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

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

26 de janeiro de 2021, Duração: 1h00m

- **Confira as instruções fornecidas no site da disciplina sobre a realização online deste 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]

1. [2,4v] Considere um programa servidor que recebe pedidos através de um *socket datagram* do domínio Unix.
   - Os pedidos são do tipo *struct request_t*, definido assim:

```
struct request_t {
  char msg[100];
  struct sockaddr_un replyTo;
  int replyToLen;
}
```

   - Quando recebe um pedido, o servidor processa-o, chamando a função *process* sobre o campo *msg* do pedido.
   - O resultado desse processamento é um inteiro (*int*), que por omissão deve ser devolvido ao *socket* de onde chegou o pedido.
   - Há uma exceção ao comportamento acima: caso o campo *replyToLen* seja diferente de zero, o resultado deve ser enviado para um outro *socket,* cujo endereço é indicado pelo campo *replyTo* do pedido (nesse caso, o campo *replyToLen* indica a dimensão, em bytes do endereço destino).

   Seguindo a especificação acima, complete o excerto seguinte do ciclo principal do servidor. Na sua resposta: apresente apenas as linhas que modificaria/acrescentaria; deve omitir o tratamento de erros.

```
while (1) {
    struct sockaddr_un client_addr;
    int addrlen;
    struct request_t req;
    int result;

    addrlen=sizeof(struct sockaddr_un);
    recvfrom(sockfd, &req, sizeof(req), 0,
             (struct sockaddr *)&client_addr, &addrlen);

    result = process(req.msg);

    /* RESPOSTA COMEÇA AQUI*/
    if (req.replyToLen == 0) {
      sendto(sockfd, &result, sizeof(result), &client_addr, addrlen);
    } else {
      sendto(sockfd, &result, sizeof(result), &req.replyTo, req.replyToLen);
    }
    /* RESPOSTA ACABA AQUI*/
  }
```

Considere agora um programa servidor típico, que recebe pedidos, processa-os e devolve as respetivas respostas (neste caso, sempre para o mesmo cliente). Pretende-se comparar implementações desse servidor usando *sockets stream* vs. *named pipes.*
De seguida apresenta-se um excerto (incompleto) do seu ciclo principal:

```
while (1) {

  //Em falta: estabelecer ligação(ões)

  read(......., buffer, sizeof(buffer));
  process(buffer); //Auxiliary function
  write (......., buffer, strlen(buffer));

  //Em falta: fechar ligação(ões)
}
```

2. [1,2v] Ambos os argumentos têm o mesmo valor?
   A. Sim caso se use *sockets stream*; não caso se use pipe com nome.
   B. Não caso se use *sockets stream*; sim caso se use pipe com nome.
   C. Sim, tipicamente têm o mesmo valor em ambas as alternativas.
   D. Não, são obrigatoriamente distintos em ambas as alternativas.

Justificação:
Os pipes com nome são unidirecionais, logo uma comunicação em ambos os sentidos exige o uso de dois pipes com nome (um para cliente > servidor; outro para servidor > cliente).
Em contraste, um único socket stream suporta comunicação bidirecional.

Antes do excerto acima ser executado, é necessário que seja(m) estabelecida(s) ligação(ões) com o cliente. Do lado do servidor, que função estabelece essa(s) ligação(ões)?

3. [1,2v] *Com sockets stream:*
   A. accept
   B. open
   C. select
   D. recvfrom

4. [1,2v] *Com named pipes:*
   A. accept
   B. open
   C. select
   D. recvfrom

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

Grupo II (Chamadas de Sistema e Escalonamento, 5v)

5. [1,2v] Qual das seguintes afirmações é verdadeira:
   A. A desvantagem principal dos núcleos monolíticos é que a implementação do núcleo é pouco modular e difícil de evoluir.
   B. O Linux utiliza uma arquitetura baseada em micro-núcleo.
   C. Os núcleos organizados com mais de uma camada são tipicamente mais eficientes do que os núcleos monolíticos.
   D. Nenhuma das afirmações anteriores é verdadeira.

```
for (i=0;i<MAX_ITER; i++) {
          sleep(1);
          pthread_mutex_lock(&mutex);
          count = count + 1;
          pthread_mutex_unlock(&mutex);
}
```

6. [1,9v] Considere o código acima. Qual das seguintes afirmações é verdadeira:
   A. A execução de uma iteração do ciclo não gera nenhuma interrupção.
   B. A execução de uma iteração do ciclo só gera interrupções software.
   C. A execução de uma iteração do ciclo só gera interrupções hardware.
   D. A execução de uma iteração do ciclo gera interrupções hardware e software.

7. [1,9v] Suponha que o escalonador Linux foi configurado para usar quantum_base=10ms e que um processo consome durante as suas primeiras 4 épocas de execução 2, 10, 6 e 1 ms do próprio quantum, respetivamente.
   Na próxima época de execução (a 5ª), o processo terá à disposição um quantum de:
   A. 12 ms
   B. 14 ms
   C. 16 ms
   D. 18 ms

Justificação:
Quantum disponível para esta época = quantum base + quantum resíduo da época anterior
Época 1: Quantum disponível = 10ms. Quantum usado: 2ms, Quantum restante= 8ms
Época 2: Quantum disponível = 14ms. Quantum usado:10ms, Quantum restante =4ms
Época 3: Quantum disponível = 12ms. Quantum usado:6ms, Quantum restante =6ms
Época 4: Quantum disponível = 13ms. Quantum usado:1ms, Quantum restante =12ms
Época 5: Quantum disponível = 16ms.

Grupo III (Memória Virtual, 4,5v)

8. [1,5v] Considere um sistema de gestão de memória com 64 bits de espaço de endereçamento virtual, com 8 bits (os mais significativos) para a identificação do segmento. Quantos segmentos diferentes pode mapear um processo?
   A. 8
   B. 128
   C. 256
   D. Nenhuma das anteriores

Justificação:
Com 8 bits pode-se referenciar 2^8 segmentos

9. [1,5v] Considere um sistema baseado em paginação, com páginas de 1GB, endereços de 64 bits e tabelas de páginas **com um nível** e entradas de 8 bytes. Quantos bytes pode ocupar, no máximo, a tabela de páginas de um processo?
   A. 2^37 bytes
   B. 2^32 bytes
   C. 2^8 bytes
   D. Nenhuma das anteriores

Justificação:
Sendo as páginas de 1GB (2^30=1GB), o deslocamento ocupa 30 bits. Logo,nos 64 bits do endereço virtual sobram 34 (64-30) bits para identificar as páginas.
*A tabela de páginas terá, então, até 2^34 entradas, sendo cada entrada de 8 (2^3) bytes. Portanto a tabela das páginas ocupa 2^34 * 2^3 = 2^37 bytes*

10. [1,5v] Qual das seguintes afirmações é verdadeira:
    A. O TLB é atualizado apenas pelo hardware, quando ocorre um *page fault*.
    B. O TLB é atualizado apenas pelo OS, quando ocorre um *page fault.*
    C. O TLB é atualizado pelo OS quando ocorre um *page fault* e pelo hardware quando existe uma comutação de processo.
    D. O TLB é atualizado pelo hardware quando a página acedida se encontra em memória principal mas o endereço físico não se encontra no TLB.

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

Grupo IV (Sistema de ficheiros, 4,5v)

11. [1,6v] Considere um disco com 1TB de espaço, blocos de 8KB e a referência para cada bloco ocupa 64 bits. Suponha-se que se utiliza um sistema de ficheiros FAT. Quanto espaço deverá ser reservado para a File Allocation Table?
    A. 1GB
    B. 128MB
    C. 32GB
    D. Nenhuma das anteriores

Justificação:
Número de blocos = 1TB/8KB=2^40/2^13=2^27
Cada referência ocupa 8 (2^3) bytes.
Como a File Allocation Table *precisa ter tantas entradas (de 8 bytes) quanto o número de blocos, ela ocupa* 2^27 * 2^3 = 2^30 bytes = 1 GB

12. [1,6v] Considere um ficheiro de 1000 KB guardado num sistema de ficheiros EXT3, onde o tamanho de bloco é de 1KB e a referência para cada bloco ocupa 4 bytes. Quantos blocos de dados são utilizados pelo ficheiro, incluindo não só os dados do ficheiro mas também os seus meta-dados?
    A. 1018
    B. 1017
    C. 1016
    D. Nenhuma das anteriores

Justificação:
Os dados do ficheiro ocupam 1000 blocos.
Além desses blocos, é preciso guardar também as referências para estes blocos.
Os primeiros 12 blocos de dados são referenciados diretamente pelo I-NODE (logo não contam para a resposta).

Sobram 988 blocos de dados, que precisam ser referenciados usando as referências indiretas suportadas pelo EXT3.
Usando 1 nível de indireção, podemos, através de ***um bloco de referências****, referenciar mais 256 blocos de dados (256 = tamanho de bloco divido pelo tamanho das referências aos blocos).*

Sobram ainda 732 (1000-12-256) blocos. Estes podem ser referenciados através de ***3 blocos*** *de nível 2 (cada bloco de nível 2 podendo memorizar referências para 256 blocos). É preciso também alocar* ***mais um bloco*** *de nível 1 para interligar o i-node aos 3 blocos de nível 2.*

Isto dá um total de 1005 blocos.

13. [1,3v] Entre as primeiras 3 afirmações, há alguma FALSA?
    Se sim, indique qual. Se não, responda d).
    A. Se existisse apenas uma tabela de ficheiros abertos **global** seria possível a um utilizador forjar um identificador para um qualquer ficheiro já aberto nessa tabela, quebrando assim o isolamento.
    B. Se existisse apenas uma tabela de ficheiros abertos por processo o núcleo não poderia sincronizar o acesso ao cursor de um ficheiro aberto por um processo P e herdado por um processo filho de P
    C. A tabela de ficheiros abertos **por processo** é mantida no espaço de endereçamento do utilizador.
    D. Responda D se nenhuma das afirmações acima for falsa.