Sistemas Operativos, Exame 2

IST - LEIC-A/ LEIC-T/ LETI - 2017-2018
27 de Janeiro de 2017

- Todas as respostas devem ser dadas na folha de resposta.
- Não pode sair da sala antes de passarem 60 minutos. Não é autorizada a utilização de telemóveis ou outros dispositivos electrónicos.
- O exame tem a duração de 3 horas.
- Em todas as respostas com código, pode omitir a verificação e tratamento de erros na chamada a funções.

---

1 Programação Com Tarefas Por Troca de Mensagens

Considere que tem um conjunto de ficheiros de texto e que precisa de contar quantas vezes uma dada palavra aparece no conjunto desses ficheiros. Pretende distribuir o trabalho, dividindo o trabalho por diversas tarefas escravas. A estratégia de divisão deste programa em tarefas é a seguinte. Existe uma tarefa mestre, que cria todas as outras tarefas e distribui o trabalho pelas tarefas escravas. Existem n tarefas escravas, que contam palavras em ficheiros. Finalmente, existe uma tarefa acumuladora, que vai somando os resultados que as escravas vão produzindo.

- *A tarefa mestre faz o seguinte:* i) Cria as n tarefas escravas e a tarefa acumuladora. Nos parâmetros de criação, cada tarefa escrava é informada do seu próprio identificador e da palavra a procurar nos ficheiros; ii) Espera mensagens de "pedido de trabalho" vindas das tarefas escravas. Uma mensagem de "pedido de trabalho" é constituida por um único byte, contendo o caracter `'P'`. Sempre que recebe uma mensagem de pedido vinda de uma tarefa escrava, a tarefa mestre envia para essa tarefa escrava o nome de um novo ficheiro para processar.

- *As tarefas escravas fazem o seguinte:* i) Enviam um "pedido de trabalho" ao mestre; ii) Esperam pelo nome do ficheiro a processar; iii) Contam o número de vezes que a palavra ocorre no ficheiro; iv) Enviam o resultado para a tarefa acumuladora; v) Retornam ao ponto i).

- *A tarefa acumuladora faz o seguinte:* i) Espera mensagens das tarefas escravas com o número de vezes que a palavra alvo apareceu num ficheiro; ii) Soma esse valor a uma variável designada por "total_acumulado"; iii) Imprime o valor do total acumulado no *stdout*; iv) Volta ao ponto i).

Para facilitar o código, assuma que o número de ficheiros a processar é infinito (isto é, há sempre mais ficheiros para processar), pelo que não é preciso concretizar condições de terminação.

Pergunta 1 (*2 valores*) Complete o programa apresentado na folha de respostas tendo em conta os aspectos abaixo referidos.

Deve recorrer a uma biblioteca de troca de mensagens com a seguinte interface, que permite, respetivamente, enviar e receber uma mensagem da tarefa *tarefaOrig* para a tarefa *tarefaDest*. Em caso de sucesso, ambas as funções retornam um inteiro que indica o número de bytes enviados/recebidos.

```
int enviarMensagem(int tarefaOrig, int tarefaDest, void *msg, int tamanho);
int receberMensagemDeQualquerOrigem (int tarefaDest, int *origem, void *buffer, int tamanho);
```

Assuma que, perante a biblioteca, a tarefa mestre tem identificador 0, as escravas são identificadas como $1, 2, .., n$ e a tarefa acumuladora é a $n + 1$.

Importante: note que, ao contrário do que acontecia no projeto, é possível receber uma mensagem sem indicar previamente de onde vem. Na função *receberMensagemDeQualquerOrigem* o parâmetro *origem* é prenchido pelo sistema de forma automática, e indica ao receptor quem enviou a mensagem.

O programa de cada escravo deve chamar a seguinte função auxiliar que recebe o nome de um ficheiro, uma palavra, e retorna quantas vezes a palavra aperece no ficheiro.

```
int conta_palavra (char* nomeficheiro, char* palavra);
```

Para simplificar, a função acumuladora já está completa.

§

2 Programação Com Tarefas Por Memória Partilhada

Considere um processo composto por tarefas escravas que executam unidades de trabalho, sendo que:

- Há 2 categorias de tarefas escravas, A e B. Cada tarefa trabalhadora executa um ciclo em que cada iteração: i) obtém uma unidade de trabalho da fila correspondente à sua categoria (fila A ou fila B); ii) executa a unidade de trabalho e guarda o resultado da computação; iii) avança para a próxima iteração. Para ilustrar, abaixo apresenta-se o programa executado pelas escravas da categoria A:

```
fnTrabalhadoraA(void *arg) {
    unidTrab_t *u;

    while (TRUE) {
        u = obtemProximaUnidade(fila[A]);
        iniciaAcessoCatA();
        executaEGuarda(u);
        terminaAcessoCatA();
    }
}
```

- No início do processo, são lançadas n escravas de cada categoria (ou seja, $n + n$ escravas no total).
- Para obter desempenho óptimo só devem estar n tarefas escravas no total a correr em simultâneo; ou seja, nem todas as tarefas de ambas as categorias podem estar simultaneamente ativas.
- Existem 2 parâmetros, max_A e max_B, que indicam o número máximo de escravas de cada categoria que podem estar a correr em simultâneo, sendo que $max_A + max_B = n$. Para já, assuma que max_A e max_B são constantes.

Pergunta 2 (*1,5 valores*)
Recorrendo exclusivamente a trincos e semáforos, escreva o código das funções *iniciaAcessoCatA()* e *terminaAcessoCatA()* (na folha de respostas) de forma a assegurar que, em qualquer momento, não há mais que max_A escravas A a computar unidades de trabalho. Declare e inicialize todas as variáveis de sincronização que usar.

Pergunta 3 (*1,5 valores*)
Recorrendo agora exclusivamente a trincos (*mutexes*) e a variáveis de condição, construa uma solução que garanta o mesmo requisito anterior. Tal como na pergunta anterior, escreva o código das funções *iniciaAcessoCatA()* e *terminaAcessoCatA()*, declarando e inicializando todas as variáveis de sincronização que usar.

§

Assuma agora que as variáveis max_A e max_B podem ser ajustadas a qualquer momento através das seguintes funções:

```
trinco_t m_A, m_B;

void incrementaMaxA(int v) {
    fechar(m_B);
    if (max_B > v) {
        fechar(m_A);
        max_B -= v; max_A += v;
        abrir(m_A);
    }
    abrir(m_B);
}

void incrementaMaxB(int v) {
    fechar(m_A);
    if (max_A > v) {
        fechar(m_B);
        max_A -= v; max_B += v;
        abrir(m_B);
    }
    abrir(m_A);
}
```

Pergunta 4 (*1 valor*) Quando chamadas por tarefas concorrentes, observou-se que as funções acima por vezes bloqueiam para sempre. Que alterações propõe ao código para eliminar esse problema? Basta indicar as linhas que modificaria.

3 Programação com Processos

Considere os seguintes programas, A e B.

```
//Programa A

int main() {

    int pid[N], i;

    for (i=0; i<N; i++) {
      pid[i] = fork();
      if (pid[i] == 0) {
        printf("f(%d) iniciou\n", i);
        f(i); /*função CPU-bound demorada */
        printf("f(%d) terminou\n", i);
        exit(0);
      }
      else
        wait(NULL);
    }
}
```

```
//Programa B

int main() {

    int pid[N], i;

    for (i=0; i<N; i++) {
      pid[i] = fork();
      if (pid[i] == 0) {
        printf("f(%d) iniciou\n", i);
        f(i); /* função CPU-bound demorada */
        printf("f(%d) terminou\n", i);
        exit(0);
      }
    }

    for (i=0; i<N; i++)
        wait(NULL);

}
```

Nas próximas 2 perguntas, assinale a qual ou quais dos programas a afirmação se aplica.

Pergunta 5 (*0,5 valor*) É possível observar este excerto (podem existir mensagens antes e depois do excerto, mas as duas mensagens apresentadas no excerto aparecem uma a seguir à outra):

```
f(1) terminou
f(0) terminou
```

Pergunta 6 (*0,5 valor*) É possível observar este excerto (podem existir mensagens antes e depois do excerto, mas as duas mensagens apresentadas no excerto aparecem uma a seguir à outra):

```
f(0) terminou
f(1) terminou
```

Pergunta 7 (*2 valores*) Partindo do programa B acima, apresente uma variante em que:

- Assim que o processo pai observe que um filho terminou, o processo pai envia um *signal* do tipo `SIGUSR1` a todos os restantes processos filho;
- Cada processo filho que receba esse *signal* deve imprimir a palavra *acknowledged* e terminar imediatamente.

§

4 Gestor de Processos

Considere esta variante do programa B do grupo anterior, em que foi adicionada a chamada à função `nice`:

```
//Programa B-alternativo

int main() {

    int pid, i;

    for (i=0; i<N; i++) {
      pid = fork();
      if (pid == 0) {
        nice(N - i);
        printf("f(%d) iniciou\n", i);
        f(i); //função CPU-bound demorada
        printf("f(%d) terminou\n", i);
        exit(0);
      }
    }

    for (i=0; i<N; i++)
        wait(NULL);
}
```

Pergunta 8 (*1 valor*) Numa máquina *single-core*, que impacto espera que a chamada à função *nice* tenha no comportamento observado por este programa? Justifique.

§

Considere agora o seguinte programa:

```
//Programa C
int main() {
    if (fork() == 0)  {
        f(); //função que gasta 2 segundos de CPU
    }
    else {
        sleep(1);
        printf("Passou 1 segundo desde o fork\n");
        wait(NULL);
    }
}
```

Pergunta 9 (*2 valores*) O autor deste programa tem a expectativa que a mensagem impressa pelo processo pai seja apresentada no ecrã cerca de 1 segundo depois do `fork()`. No entanto, ao experimentar este programa em sistemas operativos com escalonadores preemptivos e não preemptivos, observou que a mensagem surje no ecrã com diferentes atrasos (para além de 1 segundo).

Em que tipo de escalonador – preemptivo ou não-preemptivo – é mais provável que a mensagem seja impressa com maior atraso?

Ilustre a sua resposta apresentando nos diagramas temporais na folha de respostas: um exemplo de execução com escalonador preemptivo; um exemplo de execução com escalonador não-preemptivo. Nos diagramas, indique claramente qual o processo em execução em cada momento e o momento em que cada função é chamada. Assuma máquina *single-core* e escalonador com prioridades fixas em que o processo pai tem prioridade superior à do filho.

§

5 Gestão de Memória

Considere uma arquitectura de 32 bits que suporta páginas de 64Kbytes. Assuma que todas as páginas possuem esta dimensão.

Pergunta 10 (*1 valor*) Quantas páginas pode no máximo um processo ter?

§

Neste sistema, considere a seguinte tabela de páginas de um processo:

Página	Bit Presença	Protecção	Acedido (R)	Dirty (M)	Base
0	0	-	0	0	`0x0000`
1	1	R	1	0	`0x0011`
2	0	R	0	0	`0x0451`
3	1	RW	1	1	`0x0033`
4	1	RW	0	1	`0x0031`
5	0	RW	0	0	`0x0032`
6	0	RW	1	1	`0x0AB3`

Pergunta 11 (*1 valor*) Preencha a tabela com a tradução entre endereços reais virtuais e endereços reais que se encontra na folha de respostas.

Para cada acesso indique:

- Se ocorreu uma falta de página (coloque um "S"(im) ou um "N"(ão) na coluna FP).
- Qual o endereço físico gerado. Caso ocorra uma falta de página, indique o endereço gerado depois da falta de página ser tratada. Para isso assuma que o SO iria usar as seguintes tramas livres (por esta ordem): `0x0AAA`, `0x0BBB`, `0x0CCC`.
- Nos casos em que um endereço físico é gerado, indique o valor dos bits de Acedido (R) e Dirty (M) depois do acesso.
- Nos casos em que um endereço físico não é gerado, a causa para o erro.

§

Pergunta 12 (*1 valor*) Considere uma página que está em memória e que é acedida num determinado momento mas que depois nunca mais é acedida. O bit de Acedido (R) é alguma vez colocado de novo a "0"? Em caso afirmativo, descreva como é que isso acontece. Em caso negativo, justifique.

§

Pergunta 13 (*1 valor*) Considere uma página que etsá em memória e que é acedida para escrita num determinado momento mas que depois só é acedida para leitura. O bit de Dirty (M) é alguma vez colocado de novo a "0"? Em caso afirmativo, descreva como é que isso acontece.

6 Sistemas de Ficheiros

Considere que a diretoria raíz de um dado sistema de ficheiros Unix possui o seguinte conteúdo.

Inode	Tamanho Entrada	Tamanho do Nome	Tipo	Nome
2	12	1	2	.\0\0\0
2	12	2	2	.. \0\0
11234	12	3	2	tmp\0
11111	16	5	1	a.txt\0\0\0

Considere que existe uma outra diretoria com o seguinte conteúdo:

Inode	Tamanho Entrada	Tamanho do Nome	Tipo	Nome
11234	12	1	2	.\0\0\0
2	12	2	2	.. \0\0
22222	16	5	1	b.txt\0\0\0

Nas perguntas seguintes, considere que se o sistema necessitar de reservar inodes livres vai reservar os seguintes inodes (por esta ordem): 33333, 44444, 55555.

§

Considere que o utilizador dá o seguinte comando que se executa com sucesso.

```
>cp /a.txt /tmp/c.txt
```

Pergunta 14 (*1 valor*) Na folha de respostas, apresente as alterações às tabelas acima que resultam da execução deste comando.

§

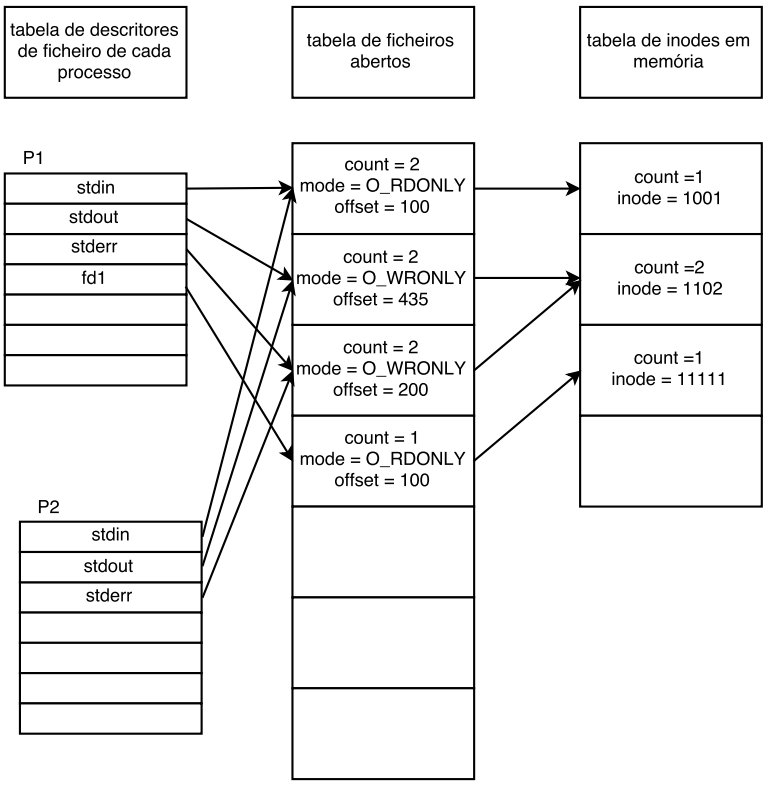
Considere que o utilizador dá o seguinte comando que se executa com sucesso.

```
>ln /a.txt /tmp/d.txt
```

Pergunta 15 (*1 valor*) Na folha de respostas, apresente quais as alterações às tabelas acima que resultam da execução deste comando (assuma o estado inicial descrito no início da página).

§

Considere um sistema ficheiros do Unix do tipo ext3 em que os inodes possuem uma tabela com 15 apontadores e os blocos 4Kbytes. Considere o seguinte estado das estruturas em memória mantidas pelo sistema de ficheiros. Considere que se o sistema necessitar de alocar blocos, reserva os seguinte blocos livres (por esta ordem): 500, 600, 700



§

Considere que o tamando do ficheiro "/a.txt" é de 3Kbytes. Considere que o processo P2 executa com sucesso a seguinte sequência de chamadas.

```
#define BUFFSZ 2048 /* 2K */

char buff[BUFFSZ];

fill_buffer(buff); /* coloca conteudo no buffer */

fd = open("/a.txt", O_APPEND);
close (stdout);
dup (fd);
close (fd);
write (stdout, buff, BUFFSZ);
```

Pergunta 16 (*1 valor*) Desenhe **no espaço reservado na folha de respostas** as alterações nas estruturas do sistema de ficheiros mantidas em memória.

§

Pergunta 17 (*1 valor*) Quais as alterações ao inode do ficheiro "/a.txt", depois da sequência de intruções acima?

Sistemas Operativos, Exame 2, 27 de Janeiro de 2018

IST - LEIC-A/ LEIC-T/ LETI - 2017-2018

Folha de Respostas (1/7)
Número: **Nome:**

§

Programação com tarefas por troca de mensagens

Pergunta 1	

```
#define N 10
#define FILENAME_SIZE 255
#define WORD_SIZE 10

typedef struct {
  int     id;
  char    alvo[WORD_SIZE];
} argsEscrava_t;

/*-------------------------------------------------------------------
| Function: fn_escrava
-------------------------------------------------------------------*/
void *fn_escrava(void *a) {
     char codigo_pedido = 'P';
     char nome_ficheiro[FILENAME_SIZE];
     argsEscrava_t *arg   = (argsEscrava_t *) a;
     int myid = arg->id;
     char *alvo = arg->alvo;
     int   n_encontradas;


     while (1) {

         /* envia mensagem a pedir nome do ficheiro */




         /* recebe nome do ficheiro */




          /* processa o ficheiro */
          n_encontradas =  conta_palavra (nome_ficheiro, alvo);

         /* envia para o acumulador */
         enviarMensagem(                    ,                  ,   &n_encontradas, sizeof(int));
     }
     return 0;
}

/*-------------------------------------------------------------------
| Function: fn_acumuladora (já completa)
-------------------------------------------------------------------*/
void *fn_acumuladora(void *a) {
     int  n_encontradas, origem, total_acumulado = 0;
     while (1) {
         receberMensagemDeQualquerOrigem (N+1, &origem, &n_encontradas, sizeof(int));
         total_acumulado = total_acumulado + n_encontradas;
         printf ("Recebeu de %d; Total acumulado=%d\n", origem, total_acumulado);
     }
     return NULL;
}
```

Folha de Respostas (2/7)

Número:
Nome:

§

```
/*-------------------------------------------------------------------
| Function: main
-------------------------------------------------------------------*/
int main (int argc, char** argv) {
    char*           nome_ficheiro;
    argsEscrava_t   escrava_args[N];
    pthread_t       escravas[N];
    pthread_t       acumuladora;
    char *palavra_alvo = argv[1]; //testes aos argumentos omitidos




    /* Inicializa biblioteca de troca de mensagens  (capacidade do canal, numero de tarefas comunicantes) */
    inicializarMPlib(CHANNEL_SZ, N+2);

    /* cria as tarefas escravas */
    for (t=1; t <= N; t++) {









       pthread_create(&escravas[t-1], NULL,                          ,                    );
    }

    /* cria a tarefa acumuladora */
    pthread_create(&acumuladora, NULL,                          , NULL );

    while (1) {
        nome_ficheiro = proximo_ficheiro ();


        /* recebe pedido de uma tarefa escrava */









        /* envia nome do ficheiro */









    }
}
```

Folha de Respostas (3/7)

Número:
Nome:

§

Programação com tarefas por memória partilhada

Pergunta 2	`//Declaração/inicialização de variáveis globais`
	
	`iniciaAcessoCatA() {`
	
	`}`
	`terminaAcessoCatA() {`
	
	`}`

Pergunta 3	`//Declaração/inicialização de variáveis globais`
	
	`iniciaAcessoCatA() {`
	
	`}`
	`terminaAcessoCatA() {`
	
	`}`

Pergunta 4	

Número:
Nome:

§

Programação com processos

Pergunta 5	Programa A ☐ Programa B ☐ Ambos ☐ Nenhum ☐
Pergunta 6	Programa A ☐ Programa B ☐ Ambos ☐ Nenhum ☐

Pergunta 7

```
int main() {
  int pid[N], i;

}
```

§

Gestor de Processos

Pergunta 8	

Folha de Respostas (5/7)

Número:
Nome:

Gestor de Processos

Pergunta 9

Exemplo sem preempção:

Exemplo com preempção:

§

Gestão de Memória

Pergunta 10

Pergunta 11

Acesso	Endereço Virtual	FP	Endereço Real	Bit Presença	Acedido (R)	Dirty (M)	Excepção lançada
Leitura	`0x0001A345`						
Leitura	`0x0002AGFC`						
Leitura	`0x0002AABD`						
Escrita	`0x0003AA4F`						
Leitura	`0x00041251`						
Escrita	`0x00000000`						

Pergunta 12

Pergunta 13

Folha de Respostas (6/7)

Número:
Nome:

§

Sistema de Ficheiros

Pergunta 14

Inode	Tamanho Entrada	Tamanho do Nome	Tipo	Nome
2	12	1	2	.\0\0\0
2	12	2	2	.. \0\0
11234	12	3	2	tmp\0
11111	16	5	1	a.txt\0\0\0

Inode	Tamanho Entrada	Tamanho do Nome	Tipo	Nome
11234	12	1	2	.\0\0\0
2	12	2	2	.. \0\0
22222	16	5	1	b.txt\0\0\0

§

Pergunta 15

Inode	Tamanho Entrada	Tamanho do Nome	Tipo	Nome
2	12	1	2	.\0\0\0
2	12	2	2	.. \0\0
11234	12	3	2	tmp\0
11111	16	5	1	a.txt\0\0\0

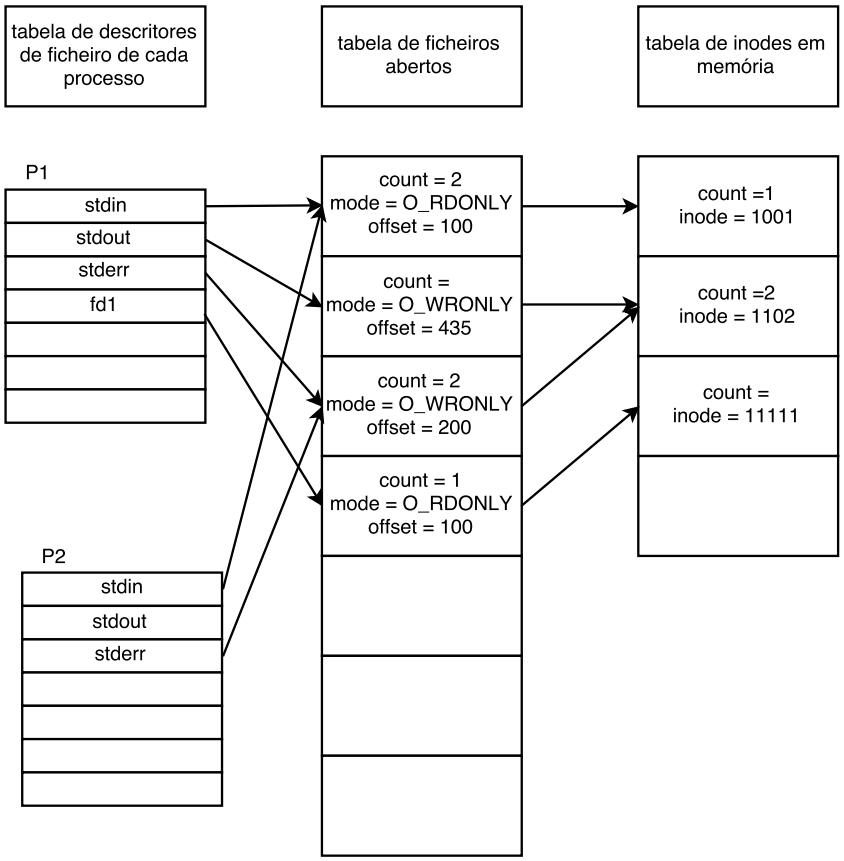
Inode	Tamanho Entrada	Tamanho do Nome	Tipo	Nome
11234	12	1	2	.\0\0\0
2	12	2	2	.. \0\0
22222	16	5	1	b.txt\0\0\0

Folha de Respostas (7/7)

Número:
Nome:

§

Pergunta 16



Pergunta 17