LEIC/LETI – 2024/25 – 1º Exame de Sistemas Operativos
24 de Janeiro de 2025, Duração: 2h00m

- Responda na **folha das respostas**, apenas no espaço fornecido. Use exclusivamente letras maiúsculas nas respostas. **Identifique a folha de resposta com número de aluno e nome legíveis.**
- Nas perguntas de escolha múltipla existe apenas uma resposta certa. Para cada pergunta pode selecionar zero ou uma opções. A nota é calculada pelas opções que escolher na sua resposta, da seguinte forma:
  - Zero opções corresponde a zero valores.
  - Uma opção correcta corresponde à cotação total da pergunta. Uma opção errada desconta 1/3 da cotação da pergunta.
  - Inserir mais do que uma opção, anula a pergunta.
  - Nas perguntas que estiverem assinaladas como "sem desconto" serão dados zero valores a respostas erradas contendo uma opção ou mais opções erradas.

Grupo 1 - Sistemas de Ficheiros [2,95 val.]

1) [0.8 v.] Considere que o tamanho do ficheiro **`/tmp/a.exe`** (com permissões **`-rw-rw-rw-`**) é de 2Kbytes num sistema de ficheiros ext3 com blocos de 1024 bytes. Após a execução do programa seguinte, qual destas alterações do ficheiro **`/tmp/a.exe`** se verifica?

```
// ...includes omitidos...
// STDOUT_FILENO está definido como 1
#define BUFFSZ 1024

void main() {
  char buff[BUFFSZ];
  memset(buff, '\0', BUFFSZ); //enche buff com BUFFSZ caracteres '\0'
  int fd = open("/tmp/a.exe", O_WRONLY | O_TRUNC);
  close(STDOUT_FILENO);
  dup(fd);
  write(STDOUT_FILENO, buff, BUFFSZ);
  close(fd);
}
```

A. O tamanho do ficheiro fica com 2 KB.
B. Um open com O_WRONLY (modo de escrita) e O_TRUNC (truncar o ficheiro) falha.
C. O número de blocos fica com o valor 3.
D. O *offset* do descritor **`fd`** imediatamente antes do **`close`** está na posição 1024.

2) [0,6 v.] Qual destas afirmações acerca dos *inodes* no ext3 é **falsa**:

A. O número de blocos de dados de um ficheiro pode ser superior ao número de entradas no *inode*.
B. O número de *hard links* que podem existir num sistema de ficheiro ext3 **não(*)**[1] pode ultrapassar o número de *inodes* existentes.
C. Os *inodes* contêm o dono do ficheiro.
D. O número de ficheiros distintos que podem existir num sistema de ficheiros ext3 não pode ultrapassar o número de *inodes* existentes.

3) [0,75 v.] No EXT3, se o tamanho de bloco de dados for 1KB 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 113KB? Na conta, excluir o espaço ocupado em disco pelo *inode* do ficheiro.

A. 2
B. 0

1 No enunciado distribuido no exame havia uma gralha (faltava o "não" nesta alinea) e portanto foram aceites todas as respostas

C. 1
D. Nenhuma das respostas anteriores.

4) [0,6 v.] Qual das seguintes afirmações sobre a chamada de sistema **`mkfifo`** é **verdadeira**:
A. A chamada de sistema **`mkfifo`** necessita que sejam sempre indicados o *path* do ficheiro e as permissões.
B. A chamada de sistema **`mkfifo`** abre um *pipe* para leitura síncrona.
C. A chamada de sistema **`mkfifo`** necessita que seja indicado o nome do ficheiro e o modo de abertura.
D. A chamada de sistema **`mkfifo`** devolve um descritor de ficheiro inteiro representando a posição do cursor do ficheiro.

Grupo 2 - Processos e Tarefas [2,75 val.]

1. [1 v.] O programa abaixo foi executado por um processo com PID=11. Durante essa execução, foi criado um processo filho com PID=12.

```
int main() {
  pid_t pid = 11;
  if (pid = fork ()) {
    wait();
    printf("%d, %d\n", pid, getpid());
  } else {
    printf("%d, %d\n", pid, getppid());
    exit(0);
  }
}
```

Qual dos seguintes outputs será apresentado?

A. 11, 11
11, 11
B. 0, 11
12, 11
C. 12, 11
0, 11
D. 0, 12
12, 11

2. [0,75 v.] No sistema Unix qual das seguintes afirmações é verdadeira quanto ao que distingue o modo utilizador e o modo núcleo?
A. As operações máquina que executam I/O (e.g. IN e OUT) são acessíveis em modo utilizador.
B. Um aplicação não pode fazer o processador comutar para modo núcleo.
C. Uma chamada de sistema usa parâmetros fornecidos pela aplicação e executa código do núcleo.
D. Para se converter um endereço de memória virtual em físico é preciso estar em modo núcleo.

3. [1 v.] Considere este programa:

```
int a=0;

void *threadFn(void *arg) {
     usleep(10); //dormir 10 microsegundos
     a++;
     printf("T: %d\n", a);
}

int main (void) {
    pthread_t tid[3];
    for (i=0; i< 3; i++){
         if(pthread_create (&tid[i], 0, threadFn, )< 0) { ... }
     }
    printf("M: %d\n", a);
}
```

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

A. T: 1
T: 2
M: 2
T: 3

B. M: 0
T: 3
T: 3
T: 3

C. T: 1
T: 1
T: 2
M: 2

D. Indique **D** se todos os outputs anteriores forem possíveis.

Grupo 3 - Exclusão Mútua [1,8 val.]

1. [0,6 v.] Qual das seguintes afirmações sobre as implementações de uma **variável de condição** é **verdadeira**:
   A. Todas as chamadas às funções de uma variável de condição (esperar, etc.) têm de ser feitas dentro de uma secção crítica do trinco associado.
   B. Não é possível desbloquear todas as tarefas bloqueadas numa variável de condição.
   C. As chamadas à função assinalar de uma variável de condição nunca são bloqueantes.
   D. Uma variável de condição é inicializada a um dado valor e nunca descer abaixo de zero.

2. [0,6 v.] Relativamente à implementação de trincos escolha a afirmação falsa.
   A. O algoritmo de Bakery implementa um trinco sem recorrer a mecanismos de *hardware* dedicados para o efeito.
   B. Não é possível implementar trincos sem espera activa se o núcleo não bloquear as tarefas sem acesso ao trinco.
   C. Um trinco implementado usando mecanismos de *hardware* dedicados (operação de *test-and-set* ou *compare-and-swap*) evita a espera activa.
   D. Nos trincos que suportam *try lock,* o *unlock* não é bloqueante.

3. [0,6 v.] Qual das seguintes afirmações é **verdadeira** acerca das soluções do problema do jantar dos filósofos apresentadas nas aulas teóricas:
   A. O problema do jantar dos filósofos só é resolúvel com uma numeração aleatória dos filósofos.
   B. A utilização do mecanismo de recuo aleatório resolve a sincronização entre filósofos estabelecendo uma ordem prévia pela qual os filósofos acedem aos garfos.
   C. O mecanismo baseado em `try_lock()` é sempre 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.

Grupo 4 - Semáforos e variáveis de condição [3,9 val.]

1. [2,1 v.] Considere a seguinte solução para o problema dos produtores-consumidores, em que os itens são inteiros positivos e -1 indica uma posição vazia :

```
int buf[N]; //todas as posições a -1
int prodptr=0, consptr=0;
trinco_t trinco;
semaforo_t pode_prod = criar_semaforo(N),
   pode_cons = criar_semaforo(0);

void colocarItem(int item) {
  esperar(pode_prod);
  fechar(trinco);
  buf[prodptr] = item;
  prodptr = (prodptr+1) % N;
  abrir(trinco);
  assinalar(pode_cons);
}
```

```
int obterItem() {
  int item;
  esperar(pode_cons);
  fechar(trinco);
  item = buf[consptr];
  buf[consptr] = -1
  consptr = (consptr+1) % N;
  abrir(trinco);
  assinalar(pode_prod);
  return item;
}
```

Indique que comportamentos incorretos poderiam ocorrer caso aplicasse as seguintes alterações ao programa acima:

a) [0,7 v.] Caso se omitisse o trinco em colocarItem; ou seja, se se eliminassem as linhas fechar(trinco) e abrir(trinco) dessa função:

A. O vector poderia ficar com posições com valores válidos (superiores a -1), uns já retirados e outros não.
B. O tampão poderia ficar com mais de N itens.
C. Poderia ser colocado um item no vector buf numa posição contendo um item ainda não retirado.
D. Poderia ser retirado o mesmo item do vector buf duas vezes.

b) [0,7 v.] Caso o semáforo pode_cons fosse inicializado a N unidades (em vez de 0 unidades):

A. Poderiam ser retirados itens com o valor -1.
B. Haveria míngua de alguns consumidores.
C. Poderiam ser colocados itens por cima de outros itens ainda não retirados.
D. Os produtores e consumidores acabariam por entrar em interblocagem (*deadlock*).

c) [0,7 v.] Caso as linhas "esperar(pode_prod);fechar(trinco);" trocassem de ordem, passando para "fechar(trinco);esperar(pode_prod);":

A. A chamada a esperar(pode_prod) passaria a causar um erro que terminaria o processo.
B. Um produtor que tentasse produzir quando o vector buf estivesse cheio causaria o bloqueio dos consumidores subsequentes.
C. Um produtor que tentasse produzir quando o buffer estivesse cheio causaria a míngua dos consumidores.
D. A chamada a esperar(pode_prod) passaria a causar um erro que terminaria a tarefa do produtor que a executasse.

2. [1,6 v. sem desconto] Pretende-se implementar um programa que representa uma instalação agro-pecuária:

- Um robot que recolhe ovos postos por galinhas. O robot recorre um número variável de ovos (entre 2 e 10) e coloca-os à disposição de uma máquina de embalar ovos a cada minuto. A embaladora tem um espaço limitado de armazenamento temporário. O robot deve bloquear-se (por ex. em **`condicaoOvosEmbaladora`**) se não houver espaço de armazenamento de ovos na embaladora.
- Outra tarefa representa a embaladora que coloca ovos em caixas com 6 ovos cada. Assim que tiver armazenado pelo menos 6 ovos, a embaladora enche uma caixa a cada 15 segundos. A embaladora também se deve bloquear se não houver ovos ou espaço para mais caixas cheias (por ex. em **`condicaoOvosEmbaladora`** e/ou em **`condicaoCaixasEmbaladora`**).
- Existem outras tarefas que rotulam as caixas a cada 10 segundos que passam. O programa em causa não contempla outros passos depois das caixas rotuladas. As rotuladoras não podem trabalhar se não houver caixas de ovos cheias (recorrendo por exemplo à **`condicaoRotuladora`**).

Para isso indique na folha de respostas, para as instruções marcada a negro no código que se segue, qual a instrução da lista abaixo que deve ser usada.

```
A. pthread_cond_signal(&condicaoOvosEmbaladora);
B. pthread_cond_broadcast(&condicaoOvosEmbaladora);
C. pthread_cond_wait(&condicaoOvosEmbaladora, &mutex);
D. pthread_cond_signal(&condicaoRotuladoras);
E. pthread_cond_broadcast(&condicaoRotuladoras);
G. pthread_cond_wait(&condicaoRotuladoras, &mutex);
K. pthread_cond_signal(&condicaoCaixasEmbaladora);
W. pthread_cond_broadcast(&condicaoCaixasEmbaladora);
X. pthread_cond_wait(&condicaoCaixasEmbaladora, &mutex);
```

```
#define LIM_OVOS_EMBALADORA 60
#define LIM_CAIXAS_EMBALADORA 100

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condicaoOvosEmbaladora = PTHREAD_COND_INITIALIZER;
pthread_cond_t condicaoCaixasEmbaladora = PTHREAD_COND_INITIALIZER;
pthread_cond_t condicaoRotuladoras = PTHREAD_COND_INITIALIZER;
int ovosRecolhidos = 0, caixasCheias = 0;

void *Robot(void *arg) {
      int ultimaRecolha;
      while (1) {
       ultimaRecolha = (rand() % 9) + 2;
       sleep(60);
       pthread_mutex_lock(&mutex);
       while (ovosRecolhidos > LIM_OVOS_EMBALADORA-ultimaRecolha) {
             1. pthread_cond_wait(&condicaoOvosEmbaladora, &mutex);
       }
       ovosRecolhidos += ultimaRecolha;
       printf("Robot recolheu %d ovos.\n Total: %d\n", ultimaRecolha,
ovosRecolhidos);
       2. pthread_cond_signal(&condicaoOvosEmbaladora);
       pthread_mutex_unlock(&mutex);
      }
}

void *Embaladora(void *arg) {
      while (1) {
       pthread_mutex_lock(&mutex);
       while ( caixasCheias == LIM_CAIXAS_EMBALADORA)
             3. pthread_cond_wait(&condicaoCaixasEmbaladora, &mutex);
       while ( ovosRecolhidos < 6)
             4. pthread_cond_wait(&condicaoOvosEmbaladora, &mutex);
      pthread_mutex_unlock(&mutex);

      sleep(15);

      pthread_mutex_lock(&mutex);
       ovosRecolhidos -= 6;
       caixasCheias += 1;
       printf("Embaladora encheu uma caixa\nOvos Armazenados: %d, Caixas Cheias:%d\
n", ovosRecolhidos, caixasCheias);
      pthread_mutex_unlock(&mutex);

       if (ovosRecolhidos <= LIM_OVOS_EMBALADORA-2)
             5. pthread_cond_signal(&condicaoOvosEmbaladora);

       if (caixasCheias >= 1)
             6. pthread_cond_broadcast(&condicaoRotuladoras);
      }
}

void *Rotuladora(void *arg) {
      while (1) {
       pthread_mutex_lock(&mutex);
       while (caixasCheias == 0) {
             7. pthread_cond_wait(&condicaoRotuladoras, &mutex);
       }
       caixasCheias -=1;
       printf("A rotuladora rotulou uma caixa.\n");
       if (caixasCheias == LIM_CAIXAS_EMBALADORA-1) {
             8. pthread_cond_signal(&condicaoCaixasEmbaladora);
       }
```

```
        pthread_mutex_unlock(&mutex);
        sleep(10);
       }
}
```

Grupo 5 - Signals [1,8 val.]

1. [0,6 v.] Qual das seguintes chamadas de sistema permite que um processo suspenda sua execução até que um sinal seja recebido?

A. `pause()`
B. `kill()`
C. `wait()`
D. `sigprocmask()`

2. [0,6v] Quais das seguintes afirmações sobre *signals* é verdadeira:

A. Nos *signal handlers* não é seguro usar a função `printf`.
B. A *system call* `kill()` permite enviar só *signals* que causam a terminação do processo que recebe o *signal.*
C. Escrever para um *pipe* onde já não existem leitores pode gerar um `SIG_CHILD`.
D. Nenhuma das anteriores é verdadeira.

3 [0,6] Em sistemas Unix, quando um processo recebe um *signal* e o *handler* de *signal* associado àquele *signal* é executado, qual das seguintes afirmações sobre o comportamento do processo após a execução do *handler* de *signal* é verdadeira?

A. O processo continuará imediatamente após a execução do *handler*, como se o signal nunca tivesse ocorrido.
B. O processo será reiniciado, após a execução do *handler* de signal.
C. O processo será imediatamente encerrado após a execução do *handler*, independentemente do que o *handler* fez.
D. O processo pode retormar a execução a partir do ponto onde foi interrompido, mas somente se o *signal* não for bloqueado durante a execução do *handler*.

Grupo 6 - *Pipes* e Escalonamento [3,9 val.]

1. [0,6 v.] Qual das seguintes afirmações sobre *pipes* e FIFOs (ou *pipes* com nomes) é **verdadeira**?
   A. Por omissão, o `open()` de um FIFO é bloqueante, enquanto o `open()` de um *pipe* é não bloqueante.
   B. Os *pipes* são identificados por nomes de ficheiros.
   C. Numa aplicação cliente-servidor, em que o servidor oferece apenas um FIFO para a recepção dos pedidos gerados por todos os clientes, as mensagens de pedido podem ter uma dimensão arbitrária, desde que seja constante.
   D. Ao escrever para um *pipe* onde não existem leitores, o processo escritor recebe um `SIGPIPE`.

2. [0,6 v.] Qual das seguintes afirmações é **verdadeira**:
   A. Quando é instalado um sistema operativo é também atualizado o código do BIOS de forma a ser compatível com o tipo de núcleo a ser ativado na sequência de *boot*.
   B. Um núcleo monolítico é mais resiliente aos *bugs* (por exemplo em *device drivers)* do que um núcleo que utiliza múltiplas camadas com níveis de proteção diferente.
   C. Se fosse possível disabilitar o mecanismo das interrupções em *user* mode, seria possível comprometer o funcionamento do núcleo do sistema operativo.
   D. A utilização de duas pilhas, uma para o núcleo e outra para as aplicações, visa impedir o acesso ao estado interno do sistema operativo às aplicações que correm em *user mode.*

3. [0,6v] Qual das seguintes afirmações é **verdadeira**:
   A. No Unix, a utilização das estruturas `proc` e `u` visa minimizar a CPU que o sistema operativo gasta para gerir processos.
   B. No Unix prioridades numéricas negativas correspondem a prioridade baixas.
   C. O Unix atribui maior prioridade aos processos CPU-bound.
   D. No Unix, se um processo se bloquear indefinidamente a sua prioridade converge para a prioridade base, caso o utilizador não tenha alterado a prioridade do processo usando a *system call* `nice`.

4. [0,75 v.] Numa execução do algoritmo CFS supõe-se que, numa maquina equipada com apenas um CPU core, existam só 2 processos executáveis, onde:
- um processo P1 que nunca se bloqueia;
- um outro processo P2 que executa durante 2ms, bloqueia durante 10 ms, e, finalmente, executa durante mais 5msec, antes de terminar.

Supondo que *minimum granularity* e *targeted scheduling latency* sejam configurados com 1ms e 10ms, respeitivamente.

Qual é a efetiva latência de escalonamento deste sistema:
   A. 10 msec
   B. 20 msec
   C. 1 msec
   D. Nenhuma das respostas anteriores

5. [0,6 v.] Considere o mesmo sistema da pergunta anterior, supondo que : i) P2 é arrancado depois de P1, ii) o vruntime do vruntime do P1 é igual a 100 quando o P2 é iniciado, e iii) P2 recebe o CPU assim que for iniciado. Qual é o valor de vruntime atribuído ao processo P2 quando ele inicia?

   A. 0 msec
   B. 2 msec
   C. 10 msec
   D. Nenhuma das respostas anteriores

6. [0,75 v.] Considere o mesmo sistema da pergunta anterior, depois de quanto tempo real desde o início do processo termina o processo P2?
   A. 17 msec
   B. 20 msec
   C. 22 msec

D. Nenhuma das anteriores

Grupo 7 - Gestão de Memória [3,3 val.]

1. Considere um sistema com uma arquitectura paginada de memória virtual de 32 bits. Neste sistema, cada endereço virtual é dividido em 10 bits (menos significativos) de **deslocamento** e 22 bits de **base** (mais significativos).

a) [0.6 v.] Qual a dimensão das páginas deste sistema?
A. 2^{12} bytes
B. 2^{20} bytes
C. 4k bytes
D. Nenhuma das respostas anteriores é correta.

b) [0,6 v.] Quantas linhas pode ter no máximo a tabela de páginas de um dado processo?
A. 2^12 linhas
B. 2^20 linhas
C. 2^8 linas
D. Nenhuma das respostas anteriores é correta.

c) [0,6 v.] Escolha a afirmação **correcta**. Em sistemas baseados em paginação:
A. O núcleo é encarregue de traduzir todos os acessos a endereços de mémoria virtual para endereços físicos.
B. Em sistemas baseados em paginação podem coexistir páginas de tamanho diferente.
C. Em sistemas baseados em segmentação não pode haver fragmentação interna.
D. O núcleo necessita de ser invocado quando surgir uma falta de página (*page fault)*.

2. [1,5 v.] Considere um sistema de memória paginada de 16 bits, com um único nível de paginação, em que cada página tem 256 bytes. Mais precisamente, o formato do endereço é:

Página (8 bits)	Deslocamento (8 bits)

Considere que a tabela de páginas de um processo contenha o seguinte conteúdo e a TLB está limpa:

Número de Página	Base	Proteções	Presente
0	0x10	r-x	Y
1	0x11	rwx	Y
2	-	r-x	N
3	0x00	rwx	Y

Considere que um processo começa a sua execução, que a TLB está limpa.

Supondo que exista uma TLB completamente associativa com 32 entradas de capacidade. Para cada acesso indique se há um *hit* na TLB, se é necessário aceder à tabela de páginas para a tradução do endereço e se houve excepções (inclusive faltas de páginas) na tradução. Para resolver o problema poderá eventualmente necessitar de novas tramas: nesse caso considere que o núcleo aloca tramas a partir do endereço 0x3000.

Endereço virtual	Tipo de acesso	Endereço Físico	TLB HIT	Necessita aceder à tabela de páginas	Exceções (inclusive faltas de páginas)
0x00FF	execute				
0xFF00	read				
0x0303	write				
0x0000	write				
0x02FF	read				
0x0222	write				

Preencher na folha de respostas!