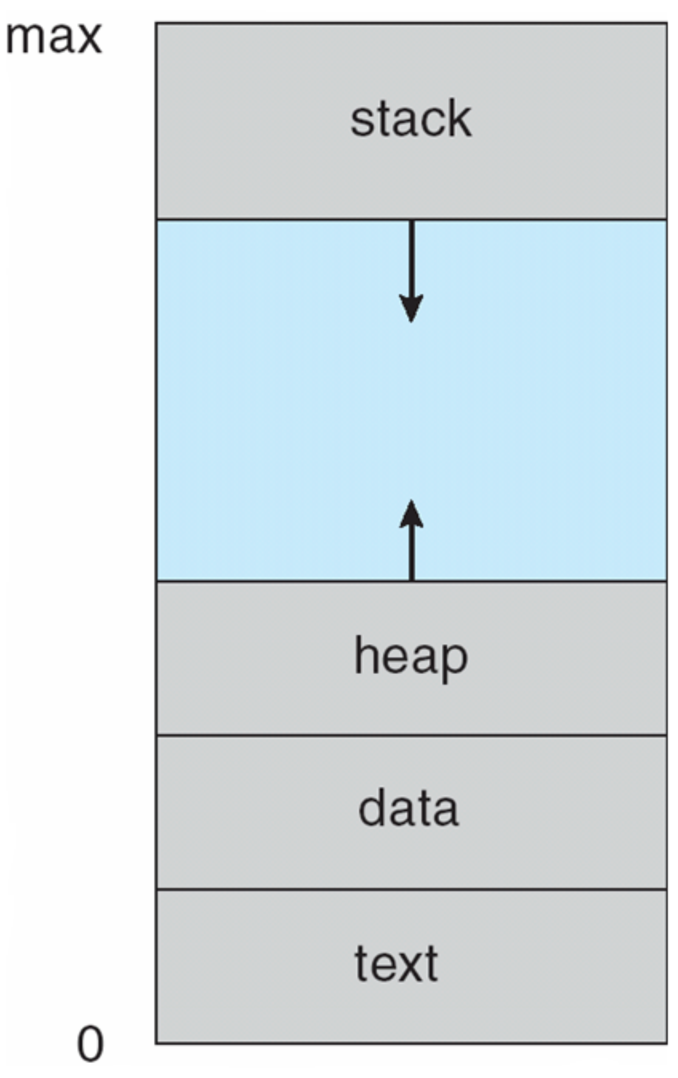
Operating System Review (Chapter 3-5)

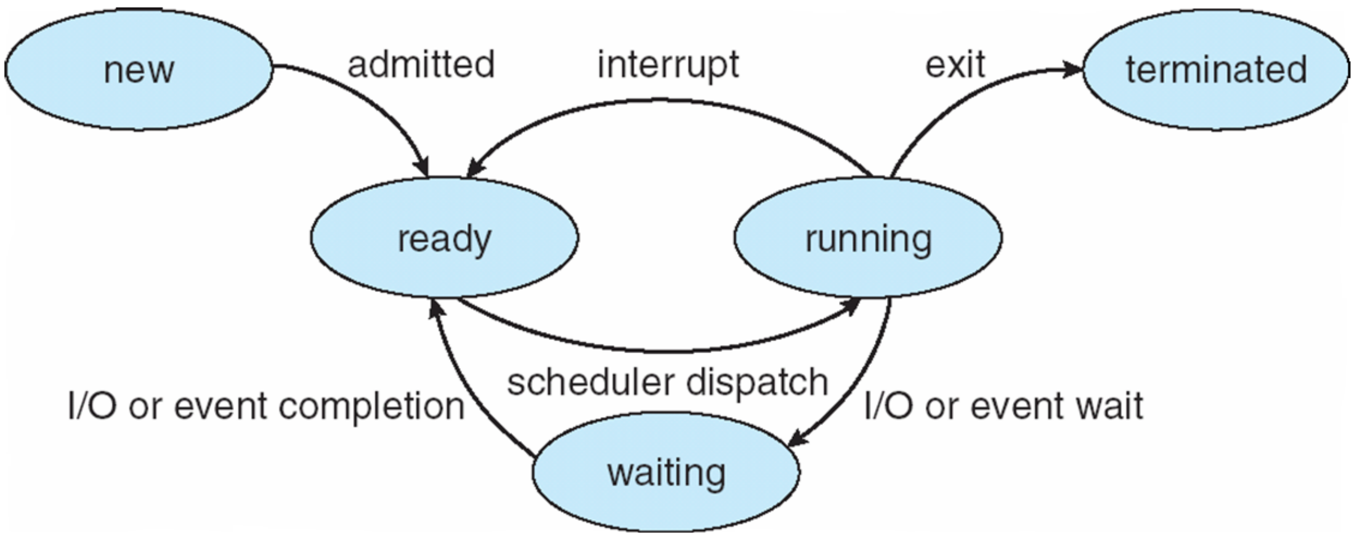
3.1 Process Concept

- What are the most important four components of a process?
- Text section: program code itself
- Stack: temporary data (function parameters, return addresses, local variables)
- Data section: global variables
- Heap: contains memory dynamically allocated during run-time



3.1 Process Concept

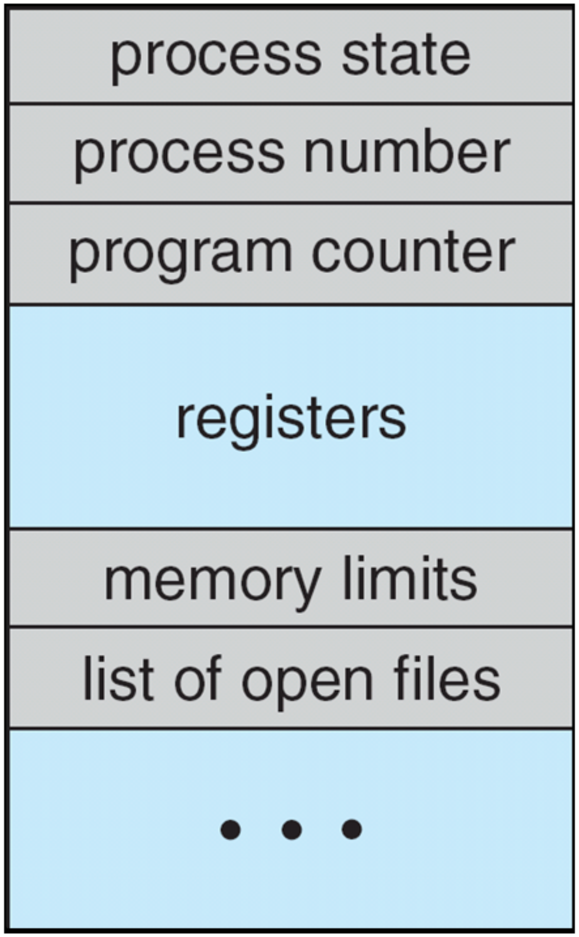
- Provide at least three possible states a process may be in



3.1 Process Concept

- What is a Process Control Block (PCB) ?

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

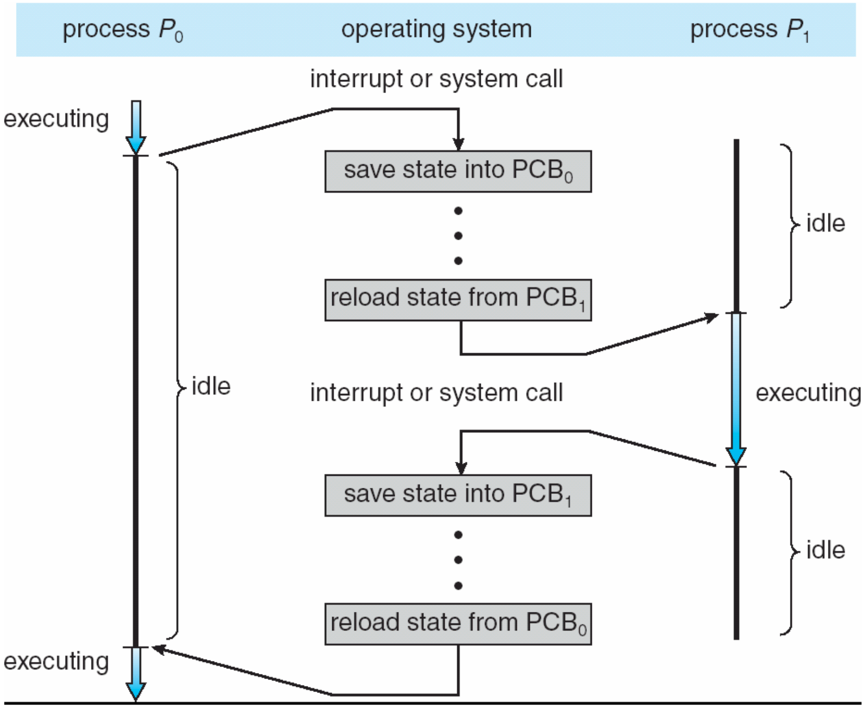


3.1 Process Concept

- List two process types
  - ❖ I/O Bound: spends more time doing I/O than computations, many short CPU bursts
  - ❖ CPU Bound: spends more time doing computations, few very long CPU bursts

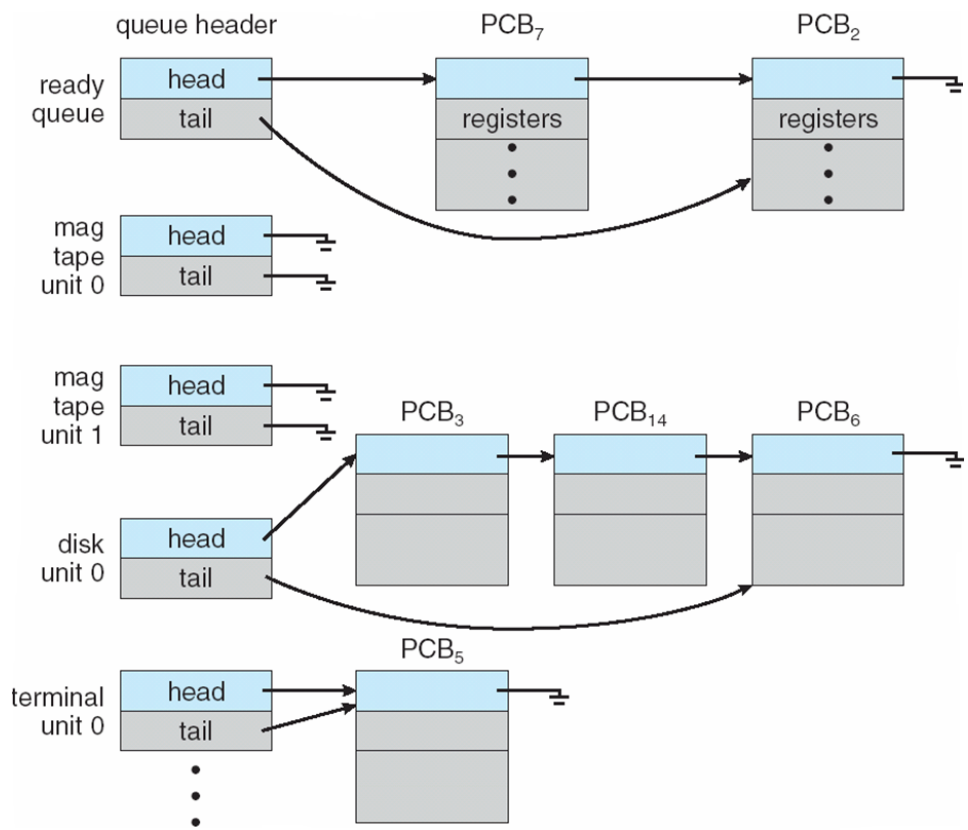
3.2 Process Scheduling

- What is the term that describes saving the state of one process, and restoring the state of another?
  - ❖ When CPU switches to another process, the system must save the state of the old process (to PCB) and load the saved state (from PCB) for the new process via a **context switch**



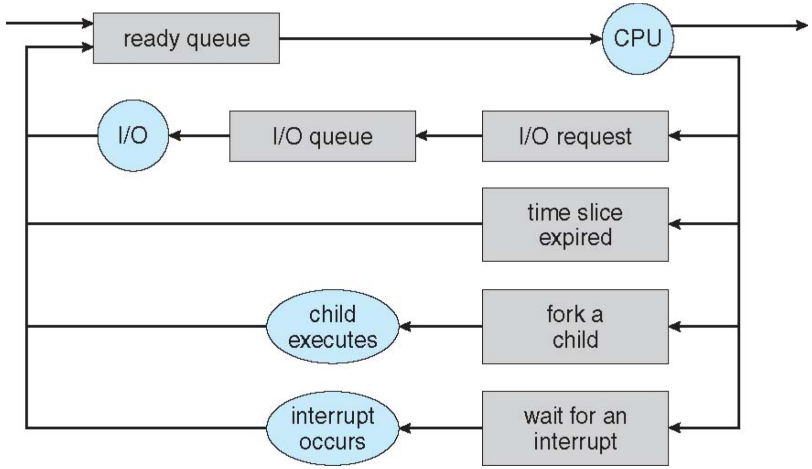
3.2 Process Scheduling

- How many queues for the process scheduler?
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device



3.2 Process Scheduling

- Show the difference between short-term and long-term scheduling?
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue



3.3 Process Creation

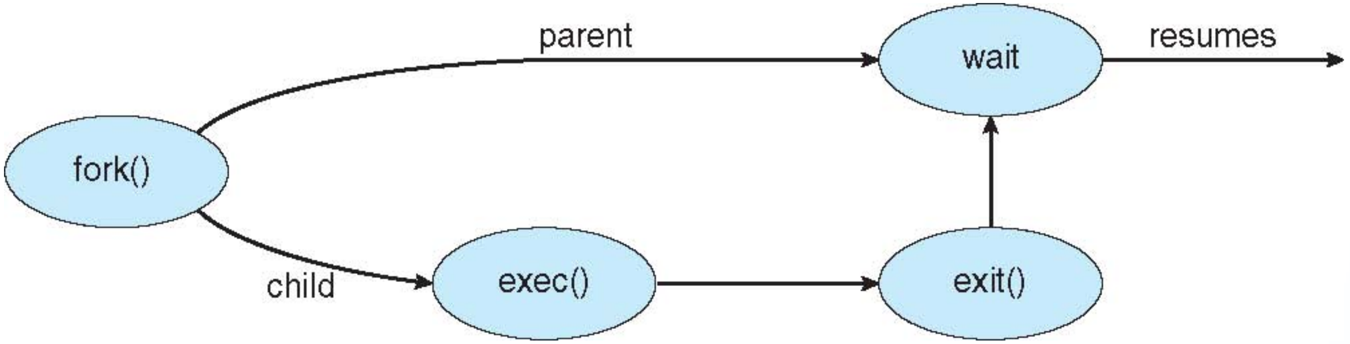
- How to create a new process?
    - **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
    - Generally, process identified and managed via a **process identifier** (**pid**)
    - Resource sharing options
        - Parent and children share all resources
        - Children share subset of parent’s resources
        - Parent and child share no resources

3.3 Process Creation

- What is the difference between fork and exec?

  `fork()` system call creates new process

  `exec()` system call used after a **`fork()`** to replace the process' memory space with a new program

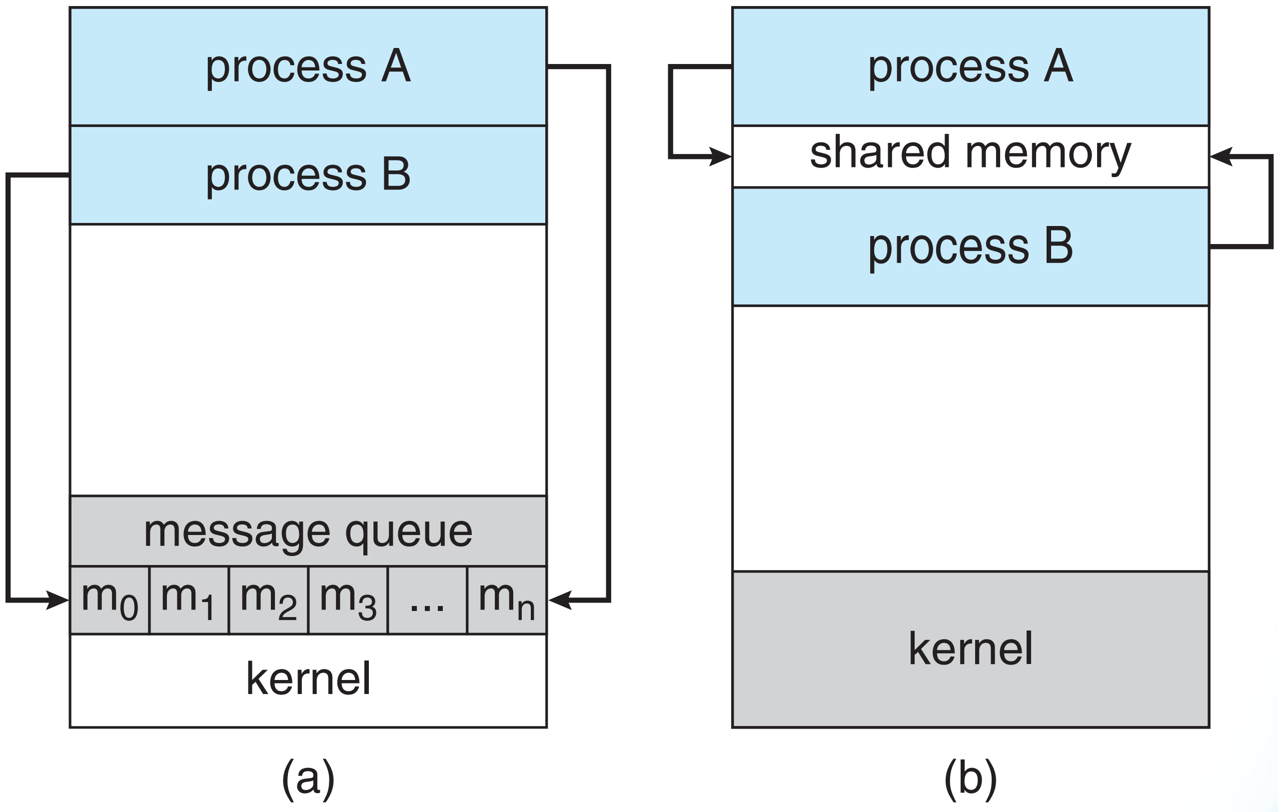


3.3 Process Creation

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       return 1;
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
    }

    return 0;
}
```

3.4 Process Communication

- What are the two models of IPC (**inter process communication**)?
  - **Shared memory**
  - **Message passing**



3.4 Process Communication

- What are the two models of message passing?
  - ❖ Direct Communication
    - **`send`** (*P, message*) – send a message to process P
    - **`receive`**(*Q, message*) – receive a message from process Q
  - ❖ Indirect Communication
    - send(A, message) – send a message to mailbox A
    - receive(A, message) – receive a message from mailbox A

3.4 Process Communication

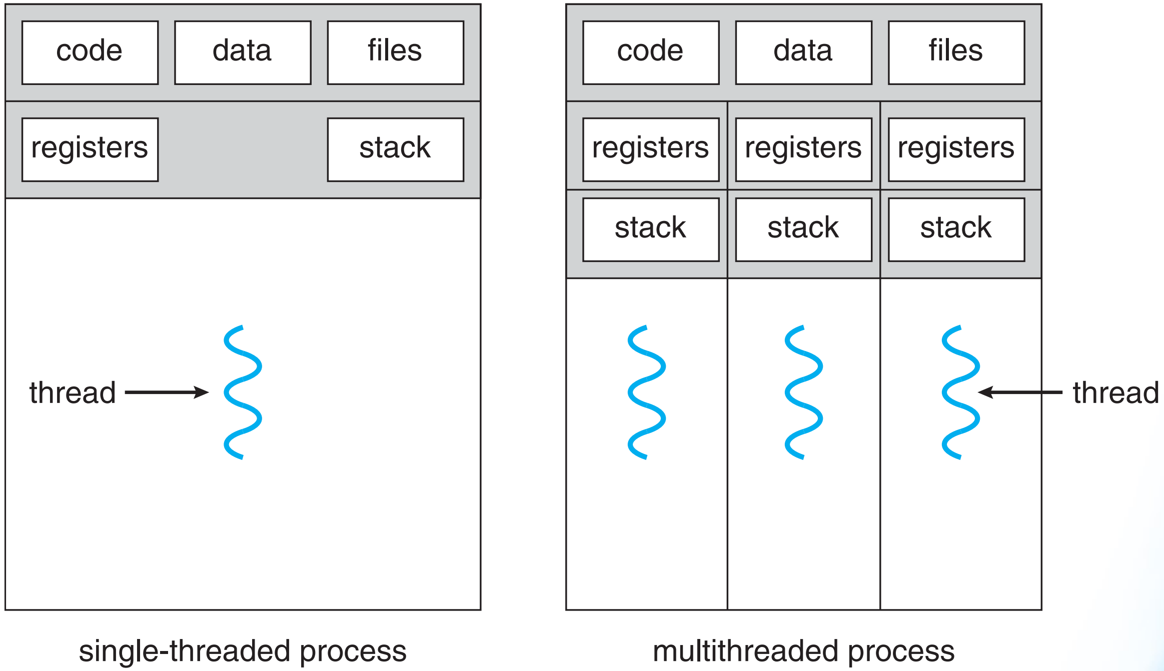
- What is the difference between blocking model and non-blocking model?
  - **Blocking** is considered **synchronous**
    - **Blocking send** -- the sender is blocked until the message is received
    - **Blocking receive** -- the receiver is blocked until a message is available
  - **Non-blocking** is considered **asynchronous**
    - **Non-blocking send** -- the sender sends the message and continue
    - **Non-blocking receive** -- the receiver receives:
      - A valid message, or
      - Null message

3.4 Process Communication

- What are the most popular three methods to communicate between client and server?
  - ❖Sockets
  - ❖Remote Procedure Calls
  - ❖Remote Method Invocation (Java)

4.1 Thread Concept

- What data are shared between threads inside the same process?
  - Code
  - Data
  - Files

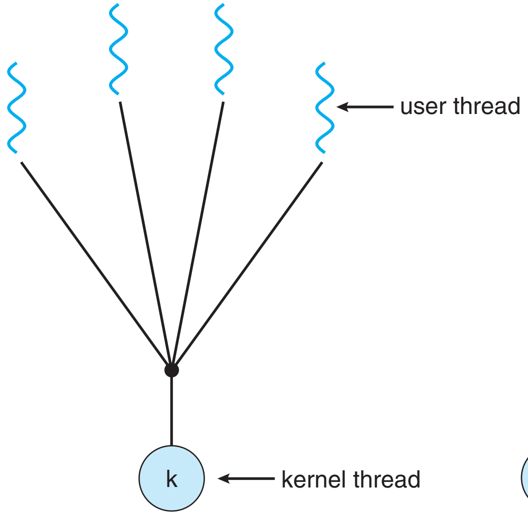


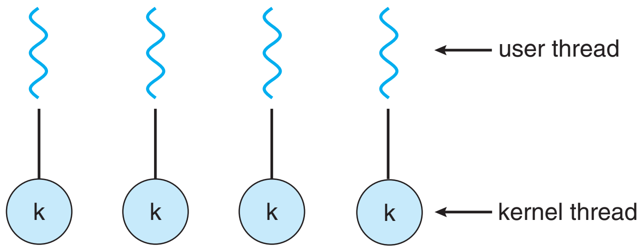
4.1 Thread Concept

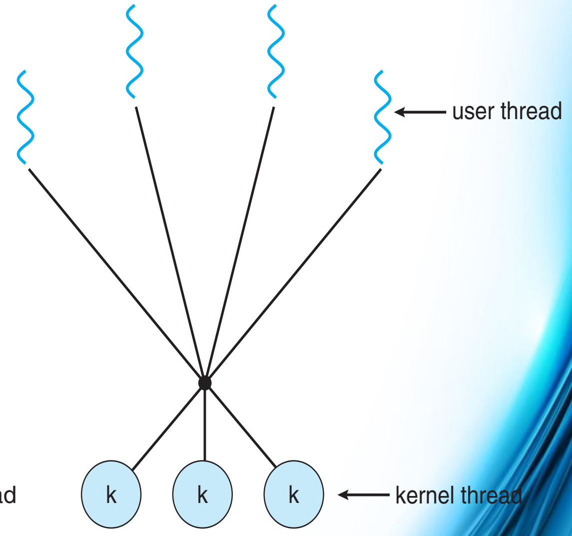
- Provide at least three benefits of multithreaded programming
  - **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
  - **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
  - **Economy –** cheaper than process creation, thread switching lower overhead than context switching
  - **Scalability –** process can take advantage of multiprocessor architectures

4.1 Thread Concept

- What are the three basic mapping relationship between kernel thread and user thread?
  - ❖ Many-to-One
  - ❖ One-to-One
  - ❖ Many-to-Many





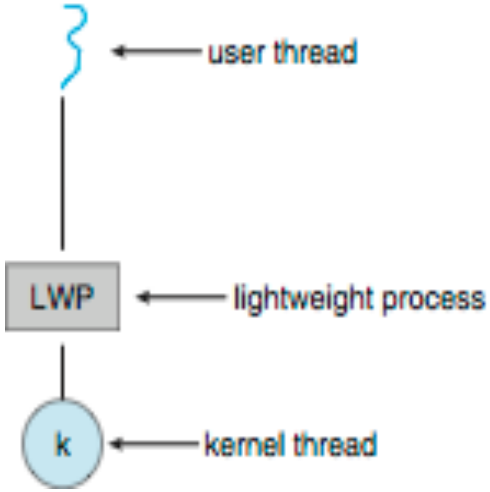


4.1 Thread Concept

- What is the LWP (lightweight process)?

  Typically use an intermediate data structure between user and kernel threads – **lightweight process** (**LWP**)

  - Appears to be a virtual processor on which process can schedule user thread to run
  - Each LWP attached to kernel thread
  - How many LWPs to create?



4.2 Thread Creation

- Pthread

```
#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);
}
```

4.2 Thread Creation

- Windows

```
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }

/* create the thread */
ThreadHandle = CreateThread(
   NULL, /* default security attributes */
   0, /* default stack size */
   Summation, /* thread function */
   &Param, /* parameter to thread function */
   0, /* default creation flags */
   &ThreadId); /* returns the thread identifier */

if (ThreadHandle != NULL) {
    /* now wait for the thread to finish */
   WaitForSingleObject(ThreadHandle,INFINITE);

   /* close the thread handle */
   CloseHandle(ThreadHandle);

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

4.2 Thread Creation

- Java

```
class Sum
{
  private int sum;

  public int getSum() {
   return sum;
  }

  public void setSum(int sum) {
   this.sum = sum;
  }
}

class Summation implements Runnable
{
  private int upper;
  private Sum sumValue;

  public Summation(int upper, Sum sumValue) {
   this.upper = upper;
   this.sumValue = sumValue;
  }

  public void run() {
   int sum = 0;
   for (int i = 0; i <= upper; i++)
     sum += i;
   sumValue.setSum(sum);
  }
}
```

```
public class Driver
{
  public static void main(String[] args) {
   if (args.length > 0) {
    if (Integer.parseInt(args[0]) < 0)
     System.err.println(args[0] + " must be >= 0.");
    else {
     Sum sumObject = new Sum();
     int upper = Integer.parseInt(args[0]);
     Thread thrd = new Thread(new Summation(upper, sumObject));
     thrd.start();
     try {
       thrd.join();
       System.out.println
              ("The sum of "+upper+" is "+sumObject.getSum());
    } catch (InterruptedException ie) { }
    }
   }
   else
    System.err.println("Usage: Summation <integer value>"); }
}
```

4.2 Thread Creation

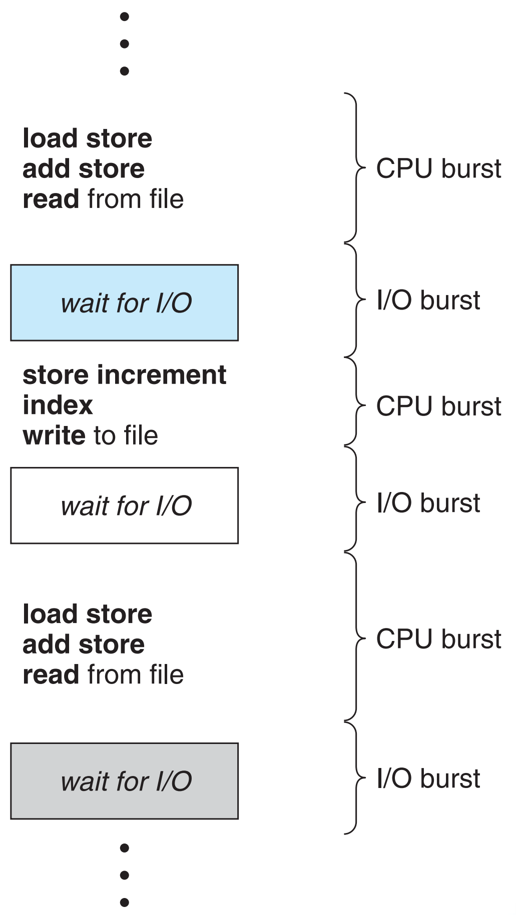
- Why do we need thread pool?
  - Create a number of threads in a pool where they await work
  - Advantages:
    - Usually slightly faster to service a request with an existing thread than create a new thread
    - Allows the number of threads in the application(s) to be bound to the size of the pool
    - Separating task to be performed from mechanics of creating task allows different strategies for running task
      - i.e.Tasks could be scheduled to run periodically

4.2 Thread Creation

- What are the two models of thread cancellation?
  - Asynchronous cancellation terminates the target thread immediately
  - Deferred cancellation allows the target thread to periodically check if it should be canceled

5.1 CPU Scheduling

- What are the two bursts that CPU schedulers are designed around?
  - CPU–I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
  - **CPU burst** followed by **I/O burst**



5.1 CPU Scheduling

- True or False? Under preemptive scheduling, when a process switches from the running to the ready state, it may lose control of the CPU.
    - Switches from running to waiting (nonpreemptive)
    - Switches from running to ready (preemptive)
    - Switches from waiting to ready (preemptive)
    - Terminates (nonpreemptive)

5.2 Scheduling Algorithm

- List at least three different criteria for designing a CPU scheduling algorithm
  - **CPU utilization** – keep the CPU as busy as possible
  - **Throughput** – # of processes that complete their execution per time unit
  - **Turnaround time** – amount of time to execute a particular process
  - **Waiting time** – amount of time a process has been waiting in the ready queue
  - **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

5.2 Scheduling Algorithm

- What scheduling algorithm assigns the CPU to the process that first requested it?
  - First- Come, First-Served (FCFS) Scheduling
- What scheduling algorithm assigns the CPU to the process with the shortest burst?
  - First Job Scheduling (FJS)
- What scheduling algorithm assigns the CPU to a process for only its time slice (or time quantum?)
  - round robin scheduling (RR)

5.2 Scheduling Algorithm

- What scheduling algorithm assigns the CPU to the process with the highest priority?
  - Priority Scheduling

- True or false, Shortest Job First is a specific priority scheduling algorithm?

是的, 优先级= 时间

- True or False? The multilevel feedback queue scheduling algorithm allows processes to migrate between different queues.

对的