Lecture 3
Threads

I233E OPERATING SYSTEMS

RAZVAN BEURAN

Today's Topics

Threads

- Why they exist
- What they are
- How they work

Threading models

- Many-to-one
- One-to-one
- Many-to-many

Thread libraries

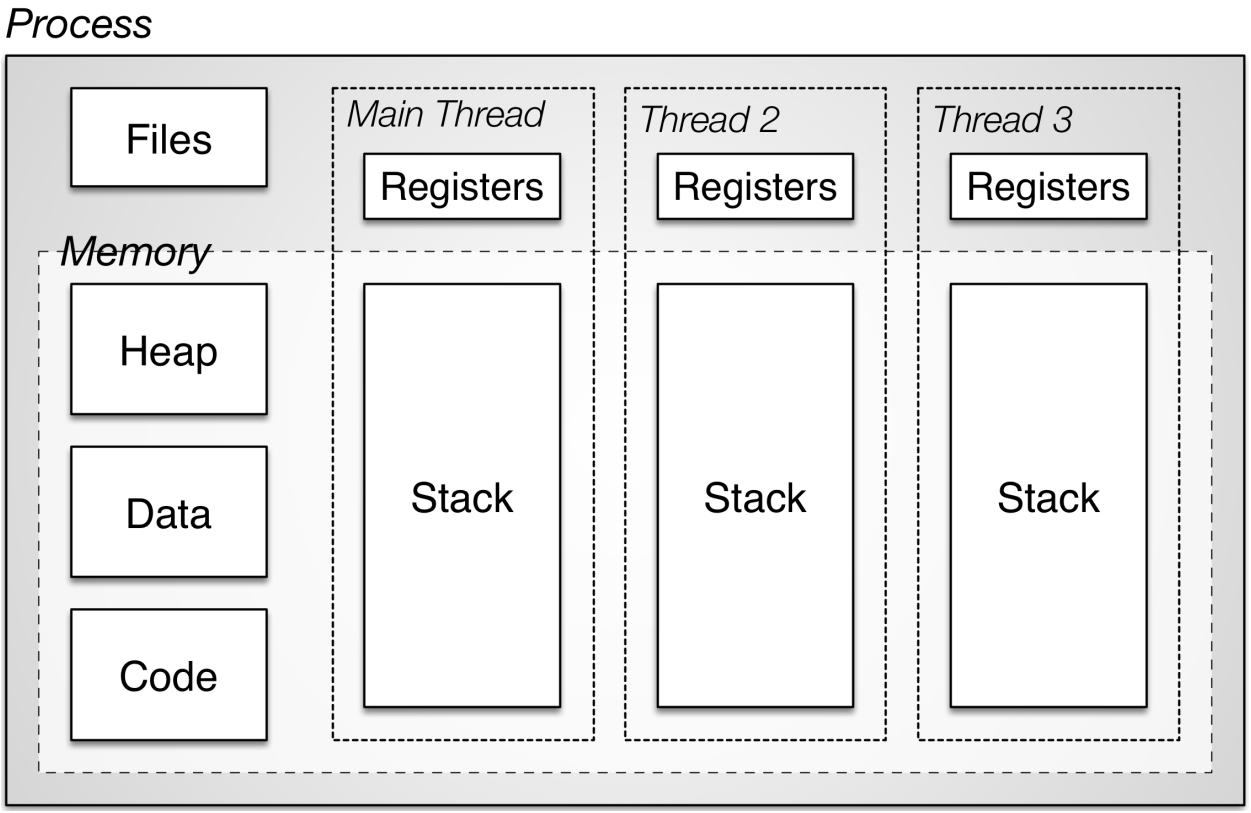
- `pthread`

Threads

Motivation

- Responsiveness
- Resource sharing
- Economy
- Scalability
- Performance (multi-processors, multi-core)

Threads in a Process



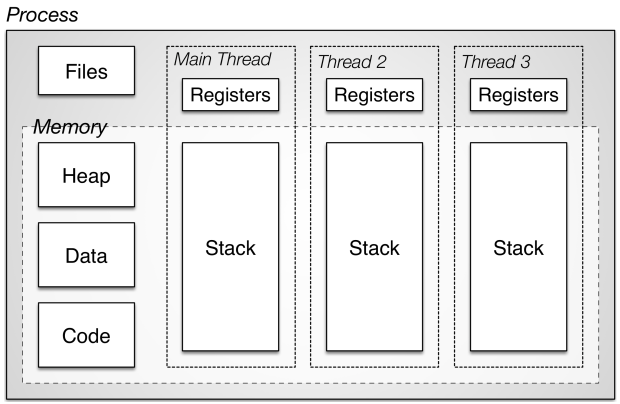
Threads vs. Processes

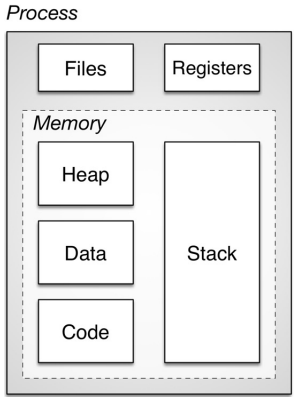
Threads

- Context of execution
  - Thread id
  - Program counter
  - Register set
  - Stack

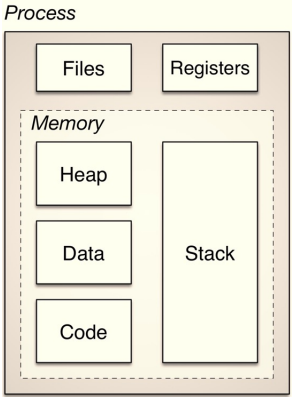
Processes

- Address space
- At least one thread





Parent process



Child process

Differences

Processes	Threads
Heavy weight, resource intensive	Light weight, take less resources
Process switching needs interaction with operating system	Thread switching does not require interaction with operating system
Each process executes the same code, but has its own memory and file resources → independent	All threads share same resources → one thread can read/write another thread's data
If one process is blocked, no other portion of it can execute until the process is unblocked	While one thread is blocked and waiting, a second thread in the same task can run
Multi-process applications may use many resources, but provide more development freedom	Multi-thread processes use fewer resources, but require more careful development

Threading Models

Thread Types

Types of threads

- User-level threads
- Kernel-level threads

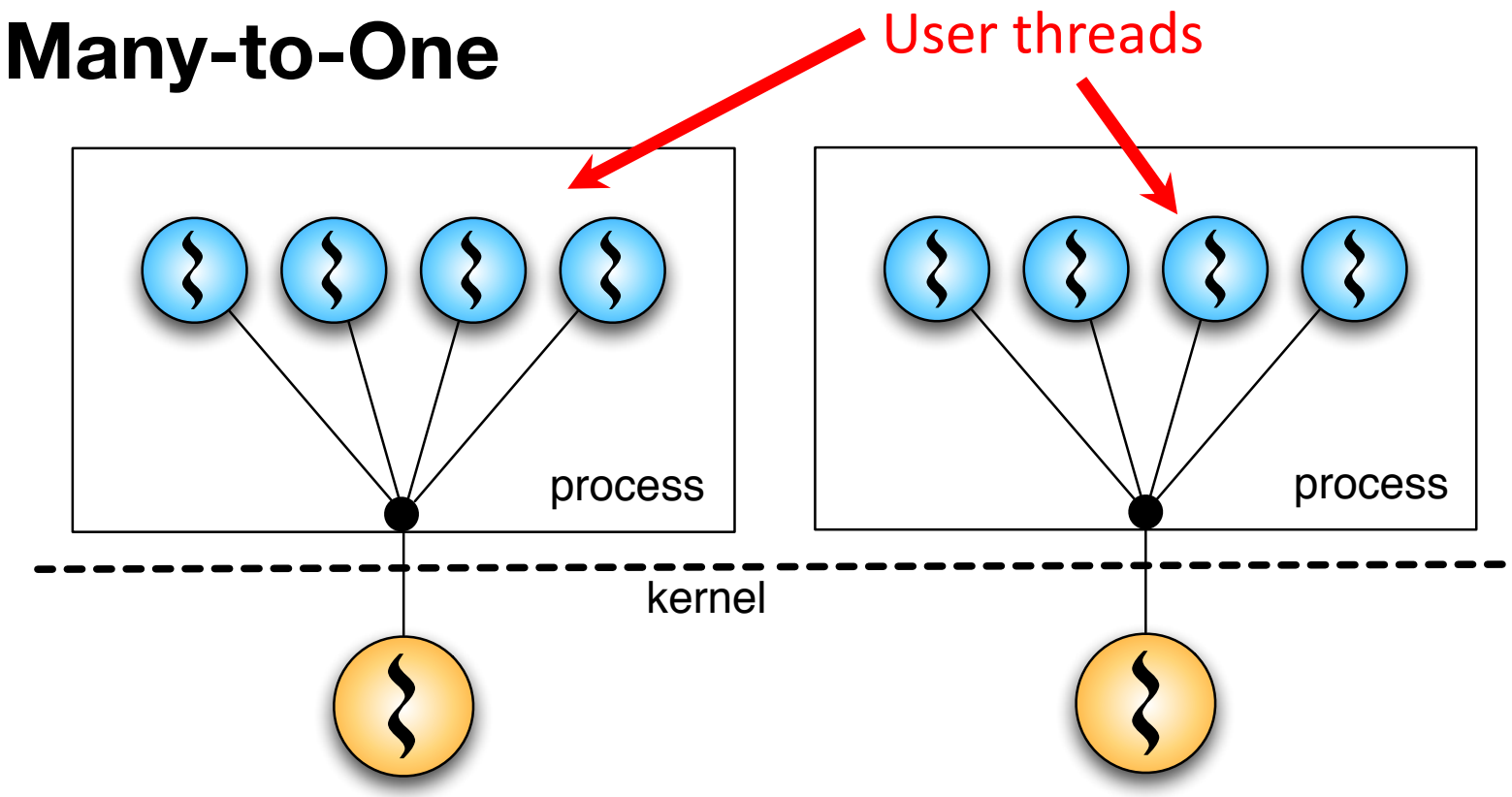
User-level threads

- Managed through library
- Multiple threads inside one process

Kernel-level threads

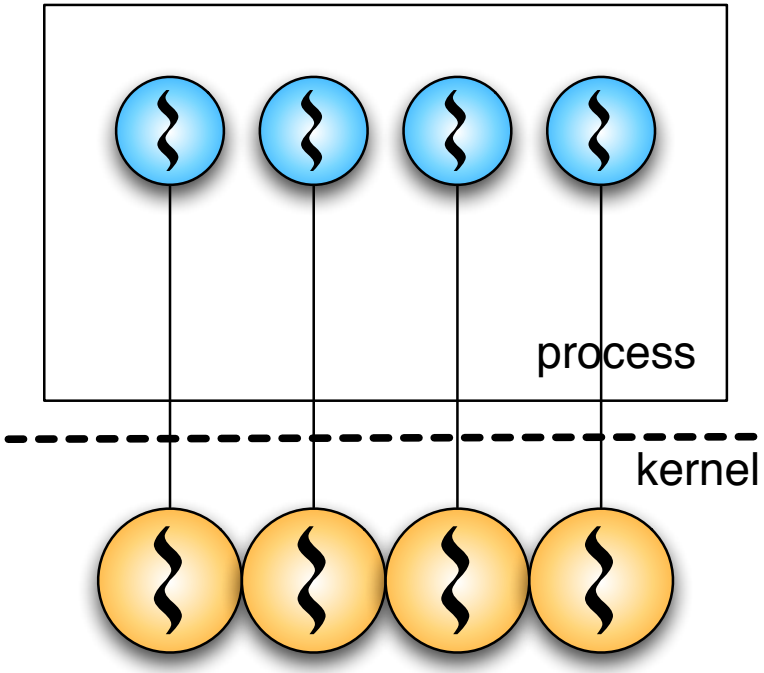
- Execution context managed by OS kernel
- Handle actual concurrent processes

Threading Models



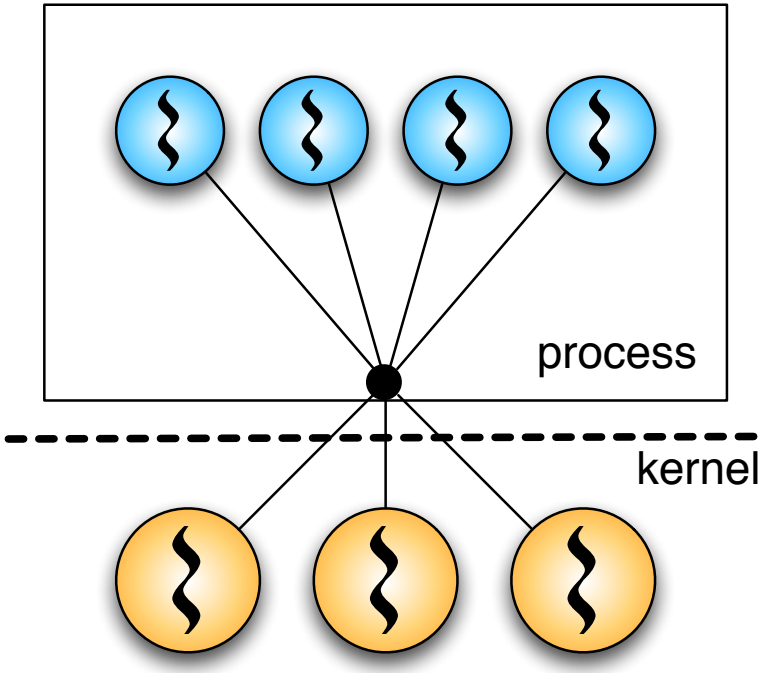
Threading Models (2)

One-to-One



More concurrency

Many-to-Many



Concurrency + better management

Differences

User-level threads	Kernel-level threads
Faster to create and manage	Slower to create and manage
Implementation done via a thread library at user level	Operating system supports creation of kernel-level threads
User-level threads are generic, and can run on any operating system	Kernel-level threads are specific to each operating system
Multi-threaded applications do not take advantage of multi-processing features of the OS	Kernel routines themselves can be implemented multi-threaded to improve performance

Thread Libraries

Thread Libraries

Main libraries

- POSIX pthreads
- Win32 threads
- Java threads

Pthread

- C language library
- Available in many OSs
- Can be user-level or kernel-level
- Type “man pthread” in any Unix shell

Main Functions

Create thread

- Implemented by calling a specific function
  `pthread_create(&thread_id, ATTRS, &start_routine, ARG)`

Define the start routine for thread

- Represents the body of the thread
- Like any other regular function, but with a predefined signature
  `void *start_routine(void *arg)`

Thread end of life

- Parent waits for the thread to terminate
  `pthread_join(thread_id, &VALUE_PTR)`

Thread Example

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

struct timespec quarter_sec = {0, 250000000};
struct timespec tenth_sec   = {0, 100000000};

void *child_thread(void* arg)
{
  printf("\t\t Child: I am the child thread.\n");
  for (int i = 0; i < 26; i++) {
    printf("\t\t Child: %d\n", i);
    nanosleep(&quarter_sec, NULL);
  }
  printf("\t\t Child: Thread finished.\n");
  pthread_exit(0);
}
```

Thread Example (cont.)

```
int main(int argc, char *argv[])
{
  pthread_t thr_id;

  pthread_create(&thr_id, NULL, &child_thread, NULL);

  /* Main thread */
  for (int i = 0; i < 26; i++) {
    printf("Parent: %c\n", i + 'A');
    nanosleep(&tenth_sec, NULL);
  }
  printf("Parent: Waiting for the child to finish...\n");
  pthread_join(thr_id, NULL);
  printf("Parent: Child thread has finished.\n");
  return 0;
}
```

DEMO

Summary

Threads

- Light-weight alternative to processes

Threading models

- Threads can be created at user level or kernel level
- Three models exist for how a thread is executed

Thread libraries

- Example of using the `pthread` library

Next Time

Scheduling principles

Scheduling algorithms

NOTES

- Term 2-1 course registration deadline: Oct. 23 at 5 PM
- Assignment #1
  - Upload to JAIST-LMS on Oct. 22
  - Due on Oct. 28 at 23:59
  - Solution during tutorial hour on Oct. 29