OPERATING SYSTEMS

THREADS

WHAT IS A THREAD?

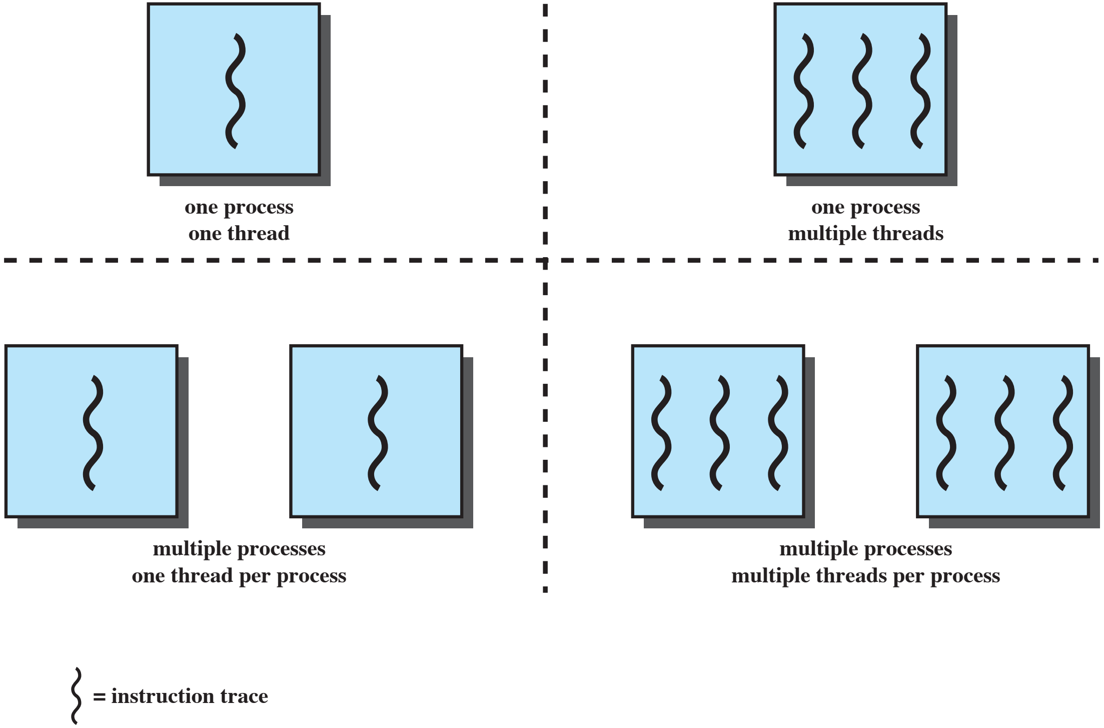
Motivations

- Execution of multiple tasks for the same goal
  - interactive applications for text, audio, video, etc.
  - servers, etc.
- Two main implementations
  - child processes
  - threads - the context switch between threads of the same process requires less resources than the context switch between processes
- Implementation of threads
  - Libraries
  - Hardware support
  - Multicore architectures

Multithreading

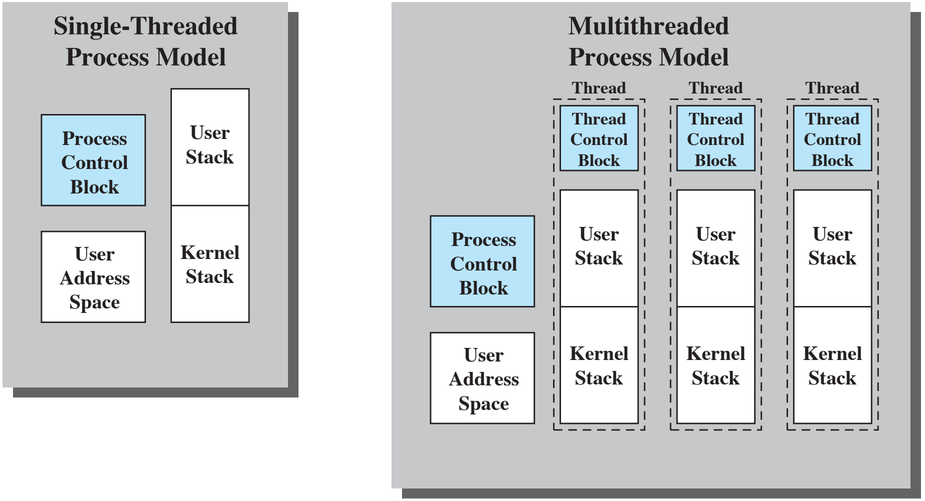
- An application may require the concurrent execution of multiple functions each devoted to a single task
  - interactive graphical (or audio) interface in multimedia production and editing apps, CAD, etc.
  - Input validation
- Server
  - one thread associated to each client process
- Operating Systems
  - some functionalities are implemented as a group of threads within the same process

Processes and Threads

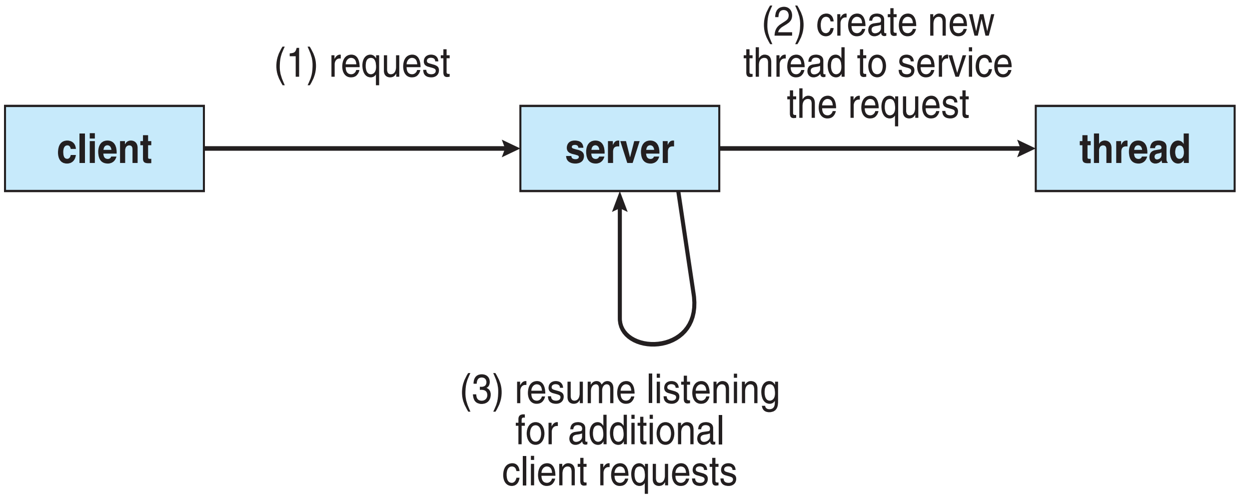


Threads

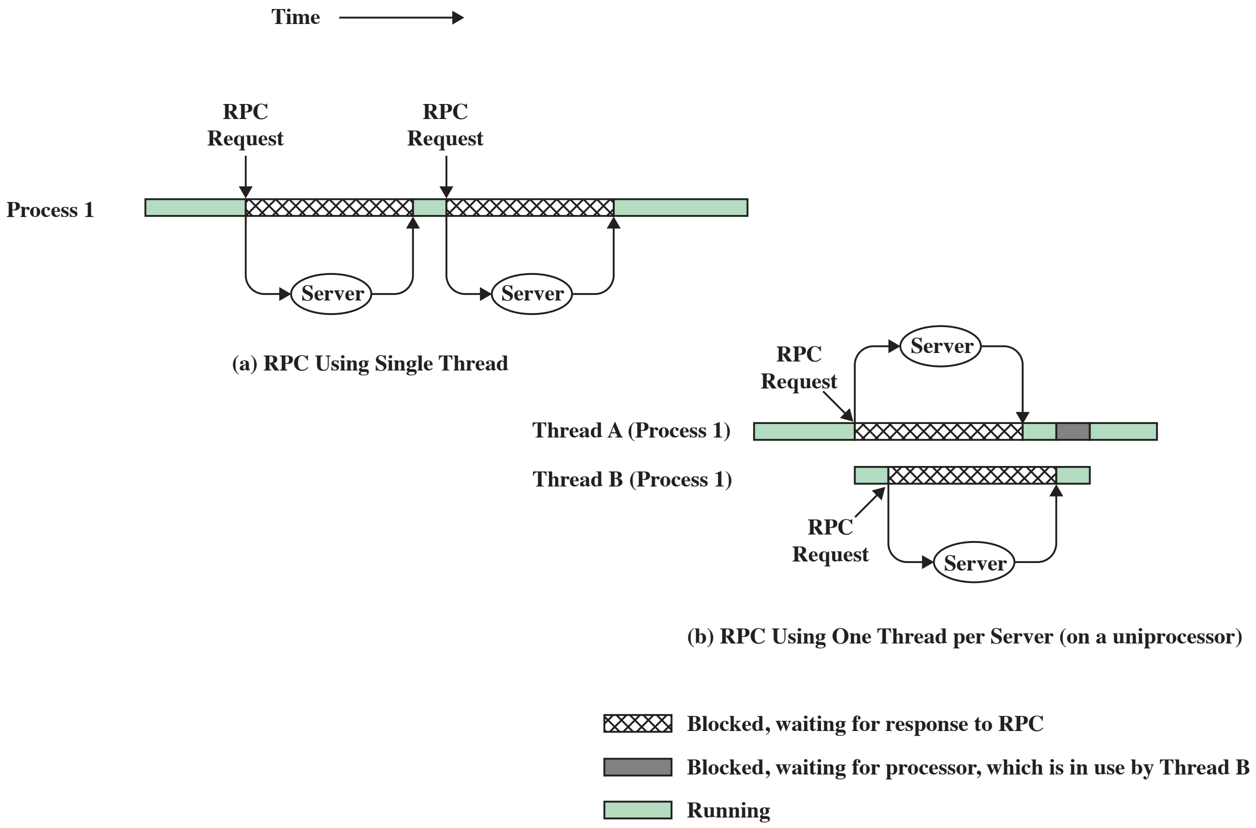
- **The CPU executes threads**
- **The process holds the resources**
  - All threads of the same process share the main memory, open files, stack, etc.
  - Each thread has a program counter, a set of registers, the stack



Multithreaded server architectures



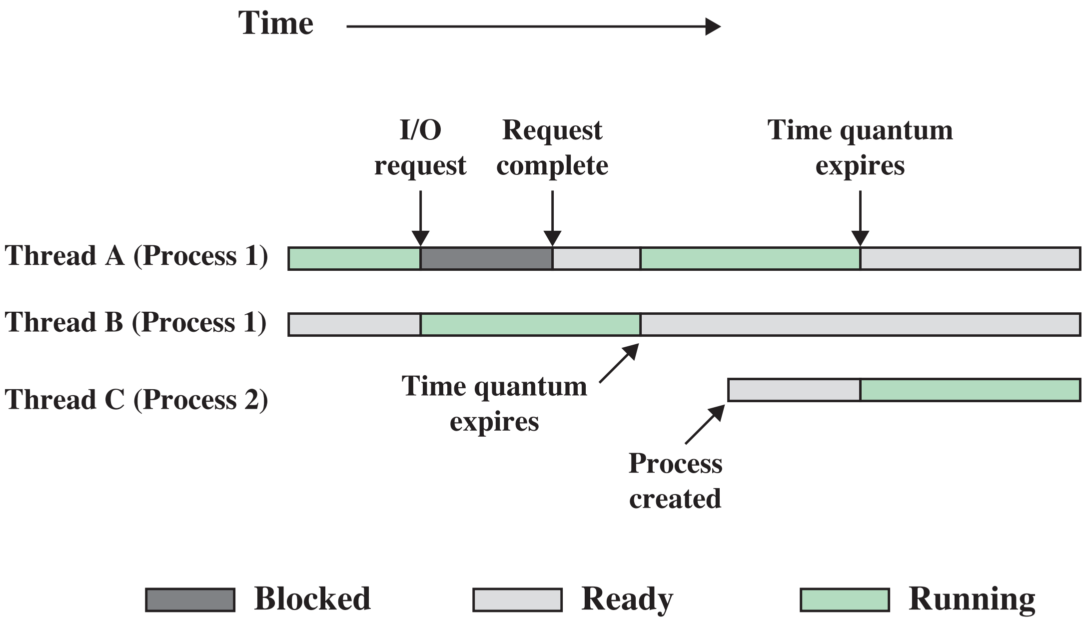
RPC and threads



Advantages of Multithreading

- Response time
  - A blocked thread does not necessarily block the process
- Resource sharing
  - Threads within the same process can easily cooperate without the help of the OS
- Lightweight management
  - The management of concurrent threads requires less computational resources than concurrent processes
- Scalability
  - Independent threads can exploit the availability of multiple cores or multiple processors

Multithreading



Pthreads

```
#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;
   }
```

Pthreads (cont.)

```
    /* 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

```
#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);
```

Threads and multicore architectures

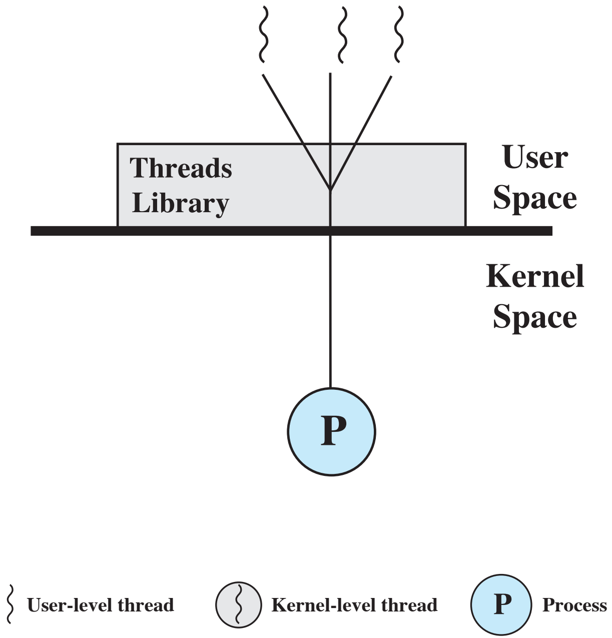
- The design of a multithreaded application that exploits multicore architectures requires
  - Isolating all independent tasks
  - Subdividing the load among tasks
  - Data separation to avoid conflicts in data access
  - Data integrity for tasks operating on a common set of data
  - A long test e debugging phase for identifying inaccuracies due to the uncertainties in the order of execution of different tasks.

Multithread Programming

User-Level Threads and Kernel-Level-Threads

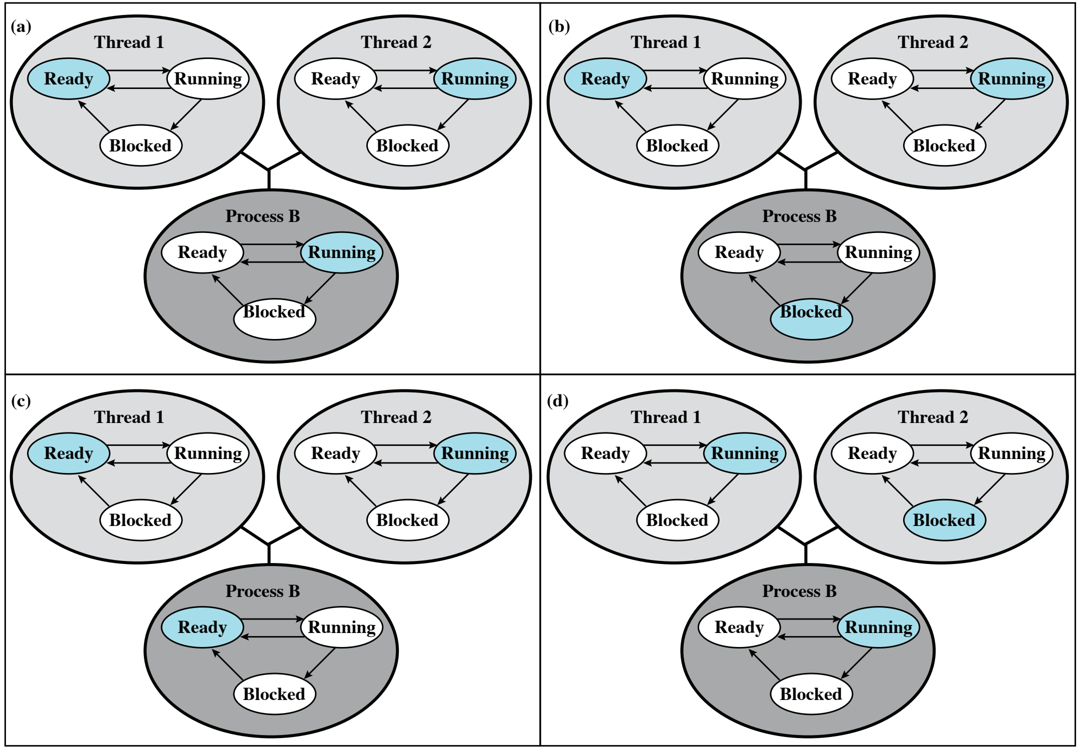
- User-Level Threads (ULT)
  - Do not require a multithreaded OS
  - Implemented through libraries of the programmng language
- Kernel-Level Threads (KLT)
  - The OS is in charge of managing threads
  - All major OS are multithreaded: Windows, Linux, macOS, Solaris, z/OS, INTEGRITY RTOS, etc.

ULT Many-to-one model



- Thread scheduling is not managed by the OS, that can only schedule processes
- If a thread makes a blocking system call, the kernel blocks the process
- Multiple threads cannot make concurrent systems calls
- No possibility of exploiting multicore architectures

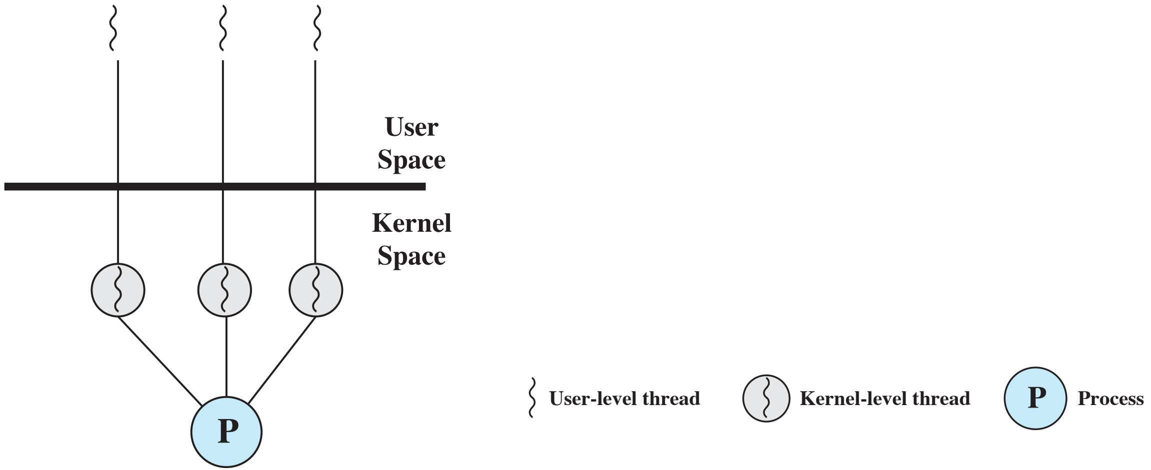
ULT
Process and Thread states



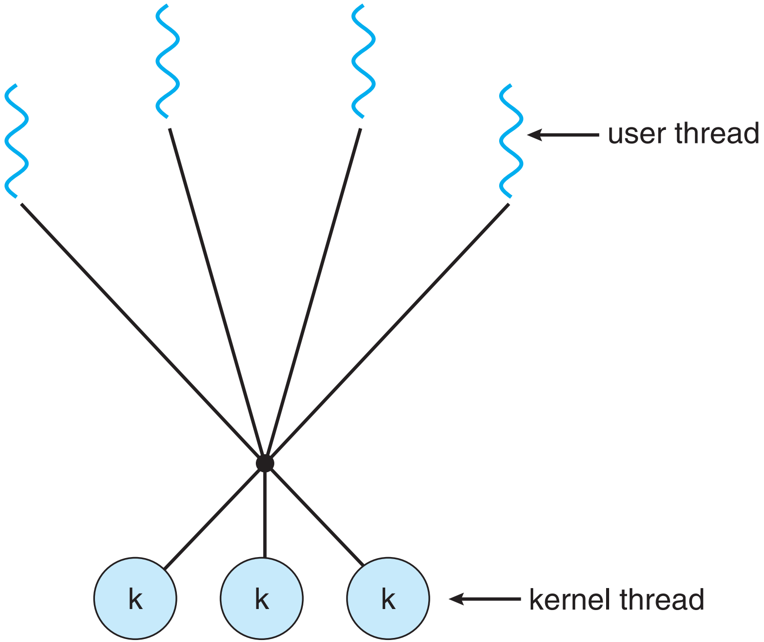
Colored state
is current state

KLT
One-to-one model

- The OS schedule threads and assigns resources to processes
- Risk: creating too many threads
- This is the model followed da Windows and Linux

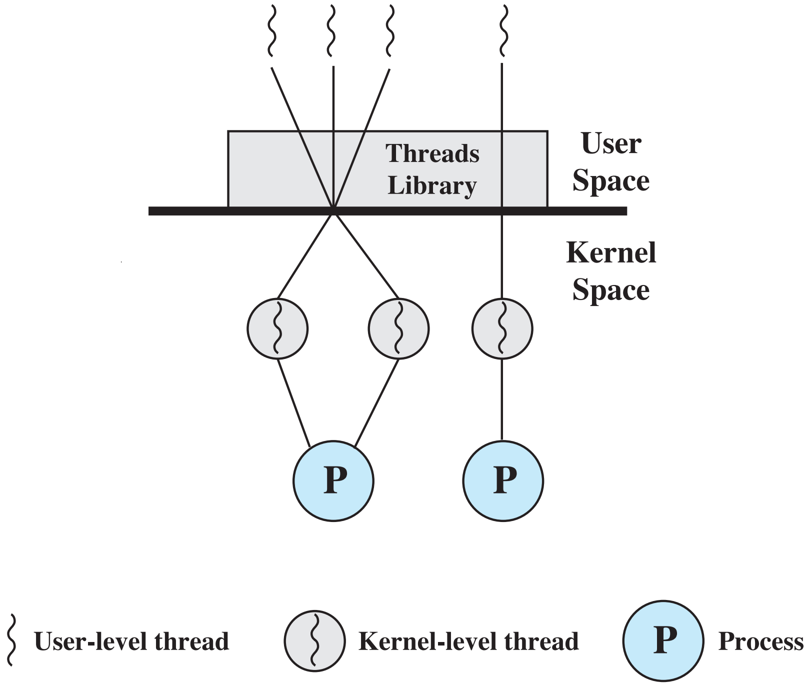


KLT Many-to-many model



The number of kernel threads can be less than or equal to the number of user threads

Combined model ULT and KLT



Thread Libraries

Thread Libraries

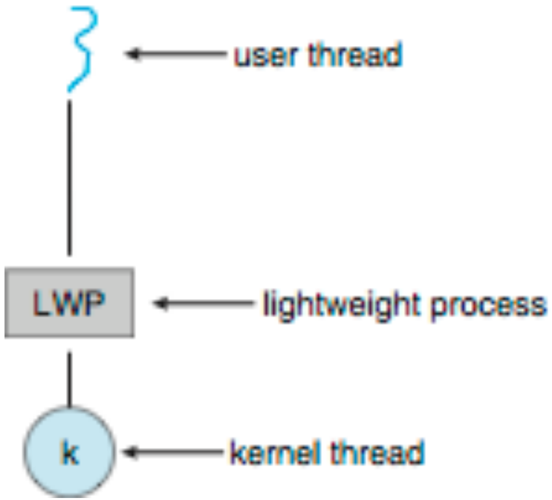
- Multithread programming requires the use of specific APIs
  - User libraries (they do not require any action by the kernel)
  - Kernel libraries
- Example of thread libraries
  - Pthreads - POSIX (user and kernel threads)
  - Win32 (kernel threads)
  - Java (user level) JRE can use the kernel libraries if the host OS supports multithreading

Multithread Issues

Main issues

- What happens to a multithread process whena a thread makes a call to either `fork()` or `exec()` ?
- Which tasks has to be performed when a thread is cancelled?
- Signals are sent to processes and not to individual threads.
- Thread pools to implement the same service for multiple clients
- Permanent association between threads and a subset of data managed by the process
- Communication between kernel and user libraries (LWP and scheduler activation)

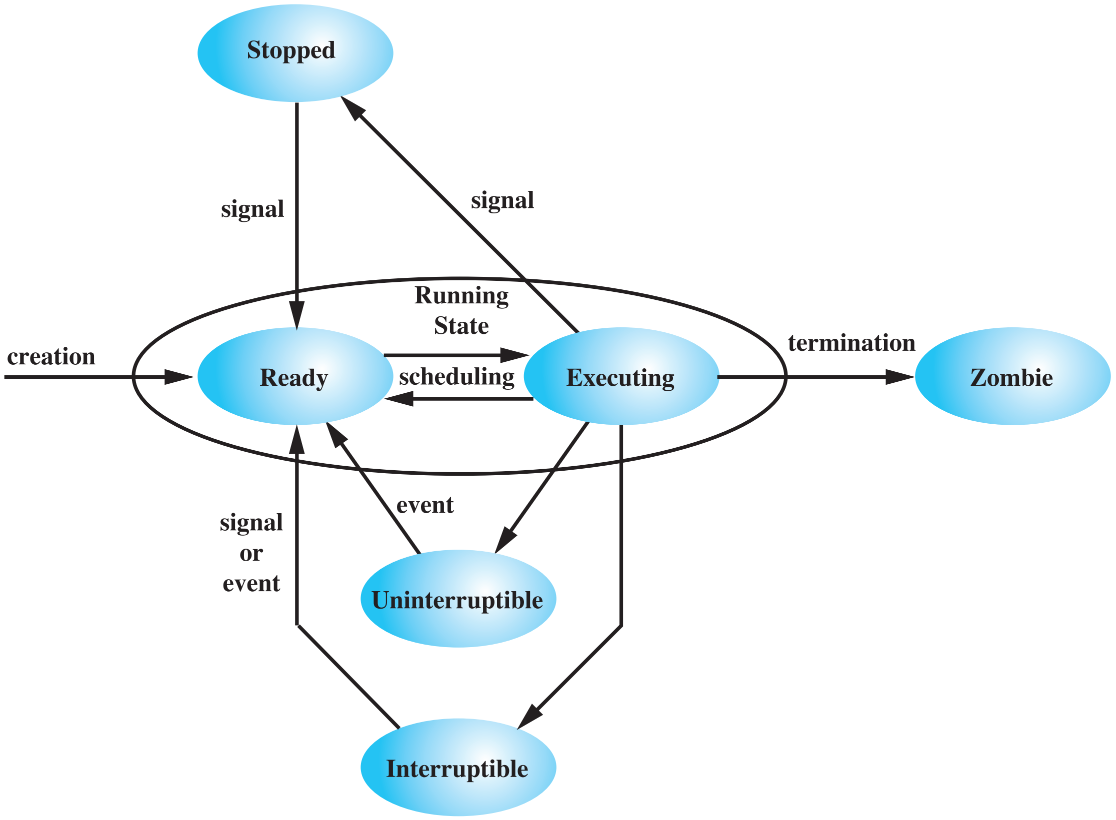
Thread, Processes and scheduling



- Many-to-many model with a third element between user and kernel threads
- LWP - LightWeight Process (introduced in Solaris)
  - This is a data structure managed by the kernel
    - Typically one LWP for each blocking system call
    - max number of LWPs set a system parameter
  - The user library schedule the user threads on the available LWPs

Examples of Thread state diagrams

Linux Process/Thread model



Windows Thread States