Threads

a new abstraction of program execution

Example

```
while (1) {
  int sock = accept();
  if (0 == fork()) {
     handle_request();
     close(sock);
     exit(0);
  }
}
```

- A web server
  - A process listens for requests
  - When a new request comes in, create a new child process to handle this request
  - Multiple requests can be handled at the same time by different child processes

Example Web Server

```
while (1) {
  int sock = accept();
  if (0 == fork()) {
     handle_request();
     close(sock);
     exit(0);
  }
}
```

This implementation is very **inefficient**.

Why?

Web Server using fork() is inefficient

- **Time**
  - creating a **new process** for each incoming request
  - **context switching** between processes
- **Space**
  - each child process has its **own copy** of the address space, but they are almost the **same**
- **Inter-process communication (IPC)**
  - the child process may generate a response which will be used by the parent process
  - **Extra overhead** to make this happen (use a file/pipe/socket/shared memory)

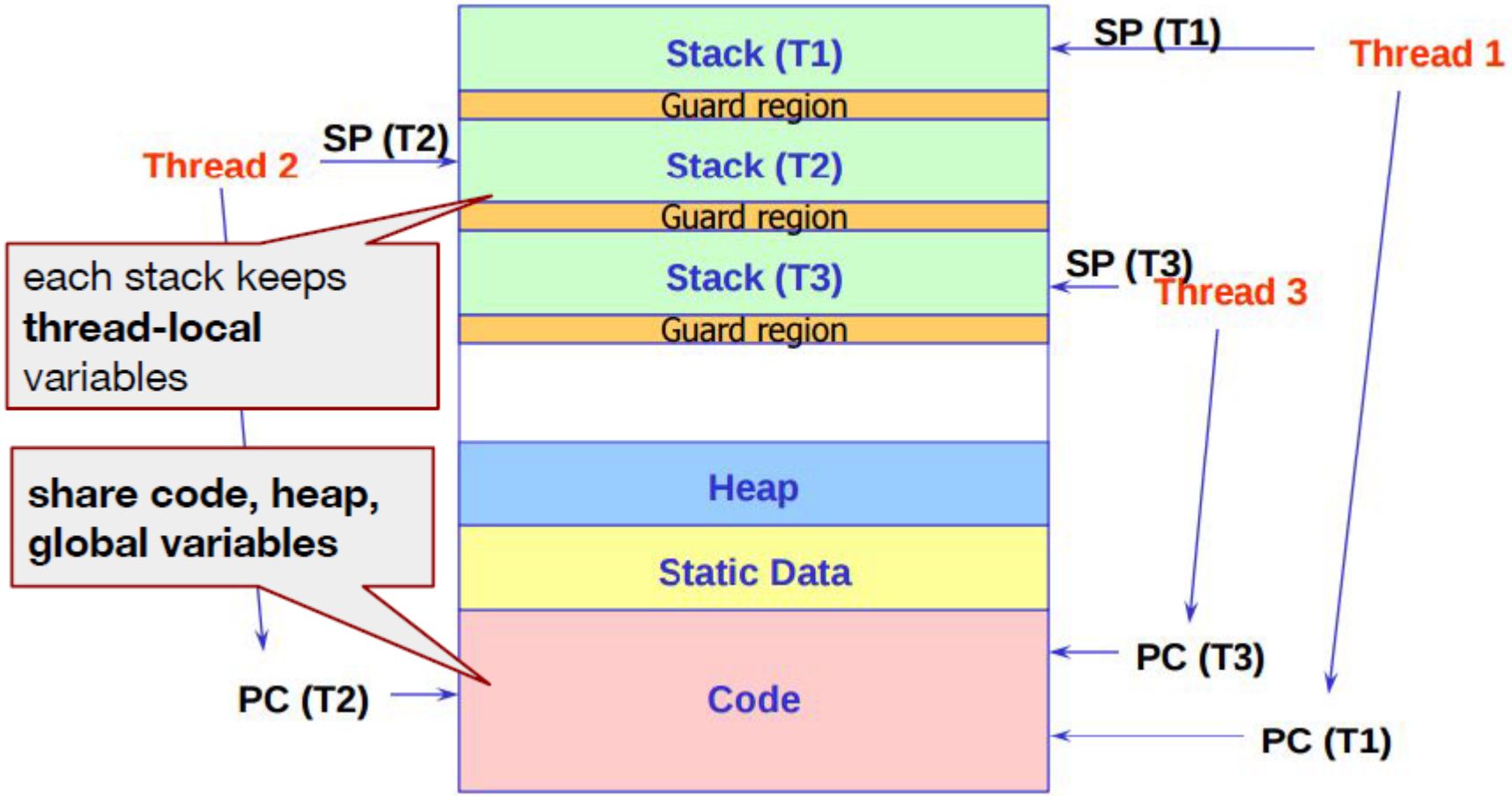
What is a Thread?

- A normal process is a running program with a **single control flow**, i.e., a single PC.
- A **multi-threaded program** has **multiple control flows**, i.e., multiple PCs.
  - One thread is a single **control flow** through a program

Multi-threaded Process Address Space

- Multiple threads share the **same address space**
  - no need to copy address space when creating thread
  - no need to switch address space on context switch
- There are **multiple stacks** in the address space, one for each thread.

Multi-threaded Process Address Space



Web Server Using Threads

- Different threads share and modify the global variable

```
int global_counter = 0;
web_server() {
  while (1) {
     int sock = accept();
     thread_create(handle_request, sock);
  }
}
```

The counter thing doesn't work using fork(). Why?

```
int global_counter = 0;
while (1) {
  int sock = accept();
  if (0 == fork()) {
     handle_request();
     ++global_counter;
     close(sock);
     exit(0);
  }
}
```

- counter is always 0 in the parent process, since changes only happen in children
- If you really want to make this work using fork(), need inter-process communication

Summary of threads

- Lighter weight
  - faster to create and destroy
  - faster context switch
- Sharing
  - threads can solve a single problem concurrently and can easily share code, heap and global variables

Thread programming

- Linux pthreads using kernel-level threads
- Thread prototypes: #include <pthread.h>
  > 60 functions
- To compile:

```
gcc -lpthread prog.c -o prog
```

OR

gcc -pthread prog.c -o prog

Basic operations on threads

- **Creating and starting a thread**
  - like an asynchronous procedure call
- **Joining with a thread**
  - blocks the calling thread until a target thread completes
  - a typical operation when a thread needs to use the return value of the target-thread's starting procedure

Thread Creation

```
int pthread_create (pthread_t *tid,
                    pthread_attr_t *attr,
                    void *(*func)(void *),
                    void *arg);
```

- **func :** input parameter, the name of the user-defined function to be executed by the newly created thread.
  - returns a void pointer, single void pointer argument
- **arg :** input parameter, the argument to be passed to the function **func**

Thread Creation

```
int pthread_create (pthread_t *tid,
                    pthread_attr_t *attr,
                    void *(*func)(void *),
                    void *arg);
```

- Why void * as argument type?
- The argument type should:
  - Work for all types and sizes of arguments
  - Even work for different numbers of arguments

Thread Creation

```
int pthread_create (pthread_t  *tid,
                    pthread_attr_t  *attr,
                    void  *(*func)(void  *),
                    void  *arg);
```

- **tid :** output parameter, pointer to location where the id of the newly created thread will be stored.
- **attr :** input parameter, pointer to structure specifying the thread's attributes (e.g. stack size, etc.)
  - If set to NULL, the system defaults are used

Threaded “hello, world”

```
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "csapp.h"

/* thread routine */
void *thread(void *vargp) {
  printf("Hello, world!\n");
  return NULL;
}

int main() {
  pthread_t tid;

  Pthread_create(&tid, NULL, thread, NULL);
  exit(0);
}
```

Wrong!

Thread coordination

- Threads can only execute as long as their containing process exists
- In the example, what could happen if the thread routine has not reached **printf** with the main thread reaching the end of main()?
- We need to use **join()** to make the main thread wait for the other thread to finish

pthread_join()

```
int pthread_join(pthread_t thread,
                 void **value_ptr );
```

- **`thread`**: input parameter, id of the thread to wait on
- **`value_ptr`**: output parameter, value given to **`pthread_exit()`** by the terminating thread (which happens to always be a **`void *`**)
  - Type is void ** because type of location to be updated is void *

Threaded “hello, world”

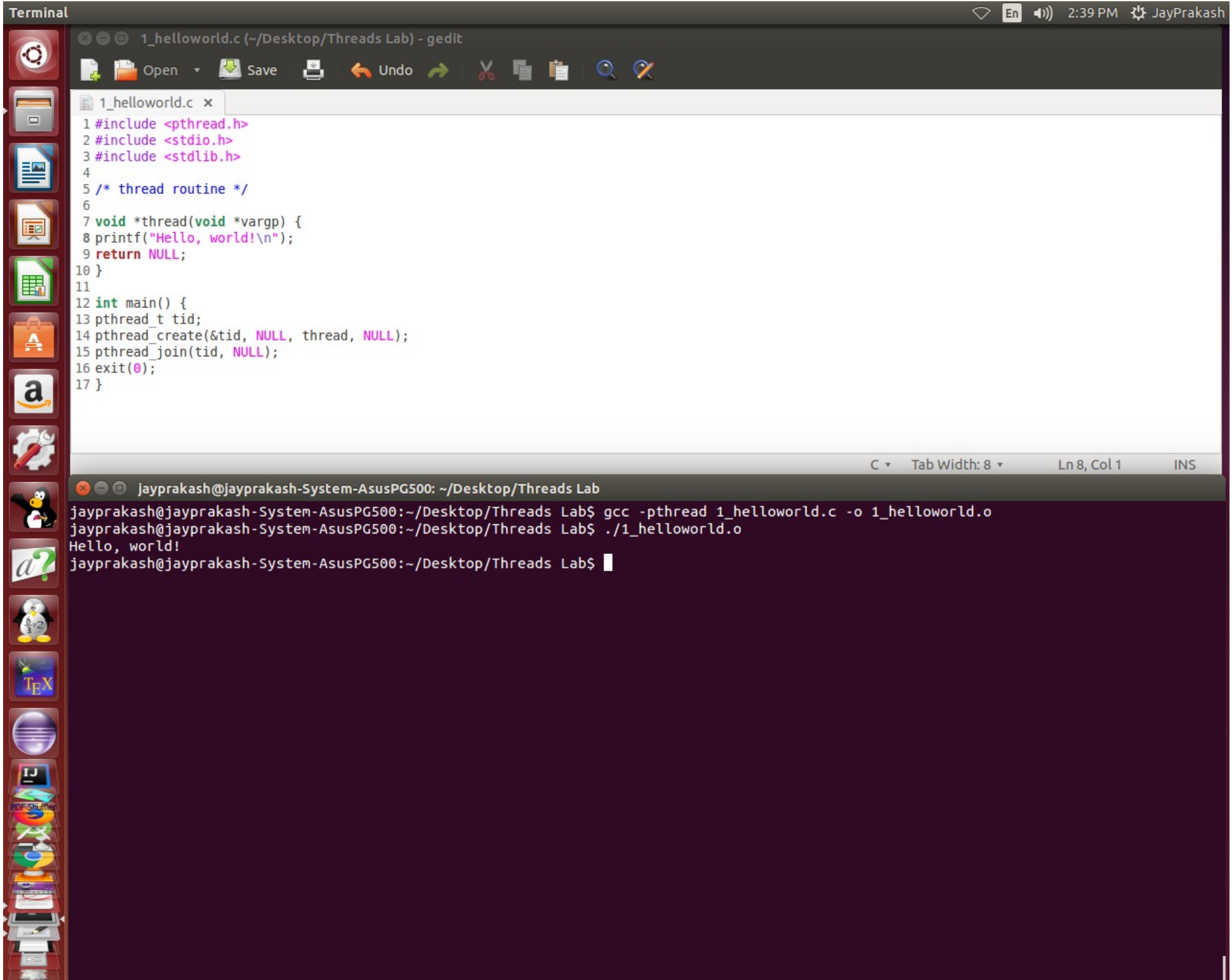
```
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "csapp.h"

/* thread routine */
void *thread(void *vargp) {
  printf("Hello, world!\n");
  return NULL;
}

int main() {
  pthread_t tid;

  Pthread_create(&tid, NULL, thread, NULL);
  Pthread_join(tid, NULL);
  exit(0);
}
```

Good!


Terminal
En
2:39 PM
JayPrakash
1_helloworld.c (~/Desktop/Threads Lab) - gedit
Open
Save
Undo
1_helloworld.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
/* thread routine */
void *thread(void *vargp) {
printf("Hello, world!\n");
return NULL;
}
int main() {
pthread_t tid;
pthread_create(&tid, NULL, thread, NULL);
pthread_join(tid, NULL);
exit(0);
}
C
Tab Width: 8
Ln 8, Col 1
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/Threads Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ gcc -pthread 1_helloworld.c -o 1_helloworld.o
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$ ./1_helloworld.o
Hello, world!
jayprakash@jayprakash-System-AsusPG500:~/Desktop/Threads Lab$

Execution of threaded “hello, world”