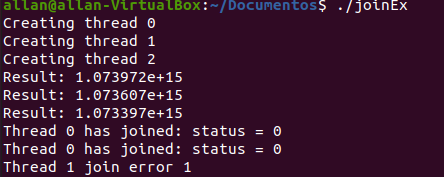
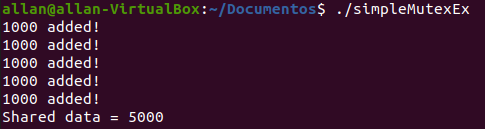
1. As partes em **negrito** devem constar no relatório
2. Faça *download* do [arquivo](https://drive.google.com/file/d/0B4HCJCgCTQHYSlR5aVZfUVdTQWM/edit?usp=sharing) para a prática
3. Descompacte o arquivo
4. Observe o programa joinEx.c. Para entender o que ele faz, procure na documentação [API da biblioteca pthreads](https://computing.llnl.gov/tutorials/pthreads/#CreatingThreads) os tipos e as funções que você não conhece:
   * pthread\_t
   * pthread\_attr\_t
   * pthread\_attr\_init
   * pthread\_attr\_setdetachstate
   * pthread\_create
   * pthread\_attr\_destroy
   * pthread\_join
5. **Dê uma descrição do que faz cada um desses tipos ou funções.**
   * pthread\_t: é o tipo que a variavel do thread é declarado.
   * pthread\_attr\_t: é o tipo que a variavel thread objeto deve ser declarada
   * pthread\_attr\_init: Inicia os atributos do thread objeto attr com o valor padrão para todos os atributos individuais usado por uma determinada implementação.
   * pthread\_attr\_setdetachstate: Ela controla se a thread deve ser criada em um estado separado a chamada ptherad\_attr\_setdetachstate contra o estado de uma thread ja criada se for separado ou juntavel.
   * pthread\_create: Cria uma nova thread e o torna executavel.
   * pthread\_attr\_destroy: Esse comando destroi os atributos de uma thread objeto.
   * pthread\_join: Ela suspende a execução da chamada thread ate que o thread de destino Termine, a menos que o thread de destino ja tenha sido encerrado.
6. **Compile (gcc -o joinEx -pthread joinEx.c) e rode o programa. O que ele faz? Explique cada linha do resultado da saída do programa.**



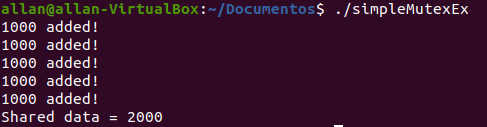
Nas primeiras 3 linhas ele cria os threads 0, 1 e 2, da 4 até a 6 ele da o resultado da função theWork de cada thread, na linha 7, 8 ela mostra a chamada da função para suspender a thread de destino e seus status, na ultima linha ela mostra que tenta chamar para a thread 1 mas retorna com erro pois o thread de detino ja terminou .

1. Observe o programa simpleMutex.c. Para entender o que ele faz, procure na documentação [API da biblioteca pthreads](https://computing.llnl.gov/tutorials/pthreads/#CreatingThreads) os tipos e as funções que você não conhece:
   * pthread\_mutex\_t
   * pthread\_mutex\_lock
   * pthread\_mutex\_unlock
2. **Dê uma descrição do que faz cada um desses tipos ou funções.**
   * pthread\_mutex\_t :as variaveis mutex são declarados dessa forma, e deve ser inicializadas antes de serem usadas e inicilmente é desbloqueado.
   * pthread\_mutex\_lock: é usado por um thread para adquirir um bloqueio exclusivo na variavel mutex especificada, se o multex ja estiver bloqueado por outro thread ele fica em espera até que o outro bloqueio termine
   * pthread\_mutex\_unlock: ele obrigatoriamente deve liberar um mutex que esteja bloqueado somente se a thread que fez a chamada é propietario daquele mutex, caso o contrario sera exibido um erro e tambem se não estiver com bloqueio.
3. **Rode o programa. O que ele faz?**



**R:** cada thread execultada adiciona 1000 a variavel global e no final é exibido o resultado da variavel global.

1. **Retire as diretivas de sincronização "lock" e "unlock". O que acontece com o programa? Você saberia explicar esse comportamento?**



R: o resultado da variavel global da incorreto, pois como o a thread não tinha acesso exclusivo à variavel global, quando era sua vez de mexer na variavel o seu tempo de CPU ja tinha esgotado guardando o seu ultimo estado, e quando voltava novamente para a sua vez ele adiciona o resultado dele, em cima do resultado de outra thread, assim criando esse resultado errado.

1. Observe o programa prodconsUnsync.c, que é uma implementação do problema do produtor/consumidor. Procure entender o código. Ele se comporta de maneira adequada? Responda às seguintes perguntas para avaliar o seu entendimento a respeito do programa:
   * **Qual o tamanho do *buffer*? 1**
   * **Pra que serve a variável *writeable* (dentro do buffer)? Para dizer se o buffer esta disponivel para escrita, ou seja cheio.**
   * **Se o programa funcionasse corretamente, qual seria sua saída no final? 55 e com encerramento.**
2. Adicione algum código usando join() e exit(), como em joinEx.c para que o programa termine de maneira adequada (sem precisar de ctrl+C). Melhore o comportamento do programa usando mutexes. Siga as seguintes dicas:
   * Lembre-se que você deve usar mutexes tanto para garantir exclusão mútua quanto para evitar condições de *buffer* cheio ou vazio.
   * Use a variável *writeable* para garantir as condições de escrita no *buffer*.
   * As regiões críticas estão dentro das funções *store* e *retrieve*. É dentro delas que você deve adicionar o código dos mutexes.
   * **É muito provável que, após corrigir seu código, às vezes no terminal apareça que um número foi retirado antes de ser armazenado. Investigue e explique por que isso acontece.**
3. **Adicione seu código corrigido ao relatório.**

/\*  
 \* Producer/Consumer demo using POSIX threads without synchronization  
 \* Linux version  
 \* MJB Apr'05  
 \*/  
#include <pthread.h>  
#include <stdlib.h>  
#include <stdio.h>  
  
/\* buffer using a shared integer variable \*/  
typedef struct  
{  
 int writeable; /\*true/false\*/  
 int sharedData;  
 int finish; /\*true/false\*/  
} buffer;  
  
buffer theBuffer; /\* global \*/  
pthread\_mutex\_t mutex;  
  
  
void delay(int secs)  
{  
 time\_t beg = time(NULL), end = beg + secs;  
 do  
 {  
 ;  
 } while (time(NULL) < end);  
}  
  
int store(int item, buffer \*pb)  
{  
 pthread\_mutex\_lock(&mutex);  
 pb->sharedData += item;  
 pb->writeable = !pb->writeable;  
 pthread\_mutex\_unlock(&mutex);  
 return 0;  
}  
  
int retrieve(buffer \*pb)  
{  
 pthread\_mutex\_lock(&mutex);  
 int data = pb->sharedData;  
 pb->sharedData -= data;  
 pb->writeable = !pb->writeable;  
   
 return data;  
}  
  
void \*producer(void \*n)  
{  
 int j = 1, tot\_prod = 0;  
 while (j <= 10)  
 {  
 store(j, &theBuffer);  
 printf("Produzido: %d | Total no buffer: %d\n", j, theBuffer.sharedData);  
 tot\_prod += j++;  
 delay(rand() % 6); /\* up to 5 sec \*/  
 }  
 theBuffer.finish = 0;  
 printf("Total produzido = %d\n", tot\_prod);  
 return n;  
}  
  
void \*consumer(void \*n)  
{  
 int j = 0, tot = 0;  
 while (theBuffer.finish)  
 {  
 if (theBuffer.sharedData > 0) {  
 j = retrieve(&theBuffer);  
 tot += j;  
 pthread\_mutex\_unlock(&mutex);  
 printf("Consumido: %d | Total no buffer: %d\n", j, theBuffer.sharedData);  
 delay(rand() % 6); /\* up to 5 sec \*/  
 }  
 }  
 printf("Total consumido = %d\n", tot);  
 return (n);  
}  
  
int main()  
{  
 pthread\_attr\_t attr;  
 pthread\_attr\_init(&attr);  
 pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);  
 pthread\_t prodThread, consThread;  
 int e, st;  
  
 theBuffer.writeable = 1;  
 theBuffer.finish = 1;  
 pthread\_mutex\_init(&mutex, NULL);  
 srand(time(NULL));  
 printf("Iniciou!\n");  
  
 if (e = pthread\_create(&prodThread, &attr, producer, NULL) != 0) {  
 printf("Criação da thread produtora falhou! Erro %d\n", e);  
 exit(-1);  
 }  
   
 if (e = pthread\_create(&consThread, &attr, consumer, NULL) != 0) {  
 printf("Criação da thread consumidora falhou! Erro %d\n", e);  
 exit(-1);  
 }  
   
 if (e = pthread\_join(prodThread, (void \*\*)&st) != 0) {  
 printf("Thread produtora erro ao juntar %d\n", e);  
 exit(-1);  
 }  
 printf("Thread produtora uniu: status = %x\n", st);  
  
 if (e = pthread\_join(consThread, (void \*\*)&st) != 0) {  
 printf("Thread consumidora erro ao juntar %d\n", e);  
 exit(-1);  
 }  
 printf("Thread consumidora uniu: status = %x\n", st);  
  
 pthread\_exit(NULL);  
}