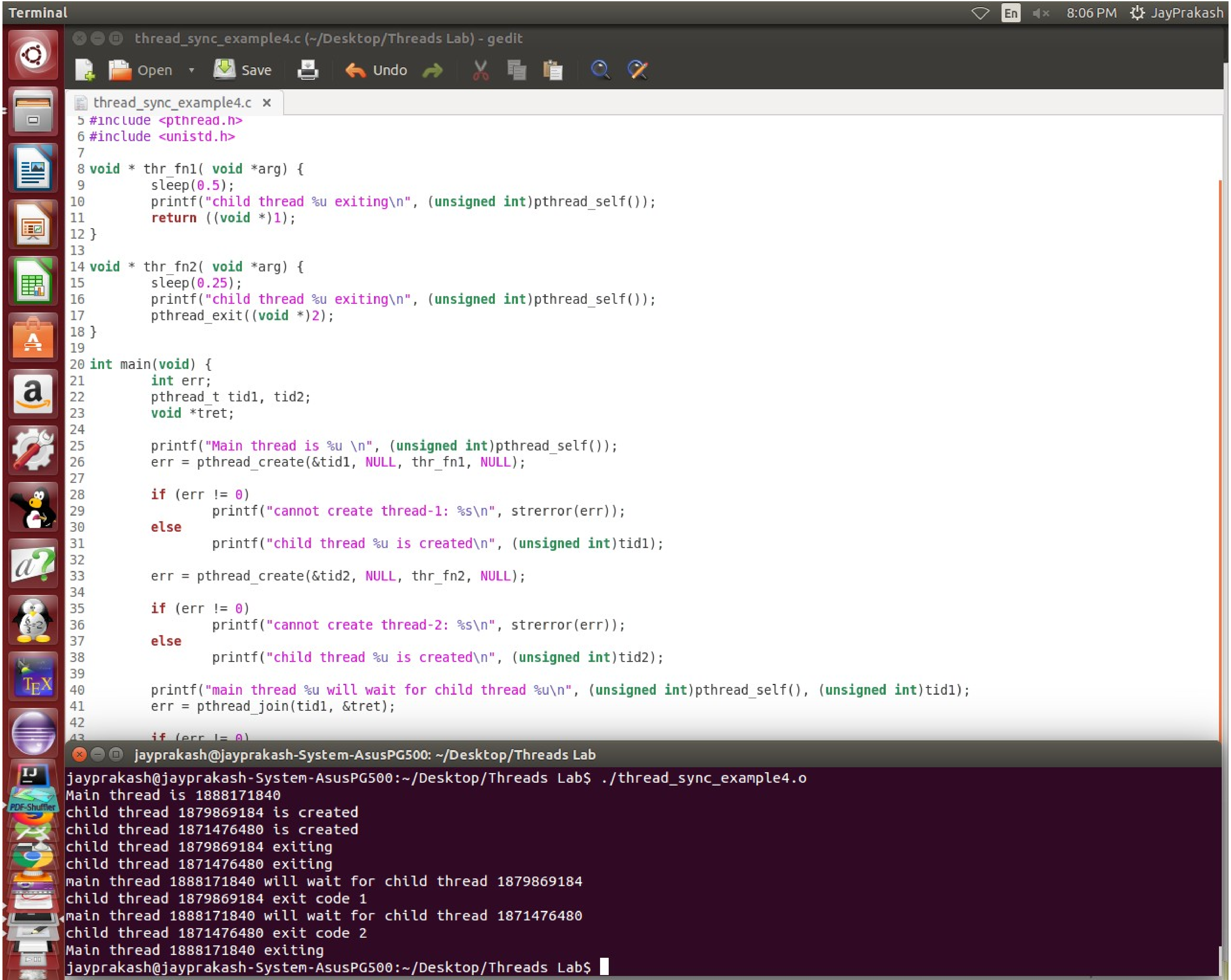
Threads

a new abstraction of program execution

Complete Thread Example with Synchronization

thread_sync_example.c

```
$ ./thread_sync_example.out
Main thread is 2739095296
child thread 2730940160 is created
child thread 2722547456 is created
main thread 2739095296 will wait for child thread 2730940160
child thread 2730940160 exiting
child thread 2722547456 exiting
child thread 2730940160 exit code 1
main thread 2739095296 will wait for child thread 2722547456
child thread 2722547456 exit code 2
Main thread 2739095296 exiting
$
```



Terminal
En
8:06 PM
JayPrakash
thread_sync_example4.c (~/Desktop/Threads Lab) - gedit
Open
Save
Undo
thread_sync_example4.c
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/Threads Lab

```c
#include <pthread.h>
#include <unistd.h>

void * thr_fn1( void *arg) {
        sleep(0.5);
        printf("child thread %u exiting\n", (unsigned int)pthread_self());
        return ((void *)1);
}

void * thr_fn2( void *arg) {
        sleep(0.25);
        printf("child thread %u exiting\n", (unsigned int)pthread_self());
        pthread_exit((void *)2);
}

int main(void) {
        int err;
        pthread_t tid1, tid2;
        void *tret;

        printf("Main thread is %u \n", (unsigned int)pthread_self());
        err = pthread_create(&tid1, NULL, thr_fn1, NULL);

        if (err != 0)
                printf("cannot create thread-1: %s\n", strerror(err));
        else
                printf("child thread %u is created\n", (unsigned int)tid1);

        err = pthread_create(&tid2, NULL, thr_fn2, NULL);

        if (err != 0)
                printf("cannot create thread-2: %s\n", strerror(err));
        else
                printf("child thread %u is created\n", (unsigned int)tid2);

        printf("main thread %u will wait for child thread %u\n", (unsigned int)pthread_self(), (unsigned int)tid1);
        err = pthread_join(tid1, &tret);

        if (err != 0)
```

```
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./thread_sync_example4.o
Main thread is 1888171840
child thread 1879869184 is created
child thread 1871476480 is created
child thread 1879869184 exiting
child thread 1871476480 exiting
main thread 1888171840 will wait for child thread 1879869184
child thread 1879869184 exit code 1
main thread 1888171840 will wait for child thread 1871476480
child thread 1871476480 exit code 2
Main thread 1888171840 exiting
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$
```

Example Program

```
#include "csapp.h"

/* thread routine */
void *mythread(void *vargp) {
  printf("T.a\n");
  printf("T.b\n");
  return NULL;
}

int main() {
  pthread_t tid;

  printf("M.a\n");

  Pthread_create(&tid, NULL,
                    mythread, NULL);
  printf("M.b\n");
  printf("M.c\n");
  Pthread_join(tid, NULL);

  printf("M.d\n");
  exit(0);
}
```

- Impossible statement orderings (printf output):

```
T.a
M.a
…
```

```
T.a
T.b
M.a
…
```

```
…
M.d
T.b
…
```

```
…
M.d
T.a
T.b
```

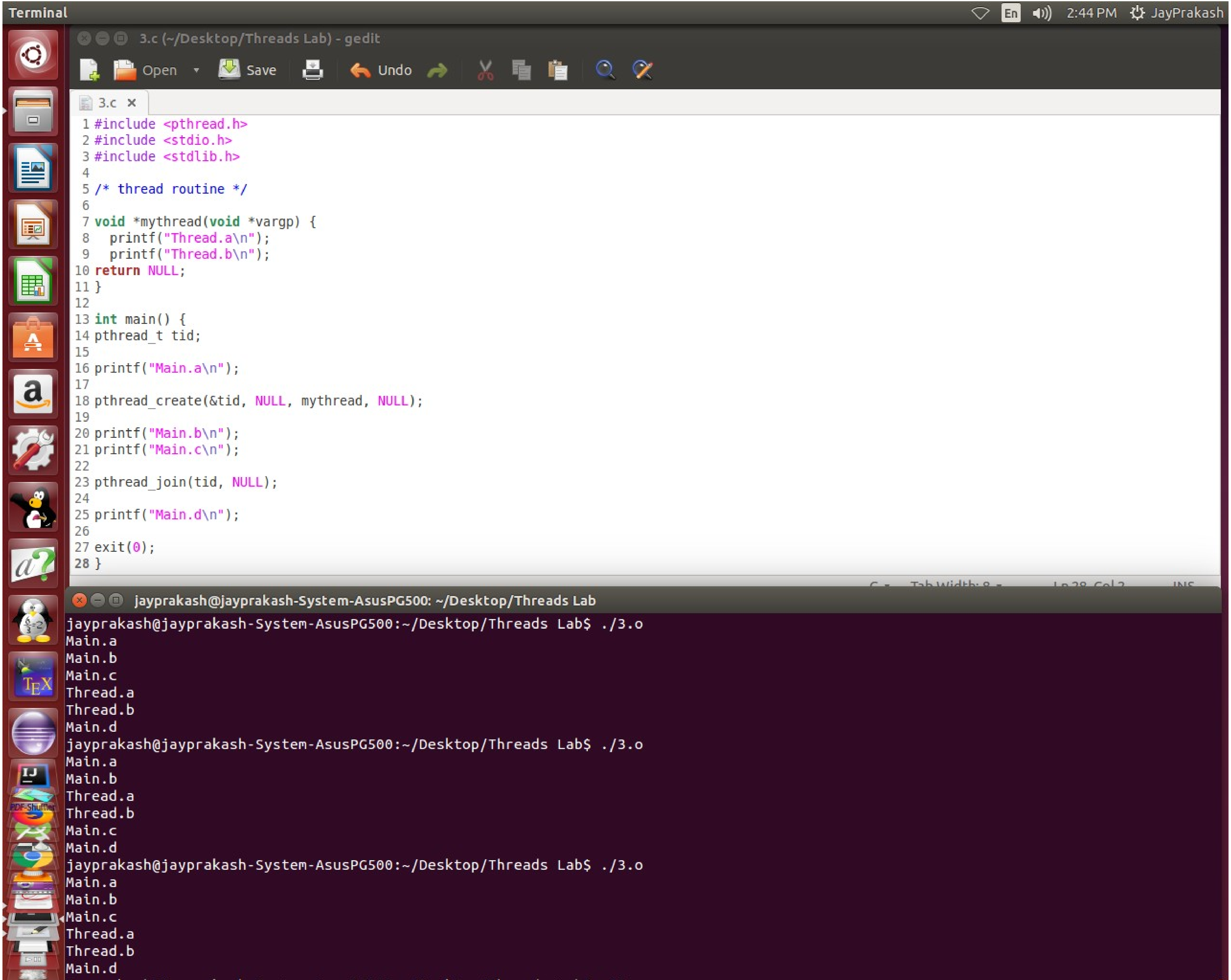
Example Program

```
#include "csapp.h"

/* thread routine */
void *mythread(void *vargp) {
  printf("T.a\n");
  printf("T.b\n");
  return NULL;
}

int main() {
  pthread_t tid;

  printf("M.a\n");

  Pthread_create(&tid, NULL,
                       mythread, NULL);
  printf("M.b\n");
  printf("M.c\n");
  Pthread_join(tid, NULL);

  printf("M.d\n");
  exit(0);
}
```

- Possible statement orderings (printf output):

M.a	M.a	M.a
M.b	T.a	T.a
M.c	M.b	M.b
T.a	M.c	T.b
T.b	T.b	M.c
M.d	M.d	M.d
…	…	…

several more?



Terminal
En
2:44 PM
JayPrakash
3.c (~/Desktop/Threads Lab) - gedit
Open
Save
Undo
3.c
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/Threads Lab

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

/* thread routine */

void *mythread(void *vargp) {
  printf("Thread.a\n");
  printf("Thread.b\n");
return NULL;
}

int main() {
pthread_t tid;

printf("Main.a\n");

pthread_create(&tid, NULL, mythread, NULL);

printf("Main.b\n");
printf("Main.c\n");

pthread_join(tid, NULL);

printf("Main.d\n");

exit(0);
}
```

```
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./3.o
Main.a
Main.b
Main.c
Thread.a
Thread.b
Main.d
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./3.o
Main.a
Main.b
Thread.a
Thread.b
Main.c
Main.d
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./3.o
Main.a
Main.b
Main.c
Thread.a
Thread.b
Main.d
```

Data race

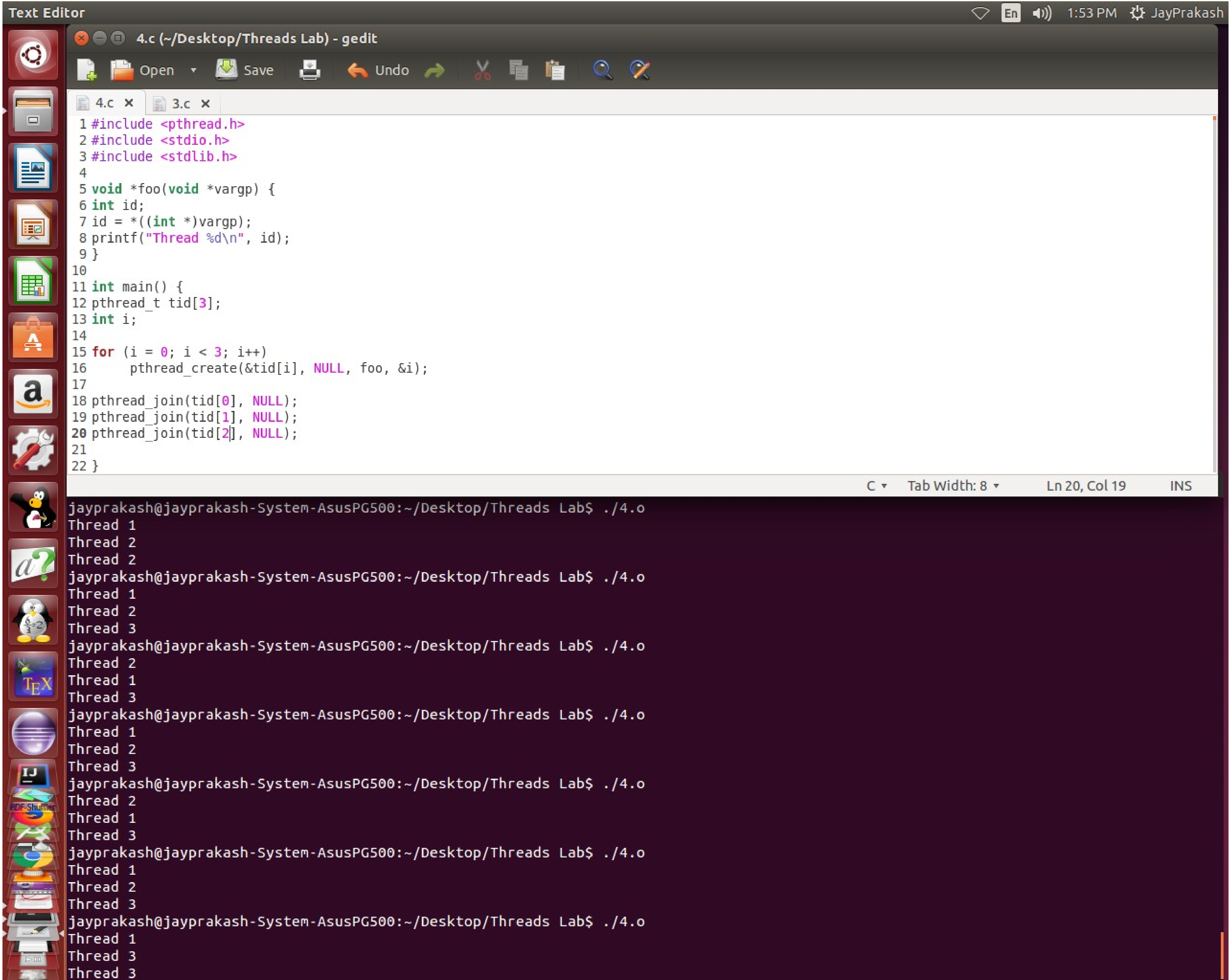
- A program has a data race if it is possible for a thread to modify an addressable location at the same time that another thread is accessing the same location
- The result/correctness depends on the sequence/timing of events; i.e., how things end up being scheduled.

Example Solution with race-condition

Q: Are there any race conditions in this code?

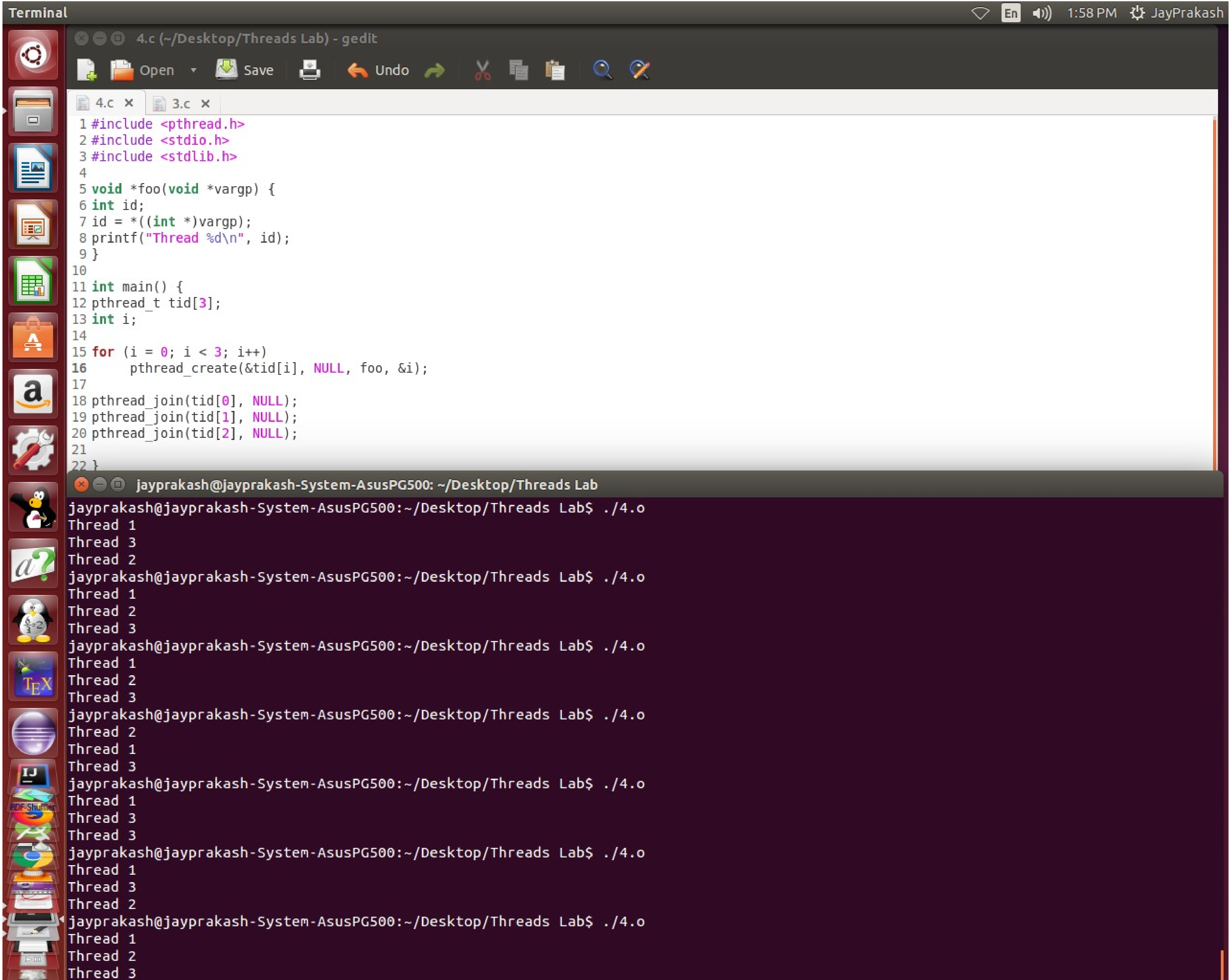
```
void *foo(void *vargp) {
    int id;
    id = *((int *)vargp);
    printf("Thread %d\n", id);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], NULL, foo, &i);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

Yes!



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

void *foo(void *vargp) {
int id;
id = *((int *)vargp);
printf("Thread %d\n", id);
}

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

for (i = 0; i < 3; i++)
	pthread_create(&tid[i], NULL, foo, &i);

pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_join(tid[2], NULL);

}
```

```
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 2
Thread 1
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 2
Thread 1
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 3
Thread 3
```


Terminal
En
1:58 PM
JayPrakash
4.c (~/Desktop/Threads Lab) - gedit
Open
Save
Undo
4.c
3.c

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

void *foo(void *vargp) {
int id;
id = *((int *)vargp);
printf("Thread %d\n", id);
}

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

for (i = 0; i < 3; i++)
     pthread_create(&tid[i], NULL, foo, &i);

pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_join(tid[2], NULL);

}
```

jayprakash@jayprakash-System-AsusPG500: ~/Desktop/Threads Lab

```
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 3
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 2
Thread 1
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 3
Thread 3
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 3
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./4.o
Thread 1
Thread 2
Thread 3
```

Flaw in code

- Inside the loop several threads are created
    - Variable i is visible in all threads
    - Variable i is modified in main after each thread is created
- The threads may not access the parameter fast enough to read the value “assigned” to them

Thread Synchronization

- Mechanism that allows programmer to control relative order of operation occurrence in different threads or processes
- How thread synchronization works:
  - Programmer identifies critical section in the code
  - Implements mutual exclusion to ensure that critical section is mutually exclusive i.e. atomic

Mutual Exclusion Implementation in POSIX Thread - MUTEX

MUTEX is like a key to access the critical section that has access to only one thread at a time

#include <pthread.h>

MUTEX variable (containing union of structures) is represented as pthread_mutex_t data type defined in /usr/include/bits/pthreadtypes.h

Before we can use mutex variable memory allocation has to be done for which use function:

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr) → return 0 on success

When mutex variable is no longer required memory should be freed for which we use function:

int pthread_mutex_destroy(pthread_mutex_t *mutex) → return 0 on success

Mutual Exclusion Implementation in POSIX Thread - MUTEX

#include <pthread.h>

To lock mutex: if mutex is already locked by another thread, thread trying to load mutex will be blocked

int pthread_mutex_lock(pthread_mutex_t *mutex) → return 0 on success

To lock mutex: if mutex is already locked by another thread, function will rerurn error code EBUSY

int pthread_mutex_trylock(pthread_mutex_t *mutex) → return 0 on success

To unlock mutex:

int pthread_mutex_unlock(pthread_mutex_t *mutex) → return 0 on success

Avoidance of Race Condition using MUTEX

thread_racecond_without_mutex.c

Race condition output without using MUTEX (remove all pthread_mutex_* function calls)

```
$ ./thread_racecond.out  → Race condition exists without using MUTEX
12345678901abcde23fgh456789012345ij6789012345678kl90mnopqrstuvwxyzabcdefghijkl
mnopqrstuvwxyz
```

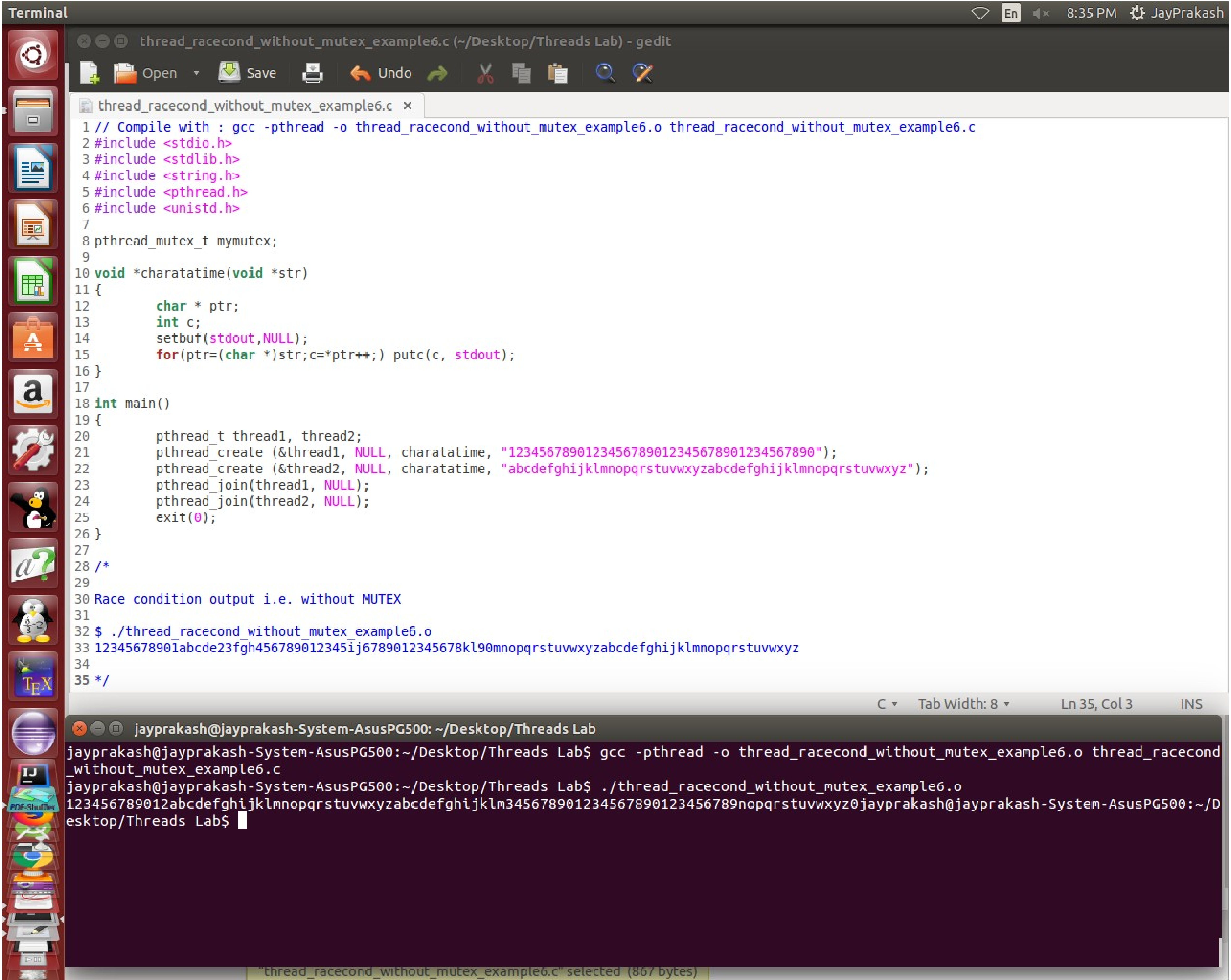
thread_racecond_with_mutex.c

With MUTEX implementation never get race condition

```
$ ./thread_racecond.out
123456789012345678901234567890abcdefghijklmnopqrstuvwxyzabcdefghijkl
mnopqrstuvwxyz
```

OR

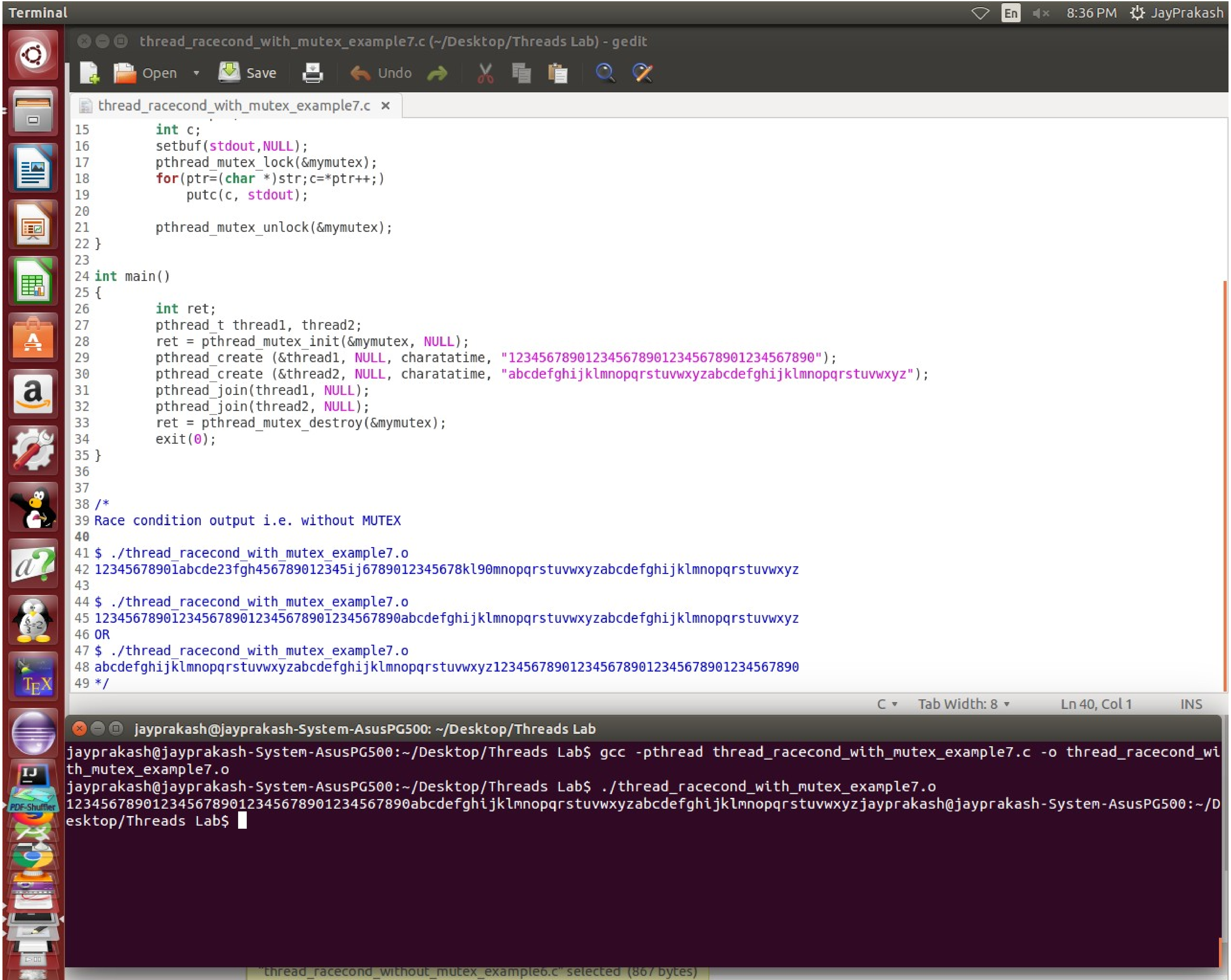
```
$ ./thread_racecond.out
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz12345678901234567890123456
78901234567890
```



Terminal
En
8:35 PM
JayPrakash
thread_racecond_without_mutex_example6.c (~/Desktop/Threads Lab) - gedit
Open
Save
Undo
thread_racecond_without_mutex_example6.c ×
C ▾
Tab Width: 8 ▾
Ln 35, Col 3
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/Threads Lab
"thread_racecond_without_mutex_example6.c" selected (867 bytes)

```c
// Compile with : gcc -pthread -o thread_racecond_without_mutex_example6.o thread_racecond_without_mutex_example6.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>

pthread_mutex_t mymutex;

void *charatatime(void *str)
{
        char * ptr;
        int c;
        setbuf(stdout,NULL);
        for(ptr=(char *)str;c=*ptr++;) putc(c, stdout);
}

int main()
{
        pthread_t thread1, thread2;
        pthread_create (&thread1, NULL, charatatime, "1234567890123456789012345678901234567890");
        pthread_create (&thread2, NULL, charatatime, "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz");
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        exit(0);
}

/*

Race condition output i.e. without MUTEX

$ ./thread_racecond_without_mutex_example6.o
12345678901abcde23fgh4567890123451j6789012345678kl90mnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz

*/
```

```
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ gcc -pthread -o thread_racecond_without_mutex_example6.o thread_racecond
_without_mutex_example6.c
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./thread_racecond_without_mutex_example6.o
123456789012abcdefghijklmnopqrstuvwxyzabcdefghijklm3456789012345678901234567890nopqrstuvwxyz0jayprakash@jayprakash-System-AsusPG500:~/D
esktop/Threads Lab$ 
```


Terminal
En
8:36 PM
JayPrakash
thread_racecond_with_mutex_example7.c (~/Desktop/Threads Lab) - gedit
Open
Save
Undo
thread_racecond_with_mutex_example7.c
C
Tab Width: 8
Ln 40, Col 1
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/Threads Lab
"thread_racecond_without_mutex_example6.c" selected (867 bytes)

```
        int c;
        setbuf(stdout,NULL);
        pthread_mutex_lock(&mymutex);
        for(ptr=(char *)str;c=*ptr++;)
            putc(c, stdout);

        pthread_mutex_unlock(&mymutex);
}

int main()
{
        int ret;
        pthread_t thread1, thread2;
        ret = pthread_mutex_init(&mymutex, NULL);
        pthread_create (&thread1, NULL, charatatime, "1234567890123456789012345678901234567890");
        pthread_create (&thread2, NULL, charatatime, "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz");
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        ret = pthread_mutex_destroy(&mymutex);
        exit(0);
}


/*
Race condition output i.e. without MUTEX

$ ./thread_racecond_with_mutex_example7.o
12345678901abcde23fgh456789012345ij6789012345678kl90mnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz

$ ./thread_racecond_with_mutex_example7.o
1234567890123456789012345678901234567890abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
OR
$ ./thread_racecond_with_mutex_example7.o
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz1234567890123456789012345678901234567890
*/
```

```
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ gcc -pthread thread_racecond_with_mutex_example7.c -o thread_racecond_wi
th_mutex_example7.o
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./thread_racecond_with_mutex_example7.o
1234567890123456789012345678901234567890abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzjayprakash@jayprakash-System-AsusPG500:~/D
esktop/Threads Lab$
```