STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 492: Operating Systems

Threads

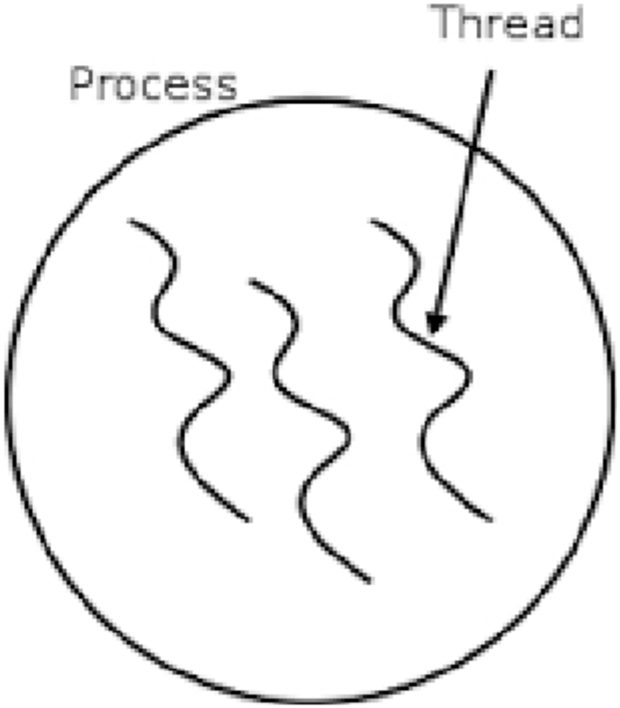
Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Exercise

- A computer has 4 GB of RAM of which the operating system occupies 512 MB. The processes are all 256 MB (for simplicity) and have the same characteristics. If the goal is 99% CPU utilization, what is the maximum I/O wait that can be tolerated?

Threads



Goals for Today

- Threads
  - Concept
  - Multiple-threading
  - Implementation of threads in kernel

Processes Continues…

- A process is created to run a program to perform a duty.

- What if we need to perform two or more similar duties?

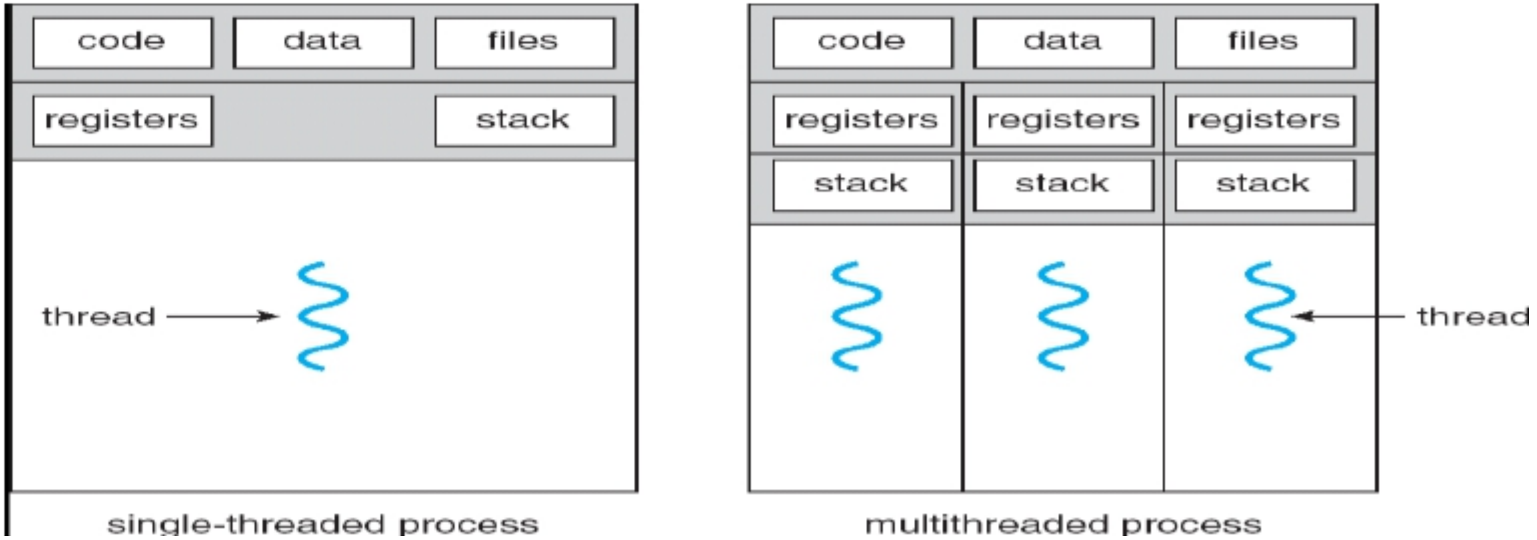
- One approach: create multiple processes, each handling one of the duties.

Why processes are not always ideal...

- Processes are not very efficient
  - Each process has its own PCB and OS resources
  - Typically high overhead for each process: e.g., 1.7 KB per task_struct on Linux!
- Processes don't (directly) share memory
  - Each process has its own address space
  - Parallel and concurrent programs often want to directly manipulate the same memory

- Can we do better?

Threads

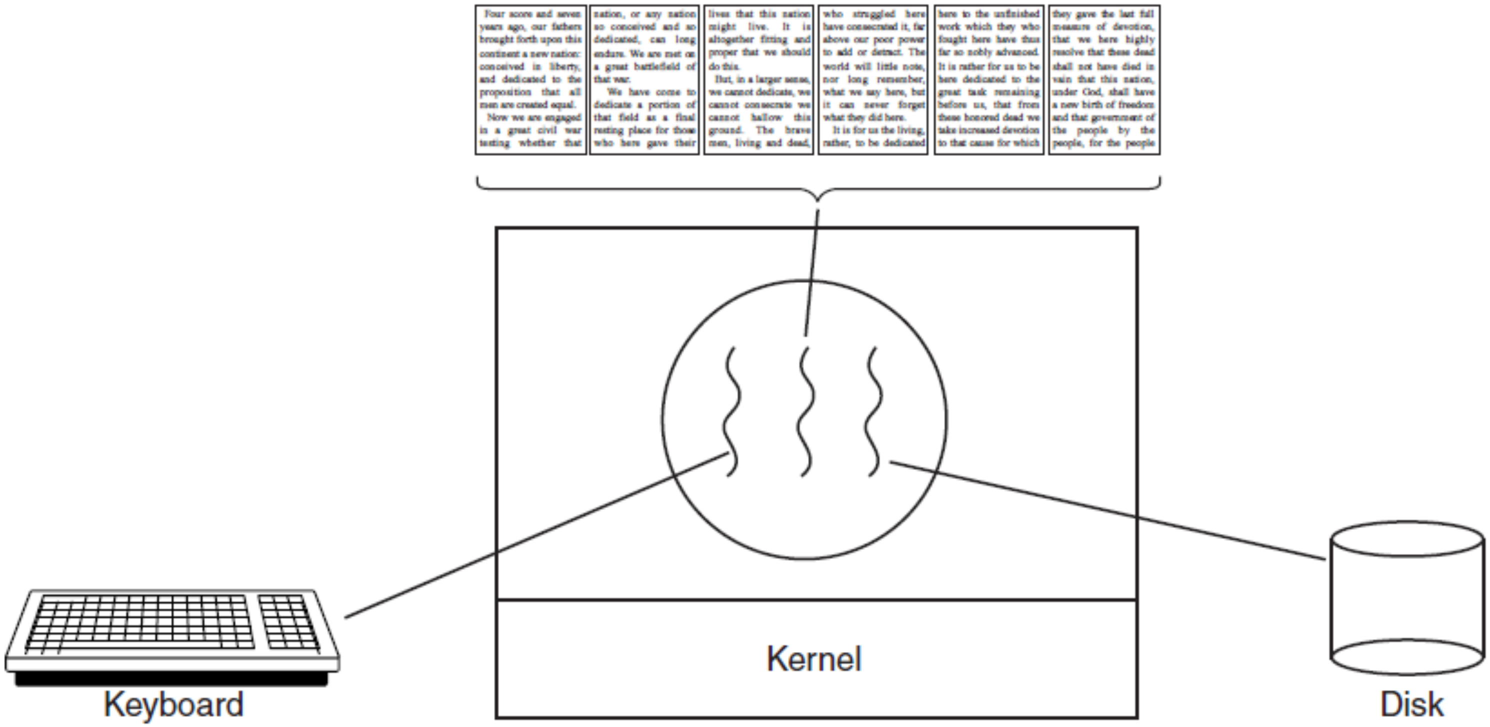


- Thread = an independent sequential execution stream within process
- Simple programs use one or multiple threads per process

Advantages of Threads

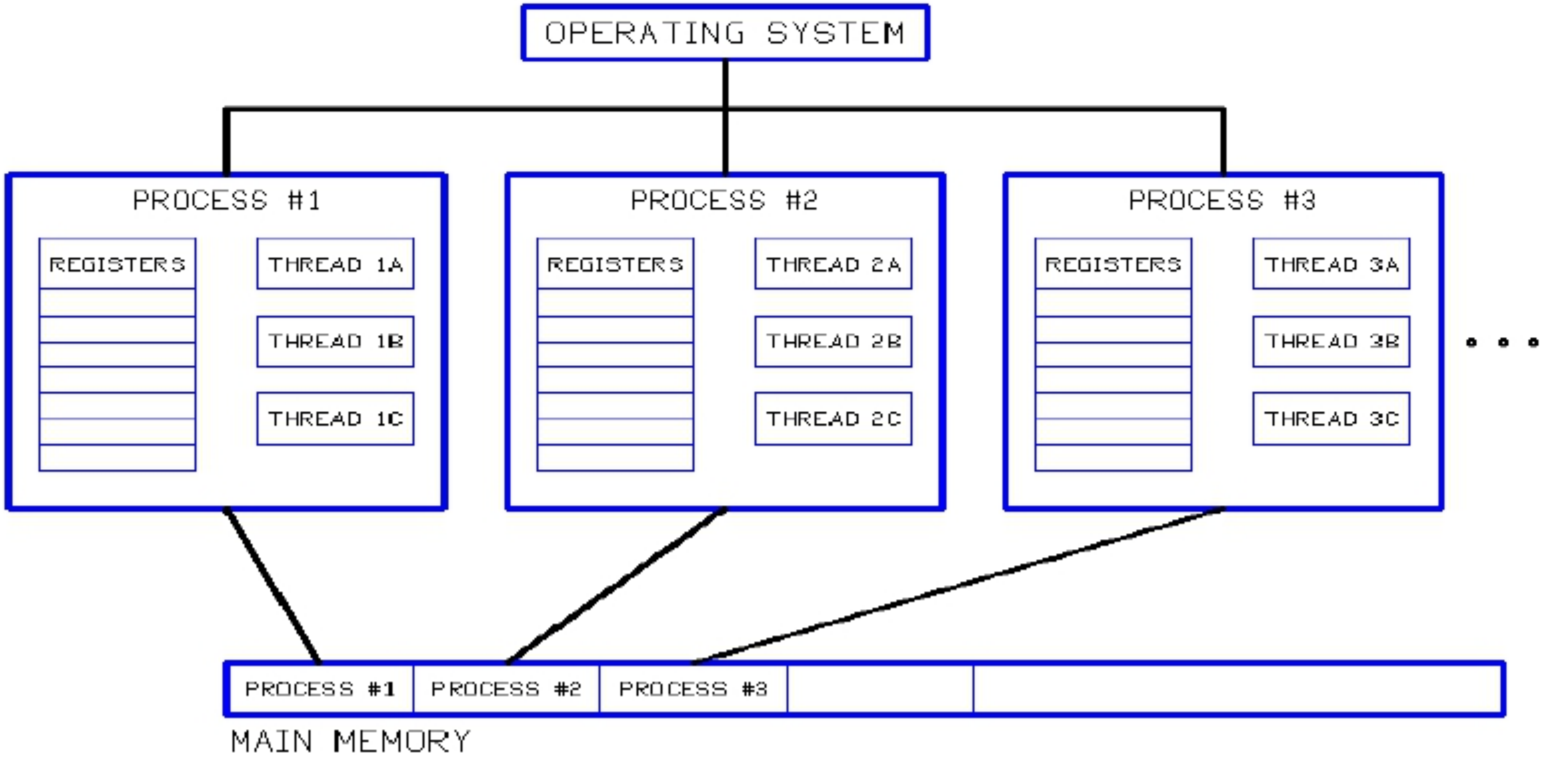
- Performance
  - Thread creation is 10-100 times faster than processes.
- Efficiency
  - Allows one process to use multiple CPUs or cores
  - Allows program to overlap I/O and computation

Thread Usage


Four score and seven years ago, our fathers brought forth upon this continent a new nation: conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war testing whether that
nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field as a final resting place for those who here gave their
lives that this nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate, we cannot consecrate we cannot hallow this ground. The brave men, living and dead,
who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated
here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which
they gave the last full measure of devotion, that we here highly resolve that these dead shall not have died in vain that this nation, under God, shall have a new birth of freedom and that government of the people by the people, for the people
Kernel
Keyboard
Disk

- Example: word processor process
  - One thread to read from keyboard
  - One thread to format document
  - One thread to write document to disk
- Would multiple processes work here?

Process vs Thread

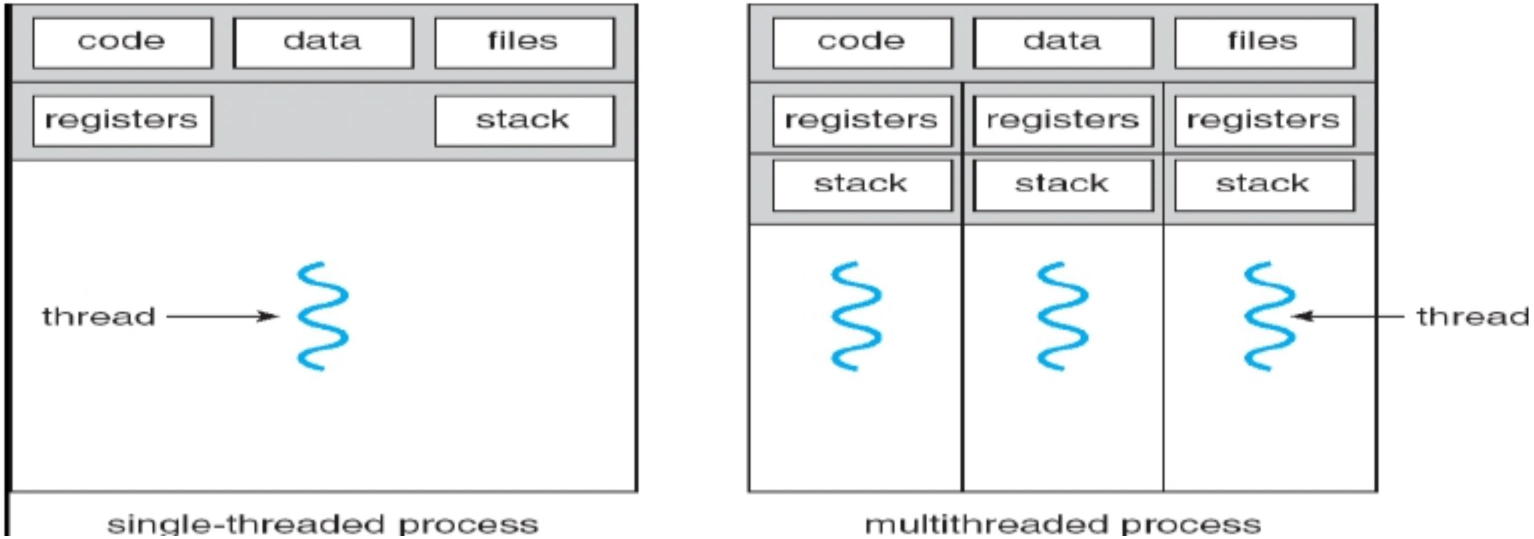


- A *thread cannot exist without a process, thus a* process is a "container" for threads
- A process may contain *multiple threads*

Goals for Today

- Threads
  - Concept
  - Multiple-threading
  - Implementation of threads in kernel

Multiple Threads in the Same Process

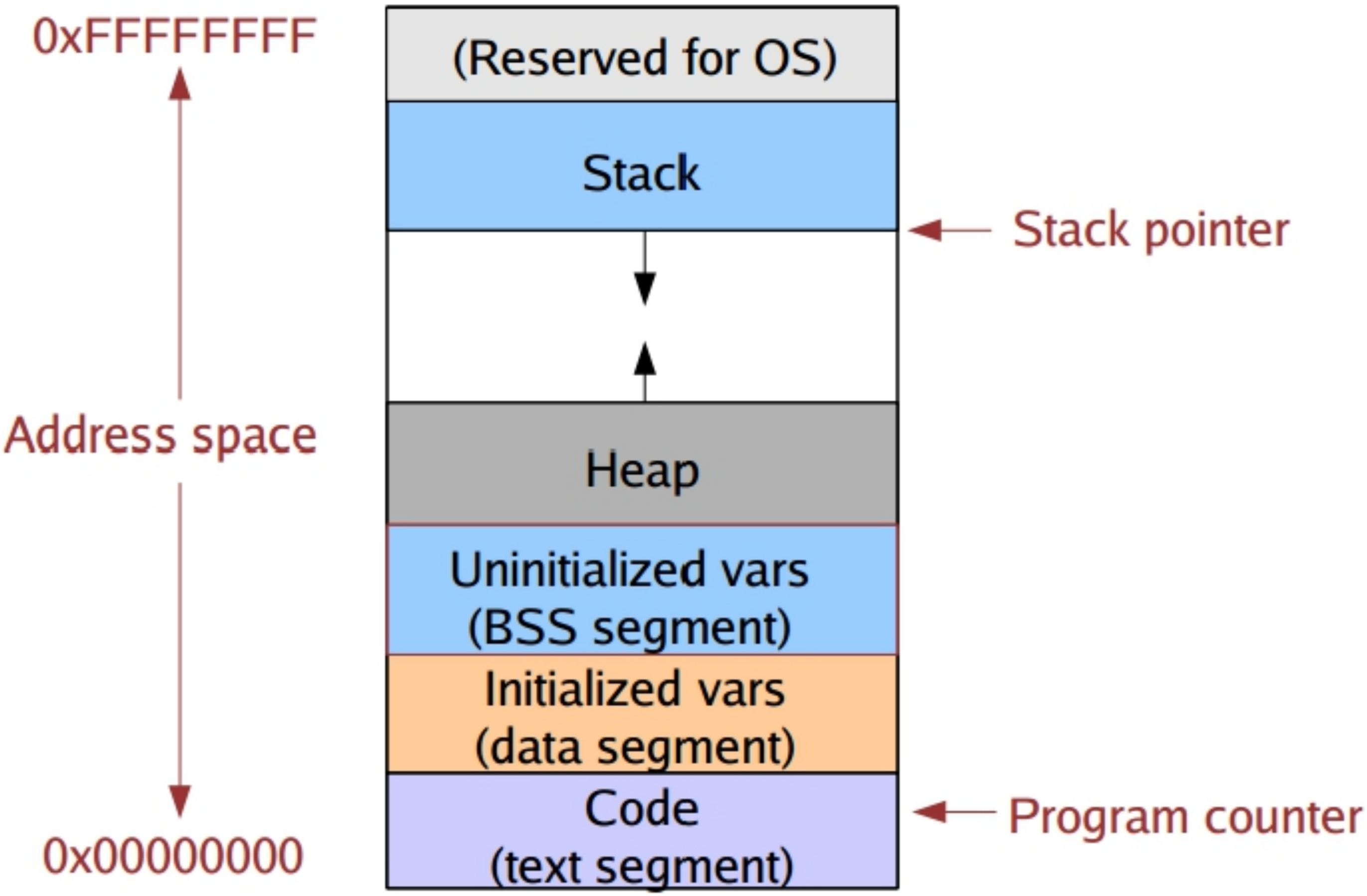


- Share same address space
- Share global variables
- Owned by a single user
- No protection between threads

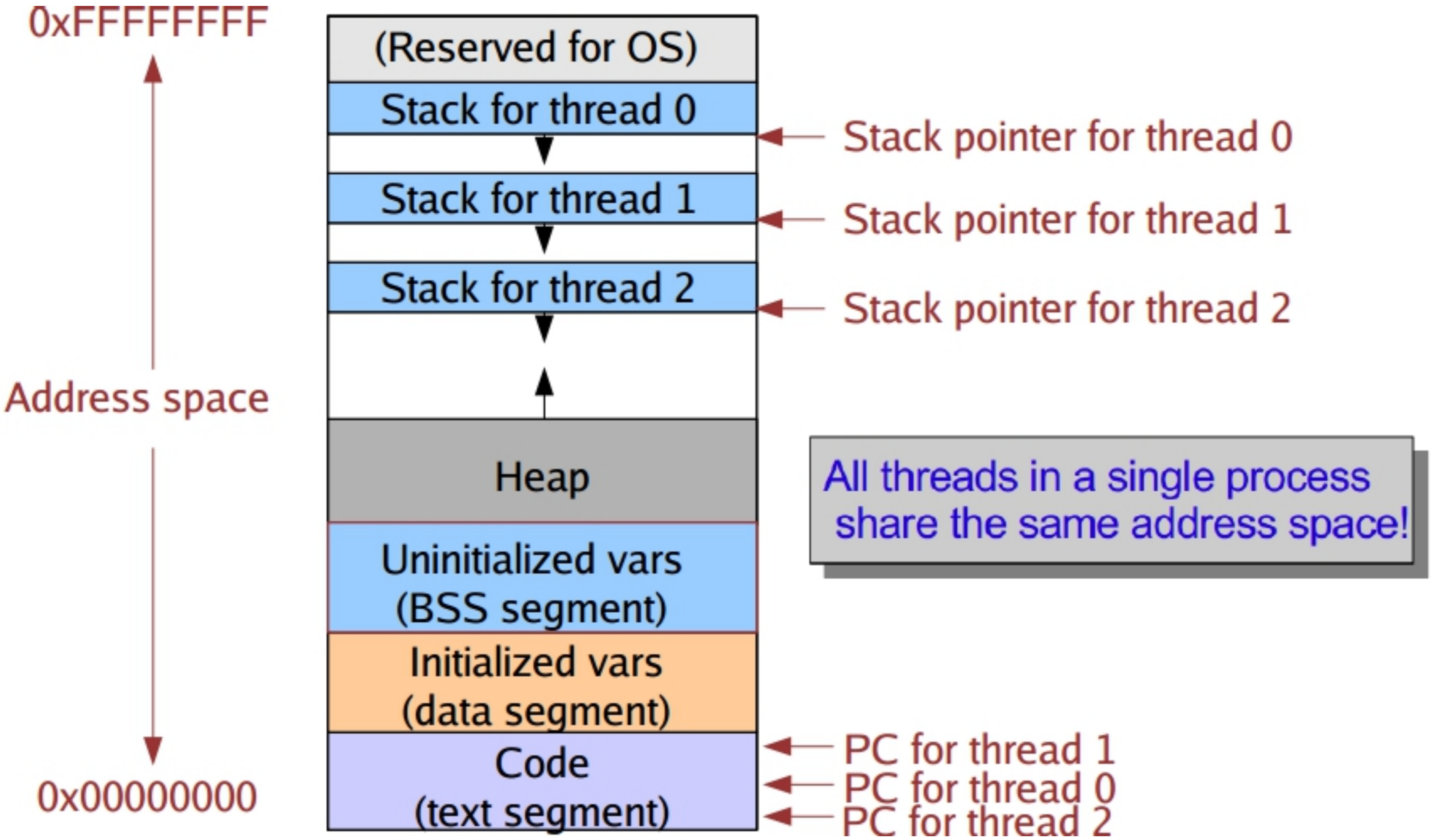
Question

- The register set is listed as a per-thread rather than a per-process item. Why? After all, the machine has only one set of registers.

(Old) Process Address Space



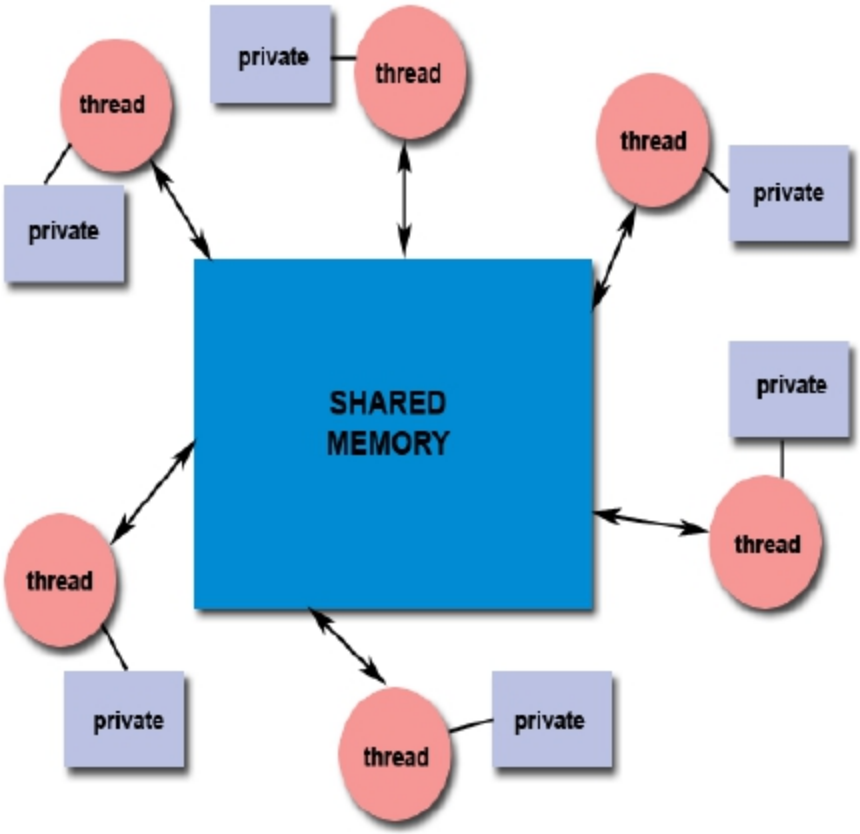
(New) Address Space with Threads



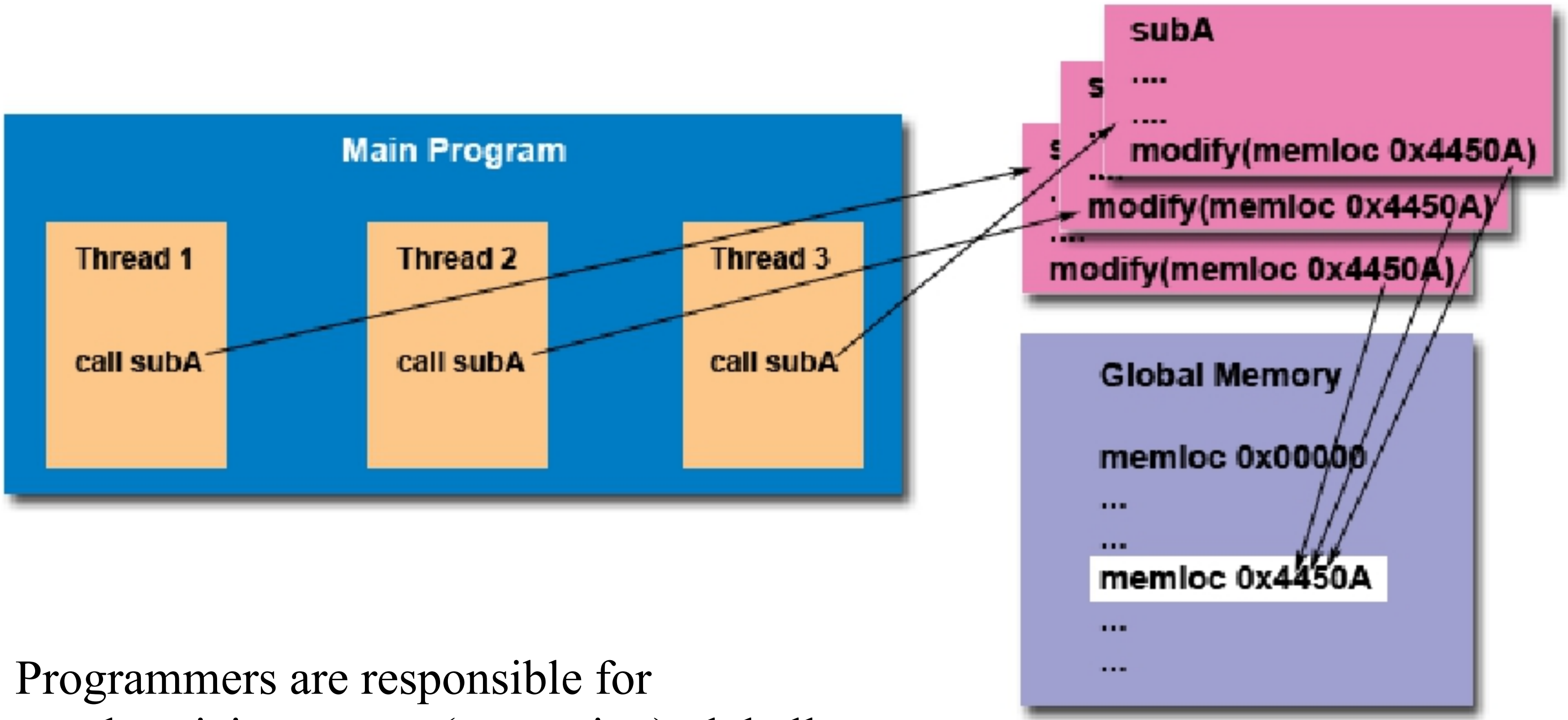
All threads in a single process share the same address space!

Local and Global Variables for Threads

- *Global variables:* the same for all threads
- *Local variables*: each thread gets a copy of the variables
- Threads within a process share memory, open files, and I/O streams.
    - File descriptors are always considered as global variables



Thread Safety



Programmers are responsible for synchronizing access (protecting) globally shared data.

An Example of Unsafe Multi-Threading

```
void push_back(LIST *l, int v) {
    NODE * n = malloc(sizeof(NODE)); // line 1
    n->value = v; // line 2
    n->next = NULL; // line 3
    l->tail->next = n; // line 4
    }
```

Assume

- Two threads running the same code on the same list at the same time.
- Thread 1 just finishes line 3 and Thread 2 is at line 4
- Thread 1 continues on to finish push_back and then Thread 2 runs next.

- What can happen?

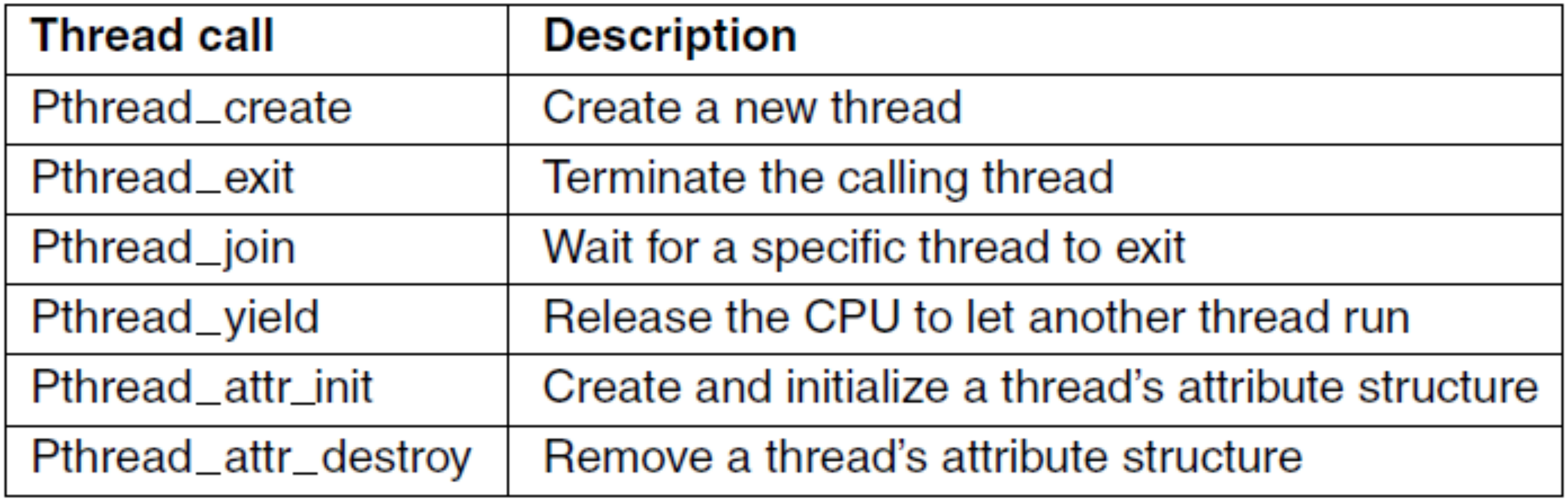
How to Avoid Mess?

- Synchronization, Lock mechanism, etc.
- Details: next lecture

Goals for Today

- Threads
  - Concept
  - Multiple-threading
  - Implementation of threads

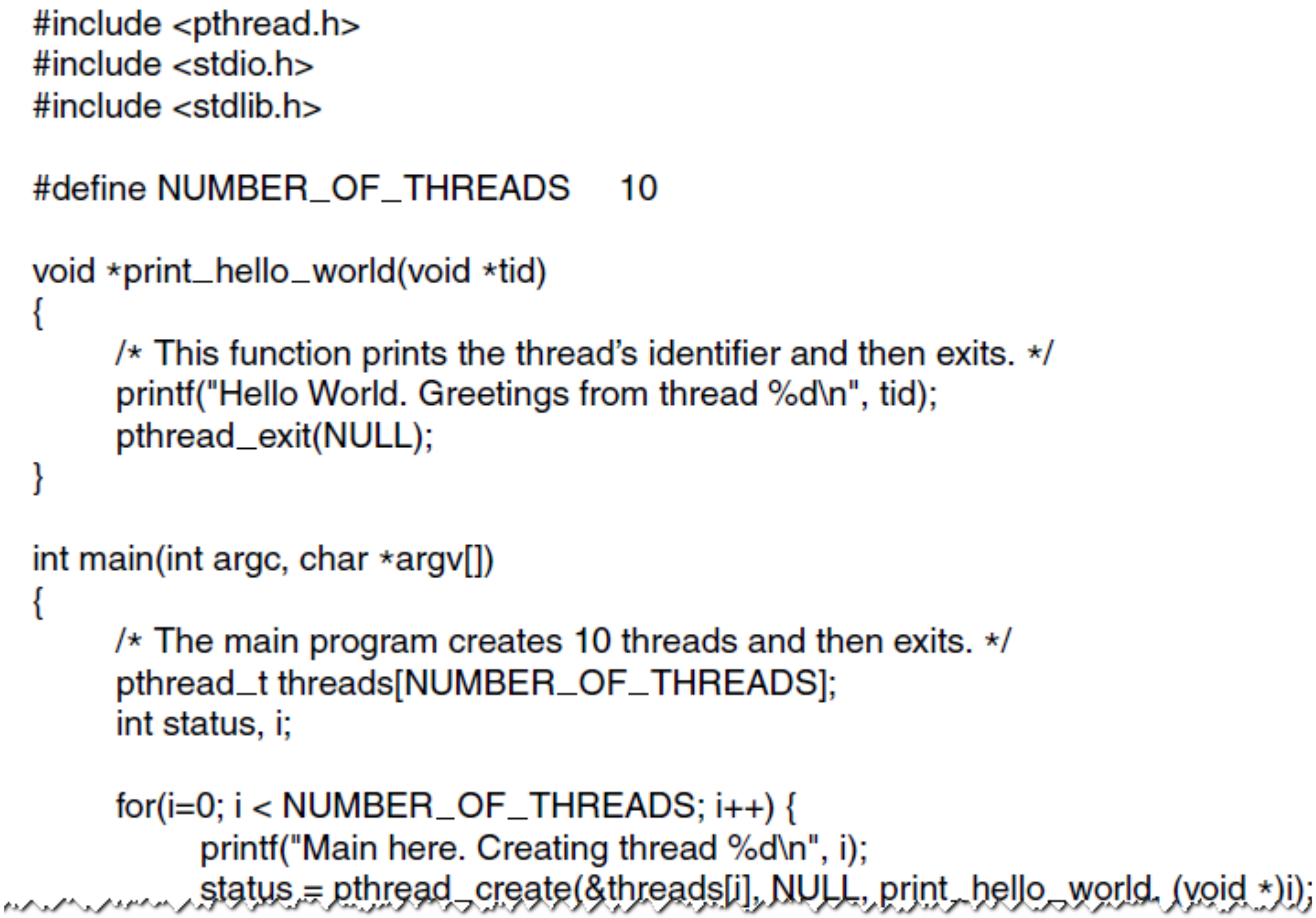
POSIX Threads



Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread’s attribute structure
Pthread_attr_destroy	Remove a thread’s attribute structure

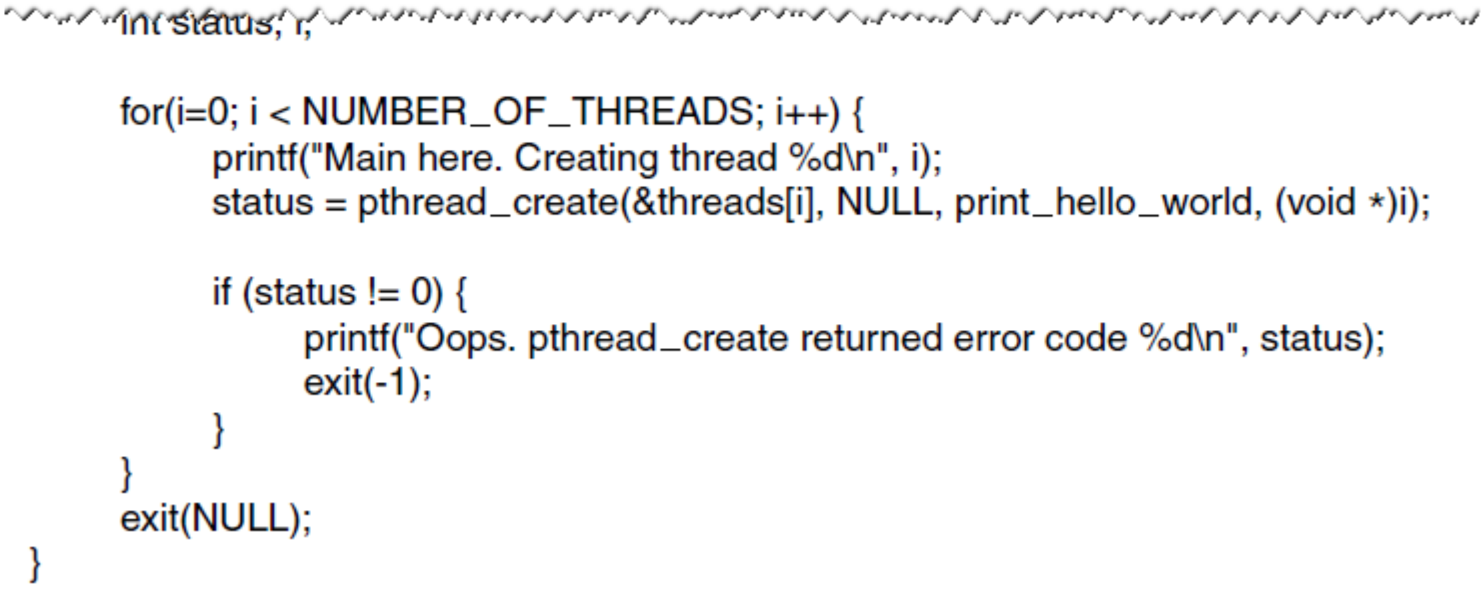
Some of the Pthreads function calls.

POSIX Threads (2)



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

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* This function prints the thread's identifier and then exits. */
        printf("Hello World. Greetings from thread %d\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        /* The main program creates 10 threads and then exits. */
        pthread_t threads[NUMBER_OF_THREADS];
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Main here. Creating thread %d\n", i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);
```

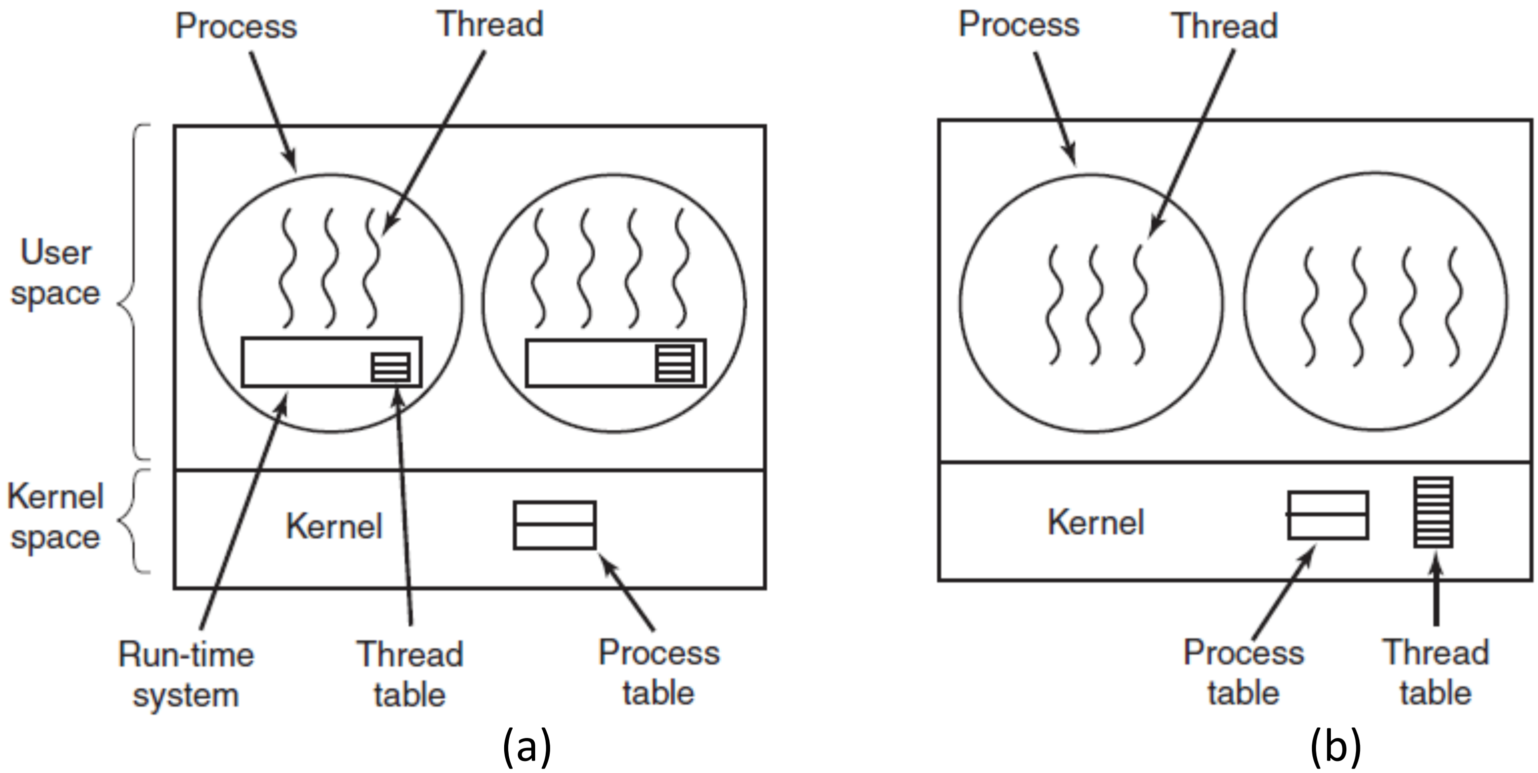
POSIX Threads (3)



```
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Main here. Creating thread %d\n", i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

                if (status != 0) {
                        printf("Oops. pthread_create returned error code %d\n", status);
                        exit(-1);
                }
        }
        exit(NULL);
}
```

An example program using pthreads.

Implementing Threads in User, or Kernel Space



(a) A user-level threads package.
(b) A threads package managed by the kernel.

Question?

- In a system with threads, is there one stack per thread or one stack per process when user-level threads are used? What about when kernel-level threads are used? Explain.