

Threads

a new abstraction of program execution



Disadvantage of MUTEX

- Deadlock can occur if:
 - Thread locking the same MUTEX twice
 - Thread1 holding lock on mutex1 wants to get lock on mutex2 while thread2 holding lock on mutex2 wants to get lock on mutex1
 - Solution: As much as possible use `pthread_mutex_trylock` function
- Only one thread is allowed to lock mutex but in some situations such as Reader-Write problem or Dining-Philosopher problem multiple locks should be possible
 - Solution: Reader-Writer Locks and Semaphores

Example Solution without race-condition

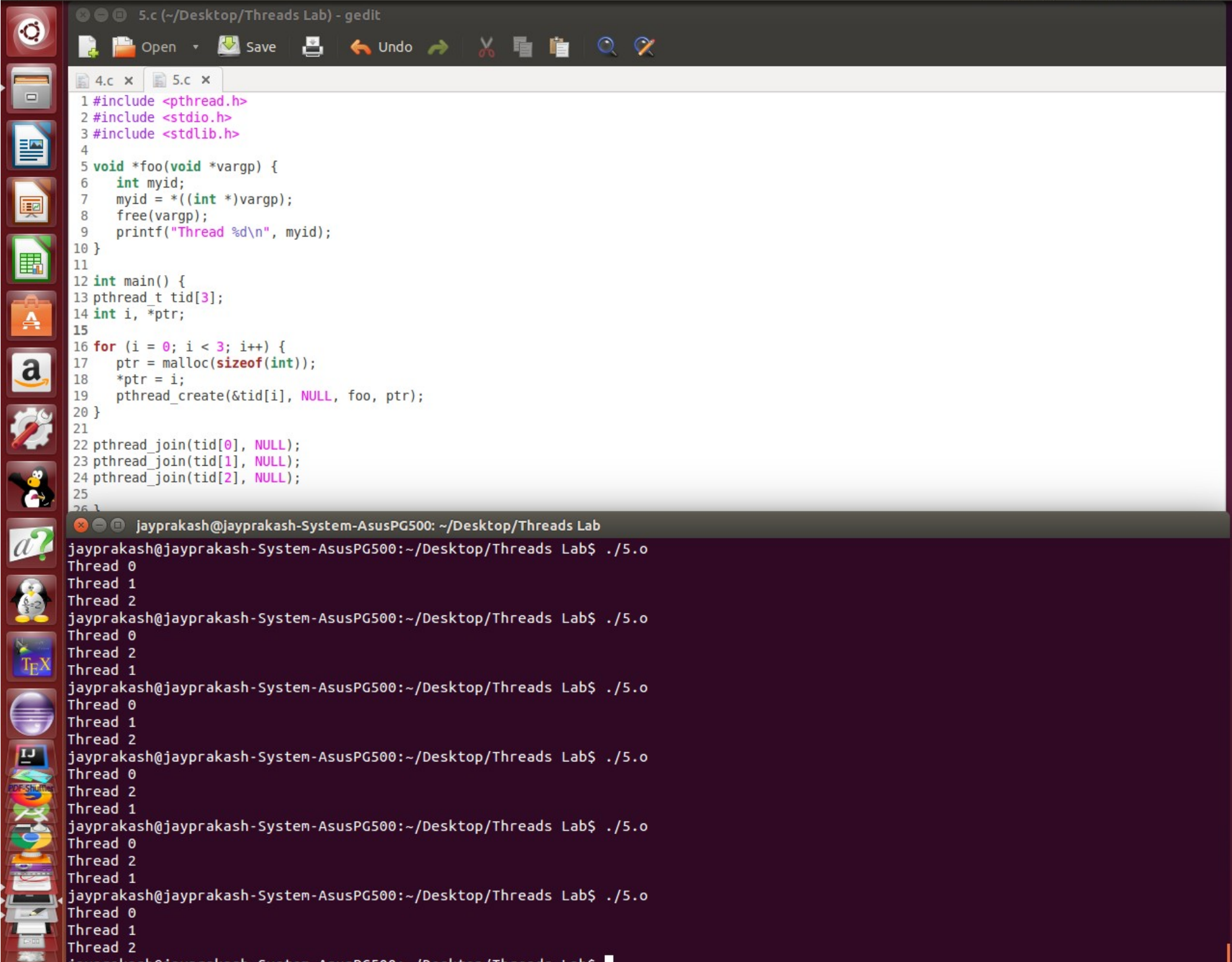
Q: Are there any race conditions in this code?

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

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

    for (i = 0; i < 2; i++) {
        ptr = Malloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], NULL, foo, ptr);
    }
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

No!



The screenshot displays a Linux desktop environment. On the left is a vertical dock containing various application icons, including a gear for settings, a folder, a document, a presentation, a spreadsheet, a shopping bag, an Amazon logo, a wrench and screwdriver, a penguin, a question mark, another penguin, a T_EX logo, a sphere, a book, a PDF viewer, a globe, and a printer. The main workspace is divided into two windows. The top window, titled '5.c (~/.Desktop/Threads Lab) - gedit', is a code editor showing a C program. The code defines a function 'foo' that takes a pointer to an integer, dereferences it to get a thread ID, prints it, and then frees the memory. The 'main' function creates an array of three pthread_t variables, allocates memory for each, creates three threads using 'pthread_create', and then joins them using 'pthread_join'. The bottom window is a terminal titled 'jayprakash@jayprakash-System-AsusPG500: ~/.Desktop/Threads Lab'. It shows the execution of the program './5.o' multiple times. Each execution results in three lines of output: 'Thread 0', 'Thread 1', and 'Thread 2', indicating that the threads are running in parallel and completing their execution.

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 void *foo(void *vargp) {
6     int myid;
7     myid = *((int *)vargp);
8     free(vargp);
9     printf("Thread %d\n", myid);
10 }
11
12 int main() {
13     pthread_t tid[3];
14     int i, *ptr;
15
16     for (i = 0; i < 3; i++) {
17         ptr = malloc(sizeof(int));
18         *ptr = i;
19         pthread_create(&tid[i], NULL, foo, ptr);
20     }
21
22     pthread_join(tid[0], NULL);
23     pthread_join(tid[1], NULL);
24     pthread_join(tid[2], NULL);
25 }
26
```

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

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 1
Thread 2

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 1
Thread 2

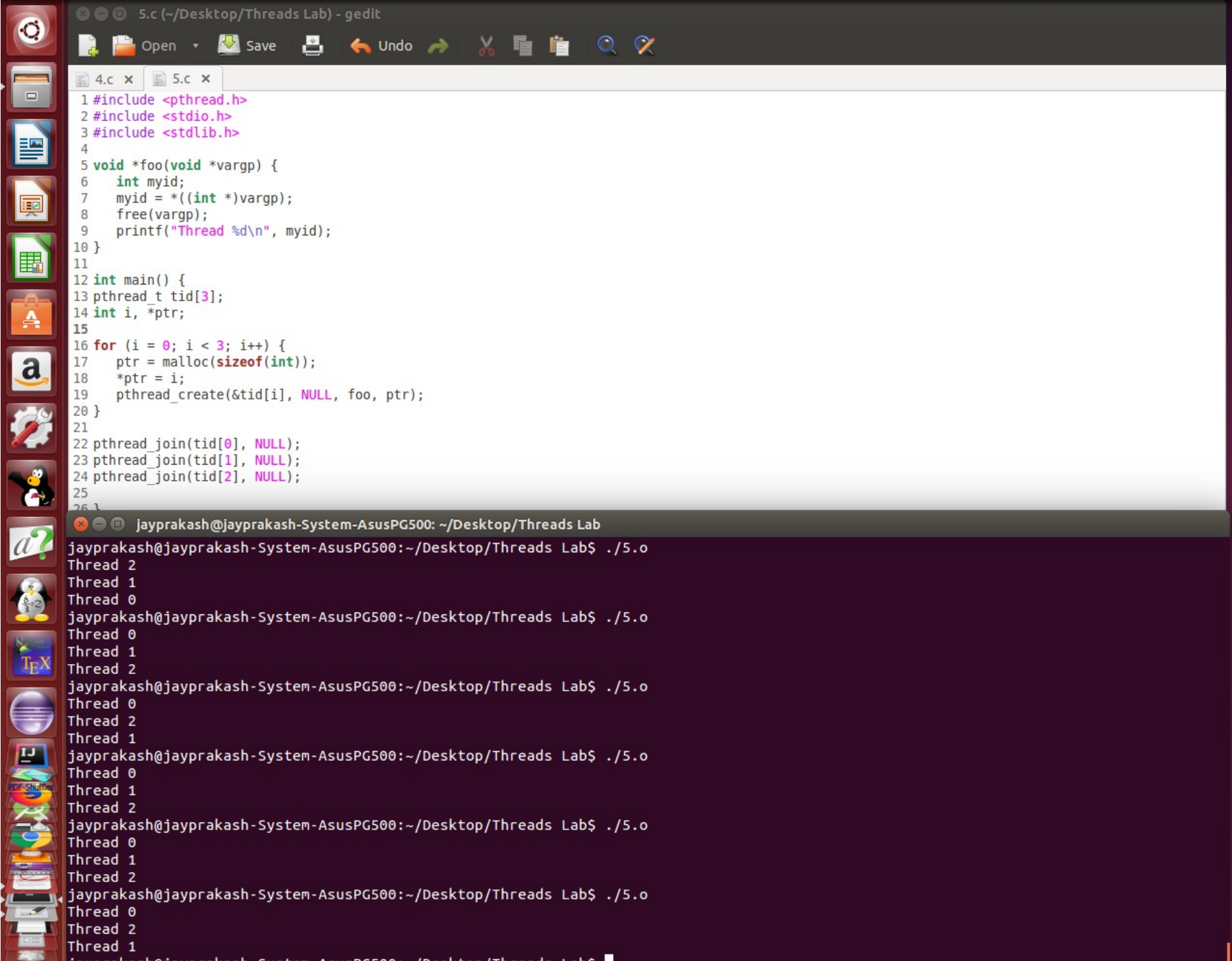
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1



The screenshot displays a Linux desktop with a dark theme. On the left is a vertical dock containing icons for various applications: a gear (system settings), a folder (file manager), a document with a magnifying glass (text editor), a presentation (slides), a spreadsheet (calc), a shopping bag (store), an Amazon logo, a wrench and screwdriver (tools), a penguin (KDE), a question mark (help), another penguin, a T_EX logo, a globe (web browser), a book (library), a PDF viewer, a file manager, a printer, and a terminal icon. The main window is a code editor titled "5.c (~/.Desktop/Threads Lab) - gedit". It contains C code for a multi-threaded program. Below the code editor is a terminal window titled "jayprakash@jayprakash-System-AsusPG500: ~/.Desktop/Threads Lab". The terminal shows the execution of the program multiple times, each time printing "Thread 2", "Thread 1", and "Thread 0" in that order.

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 void *foo(void *vargp) {
6     int myid;
7     myid = *((int *)vargp);
8     free(vargp);
9     printf("Thread %d\n", myid);
10 }
11
12 int main() {
13     pthread_t tid[3];
14     int i, *ptr;
15
16     for (i = 0; i < 3; i++) {
17         ptr = malloc(sizeof(int));
18         *ptr = i;
19         pthread_create(&tid[i], NULL, foo, ptr);
20     }
21
22     pthread_join(tid[0], NULL);
23     pthread_join(tid[1], NULL);
24     pthread_join(tid[2], NULL);
25
26 }
```

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

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 2
Thread 1
Thread 0

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 1
Thread 2

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 1
Thread 2

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab\$./5.o
Thread 0
Thread 2
Thread 1

Question

- Outline an approach to avoid race conditions that does not use malloc/free

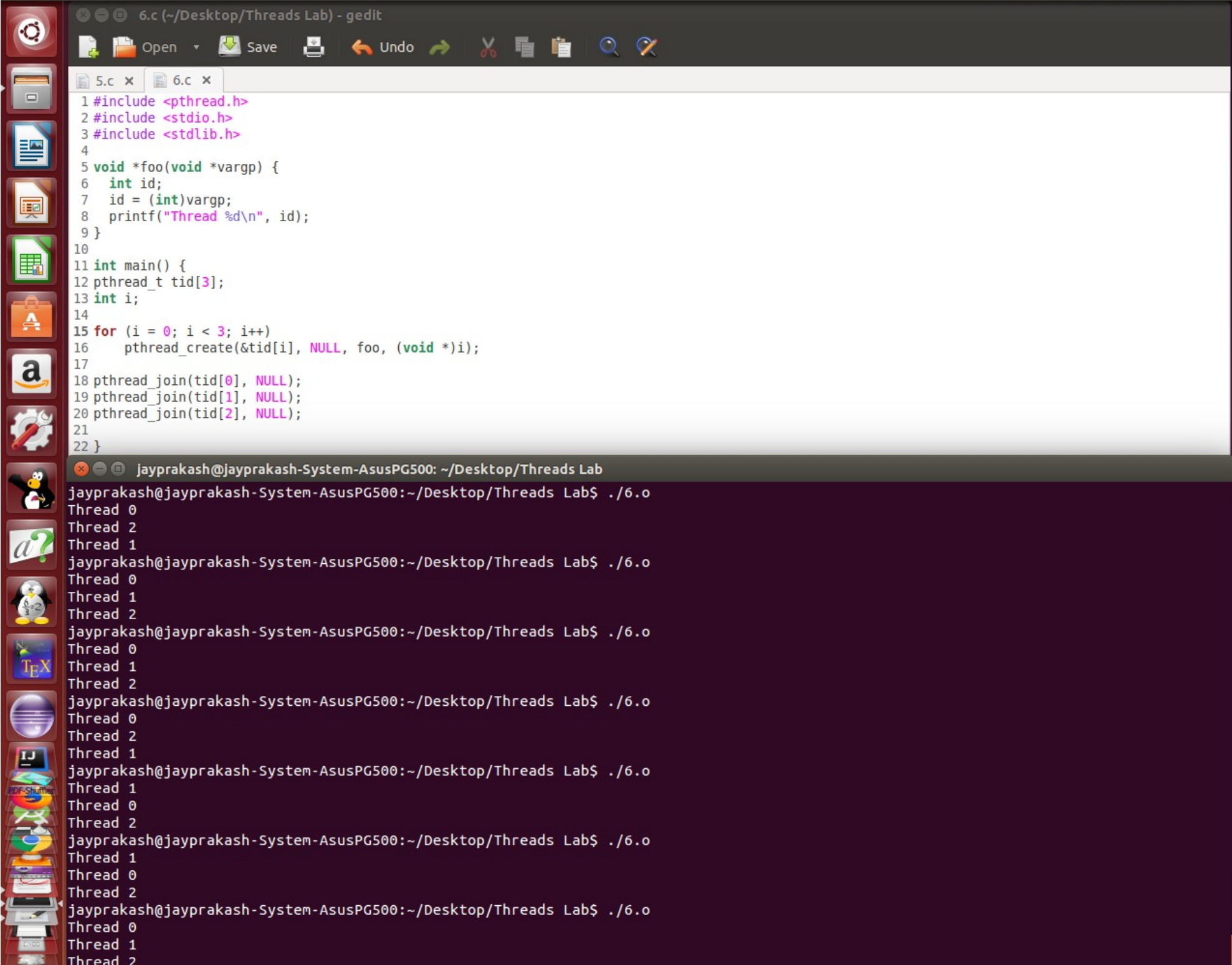
Example Solution without malloc()

Answer: Simply pass in the int directly!

```
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, (void *)i);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

```
6.c (~/.Desktop/Threads Lab) - gedit
Open Save Undo
5.c x 6.c x
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 void *foo(void *vargp) {
6     int id;
7     id = (int)vargp;
8     printf("Thread %d\n", id);
9 }
10
11 int main() {
12     pthread_t tid[3];
13     int i;
14
15     for (i = 0; i < 3; i++)
16         pthread_create(&tid[i], NULL, foo, (void *)i);
17
18     pthread_join(tid[0], NULL);
19     pthread_join(tid[1], NULL);
20     pthread_join(tid[2], NULL);
21
22 }
```

```
jayprakash@jayprakash-System-AsusPG500: ~/.Desktop/Threads Lab
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 0
Thread 2
Thread 1
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 0
Thread 1
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 0
Thread 1
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 0
Thread 2
Thread 1
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 1
Thread 0
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 1
Thread 0
Thread 2
jayprakash@jayprakash-System-AsusPG500:~/.Desktop/Threads Lab$ ./6.o
Thread 0
Thread 1
Thread 2
```


Previous example

- Pro: No added overhead due to malloc/free
- Con: Assumes that pointer datatype is at least bigger than size of int
 - May not be true on all systems

Threads are not hierarchical

- Unlike processes, a thread doesn't have a parent or children – threads are peers
- Any thread can `pthread_join` any other thread

Threads are not hierarchical

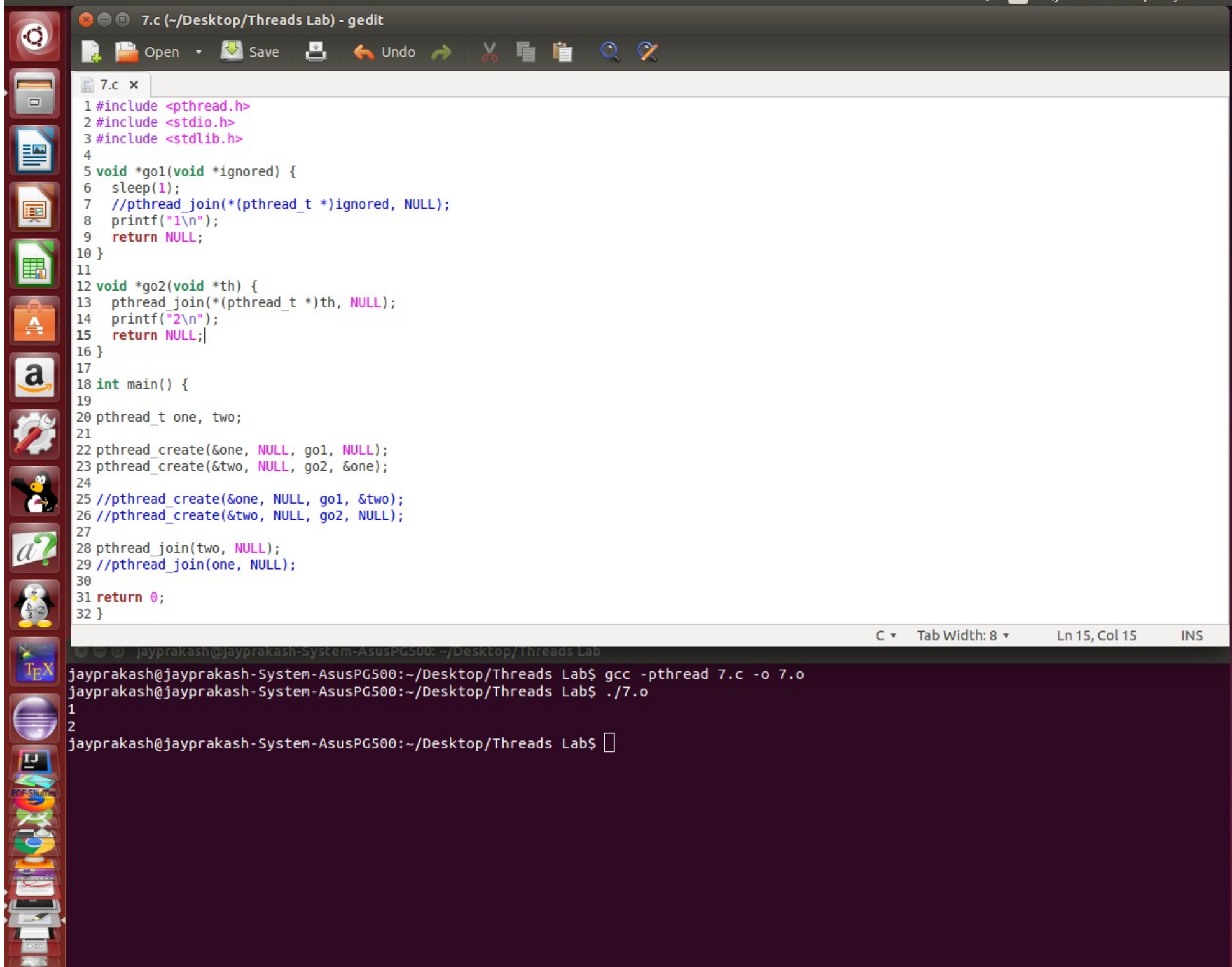
```
void *go1(void *ignored) {
    sleep(1);
    printf("1\n");
    return NULL;
}

void *go2(void *th) {
    pthread_join(*(pthread_t *)th, NULL);
    printf("2\n");
    return NULL;
}

int main() {
    pthread_t one, two;

    pthread_create(&one, NULL, go1, NULL);
    pthread_create(&two, NULL, go2, &one);
    pthread_join(two, NULL);
    return 0;
}
```

**Prints 1
then 2**



The screenshot shows a Linux desktop environment. On the left is a vertical dock with various application icons including a gear, a folder, a document, a presentation, a spreadsheet, a shopping bag, an Amazon logo, a wrench and screwdriver, a penguin, a question mark, another penguin, a terminal, a PDF viewer, and a printer. The main window is a text editor titled "7.c (~/.Desktop/Threads Lab) - gedit". It contains a C program that uses pthreads. Below the text editor is a terminal window with the same title bar. The terminal shows the user compiling the program with gcc and then running it.

```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 void *go1(void *ignored) {
6     sleep(1);
7     //pthread_join(*(pthread_t *)ignored, NULL);
8     printf("1\n");
9     return NULL;
10 }
11
12 void *go2(void *th) {
13     pthread_join(*(pthread_t *)th, NULL);
14     printf("2\n");
15     return NULL;
16 }
17
18 int main() {
19
20     pthread_t one, two;
21
22     pthread_create(&one, NULL, go1, NULL);
23     pthread_create(&two, NULL, go2, &one);
24
25     //pthread_create(&one, NULL, go1, &two);
26     //pthread_create(&two, NULL, go2, NULL);
27
28     pthread_join(two, NULL);
29     //pthread_join(one, NULL);
30
31     return 0;
32 }
```

```
jayprakash@jayprakash-System-AsusPG500: ~/.Desktop/Threads Lab$ gcc -pthread 7.c -o 7.o
jayprakash@jayprakash-System-AsusPG500: ~/.Desktop/Threads Lab$ ./7.o
1
2
jayprakash@jayprakash-System-AsusPG500: ~/.Desktop/Threads Lab$
```

Dining-Philosopher Problem

- Five philosophers spend their time eating and thinking.
- They are sitting in front of a round table with spaghetti served.
- There are five plates at the table and five chopsticks set between the plates.
- Eating the spaghetti **requires** the **use of two chopsticks** which the philosophers pick up one at a time.
- Philosophers do not talk to each other.
- Semaphore **chopstick [5]** initialized to 1



Semaphores

- Allow multiple locks
- Semaphore $S \rightarrow$ integer variable
- Modified by two operations \rightarrow wait() and signal()
 - wait() – originally called P() for Dutch word “proberen” which means try
 - signal() – originally called V() for Dutch word “verhogen” which means increase

```
wait(S) {  
    while S <= 0; // no-op  
    S--;  
}  
signal(S) {  
    S++;  
}
```

Note wait() and signal() are atomic operation

Thread Synchronization using Semaphores

```
//Thread1:
```

```
int t;
```

```
wait(sem)
```

```
sum = sum + x;
```

```
t = sum;
```

```
...
```

```
signal(sem);
```

```
//Thread2:
```

```
int t;
```

```
wait(sem)
```

```
sum = sum + y;
```

```
t = sum;
```

```
...
```

```
signal(sem);
```


Dining-Philosopher Problem: Solution

The structure of Philosopher i:

```
do {  
    wait ( chopstick[i] ); //lock  
    wait ( chopstick[ (i + 1) % 5] ); //lock  
    // eat  
    signal ( chopstick[i] ); //unlock  
    signal ( chopstick[ (i + 1) % 5] );  
    //unlock  
    // think  
} while (true) ;
```

- Problem is deadlock : when all philosopher decides to eat at the same time each will pick up one of the 2 chopsticks they need and wait for the other
- Solution: Allow to lock if both chopsticks are available at the same time
 - Lock the 1st chopstick
 - For 2nd chopstick, check if it can be locked otherwise release the 1st one

POSIX: Semaphores

- creating a semaphore:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- initializes a semaphore object pointed to by sem to integer value of “value”
- pshared is a sharing option; a value of 0 means the semaphore is local to the calling process i.e. shared by all threads of the same process; a +ve number indicates it can be shared across multiple processes using shared memory instructions

- terminating a semaphore:

```
int sem_destroy(sem_t *sem);
```

- frees the resources allocated to the semaphore sem
- usually called after pthread_join()
- an error will occur if a semaphore is destroyed for which a thread is waiting

POSIX: Semaphores

- semaphore control:

`int sem_post(sem_t *sem);` → same as `signal()`

`sem_post` atomically increases the value of a semaphore by 1, i.e., when 2 threads call `sem_post` simultaneously, the semaphore's value will also be increased by 2 (there are 2 atoms calling)

`int sem_wait(sem_t *sem);`

`sem_wait` atomically decreases the value of a semaphore by 1; but always waits until the semaphore has a non-zero value first

Semaphore Example

[semaphore1.c](#)

```
$ ./semaphore1.out
```

```
Starting thread, semaphore is unlocked.
```

```
Hello from thread!
```

```
Hello from thread!
```

```
Hello from thread!
```

```
Hello from thread!
```

```
Semaphore locked.
```

```
Semaphore unlocked.
```

```
Hello from thread!
```

```
Hello from thread!
```

```
Hello from thread!
```

```
$
```

semaphore1_example9.c (~/Desktop/Threads Lab) - gedit

Open Save Undo

semaphore1_example9.c x

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <pthread.h>
5 #include <unistd.h>
6 #include <semaphore.h>
7
8 sem_t semaphore;
9
10 void threadfunc() {
11     while (1) {
12         sem_wait(&semaphore);
13         printf("Hello from thread!\n");
14         sem_post(&semaphore);
15         sleep(1);
16     }
17 }
18
19
20 int main(void) {
21     // initialize semaphore, only to be used with threads in this process, set value to 1
22     sem_init(&semaphore, 0, 1);
23     pthread_t *mythread;
24     mythread = (pthread_t *)malloc(sizeof(*mythread));
25     // start the thread
26     printf("Starting thread, semaphore is unlocked.\n");
27     pthread_create(mythread, NULL, (void*)threadfunc, NULL);
28
29     getchar();
30
31     sem_wait(&semaphore);
32     printf("Semaphore locked.\n");
33     // do stuff with whatever is shared between threads
34     getchar();
35     printf("Semaphore unlocked.\n");
36     sem_post(&semaphore);
37
38     getchar();
39
40     return 0;
41 }
42 }
```

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

jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab\$./semaphore1_example9.o

Starting thread, semaphore is unlocked.

Hello from thread!

Semaphore locked.

abc

Semaphore unlocked.

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

Dining-Philosopher Problem: Solution

[dining-philosopher.c](#)

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 3 is thinking

Philosopher 4 is thinking

Philosopher 5 is thinking

Philosopher 1 is Hungry

Philosopher 2 is Hungry

Philosopher 3 is Hungry

Philosopher 4 is Hungry

Philosopher 5 is Hungry

Philosopher 5 takes fork 4 and 5

Philosopher 5 is Eating

Philosopher 5 putting fork 4 and 5 down

Philosopher 5 is thinking

Philosopher 4 takes fork 3 and 4

Philosopher 4 is Eating

Philosopher 1 takes fork 5 and 1

Philosopher 1 is Eating

Philosopher 4 putting fork 3 and 4 down

Philosopher 4 is thinking

Philosopher 3 takes fork 2 and 3

Philosopher 3 is Eating

Philosopher 5 is Hungry

Philosopher 1 putting fork 5 and 1 down

Philosopher 1 is thinking

Philosopher 5 takes fork 4 and 5

Philosopher 5 is Eating

.....

dining_philosopher_example10.c (~/Desktop/Threads Lab) - gedit

Open Save Undo

dining_philosopher_example10.c x

```
1 #include <pthread.h>
2 #include <semaphore.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6
7 #define N 5
8 #define THINKING 2
9 #define HUNGRY 1
10 #define EATING 0
11 #define LEFT (phnum + 4) % N
12 #define RIGHT (phnum + 1) % N
13
14 int state[N];
15 int phil[N] = { 0, 1, 2, 3, 4 };
16
17 sem_t mutex;
18 sem_t S[N];
19
20 void test(int phnum)
21 {
22     if (state[phnum] == HUNGRY
23         && state[LEFT] != EATING
24         && state[RIGHT] != EATING) {
25         // state that eating
26         state[phnum] = EATING;
```

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

jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab\$./dining_philosopher_example10.o

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
```

^C

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