Trabalho de Computação Concorrente

UFRJ

Rodrigo da Costa Passos – 115196299

Yago Alves da Costa – 115212477

Lógica do Algoritmo

A ideia do algoritmo que implementamos é que o usuário ao executar o código, ele irá informar por linha de comando o número de threads escritoras, o número de threads leitoras, quantidade de escritas, quantidade de leitoras e o nome do arquivo que ele deseja pôr o log de saída, mas esse arquivo necessita terminar com a extensão ".py" para que o nosso programa auxiliar possa executar corretamente. Abaixo na figura 1 podemos ver todas as inicializações que o programa faz na função "main" para que assim o código fique mais legível e funcione corretamente.

Arquivo Editar Formatar Exibir Ajuda

```
int main( int argc, char *argv[]){
   // Valida e recebe os valores de entrada
   if( argc < 6 ) {
     printf( "Use: %s <threads escritoras> <threads leitoras> <numero de leituras> <numero de escritas> <nome_do_arquivo.py>\n", argv[0]);
     exit( EXIT_FAILURE );}
   // Abre o arquivo de log
   pont_arquivo = fopen(argv[5], "w");
   if (pont_arquivo == NULL){
     printf("ERRO! O arquivo não foi aberto!\n");
   }else{
    printf("O arquivo foi aberto com sucesso!\n");}
   // Coloca no primeira linha do arquivo log o import necessario para executar as funcoes printadas do log
   char palavra[25] = "import teste_log as tl\n";
   fprintf(pont_arquivo, "%s", palavra);
   // Quantidades de execucoes que cada thread ira fazer
   n_leituras = atoi(argv[3]);
   n_escritas = atoi(argv[4]);
   // Verifica se quantidade minima de threads escritoras e leitoras
   if( atoi( argv[1] )<2 && atoi( argv[2] )<2 ) {
       printf("Escritoras e/ou leitoras < 2 !\n");
       exit(-1);}
   int M = atoi( argv[1] );// Quantidades de threads de escritas
   int N = atoi( argv[2] );// Quantidades de threads de leitura
   // Espaco para testes quantidades de escritas e leituras
   total_esc = M * n_escritas;
   total_leituras = N * n_leituras;
   printf("Total esc: %d\nTotal leitura: %d\n", total_esc, total_leituras);
   pthread_t tid[M];pthread_t tid_1[N];
   int *t, i;
   //--inicilaiza o mutex (lock de exclusao mutua)
   pthread_mutex_init( &mutex, NULL );
   pthread_cond_init( &cond_leit, NULL );
   pthread_cond_init( &cond_leit2, NULL );
   pthread_cond_init( &cond_escr, NULL );
   pthread_cond_init( &cond_escr2, NULL );
```

Figura 1 – Inicializações feitas na Função Main

O corpo da função "main" no geral funciona como um clássico programa concorrente, onde ele começa a execução das threads leitoras e escritoras de acordo com a quantidade informada na linha de comando e chama a função delas correspondentes. Além disso, o programa possui a "função join" para cada conjunto de threads para que dessa forma o programa só finalize quando todas as threads terminarem. A figura 2 abaixo nos mostra esse trecho do programa que foi mencionado.

```
// Cria as threads. Como inicializo as escritoras primeiro, elas tendem a ganhar a cpu primeiro sempre
for( i =0; i<M; i++ ){
    t = malloc( sizeof( int ) ); if( t==NULL ) return -1;
    *t = i;
    if ( pthread_create( &tid[i], NULL, escritor, (void *)t )) {
        printf( "--ERRO: pthread_create()\n"); exit( -1 );
    }
}
for( i =0; i<N; i++ ){
    t = malloc( sizeof( int ) ); if( t==NULL ) return -1;
    *t = i;
    if (pthread_create( &tid_1[i], NULL, leitor, ( void * )t)) {
        printf( "--ERRO: pthread_create()\n" ); exit( -1 );
    }
}
// Espera todas as threads terminarem
for ( i=0; i<M; i++ ) {
    if ( pthread_join( tid[i], NULL) ) {
        printf( "--ERRO: pthread_join() \n" ); exit( -1 );
    }
}
for ( i=0; i<N; i++ ) {
    if ( pthread_join( tid_1[i], NULL ) ) {
        printf( "--ERRO: pthread_join() \n" ); exit( -1 );
    }
}
//Escreve no final do arquivo de log a funcao que faz uma ultima verificacao no logica de leitor e escritor
char palavra_final[25] = "tl.teste_final()\n";
fprintf(pont_arquivo, "%s", palavra_final);
fclose(pont_arquivo);
pthread_mutex_destroy( &mutex );
pthread_cond_destroy( &cond_leit );
pthread_cond_destroy( &cond_leit2 );
pthread_cond_destroy( &cond_escr );
pthread_cond_destroy( &cond_escr2 );
pthread_exit(NULL);}
```

Figura 2 – Corpo da Função Main

As threads leitoras e escritoras trabalham de formas diferentes, devido às regras que eles possuem, mas nenhuma delas possuem uma prioridade no programa, de forma que não haja inanição durante o decorrer do programa. Uma característica do programa no geral é que as threads escritoras não podem escrever enquanto uma thread leitora estiver lendo e vice versa.

Começando pelas threads leitoras, elas possuem uma característica que mais de um leitor pode ler ao mesmo tempo, essa peculiaridade é garantida pelo código que a thread leitora executa. No início da execução delas, elas chamam uma função chamada “leitor” essa função garante que o leitor que está sendo executado leia a variável compartilhada pelas threads e também garante que essa função execute as funções “EntraLeitura” e “SaiLeitura” que separam a lógica do programa com variáveis de condições nessas duas funções. E também ajudam a thread a não gerar inanição nem mesmo um “deadlock”. A figura 3 abaixo apresenta a função “leitor” do programa:

```
void *leitor ( void *arg ) {
   int tid = *( int* )arg; // Id da thread
   int i = 0; // Zera o contador de vezes do while
   int var_local;
   char nome_arq[7];
   char id[2];
   char lido[2];
   FILE *arq_saida_leitora;

   sprintf(id, "%i", tid); // Converte o int tid no char e salva em id
   sprintf( nome_arq, "%s.txt", id); // Concatena o char id com a terminacao do arquivo e salva em nome_arq

   arq_saida_leitora = fopen(nome_arq, "w"); // Abre a arquivo das leitoras
   if (arq_saida_leitora == NULL){
     printf("ERRO! O arquivo da leitora %i não foi aberto!\n", tid);
   }

   while( i < n_leituras ) {
       EntraLeitura( tid );
       var_local = var_com; // Le a variavel compartilhada
       sprintf(lido, "%i", var_local); // Converte o int var_local no char e salva em lido
       fprintf(arq_saida_leitora, "%s\n", lido); // Escreve no arquivo exclusivo da thread
       printf("L%d: Li o valor %d.\n", tid, var_local);
       SaiLeitura( tid );
       i++;
   }

   fclose(arq_saida_leitora);
   pthread_exit(NULL);
}
```

Figura 3 – Função Leitor

A função “EntraLeitura” primeiramente verifica se a thread leitora está executando primeiro que a escritora ou não, para que assim no decorrer do programa ela possa seguir um fluxo correto durante a execução. Caso ela seja a primeira, ela vai seguir um fluxo de execução normal, caso contrário ela vai entrar no “wait” e gerar uma primeira fila de leitoras, essa fila vai estar contabilizada na variável “esperandoleitura”, esperando ser liberada quando um thread escritora perceber essa fila. Após a escrita perceber a fila, ela bloqueia novos escritores e ao término das escritoras restantes, libera essa primeira fila de leitores. Depois as leitoras verificam se existe alguma escritora escrevendo (lógica padrão do leitor/escritor), caso tenha, a thread será bloqueada e irá aguardar receber um sinal, podendo gerar uma segunda fila de leitoras, essa fila vai estar contabilizada na variável “esperandoleitura2”. Em seguida ela incrementa o número de leitoras e imprime no arquivo de log o que está acontecendo com ela (olhar a figura 4 abaixo).

```
void EntraLeitura(int tid) {
   pthread_mutex_lock(&mutex);
   //Essa parte  só serve no inicio e seta uma condição para a primeira leitora
   existe_leitura++;
   if ( existe_leitura  == 1 && existe_escrita == 0){
       decisao_leit = 0;
   }
   if ( decisao_leit == 1) {
       esperando_leitura++;
       printf("L%d: Bloqueio! Aguardo na fila 1!\n", tid);
       pthread_cond_wait(&cond_leit, &mutex);
       esperando_leitura--;
   }
   while ( escr > 0 ) {
       esperando_leitura2++;
       printf("L%d: Bloqueio! Aguardo na fila 2\n", tid);
       pthread_cond_wait(&cond_leit2, &mutex);
       esperando_leitura2--;
   }
   leit++;
   //Para que testar a corretude da logica pelo o log, preciso marcar quando a primeira leitora de uma nova remessa entra.
   if ( leit == 1) {
       printf("L%d: Entrei! Sou a primeira.\n", tid);
       char palavra[30] = "tl.entra_leitora_primeira()\n"; //Escreve no arquivo do log: entra_leitora_primeira()
       fprintf(pont_arquivo, "%s", palavra);
   } else {
       printf("L%d: Entrei!\n", tid);
   }
   char palavra[1000] = "tl.entra_leitora()\n";//Escreve no arquivo do log: entra_leitora()
   fprintf(pont_arquivo, "%s", palavra);
   pthread_mutex_unlock(&mutex);
}
```

Figura 4 – Função “EntraLeitura”

Para garantir que as threads leitoras não fiquem executando por uma quantidade de tempo grande e liberem as escritoras para serem executadas, temos a função “Sai Leitura” que está apresentada na Figura 5 abaixo. Após o decremento da variável “leit”, indicando término dessa leitura, ela verifica se foi a última das leitoras. Caso isso seja verdade, ela faz algumas verificações, primeiramente ele verifica se existe alguma thread escritora esperando para ser executada na fila 2 de escritoras, em caso positivo ele manda um sinal para a thread escritora e depois muda o valor das variáveis de decisão que iram ajudar a que as threads leitoras gerem uma fila nova e as escritoras sejam liberadas. Caso a primeira não seja verdade, ele verifica se tem uma thread leitora esperando e libera elas. Caso contrário, ele verifica se existe uma thread escritora esperando na fila 1 e depois dá um broadcast liberando elas para a fila 2.

Caso as leituras não sejam iguais a zero quando chegar na função “Sai Leitura”, ela segue um fluxo normal, imprimindo as coisas que são necessárias no arquivo de log para utilidade do nosso programa auxiliar e verifica se há escritoras na fila 1 e muda a variavel “decisão_leit” para 1 gerando fila de novas leitoras .

```
void SaiLeitura(int tid) {
   pthread_mutex_lock(&mutex);
   leit--;
   char palavra[25] = "tl.sai_leitora()\n";//Escreve no arquivo do log: sai_leitora()
   fprintf(pont_arquivo, "%s", palavra);
   if(leit==0){
       if ( esperando_escrita2!=0 ) {
           printf("L%d: Terminei! Sou a ultima e acordei escritora da fila 2\n", tid);
           pthread_cond_signal(&cond_escr2);
           decisao_esc = 1; // Gero fila 1 de escritores
           decisao_leit = 1; // Gero fila leitores
       } else if ( esperando_leitura!= 0) {
           printf("L%d: Terminei! Sou a ultima e acordei mais leitoras da fila 1\n", tid);
           pthread_cond_broadcast(&cond_leit);
       } else if ( esperando_escrita!= 0 ) {
           printf("L%d: Terminei! Sou a ultima e acordei escritoras da fila 1\n", tid);
           decisao_leit = 1; // Gero fila de leitores
           pthread_cond_broadcast(&cond_escr);
       } else {
           printf("L%d: Terminei! Sou a ultima.\n", tid);
           decisao_leit = 0; // Bloqueio geracao de fila leitores
           decisao_esc = 0;  // Se for a ultima leitura libero as escritoras }
       char palavra[30] = "tl.sai_leitora_ultima()\n"; //Escreve no arquivo do log: sai_leitora_ultima()
       fprintf(pont_arquivo, "%s", palavra);
   } else {
       printf("L%d: Terminei!\n", tid);
       if ( esperando_escrita != 0){
           printf("L%d: Há escritoras na fila 1, bloqueio novas leitoras!\n", tid);
           decisao_leit = 1;}
   }
   //Variavel para contabilizar leituras realizadas, não está aplicada na logica do programa
   cont_leit++;
   printf("Leituras até agora: %d  Total: %d\n", cont_leit, total_leituras);
   pthread_mutex_unlock(&mutex);}
```

Figura 5 – Função Sai Leitura

Já as threads escritoras, possuem algumas restrições que são características dela. Apenas um escritor pode escrever de cada vez e durante o desenvolvimento do nosso programa nos foi possível garantir que essa característica estivesse presente no programa.

Conforme a Figura 6 abaixo, as threads escritoras começam executando a função escritor. Essa função é bem simples, pois ela só chama as funções “EntraEscrita” e “SaiEscrita”, que são as funções que garantem com que as características do programa sejam atendidas. Além disso, ela também escreve o id da thread escritora na variável compartilhada.

```
void *escritor ( void *arg ) {
   int tid = *(int*)arg;
   int i=0;
   while( i < n_escritas ) {
       EntraEscrita(tid);
       var_com = tid; // Escreve a id da thread escritora na varivel compartilhada por todas ( escritoras e leitoras ).
       printf("E%d: Escrevi o valor %d\n", tid, tid);
       SaiEscrita(tid);
       i++;
   }
   pthread_exit(NULL);
}
```

Figura 6 – Função Escritor

A função “EntraEscrita” primeiramente verifica se a thread está sendo a primeira entre todas as threads escritoras e leitoras, para que no caso disso ser verdade, ela poder seguir o fluxo do programa sem nenhum impedimento. Em seguida, ela verifica pela variável “decisao_esc” se ela vai ser bloqueada logo nessa primeira fila de escritoras, ou se ela pode

seguir o fluxo do programa. Logo após, ela verifica se existe alguma escritora ou leitora sendo executada, em caso positivo, elas ficam paradas em uma segunda fila nesse trecho, esperando até o momento que elas recebam um sinal de uma thread leitora ou escritora. Caso a thread não fique bloqueada em nenhuma dessas filas, ela segue o fluxo normal, imprimindo as coisas necessárias no arquivo de log e incrementando a variável de escritas. A figura 7 abaixo apresenta a função descrita acima.

```
void EntraEscrita (int id) {
   pthread_mutex_lock(&mutex);
   //Essa parte  só serve no inicio e seta uma condição para a primeira escritora
   existe_escrita++;
   if ( existe_escrita == 1 && existe_leitura == 0 ) {
       decisao_esc = 0;
   }
   if( decisao_esc == 1) {
       esperando_escrita++;
       printf("E%d: Bloqueio! Aguardo na fila 1.\n", id);
       pthread_cond_wait(&cond_escr, &mutex);
       esperando_escrita--;
   }

   while ( escr>0 ||  leit>0 ) {
       esperando_escrita2++;
       printf("E%d: Bloqueio! Aguardo na fila 2.\n", id);
       pthread_cond_wait(&cond_escr2, &mutex);
       esperando_escrita2--;
   }

   escr++;
   printf("E%d: Entrei!\n", id);
   char palavra[25] = "tl.entra_escrita()\n"; //Escreve no arquivo do log: entra_escrita()
   fprintf(pont_arquivo, "%s", palavra);

   pthread_mutex_unlock(&mutex);
}
```

Figura 7 – Função Entra Escrita

A função “SaiEscrita” primeiro verifica se existe alguma escritora esperando na segunda fila, em caso positivo ela usa as variáveis de decisão para gerar uma fila nas leitoras e faz as escritoras esperarem na fila 1 de escritoras, além disso ela libera uma thread escritora que está presente nessa segunda fila de escritoras. Caso contrário, ele verifica se tem alguma leitora esperando na segunda fila 2 e as liberam, bloqueando novos escritores na primeira fila. No caso em que nenhum desses três sejam verdade, ele verifica se tem alguém esperando na primeira fila de escritores, faz as variáveis de decisão gerar uma fila de leitoras e libera todas as escritoras presentes na primeira fila. Se nenhuma dessas condições acima foram atendidas, ele apenas muda as variáveis de decisão para que assim o programa siga o seu fluxo normal e faz uma última verificação de possíveis leitoras na fila 1. Durante todo esse processo ele imprime no arquivo de log conforme o fluxo do programa. A Figura 8 abaixo apresenta essa função.

```
void SaiEscrita (int id) {
   pthread_mutex_lock(&mutex);
   escr--;
   char palavra[25] = "tl.sai_escrita()\n"; // Escreve no arquivo do log: sai_escrita()
   fprintf(pont_arquivo, "%s", palavra);
   if(esperando_escrita2 != 0){
       decisao_leit = 1; // Gera fila de leitoras
       decisao_esc = 1; // Gera fila de escritoras
       pthread_cond_signal(&cond_escr2); // Ainda tenho escritoras na fila 2 para terminarem
       printf("E%d: Terminei! Acordei uma escritora fila 2.\n", id);
   }else if (esperando_leitura2 != 0 ) {
       printf("E%d: Terminei! Acordei leitoras da fila 2.\n", id);
       decisao_esc = 1; // Gera fila 1 das escritoras
       pthread_cond_broadcast(&cond_leit2);
   } else if ( esperando_escrita!=0 ) {
       decisao_leit = 1; // Gera fila leitoras
       printf("E%d: Termenei! Acordei escritoras da fila 1.\n", id);
       pthread_cond_broadcast(&cond_escr); // Escritoras passam da primeira fila e novas escriras ficaram nessa fila até essas passarem pela fila
   } else {
       printf("E%d: Termenei!\n", id);
       // Se ultima escrita e não tem fila na escrita nem na leitora, permite qualquer uma ganhar a condição de corrida
       decisao_esc = 0;
       decisao_leit = 0;
       if (esperando_leitura!= 0){
           printf("E%d: Ainda há leitoras na fila 1, vou acordalas!\n", id);
           pthread_cond_broadcast(&cond_leit);
       }
   }
   //Variavel para contabilizar escritas realizadas, não está aplicada na logica do programa
   cont_esc++;
   printf("Escritas até agora: %d de total: %d\n", cont_esc, total_esc);
   pthread_mutex_unlock(&mutex);
}
```

Figura 8 – Função Sai Escrita

A figura 9 abaixo, apresenta todas as inicialização feita das variáveis de condições, das variáveis de mutex e as variáveis locais para que dessa forma o programa seja executada de forma correta.

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

pthread_mutex_t mutex;
pthread_cond_t cond_escr, cond_leit, cond_leit2, cond_escr2;

//Globais
int leit=0,escr=0, esperando_escrita = 0, esperando_leitura = 0, esperando_leitura2 = 0, esperando_escrita2 = 0, existe_escrita = 0, existe_leitura = 0;
int n_leituras = 0;
int n_escritas = 0;

int decisao_leit = 0; // Sinalizador que pode assumir valor 0 ou 1, se 0 leitoras não entram na fila ou se for 1 entram na fila
int decisao_esc = 0;  // Mesmo caso do de cima, mas para escritores

int var_com = -1; //Variavel compartilhada entre as threads escritoras e leitoras.

FILE *pont_arquivo; // Arquivo de log

//Espaco para testes quantidades de escritas e leituras. Não aplicado na logica do programa
int total_esc = 0, cont_esc = 0;
int total_leituras = 0, cont_leit =0;
```

Figura 9 – Inicializações do programa

Programa Auxiliar

O programa auxilar ***teste_log.py*** verifica se a lógica de leitor e escritor foi respeitada após execução do programa ***leit_escr.c*** .

Variáveis globais do programa:

Utilizamos duas listas (***leitoras*** e ***escritoras***) para indicar quantas threads leitoras e threads escritoras estão em realizando suas devidas tarefas. E também os sinalizadores ***l*** e ***s***, eles indicam que tenho as condições necessárias para ler ou escrever.

```
#Listas que indicam quantos leitores ou escri
leitoras = []
escritoras = []

# Sinalizadores para l(leitor) e s(escritor).
l = 0
s = 0
```

Funções:

1) **entra_leitora_primeira( ) -**

```
def entra_leitora_primeira():
    global l, leitoras
    n = len(leitoras)        # Quantidade de leitoras na lista
    if ( n == 0 and l == 0): # Se lista de leitoras vazias e variavel de condicao l = 0
        l =  1               # Variavel l vai para 1 e permite leitoras e "bloqueia" escritores
    else:
        print "Erro na entrada da primeira leitura, não atendeu as condicoes necessarias"
        quit()
```

2) **entra_leitora( ) -**

```
def entra_leitora():
    global l, s, leitoras
    if (l == 1 and s == 0): # Se s = 0 indica que não tenho escrita sendo realizadas
        leitoras.append(1)  # Insere 1 na lista leitora, indicando um leitor lendo
        print "leitoras: "
        print leitoras      # Printa lista de leitoras naquele momento
    else:
        print "Erro na leitura: ainda pode ter escritoras em execucao"
        quit()
```

3) sai_leitora( ) -

```
def sai_leitora():
    global l, s, leitoras
    if (l == 1 and s == 0): # Mesma condicao do entre_leitora().
        del(leitoras[0])    # Retira 1 da lista leitora, indicando o termino de leitor
        print "leitoras: "
        print leitoras      # Printa lista de leitoras naquele momento
    else:
        print "Erro na saida da leitura"
        quit()
```

4) sai_leitora_ultima( ) -

```
def sai_leitora_ultima():
    global l, leitoras
    n = len(leitoras)       # Pega a quantidade de leitoras na lista
    if ( n == 0 and l == 1):# Se todas as leitoras terminaram de le
        l =  0              # Indico que escritoras podem excutar e novas leitoras no entra_primeira_leitora()
    else:
        print "Erro na saida da ultima leitura: ainda pode ter leitoras na fila"
        quit()
```

5) **entra_escrita( ) -**

```
def entra_escrita():
    global l, s, escritoras
    n = len(escritoras)     # Pega a quantidade de escritoras na fila
    if (n == 0 and s == 0): # Condicao sera aceita se nao tiver escritoras em execucao
        s =  1              # Sinalizo que tem escrita em execucao
        if (s == 1 and l == 0 and n < 1):
            escritoras.append(1)    # Insere 1 na lista de escritores
            print "Escritoras: "
            print escritoras        # Printa lista de escritores no momento
        else:
            print "Erro na escrita: ainda ha outra escrita em execucao"
            quit()
    else:
        print "Erro na escrita: condicoes nao atendidas"
        quit()
```

6) sai_escrita( ) -

```
def sai_escrita():
    global l, s, escritoras
    if (s == 1 and l == 0): # Se condicao para sair da escrita atendidas
        del(escritoras[0])  # Retira a escritora da lista
        print "Escritoras: "
        print escritoras    # Printa lista de escritores no momento
        n = len(escritoras) # Pega a quantidade de escritoras na fila
        if ( n == 0 and s == 1): # Se quantidade n = 0 e condicao de escrita em execucao
            s =  0          # Indico que nao ha mais escritas, permitindo novas escritas ou leituras
        else:
            print "Erro na escrita"
            quit()
    else:
        print "Erro saida da escrita"
        quit()
```

7) teste_final( ) -

```
# Fucao inserida no final do arquivo de log pela da execucao do programa em C
def teste_final():
    global leitoras, escritoras
    t1 = len(leitoras)          # Pego a quantidade de leitoras
    t2 = len(escritoras)        # Pego a quantidade de escritoras
    if ( t1 == 0 and t2 == 0):  # Se as duas terminaram vazias
        print "Ok!"             # Condição final aceita
    else:
        print "Erro no final!"
```

Corretude do programa auxiliar:

Caso 1)

```
#Testa o programa aqui simulando os log
def main():
    entra_leitora_primeira() # Necessariamente uma entra_leito
    entra_leitora()
    entra_leitora()
    sai_leitora ()
    entra_leitora()
    sai_leitora ()
    entra_escrita()
    sai_leitora ()
    sai_leitora_ultima()
    sai_escrita()

# Descomentar aqui em baixo para executar os testes da main()
main()
```

Simulo esse possível caso de erro na saída do arquivo de log (imagem logo acima) do programa em c, onde uma escritora tenta escrever antes da finalização da leitura. Executo essa ordem das chamadas das funções (imagem logo abaixo).

Obs.: Necessariamente uma entra_leitora_primeira() vem seguida de uma entra_leitora(), segundo a lógica do programa que gera os logs. E as duas juntas representam os log de uma única threads leitora entrando na leitura. O mesmo acontece com uma última thread leitora, só que com as funções sai_leitora() e sai_leitora_ultima().

```
yago@yagoadc:~/Downloads/Conc/T2$ python2.7 teste_log.py
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
Erro na escrita: Pode haver outra escrita em execucao ou leitora
```

Caso 2) Agora ao contrário, uma leitora tenta lê antes da saída da escritora.

```
def main():
    entra_leitora_primeira()
    entra_leitora()
    entra_leitora()
    sai_leitora ()
    entra_leitora()
    sai_leitora ()
    sai_leitora ()
    sai_leitora_ultima()
    entra_escrita()
    sai_escrita()
    entra_escrita()
    entra_leitora_primeira()
    entra_leitora()
# Descomentar aqui em baixo
main()
```

```
yago@yagoadc:~/Downloads/Conc/T2$ python2.7 teste_log.py
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[]
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Erro na leitura: ainda pode haver escritoras em execucao
```

Caso 3) Simulação da execução correto

```
def main():
    entra_leitora_primeira()
    entra_leitora()
    entra_leitora()
    sai_leitora ()
    entra_leitora()
    sai_leitora ()
    sai_leitora ()
    sai_leitora_ultima()
    entra_escrita()
    sai_escrita()
    teste_final()
```

```
yago@yagoadc:~/Downloads/Conc/T2$ python2.7 teste_log.py
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[]
Escritoras:
[1]
Escritoras:
[]
Ok!
```

Casos de Teste

Para executar o programa:

```
yago@yagoadc:~/Downloads/Conc/T2$ gcc -o saida leit_escr.c -lpthread -Wall
yago@yagoadc:~/Downloads/Conc/T2$ ./saida
Use: ./saida <threads escritoras> <threads leitoras> <numero de leituras> <numero de escritas> <nome do arquivo.py>
```

Caso 1)

```
yago@yagoadc:~/Downloads/Conc/T2$ ./saida 2 2 2 2 log.py
O arquivo foi aberto com sucesso!
Total esc: 4
Total leitura: 4
E0: Entrei!
E0: Escrevi o valor 0
E0: Termenei!
Escritas até agora: 1 de total: 4
E0: Entrei!
E0: Escrevi o valor 0
E1: Bloqueio! Aguardo na fila 2.
E0: Terminei! Acordei uma escritora fila 2.
Escritas até agora: 2 de total: 4
E1: Entrei!
E1: Escrevi o valor 1
E1: Termenei!
Escritas até agora: 3 de total: 4
E1: Entrei!
E1: Escrevi o valor 1
E1: Termenei!
Escritas até agora: 4 de total: 4
L0: Entrei! Sou a primeira.
L1: Entrei!
L0: Li o valor 1.
L0: Terminei!
Leituras até agora: 1  Total: 4
L0: Entrei!
L1: Li o valor 1.
L1: Terminei!
Leituras até agora: 2  Total: 4
L1: Entrei!
L1: Li o valor 1.
L1: Terminei!
Leituras até agora: 3  Total: 4
L0: Li o valor 1.
L0: Terminei! Sou a ultima.
Leituras até agora: 4  Total: 4
```

```
yago@yagoadc:~/Downloads/Conc/T2$ python2.7 log.py
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[]
Ok!
```

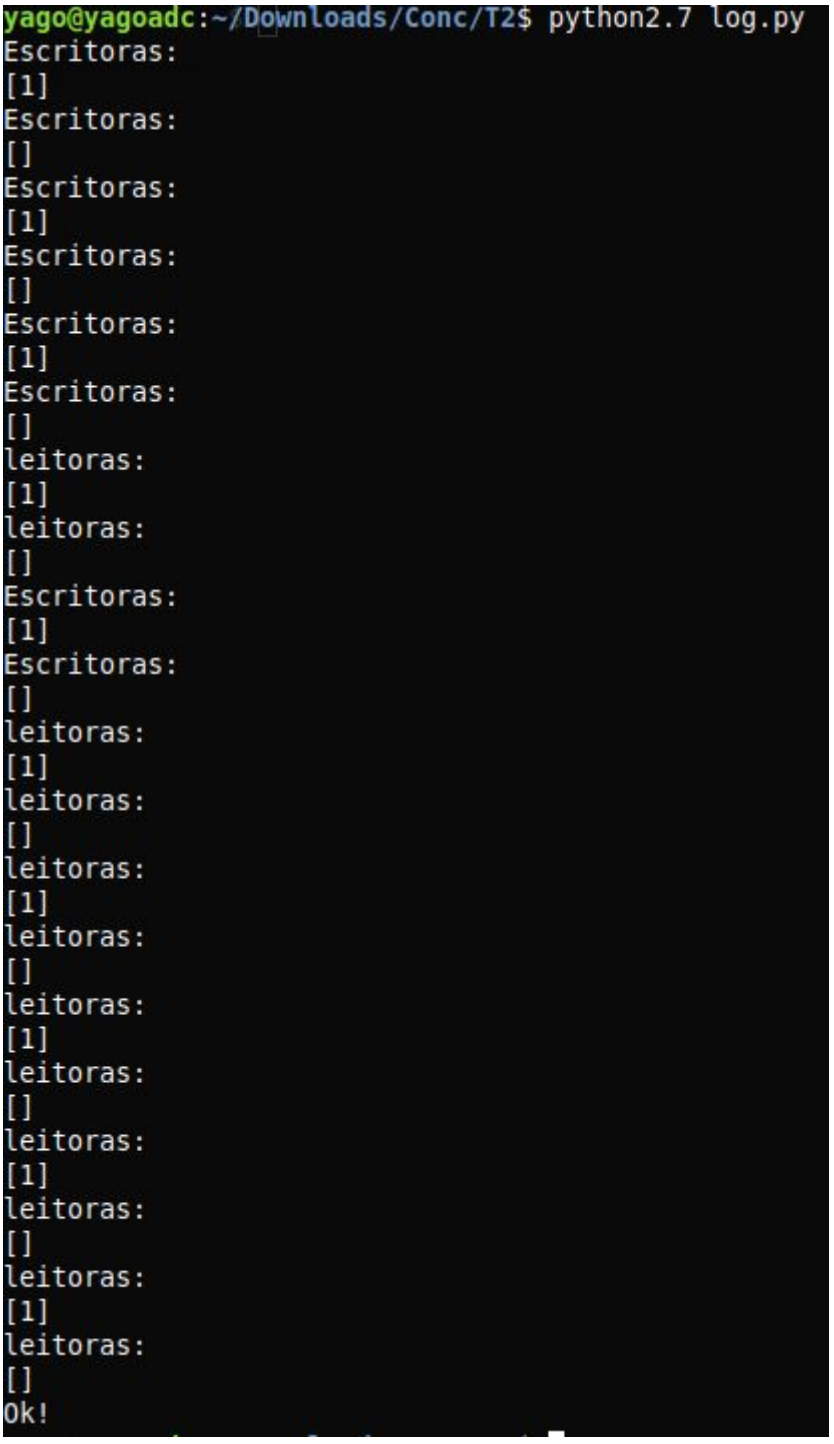
Caso 2)

```
yago@yagoadc:~/Downloads/Conc/T2$ ./saida 3 2 2 2 log.py
O arquivo foi aberto com sucesso!
Total esc: 6
Total leitura: 4
E0: Entrei!
E0: Escrevi o valor 0
E0: Termenei!
Escritas até agora: 1 de total: 6
E2: Entrei!
E2: Escrevi o valor 2
E0: Bloqueio! Aguardo na fila 2.
L0: Bloqueio! Aguardo na fila 2
L1: Bloqueio! Aguardo na fila 2
E1: Bloqueio! Aguardo na fila 2.
E2: Terminei! Acordei uma escritora fila 2.
Escritas até agora: 2 de total: 6
E2: Bloqueio! Aguardo na fila 1.
E0: Entrei!
E0: Escrevi o valor 0
E0: Terminei! Acordei uma escritora fila 2.
Escritas até agora: 3 de total: 6
E1: Entrei!
E1: Escrevi o valor 1
E1: Terminei! Acordei leitoras da fila 2.
Escritas até agora: 4 de total: 6
E1: Bloqueio! Aguardo na fila 1.
L1: Entrei! Sou a primeira.
L0: Entrei!
L1: Li o valor 1.
L1: Terminei!
L1: Há escritoras na fila 1, bloqueio novas leitoras!
Leituras até agora: 1  Total: 4
L1: Bloqueio! Aguardo na fila 1!
L0: Li o valor 1.
L0: Terminei! Sou a ultima e acordei mais leitoras da fila 1
Leituras até agora: 2  Total: 4
L0: Bloqueio! Aguardo na fila 1!
L1: Entrei! Sou a primeira.
L1: Li o valor 1.
L1: Terminei! Sou a ultima e acordei mais leitoras da fila 1
Leituras até agora: 3  Total: 4
L0: Entrei! Sou a primeira.
L0: Li o valor 1.
L0: Terminei! Sou a ultima e acordei escritoras da fila 1
Leituras até agora: 4  Total: 4
E1: Entrei!
E1: Escrevi o valor 1
E1: Termenei! Acordei escritoras da fila 1.
Escritas até agora: 5 de total: 6
E2: Entrei!
E2: Escrevi o valor 2
E2: Termenei!
Escritas até agora: 6 de total: 6
```


yago@yagoadc:~/Downloads/Conc/T2$ python2.7 log.py
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
leitoras:
[1]
leitoras:
[1, 1]
leitoras:
[1]
leitoras:
[]
leitoras:
[1]
leitoras:
[]
leitoras:
[1]
leitoras:
[]
Escritoras:
[1]
Escritoras:
[]
Escritoras:
[1]
Escritoras:
[]
Ok!

Caso 3)

```
yago@yagoadc:~/Downloads/Conc/T2$ ./saida 2 3 2 2 log.py
O arquivo foi aberto com sucesso!
Total esc: 4
Total leitura: 6
E0: Entrei!
E0: Escrevi o valor 0
E0: Termenei!
Escritas até agora: 1 de total: 4
E0: Entrei!
E0: Escrevi o valor 0
E1: Bloqueio! Aguardo na fila 2.
L0: Bloqueio! Aguardo na fila 2
L2: Bloqueio! Aguardo na fila 2
E0: Terminei! Acordei uma escritora fila 2.
Escritas até agora: 2 de total: 4
E1: Entrei!
E1: Escrevi o valor 1
E1: Terminei! Acordei leitoras da fila 2.
Escritas até agora: 3 de total: 4
E1: Bloqueio! Aguardo na fila 1.
L0: Entrei! Sou a primeira.
L0: Li o valor 1.
L0: Terminei! Sou a ultima e acordei escritoras da fila 1
Leituras até agora: 1  Total: 6
E1: Entrei!
E1: Escrevi o valor 1
E1: Terminei! Acordei leitoras da fila 2.
Escritas até agora: 4 de total: 4
L0: Bloqueio! Aguardo na fila 1!
L2: Entrei! Sou a primeira.
L2: Li o valor 1.
L2: Terminei! Sou a ultima e acordei mais leitoras da fila 1
Leituras até agora: 2  Total: 6
L2: Bloqueio! Aguardo na fila 1!
L0: Entrei! Sou a primeira.
L0: Li o valor 1.
L0: Terminei! Sou a ultima e acordei mais leitoras da fila 1
Leituras até agora: 3  Total: 6
L2: Entrei! Sou a primeira.
L2: Li o valor 1.
L2: Terminei! Sou a ultima.
Leituras até agora: 4  Total: 6
L1: Entrei! Sou a primeira.
L1: Li o valor 1.
L1: Terminei! Sou a ultima.
Leituras até agora: 5  Total: 6
L1: Entrei! Sou a primeira.
L1: Li o valor 1.
L1: Terminei! Sou a ultima.
Leituras até agora: 6  Total: 6
```



*Obs.: Por causa do forma de iniciar as threads, as escritoras provavelmente sempre ganharão a cpu primeiro.