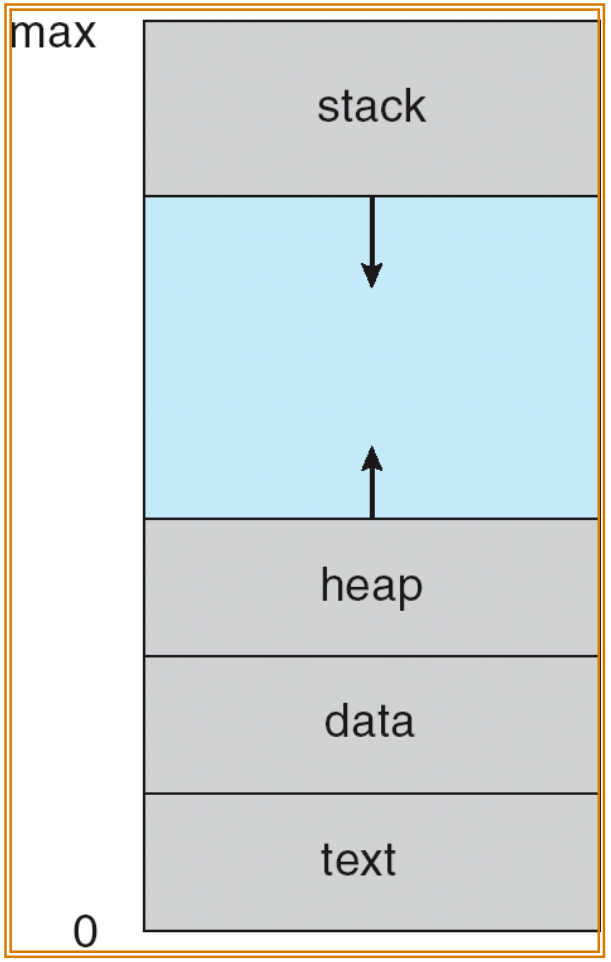
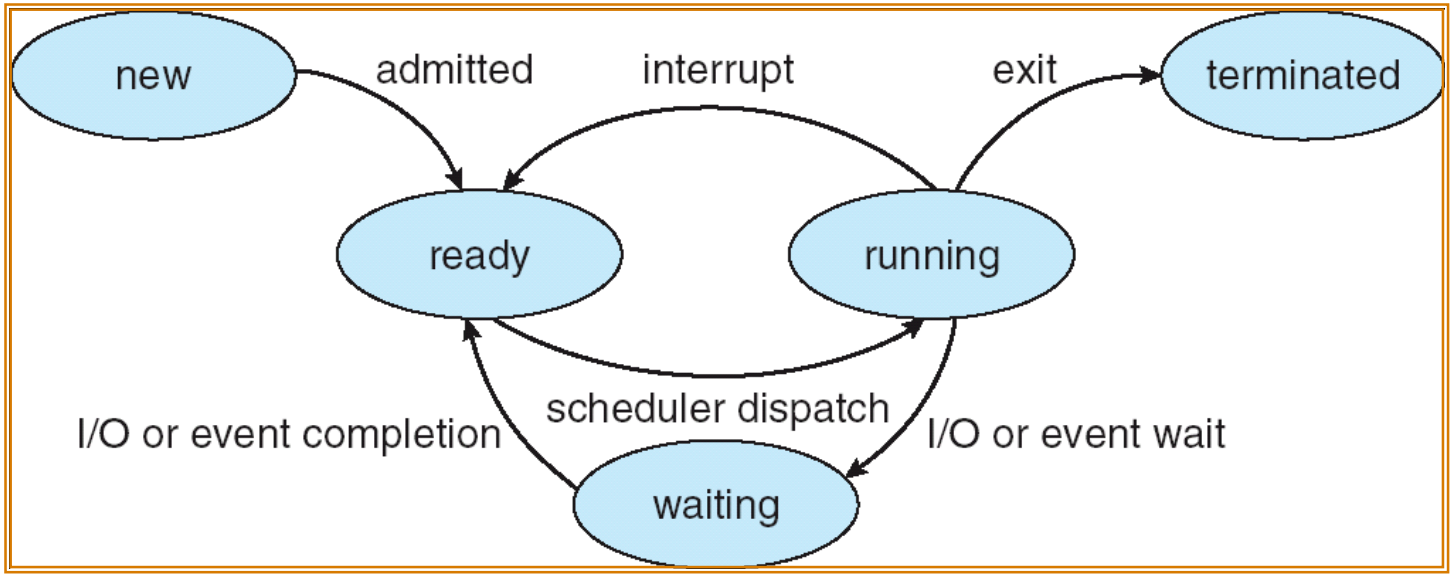
Chapter 3: Process Concept

- Process – a program (like MS Word) in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter, register
  - Stack (temporary values, function parameters) , heap (memory allocations)
  - data section (global valuables), text section (code)



Process State

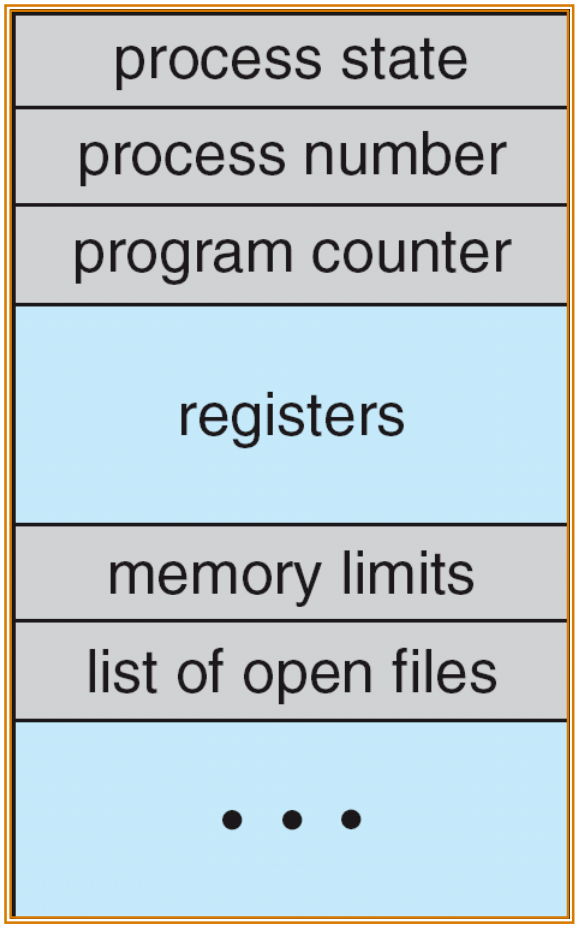
- As a process executes, it changes state
  - new: The process is being created
  - running: Instructions are being executed
  - waiting: The process is waiting for some event to occur
  - ready: process is waiting to be assigned to a processor
  - terminated: The process has finished execution



Process Control Block (PCB)

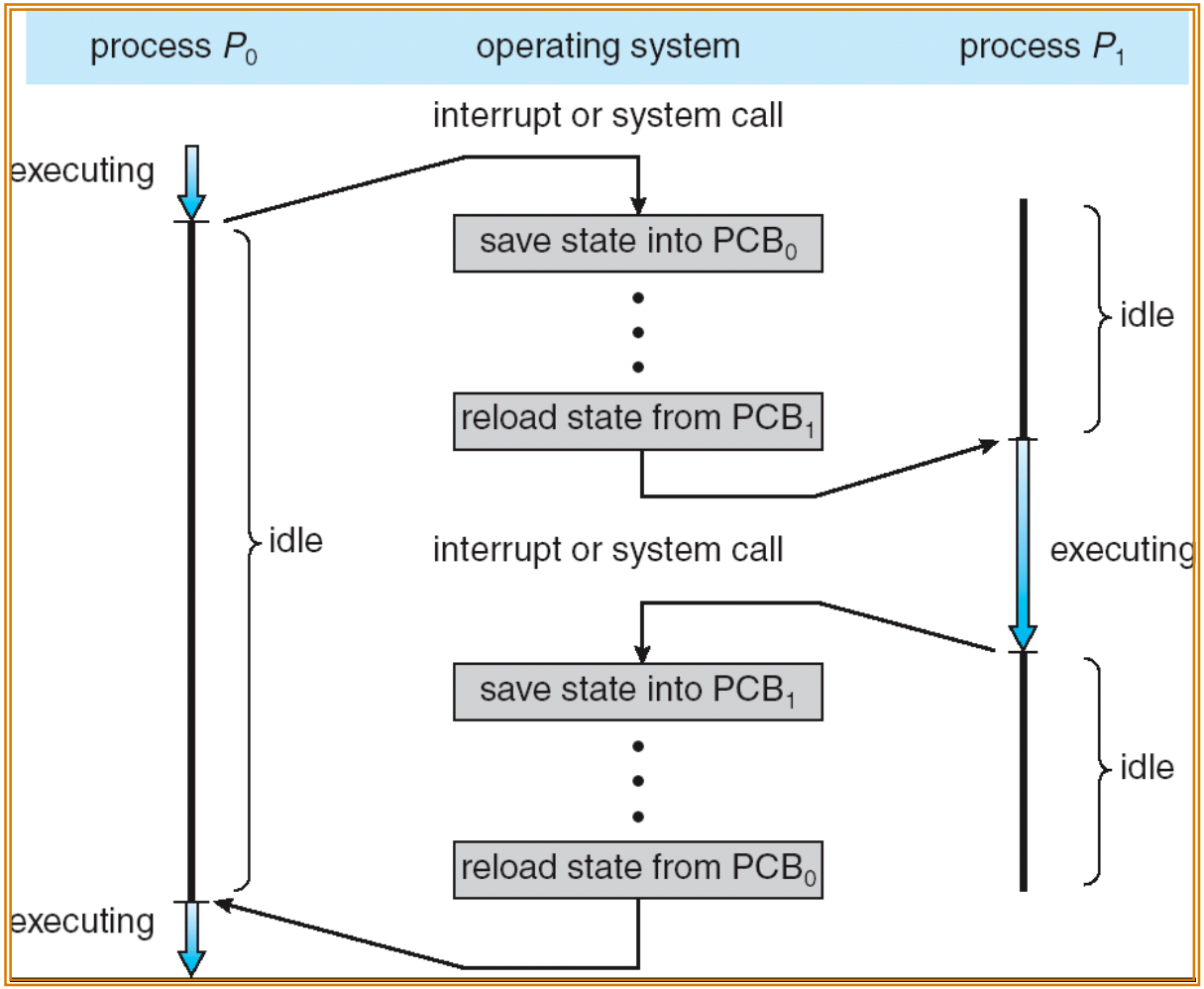
Information associated with each process and maintained by the operating system

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

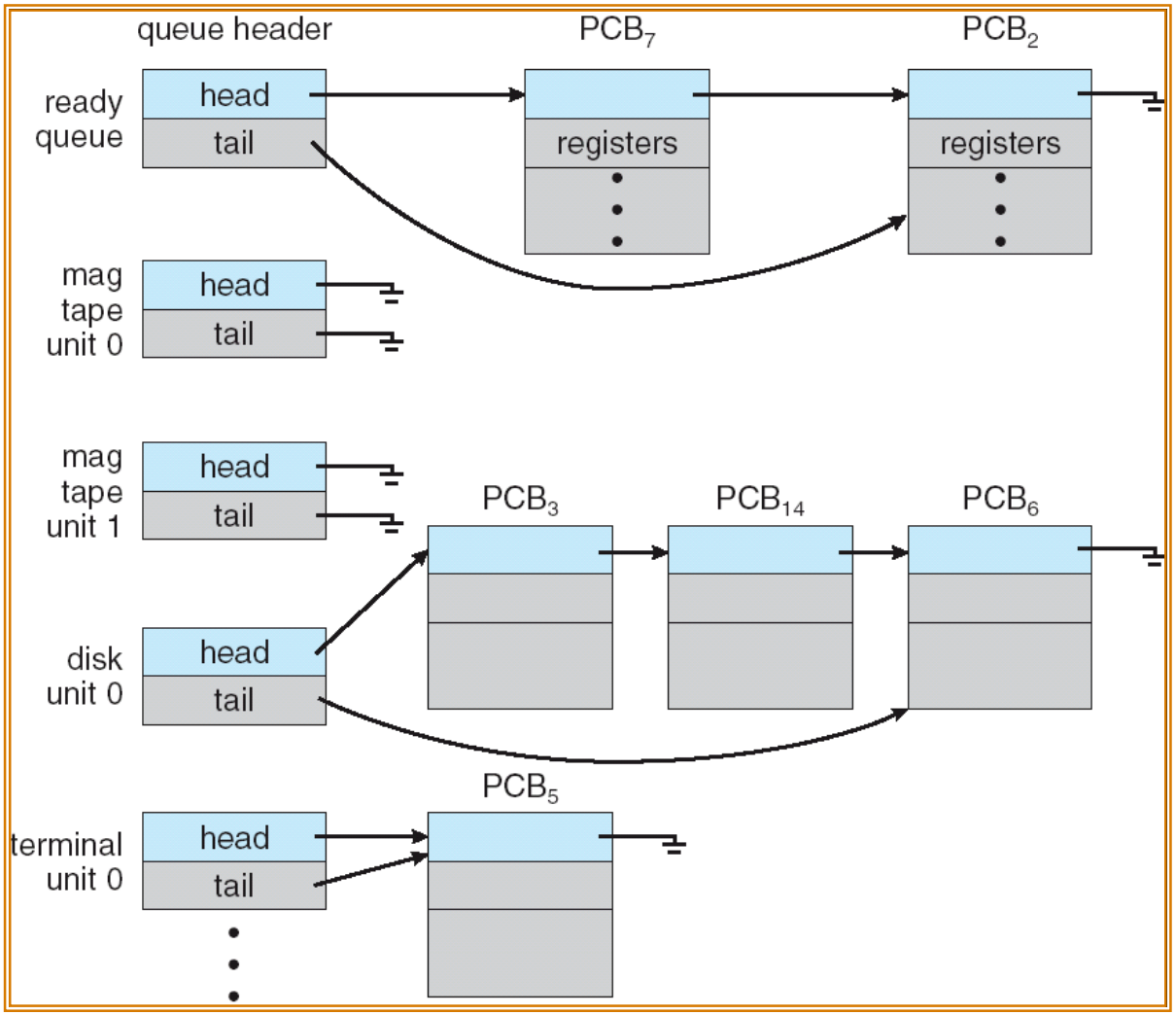


CPU switch from P_0 to P_1

- Save all state of P_0, restore all state of P_1, save ..
  - All these times are overhead



Ready queue and other device queues



Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
  - invoked very infrequently (seconds, minutes) ⇒ (may be slow)
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - invoked very frequently (milliseconds) ⇒ (must be fast)
- Medium-term scheduler moves some processes to disk
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Operations on processes

- Process creation
  - Parent creates new process forming a tree
  - Child process can run concurrently with parent or not
  - Child can share all resources, some or none at all
- Process termination
  - Exit for normal termination
    - Output data from child to parent (via **wait**)
    - exit() and _exit() functions
  - Abort for abnormal kernel initiated termination
  - Some OS require the presence of parent to allow child

C example of fork

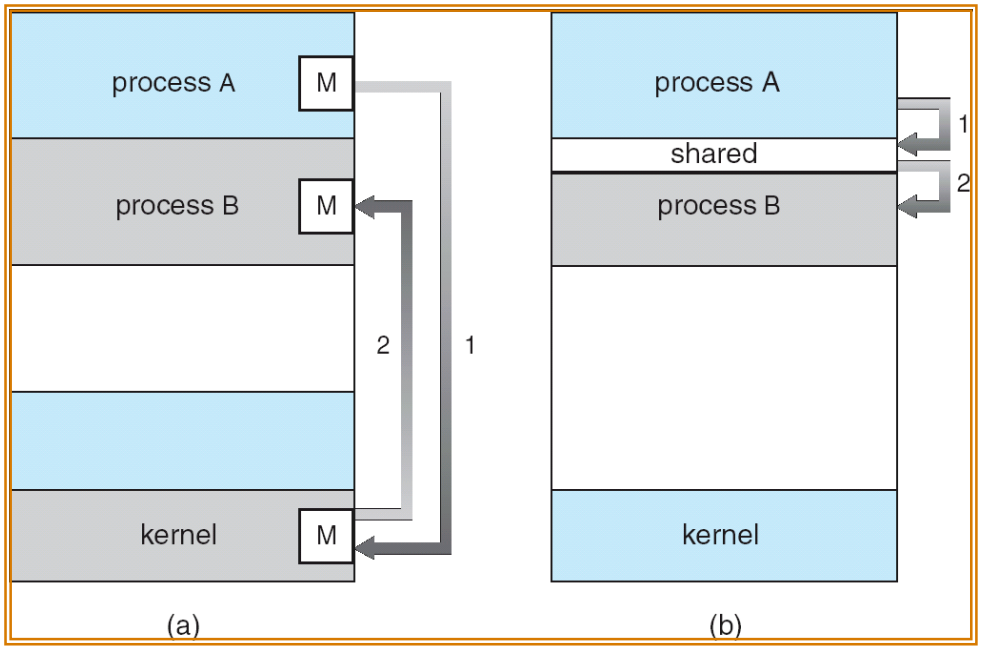
```
int main()
   {
        pid_t  pid;
        /* fork another process */
        pid = fork();
        if (pid < 0) { /* error occurred */
               fprintf(stderr, "Fork Failed");
               exit(-1);
        }
        else if (pid == 0) { /* child process */
               execlp("/bin/ls", "ls", NULL);
        }
        else { /* parent process */
               /* waits for child to complete */
               wait (NULL);
               printf ("Child Complete");
               exit(0);
        }
   }
```

Interprocess communications

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

IPC mechanisms

- Shared memory
  - Create shared memory region
  - When one process writes into this region, the other process can see it and vice versa
- Message passing
  - Explicitly send() and receive()



Producer/consumer using shared memory

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
   . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements

Insert/Remove methods

```
while (true) {
/* Produce an item */
 while (((in = (in + 1) % BUFFER SIZE count)  == out)
        ;   /* do nothing -- no free buffers */
   buffer[in] = item;
   in = (in + 1) % BUFFER SIZE;
}
```

```
while (true) {
   while (in == out)
      ; // do nothing -- nothing to consume

   // remove an item from the buffer
   item = buffer[out];
   out = (out + 1) % BUFFER SIZE;
   return item;
}
```

Message passing

- Requires ways to name objects (same machine or different machine).
- Communications can be synchronous or asynchronous.
- May need to buffer messages that are not ready to be read

Wrapup

- Processes are programs in execution
  - Kernel keeps track of them using process control blocks
  - PCBs are saved and restored at context switch
- Schedulers choose the ready process to run
- Processes create other processes
  - On exit, status returned to parent
- Processes communicate with each other using shared memory or message passing

- Tomorrow: threads