Número: [ ] Nome: _____

LEIC/LETI – 2024/25 – 2º Exame de Sistemas Operativos
7 de Fevereiro de 2025, Duração: 2h00m

- Responda na **folha das respostas**, apenas no espaço fornecido. Não use letras minúsculas nas respostas. **Identifique este enunciado e 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 [2,9 val.]

1) [1,3 v., sem desconto] Indique na caixa da folha de respostas qual é o conteúdo do ficheiro **`/tmp/a.txt`** depois da execução do programa seguinte. Assuma que o ficheiro referido existe e tem permissões de escrita.

```
// ...includes omitidos...
// STDOUT_FILENO está definido como 1


void main() {
   int fd = open("/tmp/a.txt", O_WRONLY | O_TRUNC);
   close(STDOUT_FILENO);
   dup(fd);
   printf("SO2024/25");
   fflush(std_out);
   lseek(fd,2,SEEK_SET);
   printf("**24/25!");
   close(fd);
}
```

2) [0,6 v.] Qual destas afirmações acerca do sistema de ficheiros do CP/M é **verdadeira**:

A. O sistema de ficheiros do CP/M evita fragmentação externa.
B. O sistema de ficheiros do CP/M evita fragmentação interna.
C. O CP/M armazena a lista de blocos de um ficheiro num *inode* referenciado pela directoria.
D. O CP/M organiza os dados de um ficheiro em segmentos de tamanho variável.

3) [1 v., sem desconto] No sistema de ficheiros 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 12KB ? Na conta, exclua o espaço ocupado em disco pelo *inode* do ficheiro.

Grupo 2 - Processos e Tarefas [3,35 val.]

1. [1v, sem desconto] Indique na folha de respostas qual é o *output* deste programa (não inclua mudanças de linha na resposta):

```
// includes omitidos...
int main() {
      pid_t pid;

      if (pid = fork ()) {
            wait(NULL);
            printf("A\n");
            if (pid = fork ()) {
                  wait(NULL);
                  printf("B\n");
            } else printf("C\n");
      } else printf("D\n");
}
```

2. [0,6 v.] Qual das seguintes afirmações sobre processos e tarefas é **verdadeira**?
   A. A comutação entre tarefas do mesmo processo é mais cara do que a comutação entre processos diferentes.
   B. A seguir a uma chamada **`fork()`** o processo pai e o processo filho podem comunicar lendo/escrevendo qualquer variável global que tenha sido declarada no programa.
   C. Após executar uma chamada **`fork()`**, caso esta seja bem sucedida, o filho recebe como valor de retorno o **`pid`** do processo pai.
   D. A função **`pthread_join`** permite esperar pela terminação de uma tarefa específica.

3. [1 v., sem desconto] Considere este programa:

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

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

      if (argc < 2) return 1;

      pid=fork();

      if (pid>0) {
            wait(NULL);
            i=i*10;
            printf("AA:%d\n",i);
            char *path = argv[1];
            status=execl(path, path, 0);
            i=i/2;
            printf("%d:%d\n",i,status);
      }
      else if (pid==0) {
            i=i*5;
            printf("BB:%d\n",i);
      }
}
```

   Indique na folha de respostas qual é o *output* produzido supondo que a chamada a **`exec`** seja bem sucedida e que o programa iniciado pela chamada à função **`exec`** não produza qualquer *output* (ignore as mudanças de linha).

4. [0,75 v.] Considere este programa:

```
#include <stdio.h>
#include <pthread.h>

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

void *threadFn(void *arg) {
        MyStruct *s = (MyStruct*) arg;
        printf("%d:%d\n", s->x, s->y);
}

int main (void) {
        MyStruct s;
        pthread_t tid[3];
        int a=0;

        for (int i=0; i< 3; i++){
                //Prepara argumentos da próxima thread
                s.x = ++a;
                s.y = a;

                //Cria thread, passando-lhe um user novo
                if(pthread_create (&tid[i], 0, threadFn, &s)== 0) {
                        printf ("Criada a tarefa %ld\n", tid[i]);
                }
        }
        for (int i=0; i< 3;i++){
                pthread_join(tid[i], NULL);
        }
        fflush(stdout);
}
```

Qual dos resultados seguintes **não** pode ser produzido pelas chamadas a **`printf`** na função **`threadFn`**?

A. 1:1
2:2
3:3

B. 2:2
3:3
3:3

C. 1:1
3:3
3:3

Indique **D.** caso todos os resultados anteriores sejam 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. As variáveis de condição bloqueiam quando o contador interno atinge 0.
   B. Caso a chamada à função **`esperar`** numa variável de condição seja bloqueante, o trinco não é largado antes de bloquear a tarefa que invocou esperar.
   C. Quando a chamada a **`esperar`** numa variável de condição retorna, é garantido que a tarefa invocadora possui o trinco associado à variável de condição.
   D. As afirmações acima são todas falsas.

2. [0,6 v.] Relativamente à sincronização baseada em trincos escolha a afirmação **falsa**.
   A. Usar múltiplos trincos finos com granularidade fina permite aumentar o nível de concorrência atingível face a soluções que usam apenas um trinco (granularidade grosseira).
   B. Os *read-write locks* garantem sempre maior desempenho do que mutexes.
   C. A implementação de **`pthread_mutex_lock`** não permite evitar espera ativa.
   D. Se uma aplicação tentar adquirir múltiplos *locks*, pode incorrer no problema da interblocagem.

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 exemplifica a utilização de semáforos e variáveis de condição.
   B. O problema de interblocagem surge dado que o número de filósofos é ímpar.
   C. O problema do jantar dos filósofos poderia ser resolvido se cada filósofo usasse sempre primeiro o garfo à sua direita e a seguir o garfo à sua esquerda.
   D. As afirmações acima são todas falsas.

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

1. [1,7 v., sem desconto] Considere o programa abaixo que implementa o problema do barbeiro em sincronização de tarefas, onde:

   - Um barbeiro corta o cabelo de um cliente de cada vez.
   - Os clientes têm 5 cadeiras para esperarem a sua vez após entrarem na loja.

   Complete o programa, indicando na folha de respostas as letras identificativas das linhas a colocar nas "caixas" no código abaixo, com zero ou uma instrução, entre as listadas de seguida. Cada instrução listada pode ser utilizada mais do que uma vez.

```
A. pthread_mutex_lock(&mutex);
B. pthread_mutex_unlock(&mutex);
C. sem_wait(SemaforoClientes);
D. sem_post(SemaforoClientes);
E. sem_wait(SemaforoBarbeiro);
F. sem_init(SemaforoBarbeiro, 0, 0);
G. sem_init(SemaforoBarbeiro, 0, 5);
H. sem_init(SemaforoBarbeiro, 0, 1);
I. sem_post(SemaforoBarbeiro);
```

Programa a completar:

```
#include <semaphore.h>
#include <pthread.h>
#define NCADEIRAS 5

sem_t* SemaforoClientes;
sem_t* SemaforoBarbeiro;
pthread_mutex_t mutex;
int waiting = 0;

void threadBarbeiro(){
    while(1){
        1. [redacted]
        2. [redacted]
        waiting--;
        3. [redacted]
        pthread_mutex_unlock(&mutex);
        //CortaCabelo();
    }
}

void threadCliente(){
    4. [redacted]
    if(waiting < NCADEIRAS){
        waiting++;
        5. [redacted]
        6. [redacted]
        sem_wait(SemaforoBarbeiro);
        //RecebeCorte();
    } else {
        pthread_mutex_unlock(&mutex);
        //O barbeiro está cheio, voltarei mais tarde
    }
}

int main(){
    sem_init(SemaforoClientes, 0, 0);
    7. [redacted]
    pthread_mutex_init(&mutex, NULL);
}
```

2. [1,7 v., sem desconto] Complete o programa abaixo que implementa o problema dos produtores/consumidores (usando variáveis de condição) indicando na folha de respostas as letras identificativas das linhas a colocar nas "caixas" no código abaixo, com zero ou uma instrução, entre as listadas de seguida. Cada instrução listada pode ser utilizada mais do que uma vez.

```
A. count == N
B. count == 0
C. count != 0
D. count >N
E. pthread_cond_signal(&podeCons);
F. pthread_cond_signal(&podeProd);
G. pthread_cond_broadcast(&podeCons);
H. pthread_cond_broadcast(&podeProd);
I. pthread_mutex_lock(&mutex);
L. pthread_mutex_unlock(&mutex);
M. pthread_cond_wait(&podeProd,&mutex);
N. pthread_cond_wait(&podeCons,&mutex);
```

Programa a completar:

```
int buf[N], prodptr=0, consptr=0, count=0;

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t podeProd = PTHREAD_COND_INITIALIZER;
pthread_cond_t podeCons = PTHREAD_COND_INITIALIZER;


produtor() {
  while(TRUE) {
    // A função produz retorna um inteiro que se pretende armazenar no buffer
    int item = produz();
    1. [redacted]
    while (2. [redacted]) 3. [redacted]
    buf[prodptr] = item;
    prodptr= (prodptr+1)%N;
    count++;
    5. [redacted]
    pthread_mutex_unlock(&mutex);
  }
}

consumidor(){
   while(TRUE) {
      int item;
      pthread_mutex_lock(&mutex);
      while (6. [redacted]) 7. [redacted]
      item = buf[consptr];
      consptr = (consptr+1)%N;
      count--;
      8. [redacted]
      pthread_mutex_unlock(&mutex);
   }
}
```

Grupo 5 - Signals [1,8 val.]

1. [0,6 v.] Qual das seguintes chamadas de sistema permite que o processo com o **`pid`** 10 envie um *signal* SIGINT ao processo com o **`pid`** 20?

A. `kill(10, SIGINT)`
B. `kill(20, SIGINT)`
C. `signal(10, SIGINT)`
D. `signal(20, SIGINT)`

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

A. Nos *signal handlers* não é seguro usar a função **`malloc`**.
B. A *system call* **`kill()`** volta a estabelecer o comportamento por omissão para o processamento de um dado signal.
C. Escrever para um *pipe* onde já não existem leitores gera um **`SIG_PIPE`**.
D. A função **`alarm()`** envia um *signal* ao processo que a invoca.

3. [0,6 v.] Qual das seguintes afirmações sobre *signals* é **verdadeira**:

A. O comportamento por omissão de processamento de um *signal* é SIG_IGN.
B. Um processo executará a rotina de tratamento de *signals* tantas vezes quantas as que receber esse *signal*.
C. A função **`pause`** bloqueia um processo até à chegada de um *signal*.
D. Nenhuma das opções anteriores é verdadeira.

Grupo 6 - *Pipes* e Escalonamento [3,55 val.]

1. [0,85 v.] Considere o programa abaixo e seleccione a opção **correcta e mais precisa** de entre as apresentadas:

```
//includes omitidos por brevidade
#define DIM 10

void main() {
      char msg[DIM], tmp[DIM];
      int fds[2], res;

      if (pipe (fds) < 0) exit(-1);
      write(fds[1],msg, sizeof(msg));
      res = fork ();
      if (res == 0) {
            read (fds[0], tmp, sizeof (tmp));
            printf("terminado\n");
      } else {
            read (fds[0], tmp, sizeof (tmp));
            printf("terminado\n");
            wait(NULL);
      }
}
```

A. terminado é impresso duas vezes.
B. terminado é impresso uma vez pelo processo pai.
C. terminado é impresso por um dos processos e o outro termina com erro.
D. terminado é impresso por um dos processos e o outro processo bloqueia-se.

2. [0,6 v.] Em Unix, qual das seguintes afirmações é **verdadeira**:
A. A prioridade dos processos bloqueados no núcleo depende do recurso em que se bloquearam.
B. No Unix, as aplicações de nível utilizador têm prioridades negativas.
C. Para tornarmos as aplicações mais interactivas devemos baixar o valor do seu nível de prioridade sempre que usam o CPU.
D. No Unix, os processos em execução em cada momento têm um quantum superior aos restantes.

3. [0,6 v.] No contexto dos estados de processos em Unix ,escolha a opção **verdadeira**:
A. Quando um processo bloqueado num trinco é desbloqueado, o seu estado passa para "em execução".
B. Quando um processo se bloqueia indefinidamente num semáforo, o seu estado muda para "zombie".
C. Após a execução de uma chamada de sistema **`exec`**, o processo fica no estado "bloqueado".
D. Nenhuma das anteriores.

4. [0,75 v.] Numa execução do algoritmo CFS supõe-se que, numa maquina equipada com apenas um CPU core, existam 5 processos (3 executáveis, um bloqueado e um *zombie*):
- um processo P1 e P2 que nunca se bloqueiam;
- um outro processo P3 que executa durante 3ms e se bloqueia durante 15 ms.

Supondo que *minimum granularity* e *targeted scheduling latency* sejam configurados com 5ms e 10ms, respectivamente.

Qual é a latência de escalonamento efectiva deste sistema:
A. 28 msec
B. 15 msec
C. 10 msec.
D. 13 msec.

5. [0,75 v.] Considere o programa abaixo (código simplificado) que se pretenderia que redireccionasse o **`stdout`** do processo pai para o **`stdin`** do processo filho. Selecione a opção **falsa** de entre as apresentadas:

```
01. pipe(fds);
02. p = fork();
03. if (p>0) {
04.   //close(0);
05.   close (1);
06.   dup (fds[1]);
07.   close (fds[1]);
08.   close (fds[0]);
09. }
10. else if (p==0) {
11.   close (0);
12.   dup (fds[0]);
13.   close (fds[1]);
14.   close (fds[0]);
15. }
```

A. A instrução **`dup(fds[0])`** na linha 12 coloca uma cópia da extremidade de leitura do *pipe* no **`stdin`** do processo filho.
B. A linha 14 poderia ser omitida e o *output* do processo pai seria redireccionado à mesma.
C. A linha 05 poderia ser omitida e a redirecção continuaria a funcionar.
D. Se se descomentasse a linha 04, a redirecção pretendida deixaria de funcionar.[1]

Grupo 7 - Gestão de Memória [3,2 val.]

1. Considere um sistema com uma arquitectura paginada de memória virtual de 30 bits. Neste sistema, cada endereço virtual é dividido em 12 bits (menos significativos) de **deslocamento** e 18 bits de **base** (mais significativos) e as entradas na tabela de páginas (PTEs) ocupam 8 bytes.

a) [0,6 v., sem desconto] Qual a dimensão das páginas deste sistema? Responda com o valor em KB na folha de respostas.

b) [0,6 v., sem desconto] Se quisermos ter uma sistema de paginação de dois níveis, quantos bits devemos reservar no endereço virtual para endereçar a página de primeiro nível? Responda com o número de bits na folha de respostas.

2. [0,6 v.] Escolha a afirmação **correcta**. Em sistemas baseados em segmentação:
A. Troca-se o risco de haver fragmentação interna, pelo risco de haver fragmentação externa.
B. O núcleo necessita de ser invocado quando se acede a um segmento diferente.
C. Os CPUs não precisam de ter registos para guardar limites de segmentos.
D. Cada segmento ocupa uma zona contígua de memória física.

1 No enunciado distribuído no exame a opção D era também falsa e essa escolha foi também considerada correcta.

3. [1,4 v., sem desconto] Considere um sistema de memória paginada de 32 bits, com um único nível de paginação, em que cada página tem 4k bytes. Mais precisamente, o formato do endereço é:

Página (20 bits)	Deslocamento (12 bits)

Considere que a tabela de páginas de um processo contenha o seguinte conteúdo:

Número de Página	Base	Proteções	Presente
0x00000	0x10000	rwx	S
0x00001	0x11000	r-x	S
0x00002	0x01000	rwx	S
0x00003	–	r-x	N
. . .	. . .	...	...

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

Supondo que exista uma TLB completamente associativa com 16 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 **0x12000000**.

Endereço virtual	Tipo de acesso	Endereço físico correspondente ao endereço virtual	TLB HIT (S ou N)	Necessita aceder à tabela de páginas (S ou N)	Exceções (inclusive faltas de páginas)
0x000004FF	execute	0x100004FF	N	S	-
0x00001456	write	0x11000456	N	S	PE
0x00003AAA	read	0x12000AAA	N	S	PF
0x00002010	write	0x01000010	N	S	-
0x000100FF	read	Invalid	N	S	Invalid
0x00001CC0	read	0x11000CC0	S	N	-

Preencher na folha de respostas!