Sistemas Operativos, Teste 1

IST - LEIC-A/ LEIC-T/ LETI - 2017-2018
28 de Outubro 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 teste tem a duração de 1 hora.

---

1 Tarefas e Troca de Mensagens

Considere que pretende fazer um programa paralelo usando tarefas (*threads*) e troca de mensagens. A interface da biblioteca da troca de mensagens oferece a seguinte interface:

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

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.

Assuma também que tem uma função chamada int toupper_token (char* str, char* token) que procura na cadeia de caracteres str todas as ocorrências da sub-cadeia de caracteres token e as coloca em maiúsculas, retornando quantas ocorrências encontrou. Por exemplo, se tivermos str = "eu adoro a cadeira de sistemas operativos" e token = "adoro", o resultado de chamar toupper_token(str, token) será ter str = "eu ADORO a cadeira de sistemas operativos" (e toupper_token retornará 1).

Considere que quer processar mais que uma cadeia de caracteres e que:

- Em cada cadeia de caracteres, quer substituir não 1 mas n tokens por maiúsculas;
- Pretende descartar todas as cadeias de caracteres que não têm todos os tokens e só imprimir o resultado para aquelas que possuem todos os tokens.

Por exemplo, considere que os tokens são "adoro" e "sistemas". Se o conjunto inicial de cadeias de caracteres for:

eu adoro a cadeira de sistemas operativos
eu adoro laranjas
eu sei resolver sistemas de equações
há sistemas que eu adoro

então o resultado será:

eu ADORO a cadeira de SISTEMAS operativos
há SISTEMAS que eu ADORO

Para resolver este problema vai usar a seguinte estratégia de paralelização em "linha de montagem": a tarefa principal (tarefa$_0$) lê as cadeias de caracteres do teclado e envia-as para a tarefa$_1$ que substitui o token$_1$ por maiúsculas e que, por sua vez, envia o resultado por mensagem para a tarefa$_2$; caso nenhum token seja encontrado, nenhuma mensagem deve ser enviada para a tarefa$_2$. Ao receber a cadeia de caracteres resultante, a tarefa$_2$ repete o mesmo procedimento mas agora para o token$_2$, e assim sucessivamente. No final da sequência, existe uma tarefa que imprime o resultado no ecrã.

Para simplificar, assuma que pretende substituir apenas dois tokens. Omita o código de tratamento de erros.

Pergunta 1 (*7 valores*) Complete o código que é apresentado na folha de respostas anexa, para resolver o problema acima.

2 Sincronização com Mutexes e Variáveis de Condição

Considere que possui várias tarefas que executam o seguinte código:

```
int c;

void *tarefa(void *a) {
   argsSimular_t *arg   = (argsSimular_t *) a;
   int            myid     = arg->id;


   while (1) {
     pthread_mutex_lock(&mutex);
     while (c==0)
       pthread_cond_wait(&espera, &mutex);
     pthread_mutex_unlock(&mutex);

     c--;

     printf ("thread %d conseguiu o token\n", myid);
     sleep (arg->id);
     printf ("thread %d vai libertar o token\n", myid);

     pthread_mutex_lock(&mutex);
     c++;
     pthread_cond_signal(&espera);
     pthread_mutex_unlock(&mutex);
   }
   return 0;
}
```

Pergunta 2 (*2 valores*) Considere que a variável c tem no início o valor 5. Quantas tarefas conseguem o token sem necessitarem de esperar que outras tarefas o libertem primeiro? Justifique.

Pergunta 3 (*2 valores*) O código tem um erro. Dê um exemplo concreto de uma execução que pudesse dar um resultado diferente do pretendido pelo programador.

Pergunta 4 (*2 valores*) Proponha uma correção para o erro ilustrado na alínea anterior. Não precisa apresentar o código completo, basta indicar claramente que alterações faria a quais linhas.

Pergunta 5 (*2 valores*) Assuma que a primitiva pthread_mutex_lock era concretizada sem chamar o núcleo do sistema operativo recorrendo a uma instrução atómica (CAS, BTS ou XCHG) em espera ativa. Considere também que executava o programa anterior numa máquina só com um processador.

Qual a potencial desvantagem da concretização do pthread_mutex_lock com uma espera activa? Ilustre descrevendo um exemplo de uma execução problemática.

3 Sincronização com Semáforos

Considere um sistema onde possui tarefas do tipo A e tarefa do tipo B. Estas tarefas necessitam de aceder em exclusão mútua a um recurso partilhado.

Assuma que pretende garantir que o acesso ao recurso é seriado da seguinte maneira alternada: primeiro acedem 2 tarefas do tipo A (uma depois da outra, pois não podem aceder simultaneamente), depois 2 tarefas do tipo B, depois duas tarefas do tipo A, e assim sucessivamente. Note que, depois de 2 tarefas de um tipo terem acedido ao recurso, as restantes tarefas do mesmo tipo devem ficar bloqueadas até que 2 tarefas do outro tipo acedam ao mesmo.

Antes de aceder ao recurso, as tarefas chamam o método pedeAcesso (int tipoDaTarefa), quando este método retorna acedem ao recurso (é irrelevante para a resposta a este problema o que as tarefas fazem enquanto têm acesso ao recurso), quando terminam de utilizar o recurso chamam libertaAcesso(int tipoDaTarefa).

Pergunta 6 (*5 valores*) Concretize estes dois métodos usando semáforos (pode usar tantos semáforos quantos achar necessários).

Sistemas Operativos, Teste 1, 28/outubro/2017

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

Folha de Respostas (1/4)
Número: **Nome:**

§

Tarefas e Troca de Mensagens

Pergunta 1	

```
/*-------------------------------------------------------------------
| Defines
-------------------------------------------------------------------*/
#define BUFFSZ 256

typedef struct {
  int    id;



} argsSimular_t;

/*-------------------------------------------------------------------
| Function: apply_token
-------------------------------------------------------------------*/
void *apply_token(void *a) {
   argsSimular_t *arg   = (argsSimular_t *) a;
   int           myid     = arg->id;
   char          buffer[BUFFSZ];
   int           occurrences = 0;



   while (1) {


     occurrences = toupper_token (buffer, token);



   }
   return 0;
}


/*-------------------------------------------------------------------
| Function: printer
-------------------------------------------------------------------*/
void *printer(void *a) {
   argsSimular_t *arg   = (argsSimular_t *) a;
   int            myid = arg->id;
   char           buffer[BUFFSZ];

   while (1) {



     printf ("%s\n",buffer);
   }
   return 0;
}
```

Folha de Respostas (2/4)

Número:
Nome:

§

Tarefas e Troca de Mensagens

Pergunta 1	(continuação)

```
/*-------------------------------------------------------------------
| Function: main
-------------------------------------------------------------------*/


int main (int argc, char** argv) {
  argsSimular_t     slave_args[3];
  pthread_t         slaves[3];
  char              *token1 = argv[1];
  char              *token2 = argv[2];
  char              buffer[BUFFSZ];

  if (argc < 3) {
    printf("pipeline token1 token2\n");
    return 1;
  }

  /* Inicializa biblioteca de troca de mensagens
  (capacidade do canal=10, número de tarefas comunicantes=4) */
  inicializarMPlib(10,4);

  slave_args[0].id = 1;


  pthread_create(&slaves[0], NULL,                  , &slave_args[0]);



  slave_args[1].id = 2;


  pthread_create(&slaves[1], NULL,                  , &slave_args[1]);



  slave_args[2].id = 3;


  pthread_create(&slaves[2], NULL,                  , &slave_args[2]);



  // le linhas do stdin
  while (fgets (buffer, BUFFSZ, stdin)) {




  }
}
```

Folha de Respostas (3/4)

Número:
Nome:

§

Sincronização com Mutexes e Variáveis de Condição

Pergunta 2	
Pergunta 3	
Pergunta 4	
Pergunta 5	

Folha de Respostas (4/4)

Número:
Nome:

§

Sincronização com Semáforos

Pergunta 6	

```
#define TIPOA 0
#define TIPOB 1

#define outro(x) x? 0: 1









void pedeAcesso (int tipo) {









}
void libertAcesso (int tipo) {









}
```