LEIC/LETI – 2023/24 – 1º Exame de Sistemas Operativos

19 de Janeiro de 2024, Duração: 2h00m

- Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.
- 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. Para cada pergunta pode selecionar uma ou zero 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 1 [Sistemas de Ficheiros, 3.3v]

1) [0.6v] Se o directório corrente de um processo é '/home/maria/SO' e o processo executa:
   open(fd,'../../../usr/tmp/fifo.txt')

Em que diretório deve existir o ficheiro fifo.txt para que a chamada open possa ser bem sucedida:
   a) /home/maria/tmp
   b) /usr/tmp
   c) /tmp
   d) Nenhuma das respostas anteriores é correta.

   b

2) [0.6v] Qual destas afirmações acerca dos inode no ext3 é **verdadeira**:

   a) O número de blocos disponíveis para um ficheiro é igual ao número de entradas do array i_block no inode.
   b) O número de hard links que podem existir num sistema de ficheiro ext3 não pode ultrapassar o número de inodes existentes.
   c) Os inodes permitem evitar fragmentação interna.
   d) O número de ficheiros e diretórios distintos que podem existir num sistema de ficheiros ext3 não pode ultrapassar o número de inodes existentes.

   d

3) [1,5v] No EXT3, se o tamanho de bloco for 4KB e as referências para os blocos ocuparem 64 bits, quantos blocos de disco são utilizados para armazenar as referências para os blocos de um ficheiro de tamanho 704KB? Na conta, excluir o espaço ocupado em disco pelo inode do ficheiro.
   a) 4
   b) 3
   c) 2
   d) Nenhuma das respostas anteriores.

   d

4) [0.6v] Qual das seguintes afirmações sobre a chamada de sistema *dup* é **verdadeira**:
   a) A chamada de sistema *dup* cria uma cópia de um ficheiro.
   b) A chamada de sistema *dup* lê dados do *standard input.*
   c) A chamada de sistema *dup* devolve uma cópia do descritor de ficheiro passado em input. Os dois descritores de ficheiros partilham o mesmo cursor.
   d) A chamada de sistema *dup* devolve uma cópia do descriptor de ficheiro passado em input. Os dois descritores de ficheiros têm cursores distintos.

c

Grupo 2 [Processos e tarefas, 2.85v]

1. [0.75v] Considere este programa:

```
#include <sys/wait.h>
#include <stdio.h>

int main(int argc, char **argv) {
    int pid, i;
    i=1;

    if (argc < 2) return 1;

    pid=fork();

    if (pid==0) {
        i=i*10;
        printf("F1:%d\n",i);
        char *path = argv[1];
        status=execl(path, path, 0);
        printf("F2:%d\n",status);
    }
    else if (pid>0) {
        i=i*5;
        printf("P:%d\n",i);
    }
}
```

Qual destes outputs pode ser produzido supondo que a chamada a *exec* seja bem sucedida e que o programa executado pela *exec* não produz qualquer output?

a. P:5
   F1:10
b. F1:10
   P:5
   F2:0
c. F1:10
   P:50
d. Nenhum dos outputs acima é possível.

a

2. [0.6v] Qual das seguintes afirmações é **verdadeira** acerca do bit setuid quando aplicado a um ficheiro executável num sistema operativo Unix?
    a. O executável corre com os privilégios do dono do ficheiro executável.
    b. Concede permissões de leitura e escrita a todos os utilizadores.
    c. Restringe o acesso ao arquivo apenas ao utilizador *root* .
    d. O executável corre com os privilégios de *root.*

    a

3. [0.75v] Considere este programa:

```
struct MyStruct {
    int x;
    int y;
};


void *threadFn(void *arg) {
 struct MyStruct r;
 r.x = 5;
 r.y = 5;
 return &r;
}

int main (void) {
    struct MyStruct *s;
    pthread_t tid;
     if (pthread_create (&tid, 0, threadFn, NULL) < 0) {
        ...
        }
    ...
    pthread_join(tid, &s);
    printf("Tarefa devolveu: %d, %d\n", s->x, s->y);
}
```

Qual destes outputs pode ser produzido, supondo que: i) as chamadas a pthread_create e pthread_join sejam bem sucedidas, e ii) que o único printf do programa seja o presente na última linha do programa?

    a. `Tarefa devolveu: 5, 5`
    b. Nenhum output é produzido.
    c. O output do programa não pode ser previsto.
    d. Nenhum das respostas anteriores é correta.

    c

4. [0.75v] Considere este programa:

```
void *threadFn(void *arg) {
    int* i= (int*) arg;
    printf("T:%d\n",*i);
                ...
}

int main (void) {
    pthread_t tid[3];

    for (int i=0; i< 3; i++){
        if(pthread_create (&tid[i], 0, threadFn, &i)< 0) {  ... }
    }
    ...
}
```

Qual destes outputs **não** pode ser produzido pelo programa?

a. T:2
   T:2
   T:2

b. T:0
   T:1
   T:2

c. T:0
   T:2
   T:2

Indique d. caso todos os outputs anteriores sejam possíveis.

d

Grupo 3 [Exclusão mútua, 1.8v]

1. [0,6v] Qual das seguintes afirmações sobre as implementações da abstração do mutex é **verdadeira**:
   a) As instruções que garantem acesso atómico (ex.,Test-and-Set/BTS) só funcionam em processadores single-core.
   b) As instruções que inibem as interrupções são necessárias para implementar a abstração de mutex e não podem ser invocadas em modalidade utilizador.
   c) No algoritmo da padaria de Lamport apresentado nas teóricas se não for utilizada a variável "escolha" (também referida como a "tampa" da caneta) pode ocorrer o problema da míngua (starvation)
   d) Na implementação do mutex gerida pelo núcleo do sistema operativo apresentada nas teóricas pode existir espera ativa enquanto está a ser verificado/alterado o estado do mutex, mas é evitada a espera ativa durante a execução da seção crítica que o mutex protege.

d

2. [0,6v] Qual das seguintes afirmações acerca de mutexes e read-write locks é **verdadeira**:
   a. A implementação de um read-write lock é mais leve e eficiente do que a implementação de um mutex.
   b. Mutexes e read-write locks são intercambiáveis e podem ser usados de forma equivalente em qualquer cenário.
   c. Os mutexes são mais adequados para sincronização grosseira (coarse-grained locking), enquanto os read-write locks são mais adequados para sincronização fina (fine-grained locking).
   d. As afirmações acima são todas falsas.

   d

3. [0,6v] Qual das seguintes afirmações é **verdadeira** acerca das soluções do problema do jantar dos filósofos apresentadas nas aulas teóricas:
   a. Sem a utilização do mecanismo de recuo aleatório, a sincronização baseada em try_lock() pode gerar problemas de inter-blocagem (*deadlock)*.
   b. A utilização do mecanismo de recuo aleatório visa evitar, pelo menos probabilisticamente, que a sincronização baseada em try_lock() possa sofrer de míngua (*livelock)*.
   c. O mecanismo baseado em try_lock() é mais eficiente do que a sincronização baseada na utilização ordenada de um conjunto de locks através da chamada lock().
   d. As afirmações acima são todas falsas.

   b

Grupo 4 [Semáforos e variáveis de condição, 3.8v]

1. [2v] Complete a seguinte implementação do problema dos leitores-escritores, preenchendo as "caixas" no código abaixo, com zero, uma ou mais de que uma instrução, entre as listadas de seguida. Cada instrução listada pode ser utilizada mais do que uma vez.

```
a. escritores_espera > 0
b. escritores_espera == 0
c. leitores_espera > 0
d. leitores_espera == 0
e. em_escrita
f. !em_escrita
g. nleitores == 0
h. nleitores != 0
i. nleitores>0
j. pthread_mutex_lock(&secCritica);
k. pthread_mutex_unlock(&secCritica);
l. sem_post(&leitores);
m. sem_post(&escritores);
n. sem_wait(&leitores);
o. sem_wait(&escritores);
p. &escritores
q. &leitores
```

```
fechar_leitura()
{
   1.j
   if (  2. e   || escritores_espera > 0) {
      leitores_espera++;
      3.k
      sem_wait(  4. q  );
      5. i
      if (leitores_espera >0) {
          nleitores++;
          leitores_espera--;
          6. l
      }
   }
   else
          nleitores++;
   7. k
}
```

```
abrir_leitura()
{
   8. j
   nleitores--;
   if (  9. g  &&
escritores_espera > 0){
      10 . m
      em_escrita=TRUE;
      escritores_espera--;}
   11. k
}
```

```
fechar_escrita()
{
   12. j
   if (em_escrita ||  13. i  ) {
      escritores espera++;
      14. k
      sem_wait(  15. p  );
      16. j
   }
   em_escrita = TRUE;
   17. k
}
```

```
abrir_escrita()
{
   int i;
   18. j
   em_escrita = FALSE;
   if (  19. c  ){
         20. l
         nleitores++;
         leitores_espera--;
   }
   else if (  21. a  ) {
      22. m
      em_escrita=TRUE;
      escritores_espera--;
   }
   pthread_mutex_unlock(&secCritica);
}
```

NOTA: Na alínea 13 foi atribuída cotação de 50% à opção "c" (esta opção pode gerar míngua dos escritores mas não leva a violação das propriedades de "safety").

2. [1.8v] Pretende-se implementar um vídeo jogo em que existem três classes de tarefas:
   - Uma tarefa que simula uma mina de ouro (*goldMineThread*), que produz unidades de ouro. A tarefa produz 3 unidades de ouro a cada 10 segundos e informa todos os mineiros da disponibilidade de novo ouro usando a variável de condição *minerCond*. A produção de ouro nunca é bloqueada.
   - Um conjunto de tarefas que simulam mineiros de ouro (*minerThread*). Cada mineiro bloqueia até haver ouro disponível na mina. Assim que houver ouro, o mineiro extrai uma unidade de ouro e acumula-a na variável *minedGold.* Quando a quantidade de ouro extraído atingir ou ultrapassar a constante MINERS_CAPACITY, deve-se i) bloquear a extração de ouro por todos os mineiros usando a variável de condição *minerCond* e ii) informar a tarefa banco (*bankThread*) usando a variável de condição *bankCond*. Depois de ter executados estas atividades, os mineiros dormem durante 15 segundos, antes de voltarem a repetir o ciclo do jogo.
   - Uma tarefa que simula o funcionamento de um banco (*bankThread*). O banco é notificado por um mineiro assim que a capacidade da mina for atingida e, depois de 2 segundos, transfere o valor na variável *minedGold* para a variável *bankGold* e desbloqueia, usando a variável de condição *responseCond,* o mineiro que tinha avisado o banco.

Preencha as "caixas" no código abaixo, com zero, uma ou mais de que uma instrução, entre as listadas de seguida:

A. minedGold < MINERS_CAPACITY
B. minedGold > MINERS_CAPACITY
C. pthread_cond_signal(&minerCond);
D. pthread_cond_broadcast(&minerCond);
E. pthread_cond_wait(&minerCond, &mutex);
F. pthread_cond_signal(&bankCond);
G. pthread_cond_broadcast(&bankCond);
H. pthread_cond_wait(&bankCond, &mutex);
I. pthread_cond_signal(&responseCond);
J. pthread_cond_broadcast(&responseCond);
K. pthread_cond_wait(&responseCond, &mutex);
L. pthread_mutex_lock(&mutex);
M. pthread_mutex_unlock(&mutex);
N. while
O. if

```
#define MINERS_CAPACITY 10
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t minerCond = PTHREAD_COND_INITIALIZER;
pthread_cond_t bankCond = PTHREAD_COND_INITIALIZER;
pthread_cond_t responseCond = PTHREAD_COND_INITIALIZER;

int goldMineCount = 100;
int minedGold = 0;
int bankGold = 0;
```

```
void *goldMineThread(void *arg) {
    while (1) {
        sleep(10);

        1 L

        goldMineCount += 3;
        printf("Gold Mine produced 1 unit.\n
               Total: %d\n", goldMineCount);

        2. D

        3. M

    }
}


void *minerThread(void *arg) {
    while (1) {
        4. L

        5. N  (goldMineCount == 0 ||
               minedGold >= MINERS_CAPACITY) {

            6. E

        }

        7.

        goldMineCount -= 1;
        minedGold += 1;

        printf("Miner mined 1 gold unit.
               Gold in mine: %d, Mined gold:%d\n",
               goldMineCount,minedGold);


        if (minedGold >= MINERS_CAPACITY) {
            8. F

            9. K

        }

        10. M

        sleep(15); // Time to rest
    }
}
```

```
void *bankThread(void *arg) {
 while (1) {
    11. L

    while ( 12. A ) {
            13. H
        }

    14. M

    printf("Bank processing...\n");

    sleep(2);

    15. L

    bankGold+=minedGold;
    minedGold=0;

    printf("Bank processed the request.\n
        Bank gold: %d.\n
       Notifying the Miner.\n",bankGold);

    16. I

    17. M

 }
```

NOTA: Nas alíneas 2 e 3 e 16,17 foram consideradas válidas (100% da cotação) também respostas onde a variável de condição é assinalada depois de ter largado o mutex (embora estas abordagens possam gerar "spurious wake-ups")

Grupo 5 [Signals 1.35v]

1. [0,6v] Se um signal for enviado a uma aplicação multi-tarefa (com várias *threads*), quais são as tarefas que irão executar a rotina de tratamento desse signal?

a. Todas as tarefas que não tenham chamado anteriormente sigprocmask.
b. Qualquer tarefa que não esteja bloqueada.
c. Qualquer tarefa que esteja executável.
d. Qualquer tarefa que não tenha incluído esse signal na sua máscara de signals bloqueados.

d

2. [0,75 v.] Considere o seguinte programa:

```
ApanhaSig() {
  printf("PID %d Signal tratado \n", getpid());
  signal(SIGINT, ApanhaSig);
}

main () {
  int res;
  res = fork();
  if (res ==0) {
    signal(SIGINT, ApanhaSig);
  } else {
    sleep(5);
    for (;;) {
      if (kill(res, SIGINT) == -1) exit();
    }
  }
  nice (10);
  for (;;);
}
```

Se algum processo executar a função nice, que processo será?

a. O processo pai.
b. O processo filho.
c. Ambos os processos.
d. É impossível algum dos processo executar nice devido ao ciclo infinito.

b

Grupo 6 [Pipes e escalonamento 3,6v]

1. [0.6v] Qual das seguintes afirmações sobre Pipes e FIFOs (ou Pipes com nomes) é **verdadeira**?
   a) A leitura de um pipe pode ser bloqueante e a de um FIFO também.
   b) O open do pipe nunca é bloqueante e o do FIFO pode ser.
   c) São idênticos em termos de mecanismo de comunicação e de atribuição de nome.
   d) A criação de um pipe é idêntica à de um ficheiro.

   a

2. [0,6v] Qual das seguintes afirmações é **verdadeira**:
   a. O núcleo do sistema operativo é ativado apenas caso ocorra uma interrupção causada pelo hardware.
   b. O núcleo não utiliza uma pilha (stack) diferente da usada pelas aplicações.
   c. A última instrução executada pelo despacho é Return from Interrupt (RTI).
   d. A comutação entre processos implica custos menores do que a comutação entre tarefas do mesmo processo.

   c

3. [1,8v] Em Linux, monitorizou-se a execução de um CPU durante um período de tempo e observou-se que o CPU foi partilhado entre 2 processos, P1 e P2, da seguinte forma:

a) [0,6v] Um dos processos corre um programa interativo. Pela informação acima, qual é esse processo?

   a. P1
   b. P2
   c. Ambos
   d. Nenhum

   b

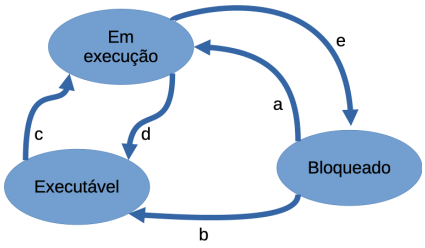
b) [0,6v] Assumindo que ambos os processos iniciaram a sua execução com prioridade base igual. Ao fim do período monitorizado, qual dos processos é mais prioritário?
   a. P1
   b. P2
   c. Ambos
   d. Nenhum

   b

c) [0,6v] Assumindo que o algoritmo de escalonamento seja o CFS, após o período monitorizado, quando P1 ou P2 voltarem a receber execução, qual receberá um quantum superior?
a. P1
b. P2
c. Ambos
d. Nenhum

b

4. [0,6v] Considere o seguinte diagrama de transições de estado de processos. Indique qual das transições representadas **não existe** no sistema Unix:



A: e
B: a
C: d
D: b

B

Grupo 7 [Gestão da Memória, 3.3v]

1. [1.8v] Considere um sistema com uma arquitectura paginada de memória virtual de 32 bits. Neste sistema, cada endereço virtual é dividido em 11 bits (menos significativos) de deslocamento e 21 bits de base (mais significativos).

a) [0.6v] Qual a dimensão das páginas deste sistema?
a. 11^2
b. 1100 bytes
c. 4k bytes
d. 2048 bytes

d

b) [0.6v] Quantas linhas pode ter no máximo a tabela de páginas de um dado processo?
a. 21k linhas
b. 2^{11} linhas
c. 4 Giga linhas
d. 2^{21} linhas

d

c) [0.6v] Escolha a afirmação **correcta**. Em sistemas baseados em paginação:

a. Ao aumentar o tamanho da página a fragmentação interna aumenta.

b. Ao reduzir o tamanho da página a fragmentação externa aumenta.

c. Em sistemas baseados em paginação, ao aumentar o tamanho das páginas reduz-se a latência caso ocorra uma falta de página.

d. De cada vez que um processo acede a um endereço de memória, o núcleo do sistema operativo efetua a tradução do endereço virtual para o correspondente endereço físico.

a

2. [1.5v] Considere um sistema de memória paginada, com um único nível de paginação, em que cada página tem 4K. Mais precisamente, o formato do endereço é:

Página (20 bits)	Deslocamento (12 bits)

Considere que na execução de um processo a sua tabela de páginas (simplificada) era a descrita em seguida e que o TLB tinha as seguintes entradas.

Tabela de Páginas

Presente	Número de página	Base	Protecção
S	00000	20000	r-x
N	00001	-	r-x
S	00002	AA100	r-x
N	00003	FE74B	rw-

TLB

Validade	Número de página	Base	Protecção
S	00002	AA100	r-x
N	00040	5B000	r-x
S	00000	20000	r-x
S	00003	FE74B	rw-

Considere que o programa gera os seguintes acessos (na ordem especificada pela tabela abaixo). Preencha todos os campos relevantes da tabela da consequência da tradução do endereço. Para resolver o problema poderá eventualmente necessitar de novas page frames, poderá arbitrar um valor ou por exemplo usar 00300 ou 30000.

Endereço	Tipo de acesso	Endereço físico	Resolvido no TLB	Resolvido na tabela de páginas	Falta de página ou Acesso invalido (especificar causa se sim)
000008A2	execução	20000802	Y	N	OK
0000209C	leitura	AA10009C	Y	N	OK
00002C65	leitura	AA100C65	Y	N	OK
00004FFA	escrita	N/A	N	Y	Endereço inválido
00002CBF	escrita	AA100CBF	Y	N	Violação da proteção
00001980	leitura	30000980	N	Y	Falta de página

Nota:

- Dado que no enunciado havia uma inconsistência entre o estado da tabela de páginas e o do TLB na entrada associada à página 2, para o penúltimo acesso foram consideradas corretas também as respostas que indicam o acesso como válido.