
ENSC 351
Processes & Threads

Dr. Matthew Stewart

**Slides for course derived from
Dr. Mohamed Hefeeda's slides
Updated by Dr. Brian Fraser**

Objectives

□ Understand

- ❖ Process concept
- ❖ Process scheduling
- ❖ Creating and terminating processes
- ❖ Interprocess communication
- ❖ Threads vs Processes

Process Concept

❑ Process is..

- ❖ Process execution must progress in sequential fashion
- ❖ A program may exist on the hard drive, but is not a process until being executed (usually from memory)

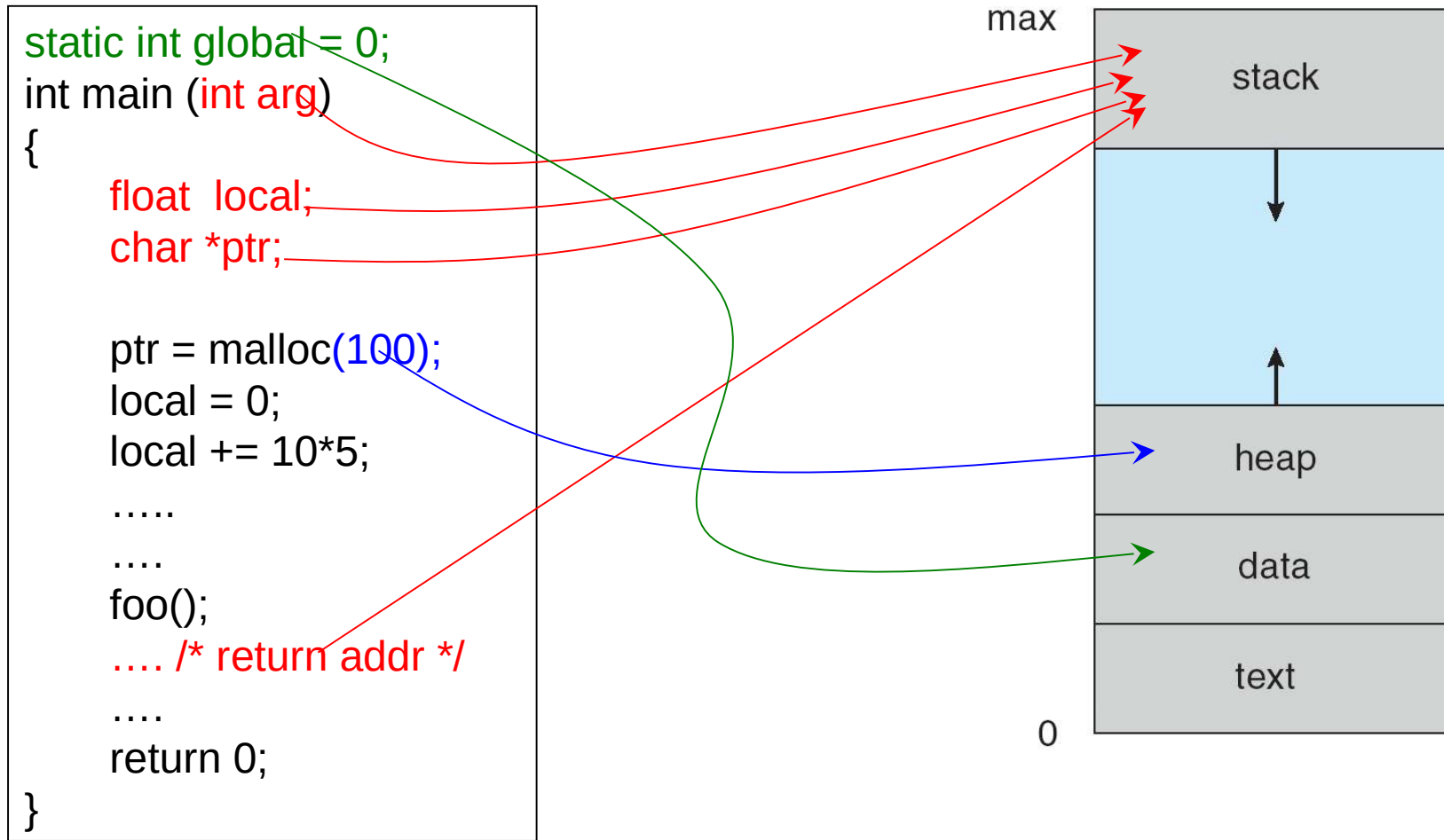
❑ Note:

- ❖ Terms.. are interchangeable

❑ A process includes:

- ❖ program counter
- ❖ stack pointer
- ❖ data section (memory)
- ❖ code section (memory)

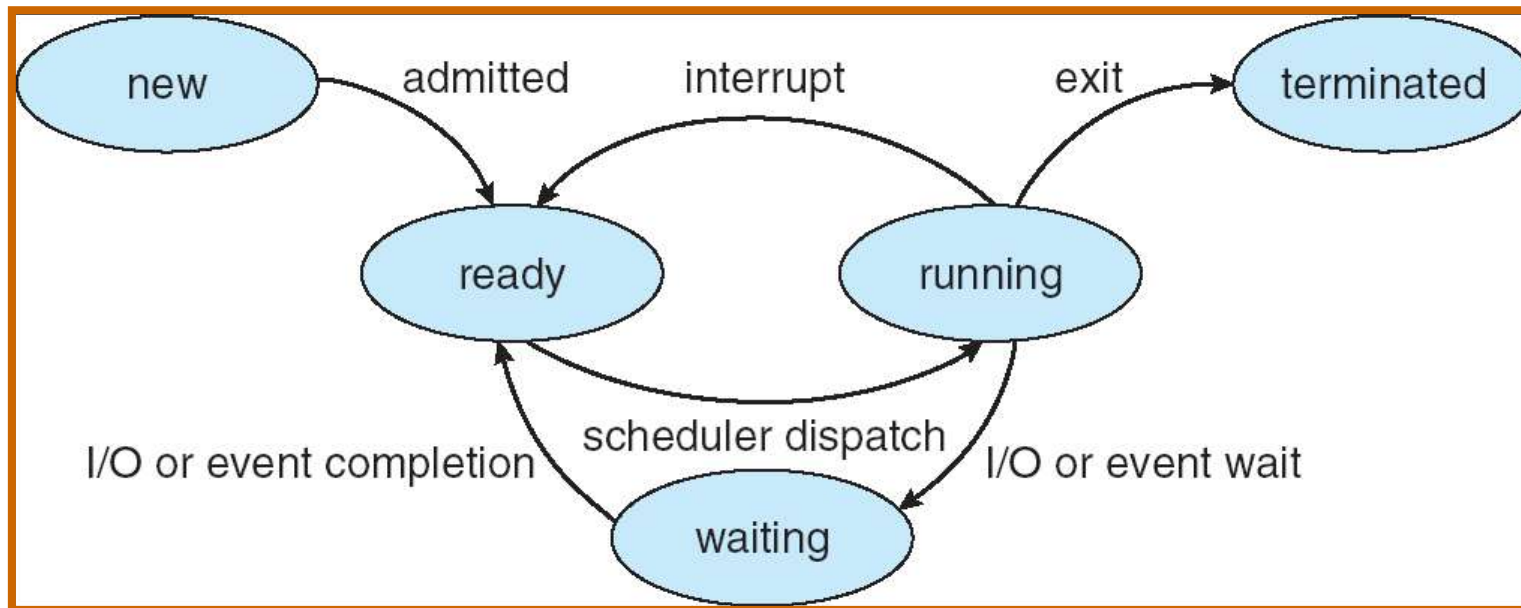
Process in Memory



Process State

❑ As a process executes,..

- ❖ **new**: just created
- ❖ instructions are being executed
- ❖ process is waiting for some event to occur
- ❖ process is waiting for CPU
- ❖ **terminated**: process has finished execution



..

(PCB)

❑ OS maintains info about process in PCB

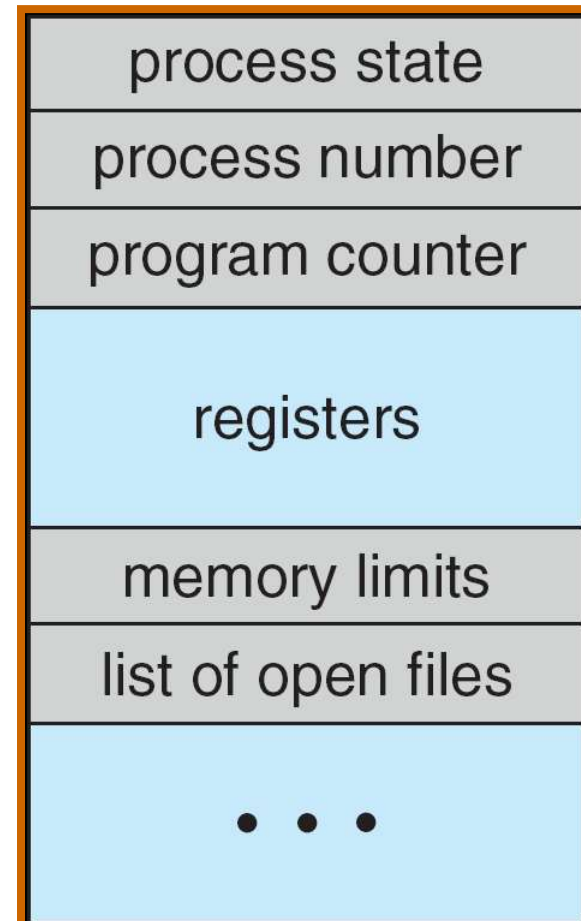
- ❖ Process state
- ❖ Program counter
- ❖ CPU registers
- ❖ CPU scheduling info
- ❖ Memory-management info
- ❖ Accounting info
- ❖ I/O status info

❑ PCB used to..

- ❖ E.g., to switch CPU from one process to another

❑ Typically, a large C structure in kernel

- ❖ Linux: `struct task_struct`

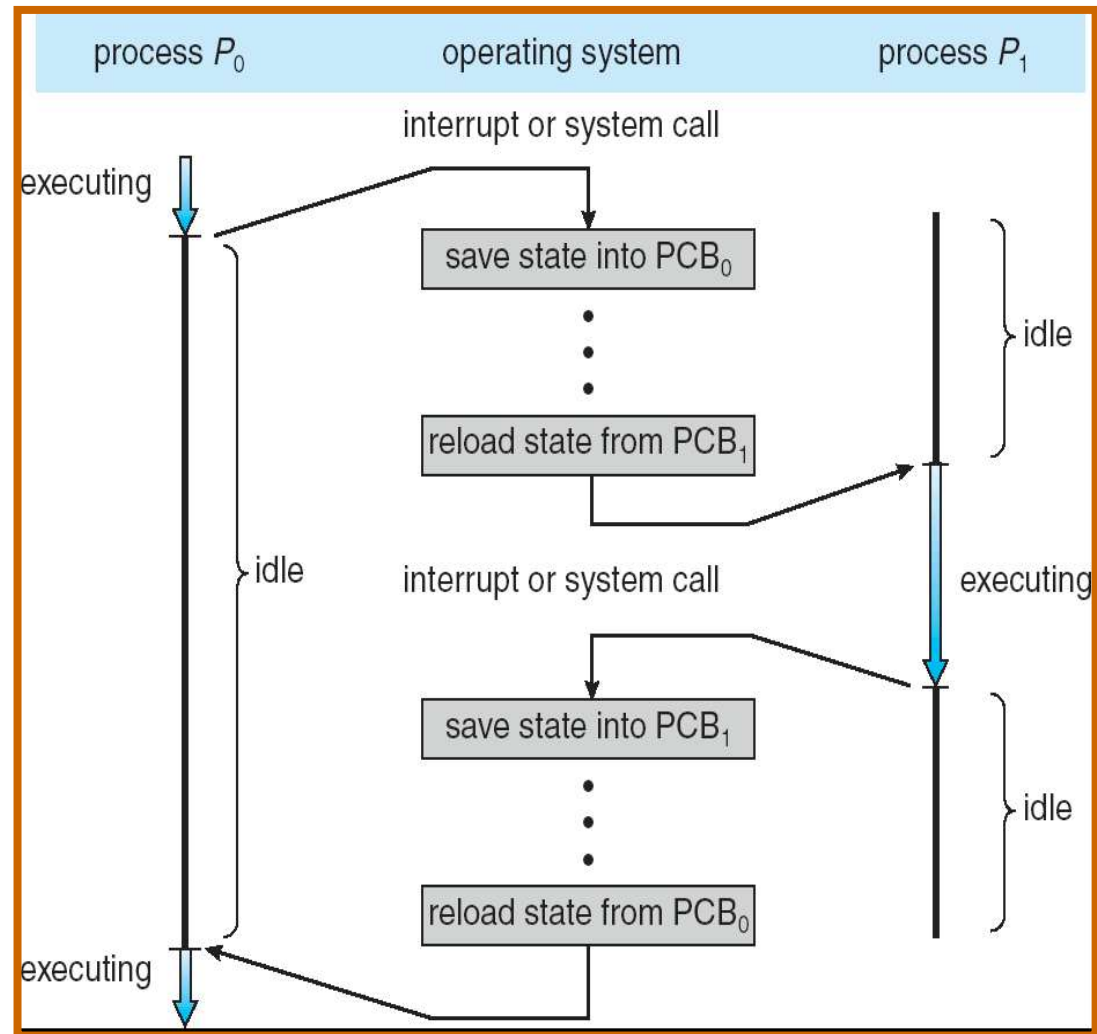


CPU Switch From Process to Process

❑ When switching from P_0 to P_1 kernel will:

- ❖ Save **state** of P_0 in PCB_0 (in memory)
- ❖ Load **state** of P_1 from PCB_1 into registers

❑ **State** = values of the..



CPU Switch From Process to Process cont'd

- ❑ Switching between processes is called a ..
- ❑ Context-switch time is..
no useful work is done
- ❑ Switching time depends on hardware support
 - ❖ Some systems (Sun UltraSPARC) provide multiple register sets → very fast switching (just change a pointer)
 - ❖ Typical systems, few milliseconds for switching

Job Types

□ Jobs (Processes) can be described as either:

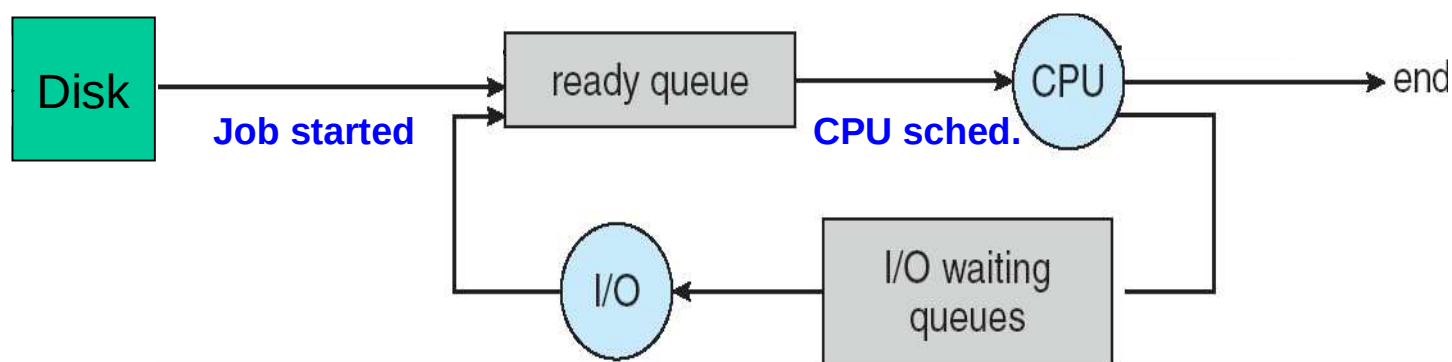


- spends more time doing I/O than computations, many short CPU bursts
- Often characteristic of interactive programs
- Example: GUI, word processor, IDE



- spends more time doing computations; long CPU bursts
- Example: factoring a large prime (cryptography)

Scheduling: The Big Picture (cont'd)



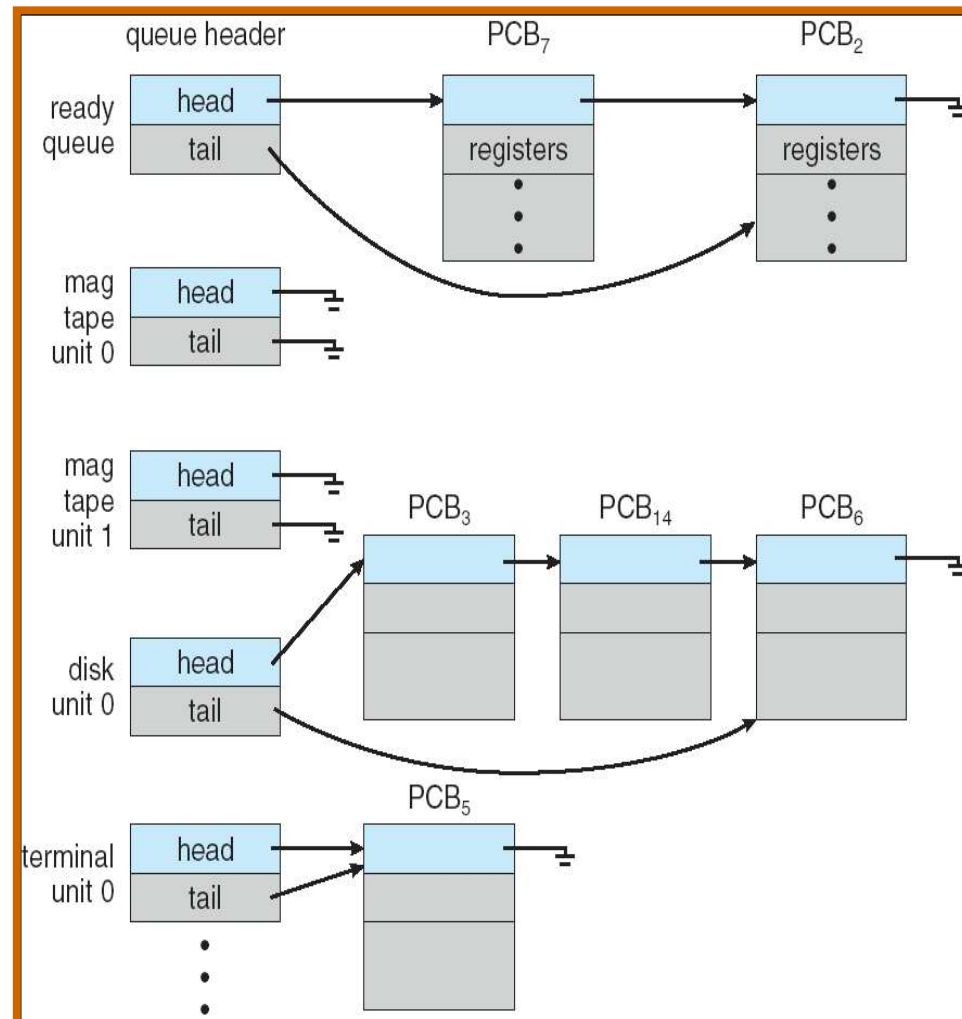
Schedulers (cont'd)

❑ **Short-term scheduler** (or CPU scheduler)

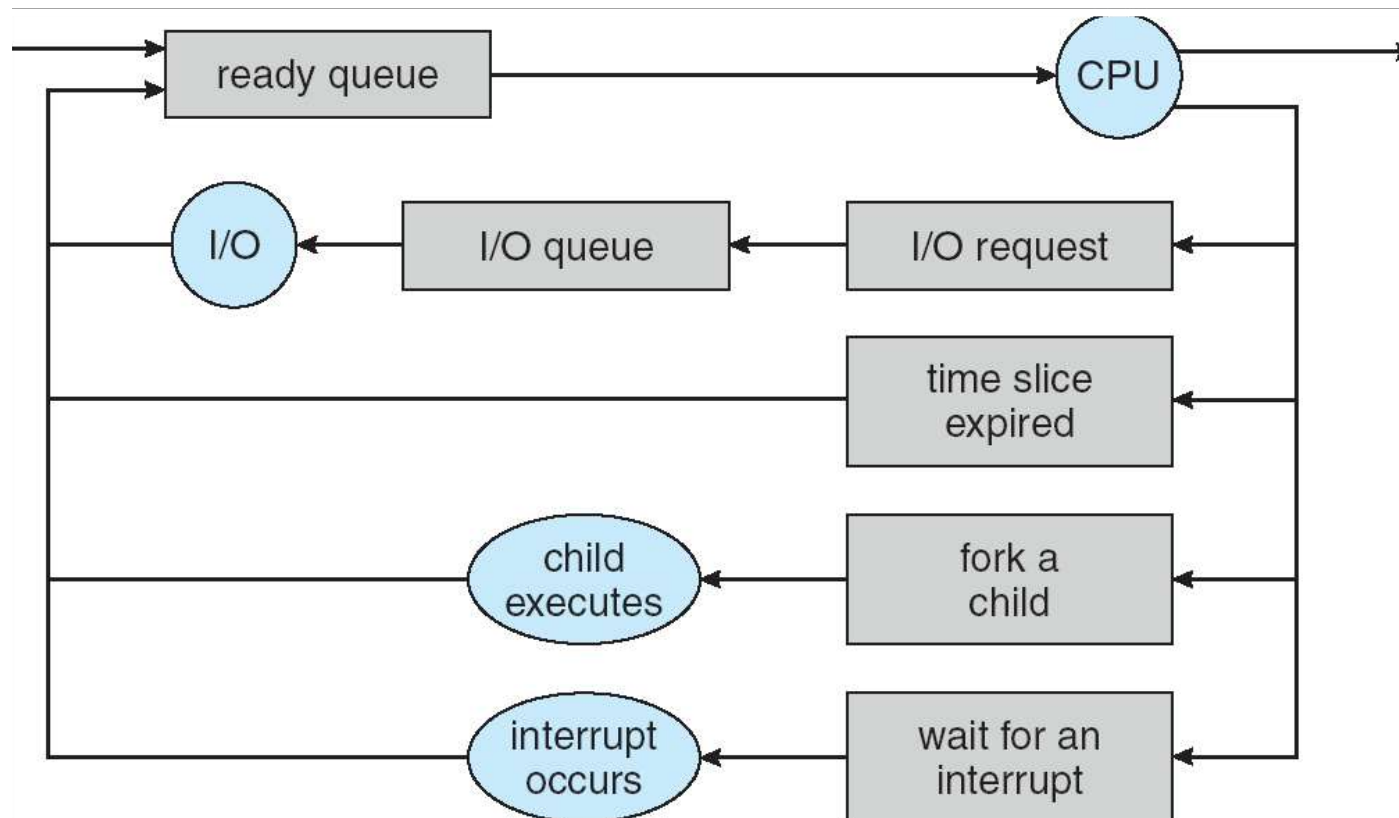
- ❖ selects which process should be..
and allocates CPU to it
- ❖ Short-term scheduler is invoked..
(milliseconds)
 - So must..

Scheduling Queues

- ❑ Processes migrate among various queues
- ❑ ..
set of all processes in the system
- ❑ ..
set of all processes residing in main memory, ready and waiting to execute
- ❑ ..
set of processes waiting for a specific I/O device

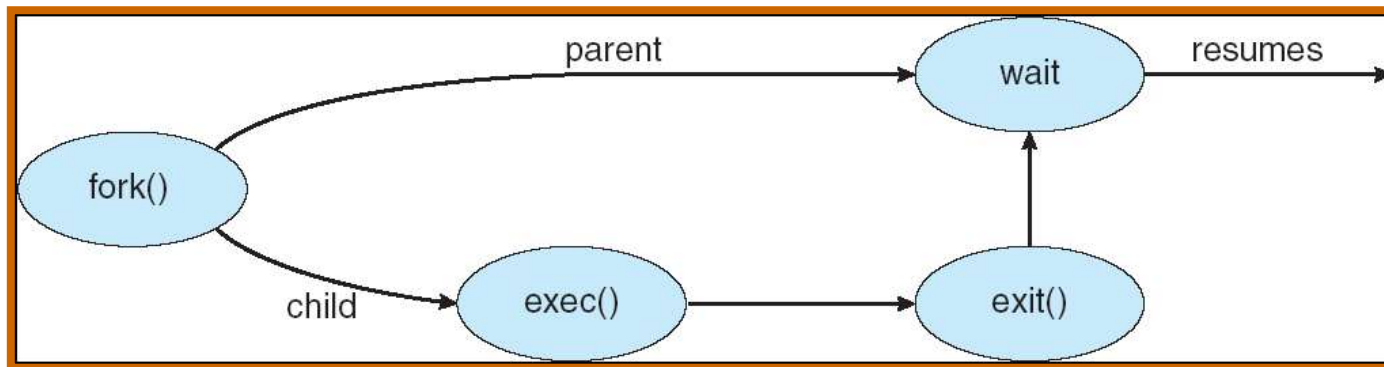


Process Lifetime



Process Creation: Unix Example

- ❑ Process creates another process (child) by using **fork** system call
 - ❖ Child is..
 - ❖ Typically, child loads another program into its address space using **exec** system call
 - ❖ Parent waits for its children to terminate



C Program Forking Separate Process

```
int main()
{
    /* fork another process */
    pid_t pid = fork();

    if (pid < 0) { /* error occurred */
        fprintf (stderr, "fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp ("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for child to complete */
        wait (NULL);
        printf ("Child %d Completed", pid);
        exit(0);
    }
}
```

Fork returns:

< 0:

0:

> 0:

Replace child with
new program.

Tree of processes on BeagleBone Green

```
debian@BeagleBone:~$ pstree -l
systemd--3*[agetty]
        --avahi-daemon--avahi-daemon
        --cron
        --dbus-daemon
        --nginx--nginx
        --node-red--10*[{node-red}]
        --rpcbind
        --rsyslogd--3*[{rsyslogd}]
        --sshd--sshd--sshd--bash--pstree
        --systemd--(sd-pam)
        --systemd-journal
        --systemd-logind
        --systemd-network
        --systemd-resolve
        --systemd-timesyn--{systemd-timesyn}
        --systemd-udev
        --wpa_supplicant
```


Process Termination (Linux)

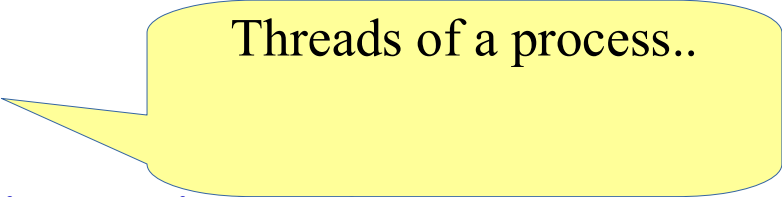
- ❑ Normal termination:..
 - Asks OS to delete the current process (itself)
 - ❖ Last statement a process executes
 - ❖ Process' resources are de-allocated by OS
 - ❖ Exit code (int) available to parent process via..
- ❑ Abnormal termination:..
- ❑ Terminate child process:..
 - ❖ Useful if:
 - Child has exceeded allocated resources
 - Task assigned to child is no longer required

Threads



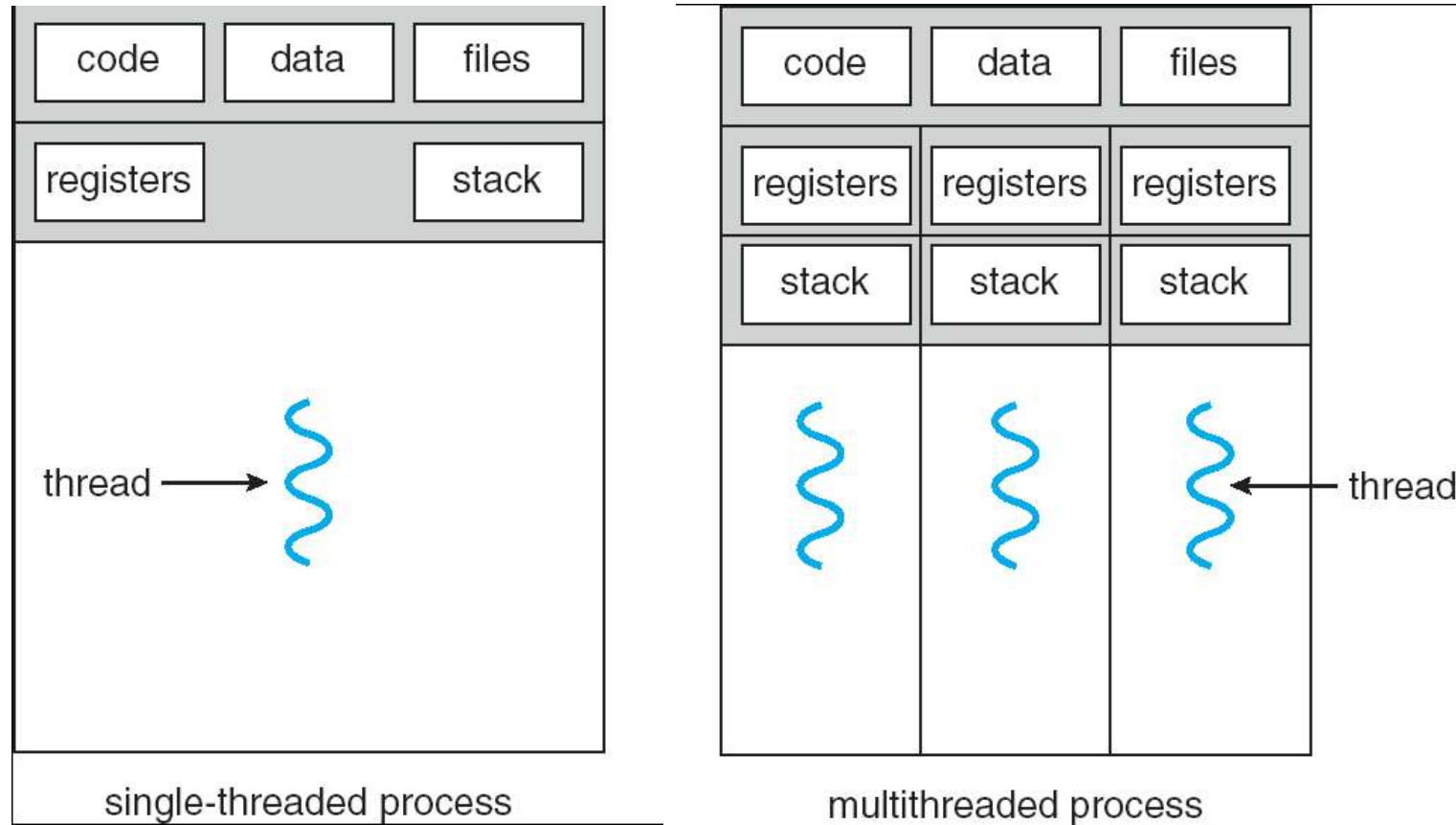
Thread Definitions

- ❑ Thread is a basic unit of CPU utilization
 - ❖ A sequence of instructions enclosed in a function which..
- ❑ Process is a program in execution
 - ❖ A process is composed of..
- ❑ Each thread has a thread control block (TCB)
 - ❖ Program counter
 - ❖ Register set, and
 - ❖ Stack
- ❑ Threads of the same process share
 - ❖ Code section
 - ❖ Data section
 - ❖ OS resources such as open files and signals



Threads of a process..

Single and Multithreaded Processes



Why Multithreading?

- ❑ **Responsiveness**: one thread for ..
- ❑ **Resource Sharing**: similar requests handled by the same code and use same files/resources
- ❑ **Economy**: threads are much cheaper to create/delete than..
- ❑ **Utilization of multiprocessors**: single threaded-process can NOT make use of multiple processors
- ❑ **Examples of multithreaded applications?**
 - ❖ Web browsers: parallel downloads
 - ❖ Web servers: handle multiple concurrent clients
 - ❖ Word processors: spell check in the background
 - ❖ Many others ...

Cooperating Processes

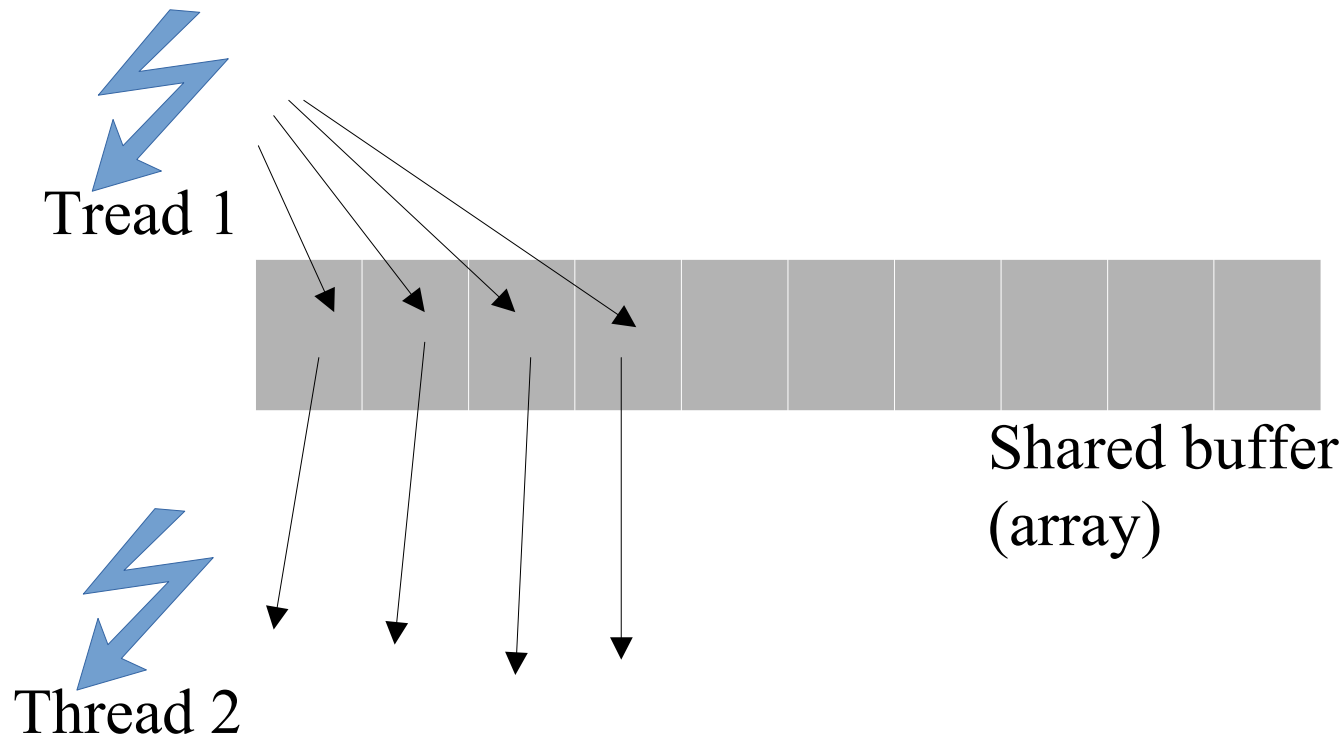
- ❑ **Cooperating** process can affect the execution of each other

- ❑ **Why processes cooperate?**
 - ❖ Information sharing
 - ❖ Computation speed-up
 - ❖ Modularity, Convenience

- ❑ **Interprocess Communication (IPC) methods**
 - ❖ Shared memory
 - ❖ Message passing

Threads & Shared Memory

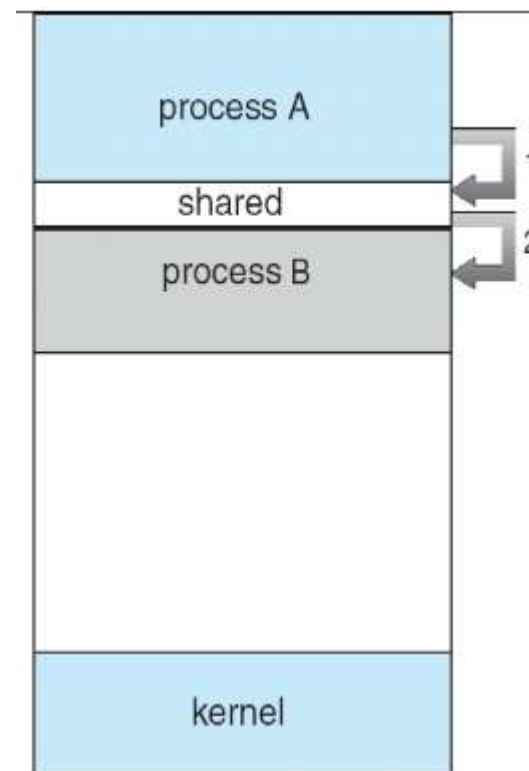
- ❑ Threads inside a process share a memory space
 - ❖ Therefore, they can just use pointers to reference shared memory



IPC: Shared Memory

❑ Processes communicate by creating a shared place in memory

- ❖ One process creates a shared memory— **shmget()**
- ❖ Other processes attach shared memory to their own address space — **shmat()**
- ❖ Then, shared memory is treated as regular memory
- ❖ Synchronization is needed to prevent concurrent access to shared memory (conflicts)



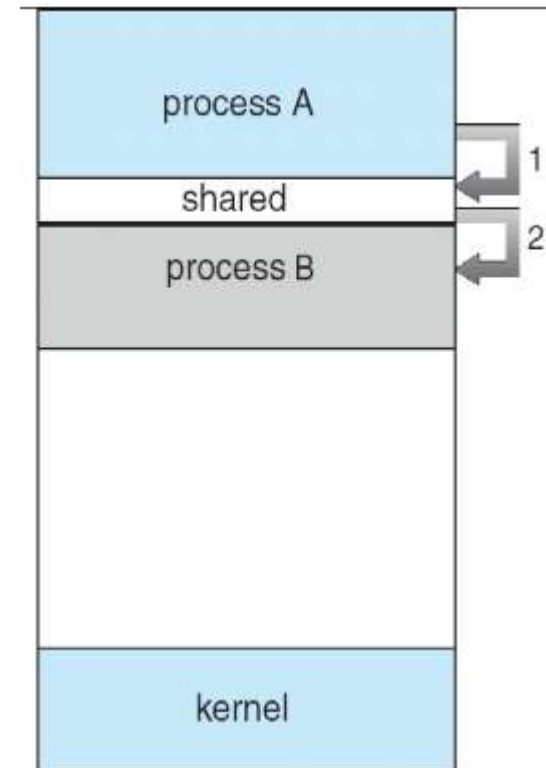
IPC: Shared Memory

❑ Pros

- ❖ ..
(use at memory speed)
- ❖ Convenient to programmers
(just regular memory)

❑ Cons

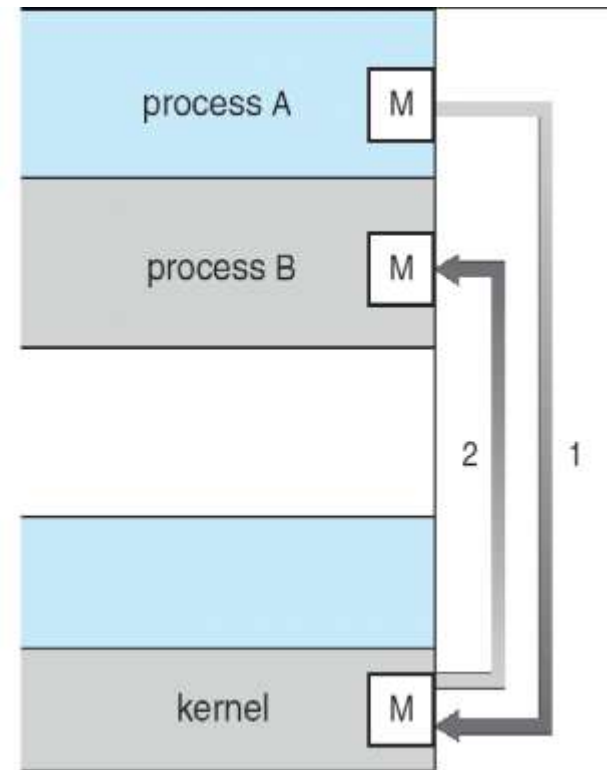
- ❖ Need to..
(tricky for distributed systems)



IPC: Message Passing

□ If processes (or threads) P and Q wish to communicate, they need to:

- ❖ establish a **communication**
- ❖ exchange messages via a pipe:
 - **send** (*message*) – message size fixed or variable
 - **receive** (*message*)



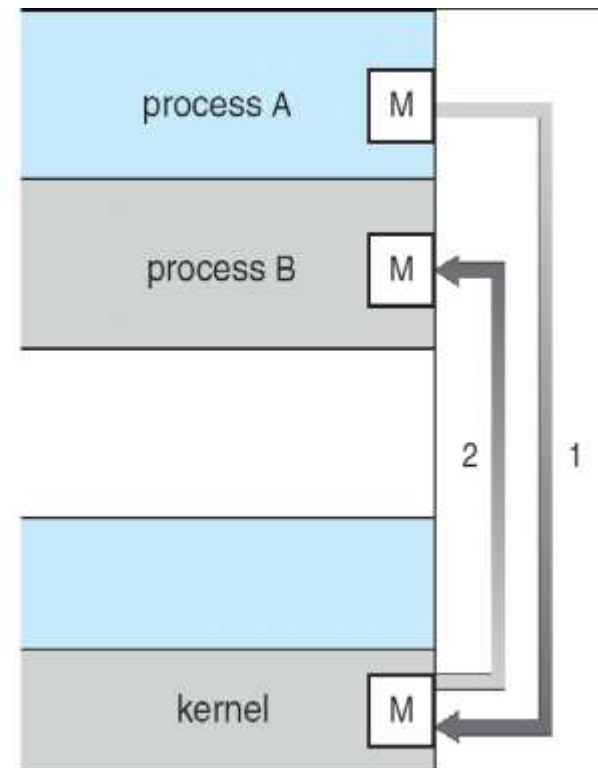
IPC: Message Passing

❑ Pros

- ❖ No conflict:
easy to exchange messages
especially in distributed systems

❑ Cons

- ❖ Overhead (message headers)
- ❖ ..
 - Sender must prepare messages;
receiver must process them.
 - ..
sender → kernel → receiver
(several system calls)



IPC: Message Passing (cont'd)

- ❑ **Synchronization:** message passing is either
 - ❖ ..
 - `send ()` has sender block until message is received
 - `receive ()` has receiver block until message is available
 - ❖ ..
 - `send ()` has sender send message and continue
 - `receive ()` has receiver receive a valid message or null
- ❑ **Buffering:** Queue of messages attached to communication channel
 - ❖ **Zero capacity** – Sender must wait for receiver (rendezvous)
 - ❖ **Bounded capacity** – Sender must wait if link full
 - ❖ **Unbounded capacity** – Sender never waits

Example: Linux Pipes

❑ Pipe:



- ❖ Good for inter-thread and inter-process communication.

❑ Needed Functions:

- ❖ **pipe()** to create file descriptors for read and write ends of pipe.
- ❖ **fdopen()** to open the pipe (from descriptor)
- ❖ **fprintf()** to write (or other functions)
- ❖ **fgets()** to read [blocking] (or other functions)
- ❖ **close()** to close the file descriptor.

Example: Linux Pipes code

```
int fds[2];           // File descriptors for two ends of pipe
pipe (fds);           // Create a pipe.

// Writer: Convert the write file descriptor to a FILE object
FILE* streamW = fdopen (fds[1], "w");
fprintf (streamW, "Hello World of Pipes!\n");
fflush (streamW);
close (fds[1]);
```

This possibly in different process/thread:

```
// Reader: Convert read file descriptor to a FILE object.
FILE* streamR = fdopen (fds[0], "r");

// Read until end of the stream.
char buffer[1024];
while (!feof (streamR) && !ferror (streamR)
      && fgets (buffer, sizeof (buffer), streamR) != NULL) {
    printf("%s", buffer);
}
close (fds[0]);
```

Summary

- ❑ **A process is a program in execution**
 - ❖ OS maintains process info in PCB
 - ❖ Process State diagram
 - ❖ Creating and terminating processes (fork)
- ❑ **Process scheduling**
 - ❖ Long-, short-, and medium-term schedulers
 - ❖ Scheduling queues
- ❑ **Interprocess communication**
 - ❖ Shared memory
 - ❖ Message passing
- ❑ **Threads**
 - ❖ Share memory between threads of a process
 - ❖ Each thread executes independently