CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

Bachelor of Engineering (Computer Science & Engineering)

Operating System (CST-328)

Subject Coordinator: Er. Kulvinder Singh(E8770)

DISCOVER . LEARN . EMPOWER

Lecture 7

Threads

- Processes and Threads
- fork() system call
- Thread Approaches
- Types of threads
- Benefits of thread
- Concept of multithreading
- Linux thread management

Processes and Threads

- The unit of dispatching is referred to as a ***thread*** or ***lightweight process***
- The unit of resource ownership is referred to as a ***process*** or ***task***
- ***Multithreading*** - The ability of an OS to support multiple, concurrent paths of execution within a single process.

- A running process may issue system calls to create new processes:

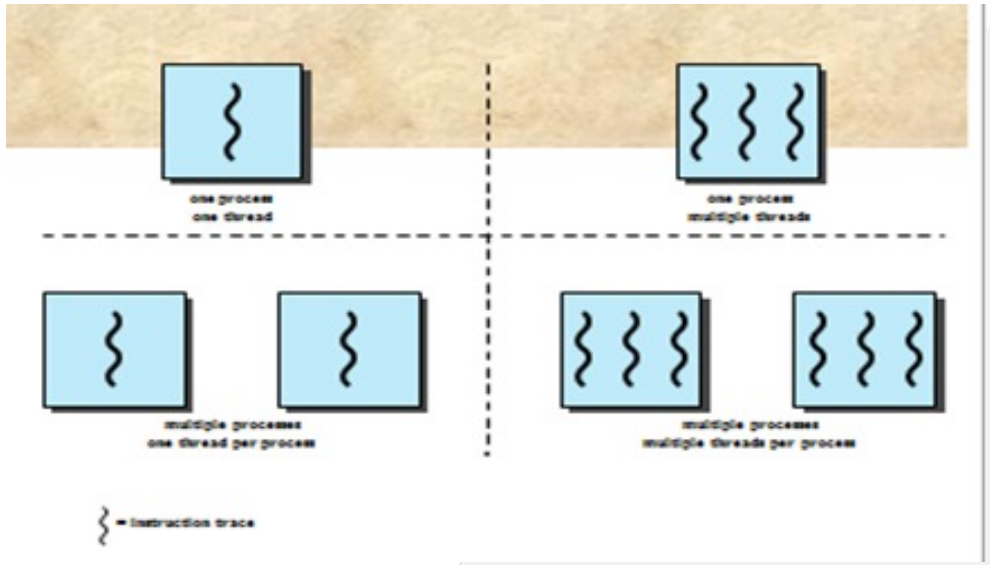
In UNIX: *fork* system call

Single Threaded Approaches

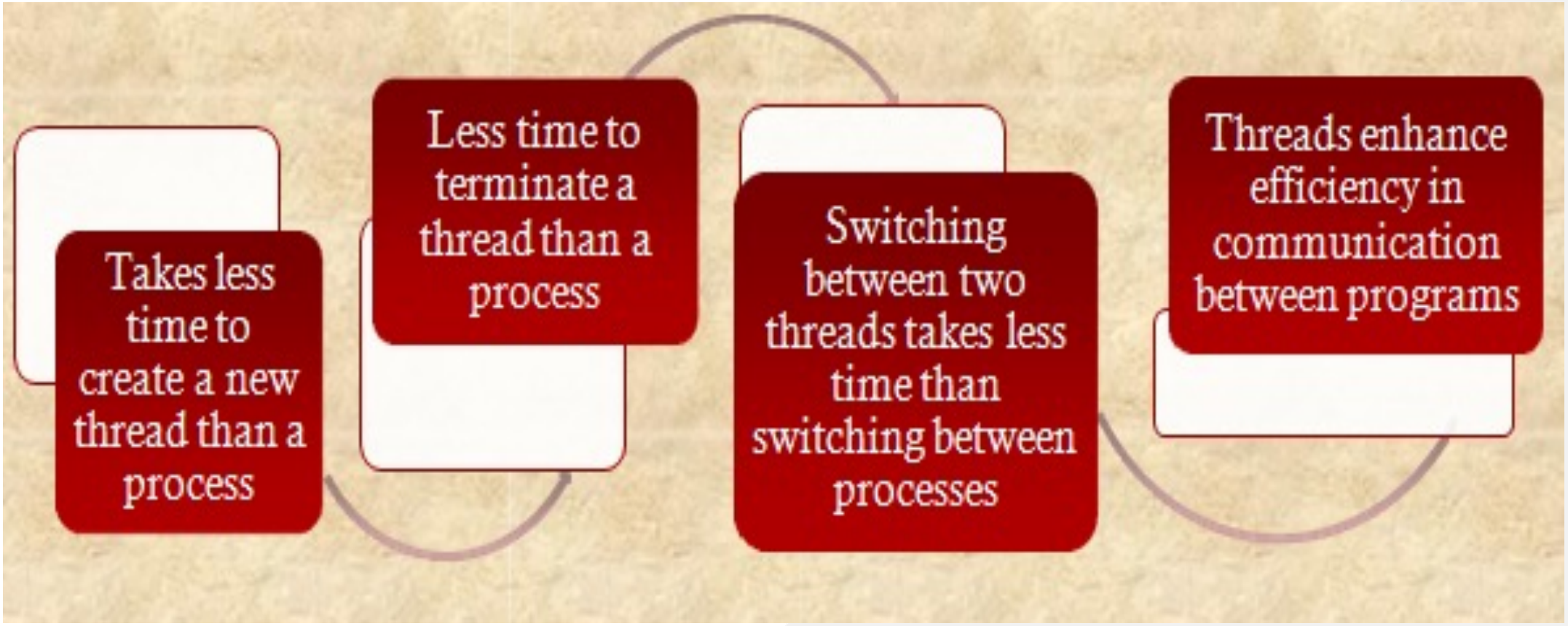
- A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
- MS-DOS is an example.

Multithreaded Approaches

- A Java run-time environment is an example of a system of one process with multiple threads



Benefits of Threads



Thread Execution States

- The key states for a thread are:
  - Running
  - Ready
  - Blocked

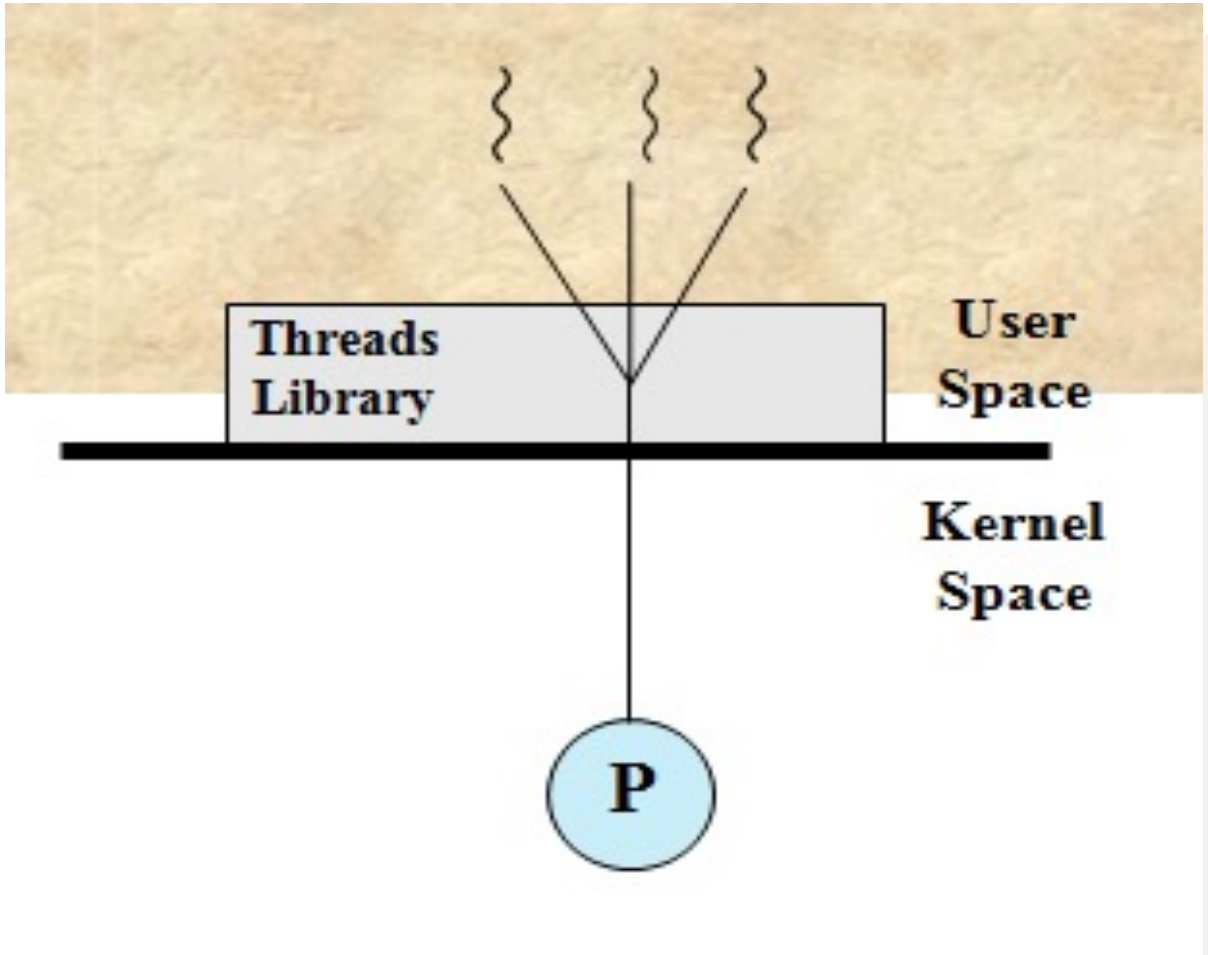
Thread operations associated with a change in thread state are:

- Spawn
- Block
- Unblock
- Finish

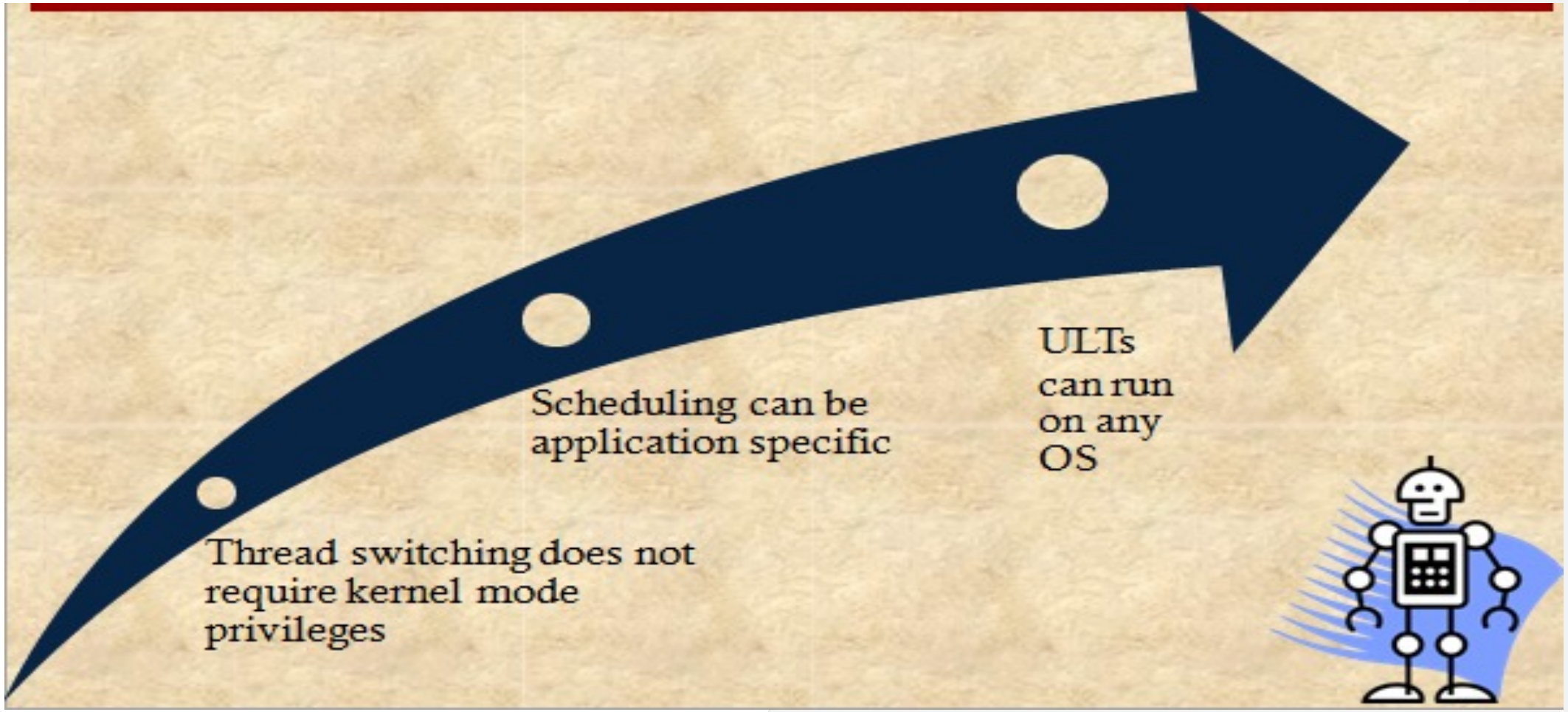
Types of Threads

User-Level Threads (ULTs)

- All thread management is done by the application
- The kernel is not aware of the existence of threads



Advantages of ULTs



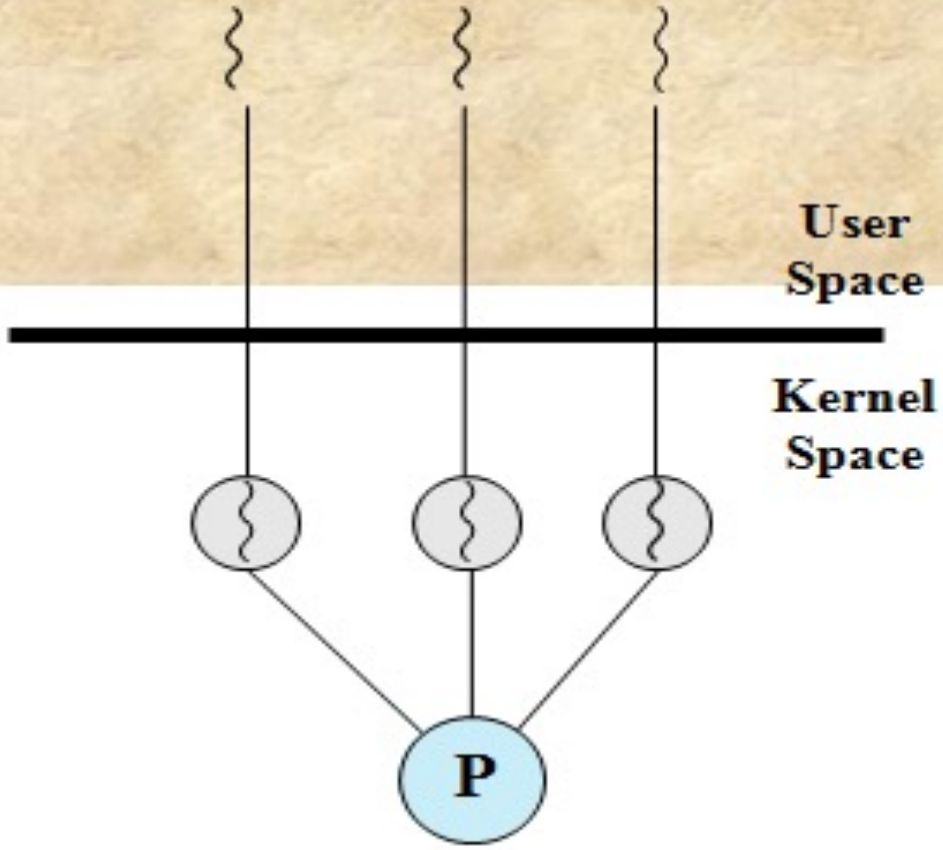
Disadvantages of ULTs

- In a typical OS many system calls are blocking
  - as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
- In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing
- Overcoming ULT Disadvantages

Jacketing
- converts a blocking system call into a non-blocking system call

Writing an application as multiple processes rather than multiple threads

Kernel-Level Threads (KLTs)



(b) Pure kernel-level

- Thread management is done by the kernel
  - no thread management is done by the application
  - Windows is an example of this approach

Advantages of KLTs

- The kernel can simultaneously schedule multiple threads from the same process on multiple processors
- If one thread in a process is blocked, the kernel can schedule another thread of the same process
- Kernel routines can be multithreaded

Disadvantage of KLTs

- The transfer of control from one thread to another within the same process requires a mode switch to the kernel
- Thread and process Operation Latencies :

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

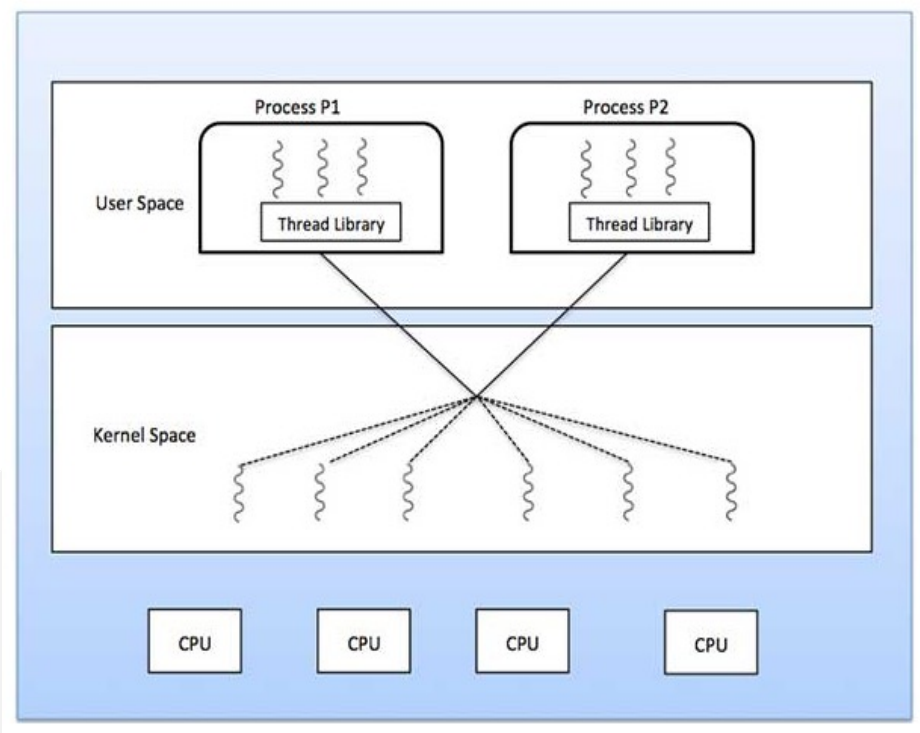
Concept of Multithreading

Multithreading Models

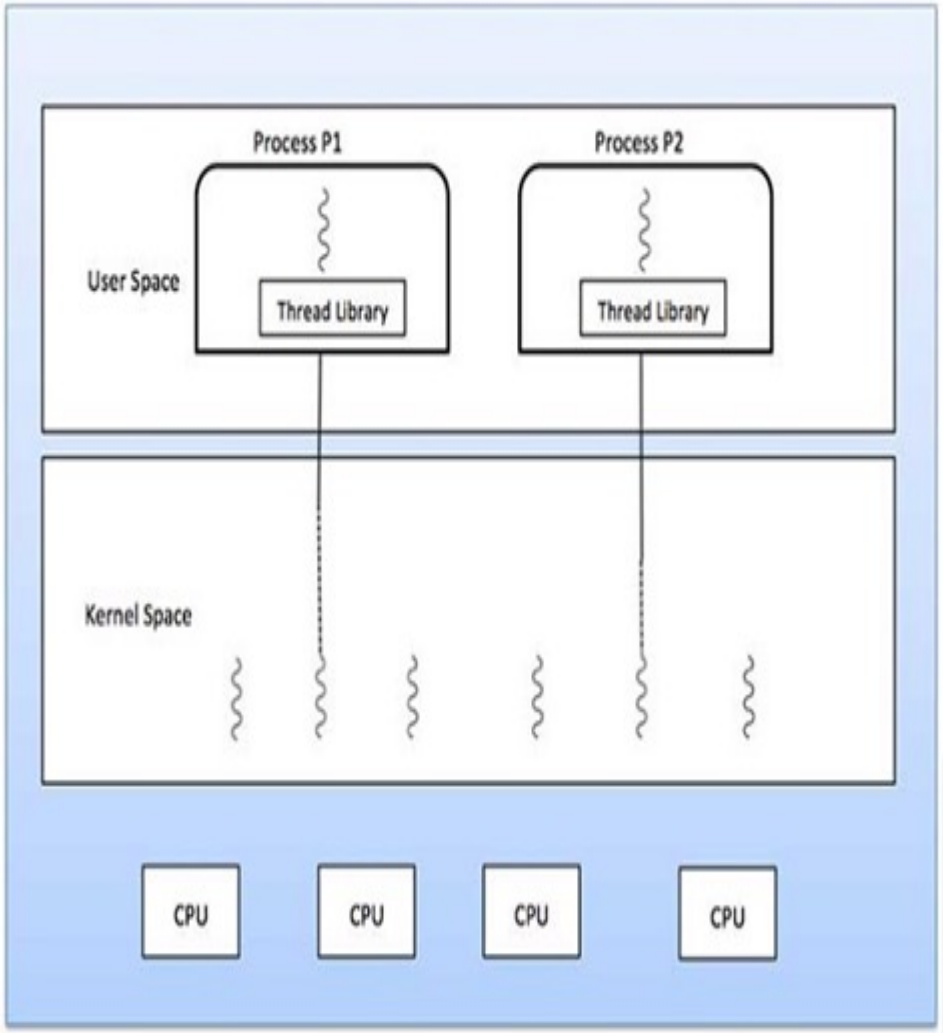
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.

Multithreading models are three types:

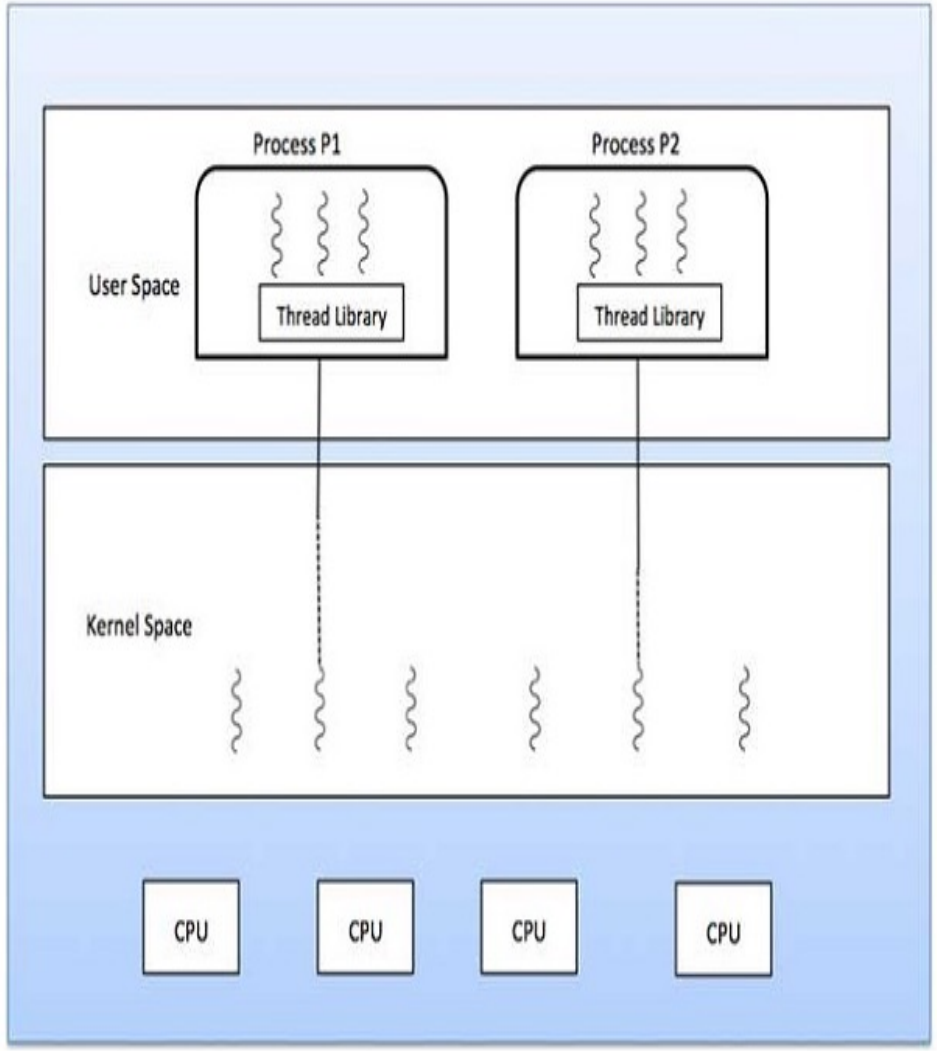
- Many to many relationship.



One to One Model



Many to One Model



Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Linux Thread Management

How are threads implemented in Unix / Linux (posix) systems? Any ideas?

- The same way that we have POSIX systems calls...
- ...We also have POSIX threads...

Care to guess how the POSIX threads are named Any ideas?

Answer : Pthreads

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

Pthread_create

Then, how to compile C program with pthread.h library?

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

/*thread function definition*/
void* threadFunction(void* args)
{
    while(1)
    {
        printf("I am threadFunction.\n");
    }
}
int main()
{
    /*creating thread id*/
    pthread_t id;
    int ret;

    /*creating thread*/
    ret=pthread_create(&id,NULL,&threadFunction,NULL);
    if(ret==0){
        printf("Thread created successfully.\n");
    }
    else{
        printf("Thread not created.\n");
        return 0; /*return from main*/
    }

    while(1)
    {
      printf("I am main function.\n");
    }

    return 0;
}
```

The command is:

gcc thread.c -o thread -lpthread

```
sh-4.3$ gcc thread.c -o thread -lpthread
sh-4.3$ ./thread

Thread created successfully.
I am threadFunction.
I am threadFunction.
I am threadFunction.
I am threadFunction.
…
…
I am threadFunction.
I am main function.
I am main function.
I am main function.
I am main function.
…
…
I am main function.
I am threadFunction.

… and so on.
```

Thread Argument Passing

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     8

char *messages[NUM_THREADS];

void *PrintHello(void *threadid)
{
   long taskid;

   sleep(1);
   taskid = (long) threadid;
   printf("Thread %d: %s\n", taskid, messages[taskid]);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
long taskids[NUM_THREADS];
int rc, t;

messages[0] = "English: Hello World!";
messages[1] = "French: Bonjour, le monde!";
messages[2] = "Spanish: Hola al mundo";
messages[3] = "Klingon: Nuq neH!";
messages[4] = "German: Guten Tag, Welt!";
messages[5] = "Russian: Zdravstvuyte, mir!";
messages[6] = "Japan: Sekai e konnichiwa!";
messages[7] = "Latin: Orbis, te saluto!";

for(t=0;t<NUM_THREADS;t++) {
  taskids[t] = t;
  printf("Creating thread %d\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *) taskids[t]);
  if (rc) {
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }

pthread_exit(NULL);
}
```

Output

```
Creating thread 0
Creating thread 1
Creating thread 2
Creating thread 3
Creating thread 4
Creating thread 5
Creating thread 6
Creating thread 7
Thread 0: English: Hello World!
Thread 1: French: Bonjour, le monde!
Thread 2: Spanish: Hola al mundo
Thread 3: Klingon: Nuq neH!
Thread 4: German: Guten Tag, Welt!
Thread 5: Russian: Zdravstvytye, mir!
Thread 6: Japan: Sekai e konnichiwa!
Thread 7: Latin: Orbis, te saluto!
```

Conclusion

This Topic enables students to understand What is difference between a thread and a process, thread types, multithreading, thread argument passing etc.

References

- https://www.includehelp.com/c-programming-questions/compiling-program-with-pthread-library-linux.aspx
- https://www.studytonight.com/operating-system/multithreading
- https://computing.llnl.gov/tutorials/pthreads/