Operating Systems

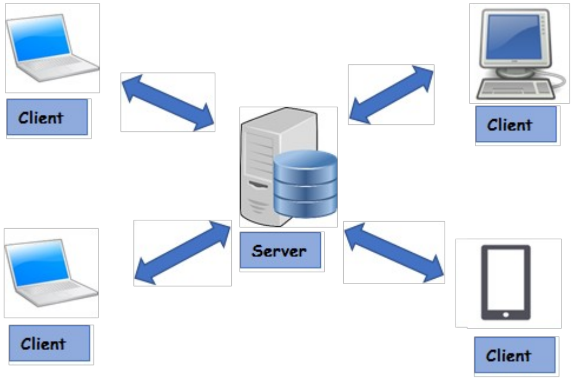
Isfahan University of Technology
Electrical and Computer Engineering Department
1400-1 semester

Zeinab Zali

Session 8: Threads Concepts and API

A Client-Server program

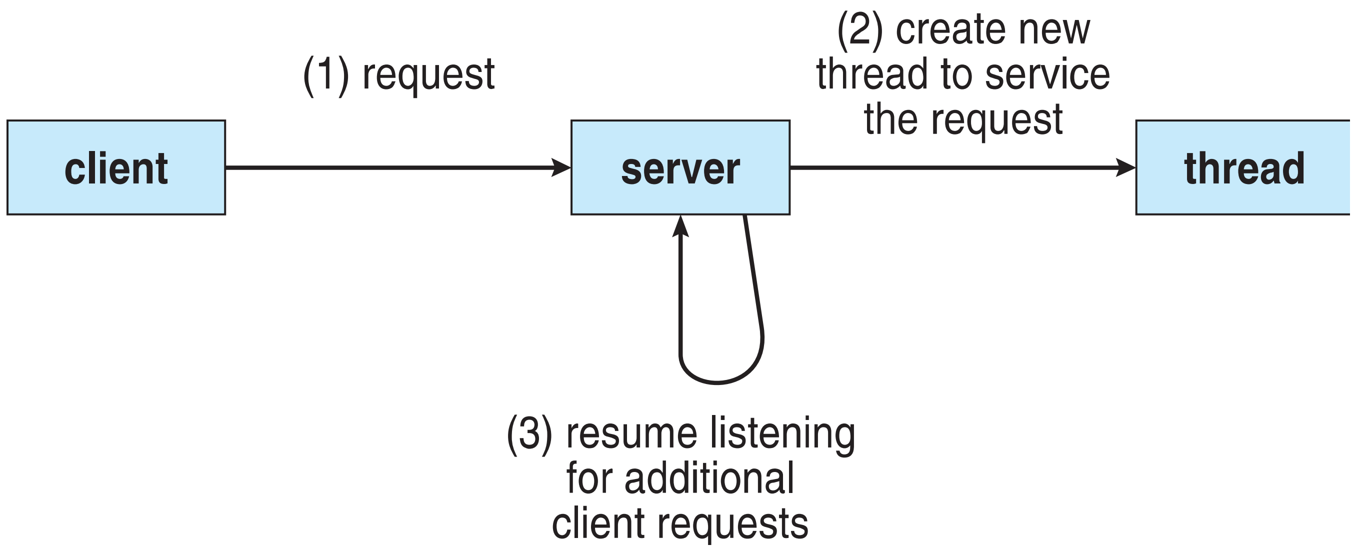
Assume you have a server that is responsible for responding some clients. The clients frequently ask the server to send a requested large file. How does server manage the requests from the clients? What is the problem? What is the solution?



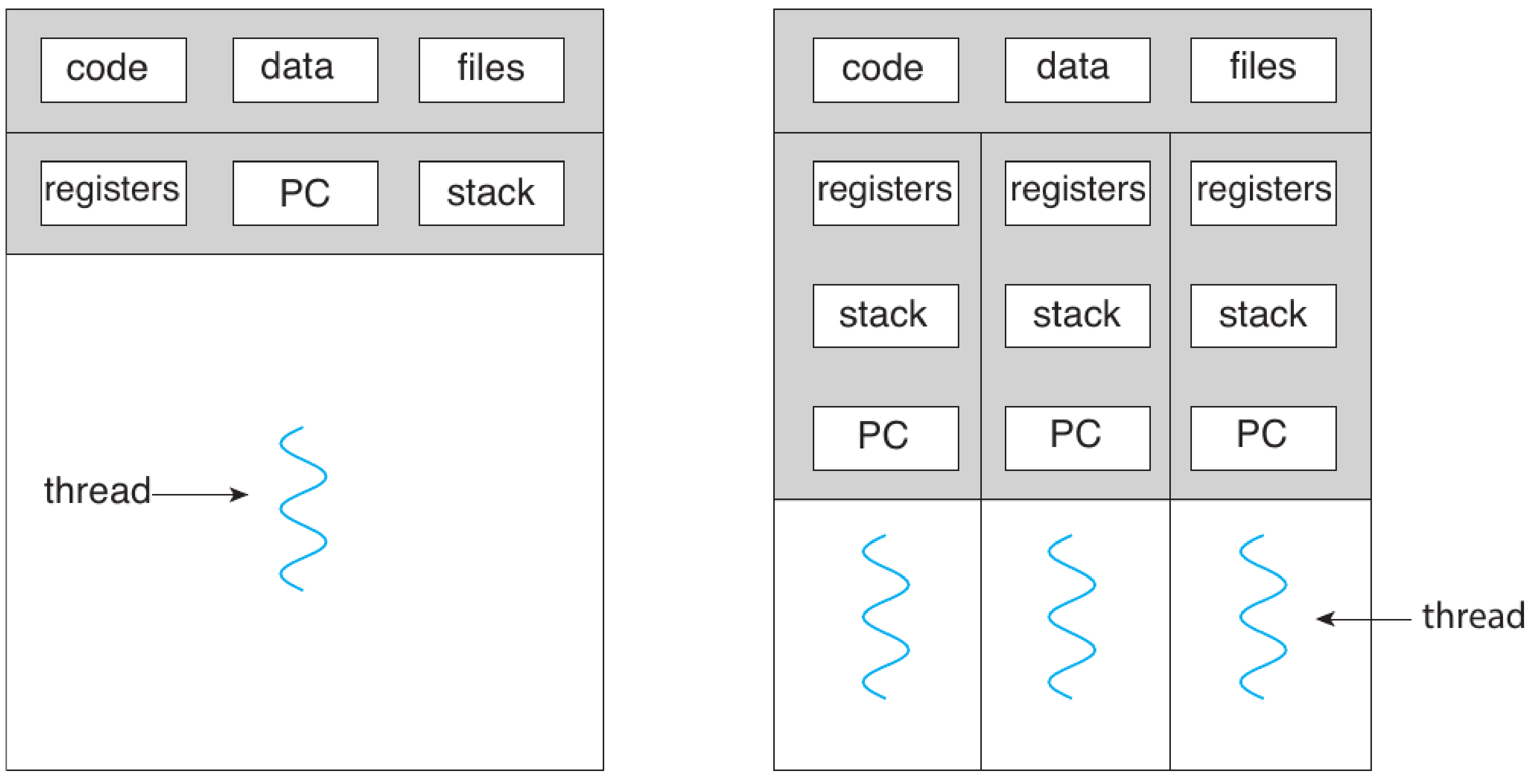
Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks with the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Examples of multi-thread applications: basic sorting, trees, and graph algorithms, programmers who must solve contemporary CPU-intensive problems in data mining, graphics, and artificial intelligence can leverage the power of modern multicore systems by designing solutions that run in parallel.

Multithreaded Server Architecture



Single and Multithreaded Processes



single-threaded process

multithreaded process

Benefits

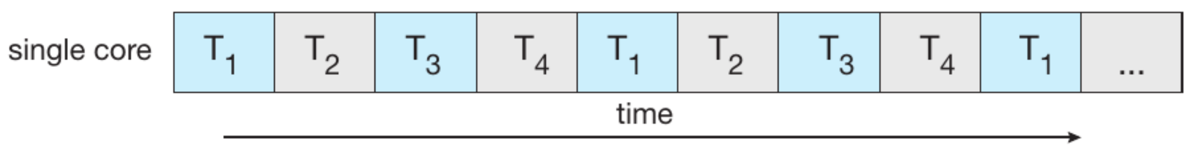
- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces (if the time-consuming operation is performed in a separate, asynchronous thread, the application remains responsive to the user)
- **Resource Sharing –** threads share the memory and the resources of the process to which they belong by default, so easier than shared memory or message passing between processes
- **Economy –** thread creation consumes less time and memory than process creation. Additionally, context switching is typically faster between threads than between processes
- **Scalability –** a single process can take advantage of multiprocessor architectures

Multi-thread kernel

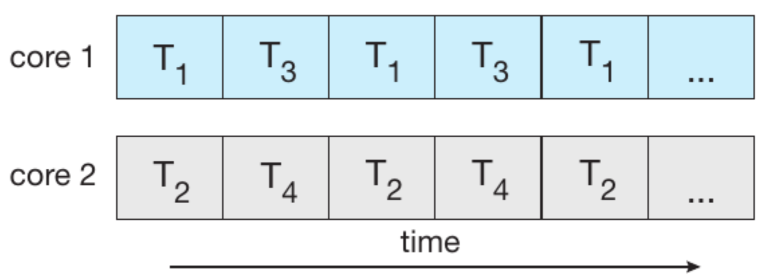
- Most operating system kernels are also typically multithreaded
- The command ps -ef can be used to display the kernel threads on a running Linux system
  - Examining the output of this command will show the kernel thread kthreadd (with pid = 2), which serves as the parent of all other kernel threads.

Concurrency vs. Parallelism

- ***Concurrency*** *supports more than one task making progress*
  - Single processor / core, scheduler providing concurrency
- ***Parallelism*** implies a system can perform more than one task simultaneously
- **Concurrent execution on single-core system:**

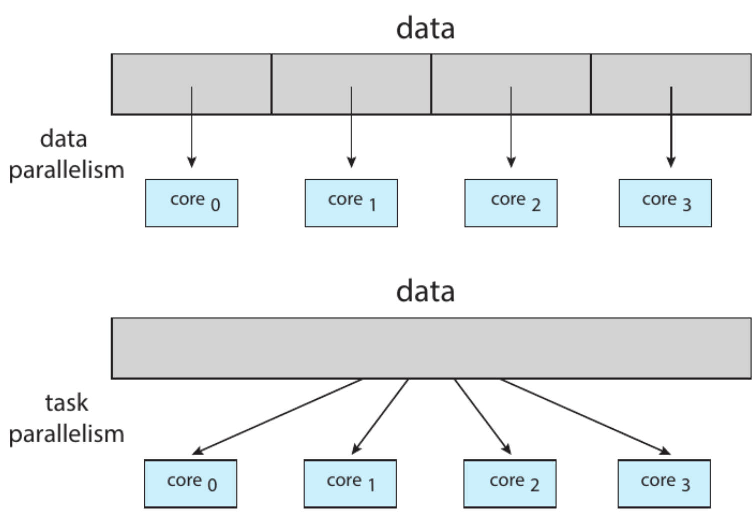


- **Parallelism on a multi-core system:**



Multicore Programming (Cont.)

- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
    - Ex: calculating an array sum or matrix multiplication
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
    - Ex: calculating different statistical operation on the array of elements
  - **Hybrid**



Multicore Programming challenges

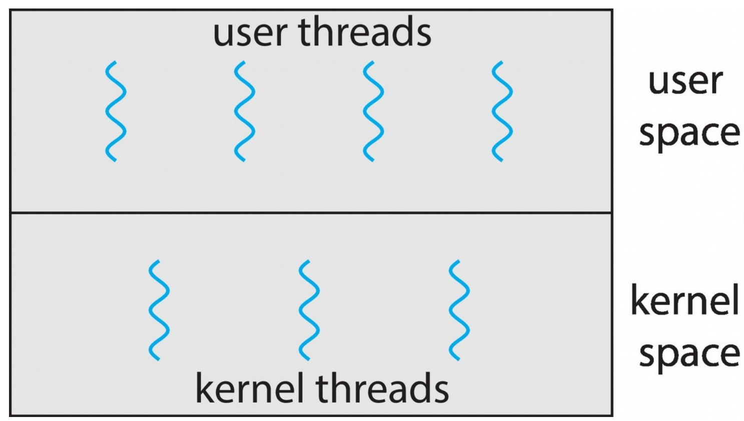
- **Dividing activities:** examining applications to find areas that can be divided into separate, concurrent tasks
- **Balance:** ensure that the tasks perform equal work of equal value.
- **Data splitting:** the data accessed and manipulated by the tasks must be divided to run on separate cores
- **Data dependency:** When one task depends on data from another, programmers must ensure that the execution of the tasks is synchronized to accommodate the data dependency
- **Testing and debugging:** When a program is running in parallel on multiple cores, many different execution paths are possible making debugging difficult

User Threads and Kernel Threads

- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX **Pthreads**
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
  - Examples – virtually all general purpose operating systems, including:
    - Windows
    - Solaris
    - Linux
    - Tru64 UNIX
    - Mac OS X

User and Kernel Threads

A relationship exists between user threads and kernel threads

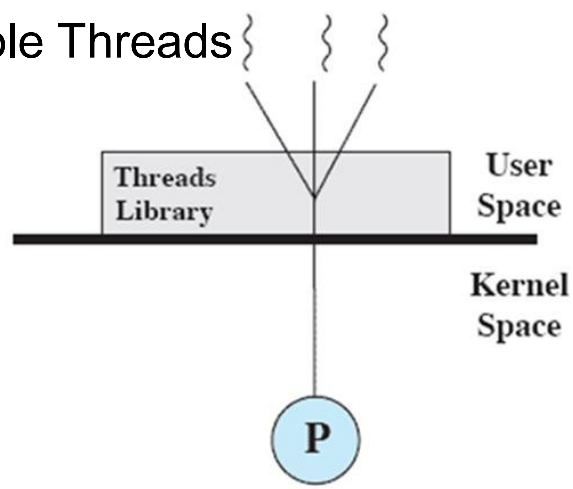


Multithreading Models

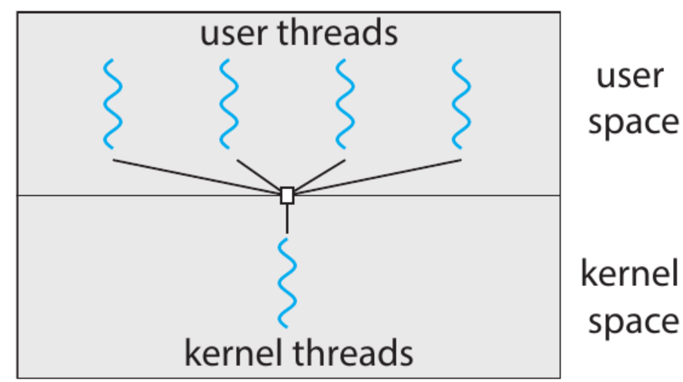
- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One

- Many user-level threads mapped to single kernel thread
  - One thread blocking causes all to block
  - Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

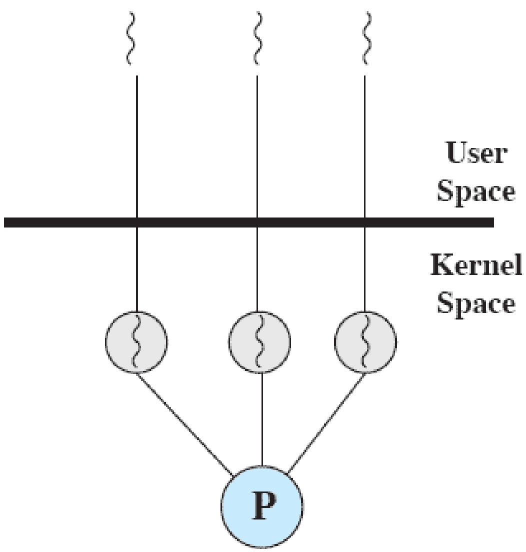


(a) Pure user-level

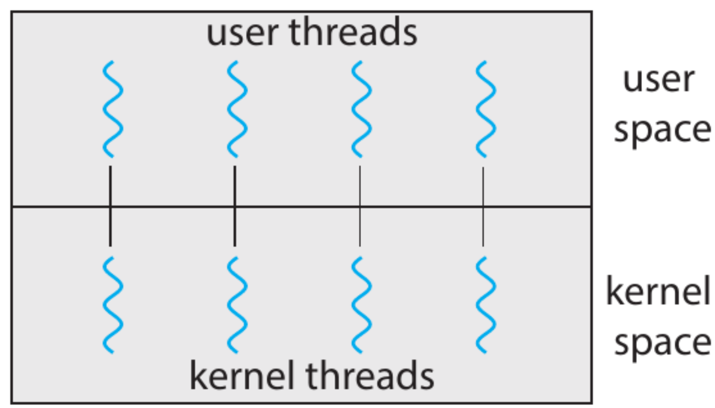


One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
  - More concurrency than many-to-one
  - Number of threads per process sometimes restricted due to overhead
- Examples: **Windows, Linux,** Solaris 9 and later

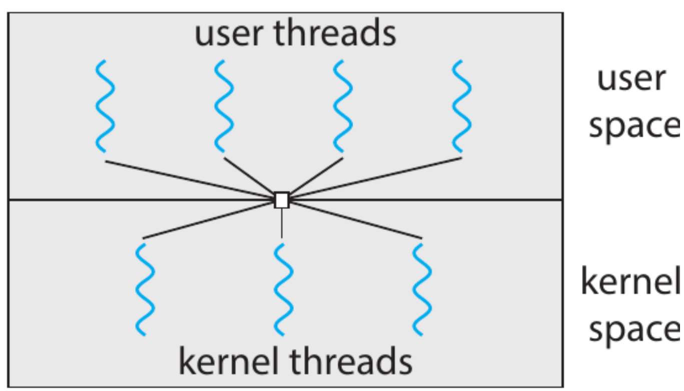


(b) Pure kernel-level



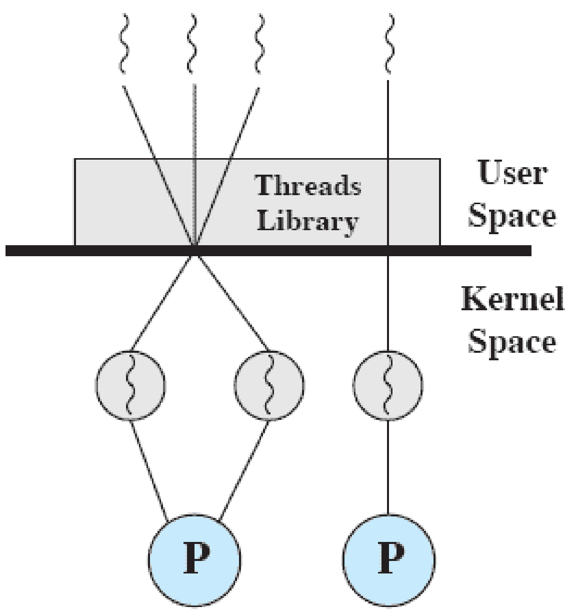
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
  - Allows the operating system to create a sufficient number of kernel threads
  - developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor
  - when a thread performs a blocking system call, the kernel can schedule another thread for execution.
  - Windows with the *ThreadFiber* package
  - Otherwise not very common

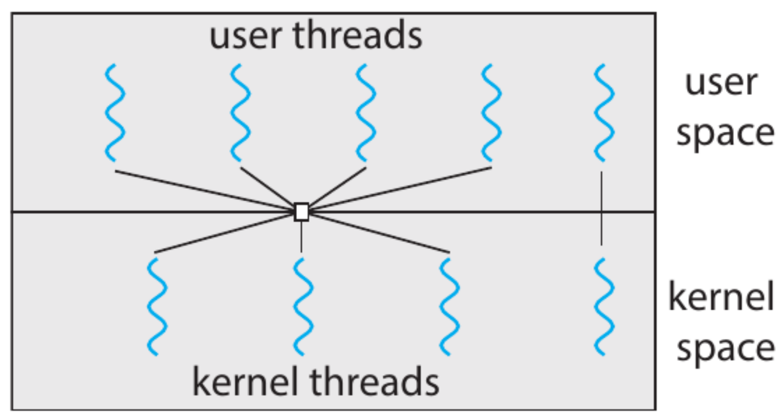


Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Although the many-to-many model appears to be the most flexible of the models discussed, in practice it is difficult to implement.



(c) Combined



تحقیق

- با بررسی در هدر فایل sched.h و جستجو، تحقیق کنید آیا برای مدیریت threadها نیز ساختاری مشابه task_struct در سیستم عامل برای هر thread استفاده میشود؟ یا روش دیگری وجود دارد؟ تفاوتها را مشخص کنید
- مدل threadها در زبانهای c، جاوا و پایتون را مقایسه کنید.
- نحوه ساخت kernel thread و user thread با استفاده از system callهایی شبیه fork به چه صورت است؟

Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- ***Specification***, not ***implementation***
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

Pthread.h

- get the default attributes
  - Pthread_attr_init(pthread_attr_t * attr)
- create the thread
  - pthread_create(pthread_t *tid, pthread_attr_t *attr_t,
    void* thread_runner, void *thread_runner_args):
- wait for the thread to exit
  - pthread_join(pthread_t *tid, void ** thread_runner_ret_val)
- Exit thread
  - pthread_exit(void * pthread_runner_ret_val)

Pthreads Example

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
    return -1;
  }
```

```
    /* get the default attributes */
    pthread_attr_init(&attr);
    /* create the thread */
    pthread_create(&tid,&attr,runner,argv[1]);
    /* wait for the thread to exit */
    pthread_join(tid,NULL);

    printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;

    for (i = 1; i <= upper; i++)
        sum += i;

    pthread_exit(0);
}
```

Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```