

LEIC/LETI – 2023/24 – 2º Exame de Sistemas Operativos
2 de Fevereiro de 2024, 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 seleccionar 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 [2,8 val.]

- 1) [1 v.] Considere que o tamanho do ficheiro `/tmp/a.exe` é de 2Kbytes num sistema de ficheiros ext3 com blocos de 512 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, 'a', BUFFSZ); //enche buff com BUFFSZ caracteres 'a'
    int fd = open("/tmp/a.exe", O_RDWR | O_APPEND);
    close(STDOUT_FILENO);
    dup(fd);
    close(fd);
    write(STDOUT_FILENO, buff, BUFFSZ);
}
```

- A. O tamanho do ficheiro fica com 3 KB.
- B. A hora de criação é actualizada para a hora actual.
- C. O número de blocos fica com o valor 2.
- D. Não se pode escrever para um ficheiro executável.

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

- A. O número de blocos de dados é igual ao número de entradas directas no *inode*.
- B. O número de *hard links* que podem existir num sistema de ficheiro ext3 pode ultrapassar o número de *inodes* existentes.
- C. Os *inodes* contêm as permissões do ficheiro.
- D. O número de ficheiros e directórios distintos que podem existir num sistema de ficheiros ext3 não pode ultrapassar o número de *inodes* existentes.

- 3) [0,6 v.] No EXT3, se o tamanho de bloco for 8KB e as referências para os blocos ocuparem 32 bits, quantos blocos de disco são utilizados para armazenar as referências para os blocos de um ficheiro de tamanho 64KB? Na conta, excluir o espaço ocupado em disco pelo *inode* do ficheiro.

- A. 4
- B. 3
- C. 2
- D. Nenhuma das respostas anteriores.

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

Grupo 2 - Processos e Tarefas [3,1val.]

1. [1 v.] Considere este programa:

```
//...includes omitidos...
```

```
int main(int argc, char **argv) {
    int pid=42;

    if (argc < 2) return 1;
    printf("%d", pid);
    pid=fork();
    printf("%d", pid);
    pid=fork();
    printf("%d", pid);
}
```

Assumindo que o processo inicial tem o identificador de processo 42, num sistema real, qual é a afirmação **verdadeira**:

- A. A variável `pid` é impressa 3 vezes, uma vez com valor 0 e duas vezes com valores impossíveis de prever.
 - B. A variável `pid` é impressa 5 vezes, 3 vezes com o valor 0, uma vez com o valor 42 e outra vez com o valor 43.
 - C. A variável `pid` é impressa 7 vezes, 3 vezes com o valor 0, 1 vez com o valor 42, e três vezes com valores impossíveis de prever.
 - D. A variável `pid` é impressa 7 vezes, 2 vezes com o valor 42, 2 vezes com o valor 0, uma vez com o valor 43 e uma vez com um valor impossível de prever.
2. [0,6 v.] Qual das seguintes afirmações é **falsa** 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.
 - B. A alteração do bit `setuid` não requer privilégios de superutilizador.
 - C. O bit `setuid` para o dono do ficheiro é alterado com a chamada de sistema `setuid g+s <nome-ficheiro>`.
 - D. O bit `setuid` altera o *effective user id* dos utilizadores que executam o executável em causa.

3. [0,75 v.] Pretende-se desenvolver um programa que recebe os seguintes argumentos (de linha de comando):
- ```
executável arg1 arg2 ... argn
```
- Ao receber os argumentos acima, o ficheiro executável indicado no 1º argumento deve ser executado com cada um dos argumentos seguintes, de forma sequencial. Ou seja:
- ```
executável arg1
executável arg2
...
executável argn
```

Considere a seguinte implementação deste programa:

```
int main(int argc, char **argv) {
    int p = 1;
    for (int i=2; i<argc; i++) {
        if (p != 0) {
            exec(argv[1], argv[1], argv[i], NULL);
        }
        p = fork();
    }
    exit(EXIT_SUCCESS);
}
```

Assuma que o programa é chamado com os seguintes argumentos: `./xpto a b c`
O comportamento observado será o que está especificado acima, supondo que todas as chamadas `exec` sejam bem sucedidas?

- A. Sim.
- B. Não. Haverá múltiplas cópias de xpto com cada um dos argumentos (a,b,c).
- C. Não, pois só executará o primeiro exec ("`./xpto a`") e não executará os restantes.
- D. Não, pois a função exec só pode ser chamada depois de uma chamada a `fork`.

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

```
int a=0;

void *threadFn(void *arg) {
    usleep(10); //dormir 10 microsegundos
    a++;
    ...
}

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

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

Qual destes outputs não pode ser produzido pelo programa?

- A. T:0
T:1
T:1
- B. T:0
T:1
T:2
- C. T:0
T:1
T:3

Indique D. caso todos os outputs anteriores sejam possíveis.

Grupo 3 - Exclusão Mútua [1,8 val.]

1. [0,6 v.] Qual das seguintes afirmações sobre as implementações da abstração do semáforo é **falsa**:
 - A. Um semáforo inicializado a 1 pode ser utilizado com a mesma semântica de um trinco.
 - B. A chamada à função *esperar* necessita de ser implementada dentro do núcleo do sistema operativo para os semaforos serem mais eficientes.
 - C. As chamadas à função assinalar de um semáforo nunca são bloqueantes.
 - D. Um semáforo é inicializado a um dado valor e nunca pode subir acima desse valor.

2. [0,6 v.] No caso de um processo Unix que corra um programa com frequentes acessos a trincos, optar entre um trinco com espera bloqueante e um trinco com espera activa tem algum impacto nas prioridades calculadas pelo escalonador Unix? Escolha a opção **verdadeira**.
 - A. Com trincos de espera bloqueante, as tarefas que tentam aceder sem sucesso ao trinco tenderão a ficar mais prioritárias.
 - B. Com trincos de espera activa, as tarefas que acedem com sucesso ao trinco tenderão a ficar mais prioritárias.
 - C. O tipo de espera (bloqueante ou activa) não tem qualquer impacto nos cálculos da prioridade pelo escalonador.
 - D. Com trincos de espera bloqueante, a prioridade das outras tarefas com acesso ao trinco tenderá a subir.

3. [0,6 v.] Qual das seguintes afirmações é **verdadeira** acerca das soluções do problema do jantar dos filósofos apresentadas nas aulas teóricas:
 - A. Com a utilização do mecanismo de recuo aleatório, a sincronização baseada em `try_lock()` pode produzir situações de interbloqueagem (*deadlock*).
 - B. A utilização do mecanismo de recuo aleatório visa evitar, pelo menos probabilisticamente, que a sincronização baseada em `try_lock()` possa sofrer de minguagem (*livelock*).
 - 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 [3,8 val.]

1. [2 v., sem desconto] Complete a seguinte implementação das funções de entrada e saída de um trinco especial que diferencia entre tarefas de duas categorias, A e B, só permitindo o acesso a tarefas de umas das categorias de cada vez mas permitindo que multiplas tarefas da mesma categoria usem a secção critica simultaneamente.

Notas:

- catA_dentro e catB_dentro são variáveis que contam as tarefas de cada categoria dentro da secção critica.

- catA_em_espera e catB_em_espera são variaveis que contam as tarefas de cada categoria a aguardar entrada na secção critica.

- categoria_dentro indica qual a categoria na secção critica.

- sem_catA e sem_catB são semaforos.

Indique na folha de respostas, para as instruções marcada a negro no código que se segue, qual a linha de código que devem ser colocadas nessa posição do programa.

```

#define A 0
#define B 1
#define LIVRE -1
int categoria_dentro = LIVRE;
int catA_dentro=0; int catB_dentro=0;
int catA_em_espera= 0; int catB_em_espera=0;
pthread_mutex_t secCritica = PTHREAD_MUTEX_INITIALIZER;
sem_t sem_catA, sem_catB; //assuma que ambos os semáforos são inicializados a 0

entrar_A() {
    pthread_mutex_lock(&secCritica);
    if ( categoria_dentro==B ) {
        catA_em_espera++;
        pthread_mutex_unlock(&secCritica);
        1. sem_wait(&sem_catA);
        pthread_mutex_lock(&secCritica);
        categoria_dentro = A;
        if (catA_em_espera > 0) {
            catA_dentro++;
            catA_em_espera--;
            2. sem_post(&sem_catA);
        }
    } else { categoria_dentro = A; catA_dentro++; }
    pthread_mutex_unlock(&secCritica);
}

sair_A() {
    pthread_mutex_lock(&secCritica);
    catA_dentro--;
    if (catA_dentro == 0 && catB_em_espera > 0){
        3. sem_post(&sem_catB);
        categoria_dentro = B;
        catB_em_espera--;
    } else if (catA_dentro == 0 && catB_em_espera == 0) {
        categoria_dentro = LIVRE;
    }
    pthread_mutex_unlock(&secCritica);
}

entrar_B() {
    pthread_mutex_lock(&secCritica);
    if (categoria_dentro==A ) {
        catB_em_espera++;
        pthread_mutex_unlock(&secCritica);
        4. sem_wait(&sem_catB);
        pthread_mutex_lock(&secCritica);
        categoria_dentro = B;
        if (catB_em_espera > 0) {
            catB_dentro++;
            catB_em_espera--;
            5. sem_post(&sem_catB);
        }
    } else { categoria_dentro = B; catB_dentro++; }
    pthread_mutex_unlock(&secCritica);
}

sair_B() {
    pthread_mutex_lock(&secCritica);
    catB_dentro--;
    if (catB_dentro == 0 && catA_em_espera > 0){
        6. sem_post(&sem_catA);
        categoria_dentro = A;
        catA_em_espera--;
    } else if (catB_dentro == 0 && catA_em_espera == 0) {
        categoria_dentro = LIVRE;
    }
    pthread_mutex_unlock(&secCritica);
}

```

2. [1,8 v. sem desconto] Pretende-se implementar uma simulação de uma cadeia de produção na qual se representam três entidades através de tarefas:
- Uma tarefa que simula uma fábrica de peças (Fábrica) que produz peças individuais para montagem. A tarefa produz continuamente 5 peças a cada 10 segundos e passa-as para uma tarefa montador, se o montador tiver espaço disponível para armazenar peças individuais. A fábrica deve bloquear-se (por ex. em `montadorIndivCond`) se não houver espaço de armazenamento de peças individuais.
 - Uma tarefa que simula o funcionamento de um montador (Montador). O montador tem uma capacidade limitada de armazenamento de peças individuais e de produtos. O montador converte, a cada 2 segundos 2 peças individuais num *produto* obtido montando as 2 peças individuais. O armazém deve bloquear-se (por ex. em `montadorIndivCond` e/ou em `montadorProdCond`) se não houver peças individuais ou espaço para armazenar produtos.
 - Um conjunto de tarefas que simulam oficinas de pintura (Oficina). Cada oficina retira um produto do montador, pinta-o e, 2 segundos depois, vai buscar um novo produto continuamente. O sistema não simula o que acontece aos produtos depois de pintados. As oficinas devem bloquear-se (por ex. em `oficinasCond`) se não houver produtos para pintar.

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(&montadorIndivCond);`
- B. `pthread_cond_broadcast(&montadorIndivCond);`
- C. `pthread_cond_wait(&montadorIndivCond, &mutex);`
- D. `pthread_cond_signal(&montadorProdCond);`
- E. `pthread_cond_broadcast(&montadorProdCond);`
- G. `pthread_cond_wait(&montadorProdCond, &mutex);`
- M. `pthread_cond_signal(&oficinasCond);`
- S. `pthread_cond_broadcast(&oficinasCond);`
- T. `pthread_cond_wait(&oficinasCond, &mutex);`

```

#define LIM_MONTADOR_INDIV 100
#define LIM_MONTADOR_PROD 100

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t montadorIndivCond = PTHREAD_COND_INITIALIZER;
pthread_cond_t montadorProdCond = PTHREAD_COND_INITIALIZER;
pthread_cond_t oficinasCond = PTHREAD_COND_INITIALIZER;
int pecasMontadorIndiv = 0, produtosMontador = 0;

void *Fabrica(void *arg) {
    while (1) {
        sleep(10);
        pthread_mutex_lock(&mutex);
        while (pecasMontadorIndiv > LIM_MONTADOR_INDIV-5) {
            1. pthread_cond_wait(&montadorIndivCond, &mutex);
        }
        pecasMontadorIndiv += 5;
        printf("Fabrica produziu 5 unid.\n Total: %d\n", pecasMontadorIndiv);
        2. pthread_cond_signal(&montadorIndivCond);
        pthread_mutex_unlock(&mutex);
    }
}

void *Montador(void *arg) {
    while (1) {
        pthread_mutex_lock(&mutex);
        while ( pecasMontadorIndiv < 2) {
            3. pthread_cond_wait(&montadorIndivCond, &mutex);
        }
        while ( produtosMontador == LIM_MONTADOR_PROD ) {
            4. pthread_cond_wait(&montadorProdCond, &mutex);
        }
        pecasMontadorIndiv -= 2;
        produtosMontador += 1;

        printf("Armazem emparelhou 2 peças\nArmazem indiv: %d, produtos:%d\n",
            pecasMontadorIndiv, produtosMontador);

        if (pecasMontadorIndiv <= LIM_MONTADOR_INDIV-5) {
            5. pthread_cond_signal(&montadorIndivCond);
        }
        if (produtosMontador == 1) {
            6. pthread_cond_signal(&oficinasCond);
        }
        pthread_mutex_unlock(&mutex);
        sleep(2);
    }
}

void *Oficina(void *arg) {
    while (1) {
        pthread_mutex_lock(&mutex);
        while (produtosMontador == 0) {
            7. pthread_cond_wait(&oficinasCond, &mutex);
        }
        produtosMontador -=1;
        printf("A oficina pintou um produto\n");
        if (produtosMontador == LIM_MONTADOR_PROD-1) {
            8. pthread_cond_signal(&montadorProdCond);
        }
        pthread_mutex_unlock(&mutex);
        sleep(2);
    }
}

```


Grupo 5 - Signals [1,30 val.]

1. [0,6 v.] Qual das seguintes afirmações sobre os *signals* é **falsa**?
- A. Não é seguro utilizar `printf` e `scanf` num *signal handler*.
 - B. Dependendo da implementação utilizada (p.ex., System V ou BSD) pode tornar-se necessário voltar a reassociar a rotina do tratamento do *signal* dentro do próprio *signal handler*.
 - C. Os *signals* permitem o envio de mensagens arbitrárias entre processos.
 - D. É possível redefinir o tratamento de apenas um subconjunto dos *signals* existentes.

2. [0,7 v.] Considere o seguinte programa:

```
void sigHandler (int s) {
    char c;
    read(0,&c,1);
    if (c == 'Y') exit(0);
    else {
        signal (SIGINT, sigHandler); }
}

int main () {
    signal (SIGINT, sigHandler);
    for (;;)
        pause();
}
```

Qual destas afirmações é **falsa**?

- A. Este programa permite evitar que, ao carregar nas teclas CTRL-C, o programa seja terminado.
- B. Este programa está escrito supondo a utilização numa plataforma cuja implementação do mecanismo dos *signals* tem semântica System V.
- C. Este programa efetua espera ativa enquanto não são entregues *signals*.
- D. Este programa pode ser terminado repentinamente caso esteja a correr numa plataforma com semântica System V para os *signals* e se carregar nas teclas CTRL-C duas vezes de seguida.

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. Os FIFOs só podem ser usados para comunicação entre processos relacionados hierarquicamente (por exemplo pai e filho).
 - B. Os FIFOs são identificados por nomes de ficheiros.
 - C. Os *pipes* oferecem dois canais unidirecionais enquanto os FIFOs expõem um canal bidirecional.
 - D. A abertura de um *pipe* é idêntica à de um ficheiro.

2. [0,6 v.] Qual das seguintes afirmações é **verdadeira**:
 - A. O núcleo do sistema operativo é ativado apenas caso seja invocada uma *system call*.
 - B. Um núcleo monolítico é mais eficiente do que um núcleo que utiliza múltiplas camadas com níveis de proteção diferente.
 - C. O mecanismo de preempção visa evitar que os processos menos prioritários perdam o processador frequentemente demais quando existem muitos processos mais prioritários.
 - D. A utilização de duas pilhas, uma para o núcleo e outra para as aplicações, visa aumentar a eficiência do sistema, reduzindo o custo de comutação de processos.

3. [0,6v] Qual das seguintes afirmações é **verdadeira**:
 - A. No Unix o núcleo corre com prioridades positivas.
 - B. No Unix prioridades numéricas negativas correspondem a prioridade altas.
 - C. No Unix, a prioridade dos processos é calculada só depois de todos os processos executáveis terem tido uma oportunidade de correr.
 - D. No Unix, todos os utilizadores podem aumentar a prioridade dos próprios processos se especificar valores negativos como *input* para a *system call nice*.

4. [0,75 v.] Numa execução do algoritmo CFS supõe-se que: existam 10 processos executáveis que nunca bloqueiam; *minimum granularity* e *targeted scheduling latency* sejam configurados com 1ms e 20ms, respetivamente. Qual é a efetiva latência de escalonamento deste sistema:
 - A. 10 msec
 - B. 20 msec
 - C. 1 msec
 - D. Nenhuma das respostas anteriores

5. [0,75 v.] Considere o mesmo sistema da pergunta anterior, mas que existam 50 processos executáveis que nunca bloqueiam. Qual é a efetiva latência de escalonamento deste sistema:
 - A. 10 msec
 - B. 20 msec
 - C. 50 msec
 - D. Nenhuma das respostas anteriores

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


```

graph TD
    Executavel([Executável]) -- a --> EmExecucao([Em execução])
    EmExecucao -- b --> Executavel
    Bloqueado([Bloqueado]) -- c --> Executavel
    Executavel -- d --> Bloqueado
    EmExecucao -- e --> Bloqueado
      
```

- A. transição a
- B. transição c
- C. transição d**
- D. transição e

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 20 bits (menos significativos) de **deslocamento** e 12 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 linhas
 - D. Nenhuma das respostas anteriores é correta.
- c) [0,6 v.] Escolha a afirmação **correcta**.
- A. Em sistemas baseados em paginação não pode haver fragmentação interna.
 - B. Em sistemas baseados em paginação não pode haver fragmentação externa.**
 - C. Em sistemas baseados em segmentação não pode haver fragmentação interna.
 - D. Nenhuma das respostas anteriores é correta.

2. [1,5 v.] 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 um processo começa a sua execução, que a TLB está limpa e que todas as páginas acedidas pelo processo se encontram carregadas em memória principal. Considere a seguinte sequência de acessos a endereços virtuais (todos os endereços estão em hexadecimal):

1. 0x32002A5C; TLB_HIT: N; Num. Acessos RAM para tradução: 1
2. 0x3100B65D ; TLB_HIT: N; Num. Acessos RAM para tradução: 1
3. 0x320022AA; TLB_HIT: S; Num. Acessos RAM para tradução: 0
4. 0x32002BF3; TLB_HIT: S; Num. Acessos RAM para tradução: 0
5. 0x3100B432; TLB_HIT: S; Num. Acessos RAM para tradução: 0
6. 0x3100B372; TLB_HIT: S; Num. Acessos RAM para tradução: 0
7. 0x3000C70E; TLB_HIT: N; Num. Acessos RAM para tradução: 1
8. 0x3100B66B; TLB_HIT: S; Num. Acessos RAM para tradução: 0
9. 0x32002800; TLB_HIT: S; Num. Acessos RAM para tradução: 0
10. 0x30000FBA; TLB_HIT: N; Num. Acessos RAM para tradução: 1

Supondo que exista uma TLB completamente associativa com 32 entradas de capacidade. Para cada acesso indique se há um *hit* na TLB e quantos acessos à RAM são necessários para a tradução do endereço.