Diferentes clientes podem ligar-se através do *socket stream* de P.
- Quando P estabelece uma nova ligação com um cliente, é criada uma tarefa escrava (em P).
- A tarefa escrava recebe os dados que o cliente envia pela ligação (até um máximo de DIM bytes, em que DIM é uma constante previamente definida).
- Logo depois, a tarefa escrava envia os dados recebidos através do socket datagram de P para um processo remoto, cujo endereço é definido na variável global *targetAddr*.

O programa seguinte implementa o comportamento especificado acima, embora tenha 3 porções incompletas.

```
int socketStream, socketDatagram;
struct sockaddr_un targetAddr;

//Assumir que socketStream, socketDatagram e targetAddr já
//foram criados/inicializados

while (1) {
  pthread_t tid;
  int *s = (int*)malloc(sizeof(int));

  *s = accept(socketStream, 0, 0);
  pthread_create(&tid, 0, fn, s);
}

void *fn(void *s) {
  char buffer[DIM];

  //1. Receber até DIM bytes da ligação estabelecida por socket stream e
coloca-los em buffer

  //2. Enviar os dados em buffer para o endereço targetAddr, através do socket
datagram

  //3. Fechar/libertar os recursos que já não sejam necessários

  return NULL;
}
```

1. [1,5v] Apresente o código que implementa o passo 1 está em falta no programa fornecido acima.

```
int newSocketStream = *((int*)s);
int c = read(newSocketStream, buffer, DIM);

//Notas:
//- como indicado no enunciado, omitimos o tratamento de
//retornos de erro das funções read e sendto (abaixo)
//- também se consideraram corretas soluções que chamam
//a função read em ciclo até ao buffer estar cheio ou à função
//read devolver 0
```

2. [1,5v] Idem para o passo 2.

```
sendto(socketDatagram, buffer, c,
       0, (struct sockaddr *) &targetAddr, sizeof(targetAddr));
```

3. [1,5v] Idem para o passo 3.

```
close(newSocketStream);
```

4. [1,5v] No programa acima, pode acontecer que, após uma ligação ser estabelecida com o processo P, o cliente que iniciou a ligação termine imediatamente. Caso isso aconteça antes da tarefa escrava de P chamar a função para receber dados do *socket*, ocorre um *signal* do tipo *SIGPIPE* e todo o processo P é terminado pelo SO.
   Pretende-se que, nesta situação, o processo P não termine. Em vez disso, o processo P deve continuar em execução, pronto a servir outros clientes.

   O que modificaria/acrescentaria ao excerto de código apresentado acima?
   A. Acrescentaria *signal(SIGPIPE, SIG_IGN)* antes do ciclo.
   B. Chamaria *wait* no início de cada iteração.
   C. Colocaria o código da iteração dentro de uma função f e, antes do ciclo, chamaria *signal(SIGPIPE, f)*.
   D. Usaria *signal(SIGPIPE, read)*.

Versão: v0 (deve corresponder ao algarismo menos significativo do número de aluno)

**Grupo II (Sistema de ficheiros, 4,5v)**

5. [1,5v] Qual é o tamanho máximo de um ficheiro num sistema de ficheiro ext3 caso o tamanho de cada bloco de dados seja de 16KB e as referências para blocos de dados ocupem 8 bytes?
   A. 1TB
   B. 1GB
   C. 32GB
   D. Nenhuma das anteriores

6. [1,5v] O *VFS Superblock* é usado para:
   A. Guardar a informação sobre os blocos livres no EXT3.
   B. Definir quais funções chamar para implementar cada operação de acesso a um ficheiro, dependendo do sistema de ficheiros em que o ficheiro se encontra guardado.
   C. Guardar persistentemente informação sobre quanto espaço é atribuído aos vários sistemas de ficheiro presentes num disco.
   D. Nenhuma das anteriores

7. [1,5v] Quantos i-nodes são acedidos no ext3 para executar a seguinte chamada de sistema, no caso em que a chamada é bem sucedida?

   *open("/usr/dir/file.txt", O_RDOONLY)*

   A. 1
   B. 2
   C. 3
   D. 4

## Grupo III (Memória Virtual, 4,5v)

8. [1v] Qual das seguintes afirmações é verdadeira:
   A. Páginas de tamanho maior tendem a gerar maior fragmentação externa.
   B. Páginas de tamanho maior implicam tabelas de páginas de dimensão maior.
   C. Páginas de tamanho menor implicam tabelas de páginas de dimensão maior.
   D. O Linux permite adaptar o tamanho das páginas em tempo real

9. [1v] Qual das seguintes afirmações é verdadeira:
   A. Em sistemas baseados em segmentação o núcleo é ativado cada vez que há um acesso à memória.
   B. Em sistemas baseados em paginação o núcleo é ativado apenas caso a página não esteja presente em memória principal.
   C. Em sistemas baseados em segmentação o TLB permite acelerar a tradução de endereços virtuais para endereços físicos.
   D. Nenhuma das anteriores.

10. [2,5 val] Considere um sistema de gestão de memória com 16 bits de espaço de endereçamento virtual, com páginas de 256B e um processo P cuja tabela das páginas contém estas entradas:

| Página | Presente | Protecção | Base |
|---|---|---|---|
| 0 | 0 | RW | |
| 1 | 1 | R | 0x02 00 |
| 2 | 1 | R | 0x01 00 |
| 3 | 1 | RW | 0x03 00 |

Na tabela abaixo, indique qual é o endereço físico correspondente aos seguintes endereços virtuais, caso possível. Caso o acesso der origem a alguma exceção, indique também o tipo da exceção.
Assuma que, em caso de falta de páginas, o SO aloca tramas (*page frames*) livres a partir do endereço 0x09 00.

| Acesso | End. Virtual | End. Físico | Eventuais exceções |
|---|---|---|---|
| Leitura | 0x00 22 | **0x09 22** | **Falta de página** |
| Escrita | 0x02 22 | | **Violação de acesso** |
| Escrita | 0x03 01 | **0x03 01** | **(Nenhuma)** |
| Leitura | 0x0F 00 | | **Página inexistente** |

Versão: v0 (deve corresponder ao algarismo menos significativo do número de aluno)

**Grupo IV (Chamadas de Sistema e Escalonamento, 5v)**

11. [2v] Suponha que o algoritmo baseado em épocas do escalonador Linux foi configurado para usar quantum_base=10ms e que existem dois processos, P1 e P2, que foram lançados simultaneamente e que se executaram concorrentemente durantes 3 épocas, gastando em cada época:

    P1: 10ms, 0ms, 0ms
    P2: 6ms, 6ms,5ms

    Na próxima época de execução (a 4ª), o processo P2 à disposição um quantum de:
    A. 18 ms
    B. 15 ms
    C. 14 ms
    D. 12 ms

12. [1,5v] Qual das seguintes afirmações é verdadeira:
    A. O algoritmo de escalonamento em Unix utiliza quantum variáveis.
    B. O algoritmo de escalonamento em Linux atualiza as prioridades dos processos periodicamente, depois de um período temporal fixo.
    C. O algoritmo de escalonamento em Linux favorece os processos "I/O-intensive" enquanto que o do Unix privilegia os processos "CPU-bound".
    D. Nenhuma das repostas anteriores.

13. [1,5v] Qual das seguintes afirmações é verdadeira:
    A. Usar mutexes implementados pelo núcleo garante melhor desempenho do que mutexes implementados em "user-level" com seções críticas curtas e com pouca contenção.
    B. Mutexes implementados em "user-mode" garantem melhor eficiência energética.
    C. Os mutexes implementados pelo núcleo permitem reduzir o consumo de CPU caso a correspondente secção crítica seja muito demorada e com elevada contenção.
    D. Nenhuma das repostas anteriores.