Get arguments…

int main(int argc,char *argv[])
{ return(0); }

<u>ex11-1 hello testing one two</u>

argc=5

argv[1]=hello argv[2]=testing argv[3]=one argv[4]=two

Strtok ()

```
int main ()
{
  char str[] ="- This, a sample string.";
  char * pch;
  printf ("Splitting string \"%s\" into tokens:\n",str);
  pch = strtok (str," ,.-");
  while (pch != NULL)
  {
    printf ("%s\n",pch);
    pch = strtok (NULL, " ,.-");
  }
  return 0;
}
```

```
Splitting string "- This, a sample string." into tokens:
This
a
sample
string
```

wait() function

- The **wait**() function suspends execution of its calling process until information is available for a terminated child process, or a signal is received.
- **Variations:**
  - **wait**, **waitpid** - wait for process termination

example

How I grade assignments?

- 80% to functionality
- 20% to style, comments, interface, etc…
- 0 if not compilable

Project Proposal
What you should include (at least)

- The report should be 1-2 pages and includes:
  - Goals
  - Background/Reviews
  - Your idea / risk analysis
  - Software/Hardware Environment
  - Tasks Allocation
  - Schedule/timeline

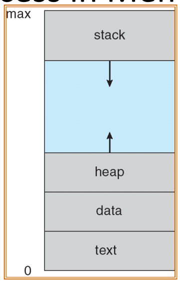
Chapter 3: Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication
- Communication in Client-Server Systems

Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section

Process in Memory

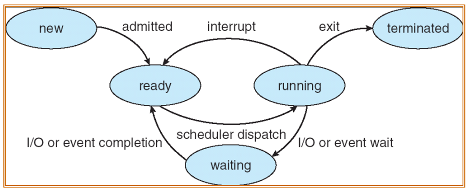


Example …

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**: The process is waiting to be assigned to a process
  - **terminated**: The process has finished execution

Diagram of Process State

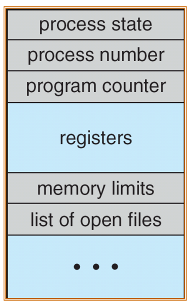


Process Control Block (PCB)

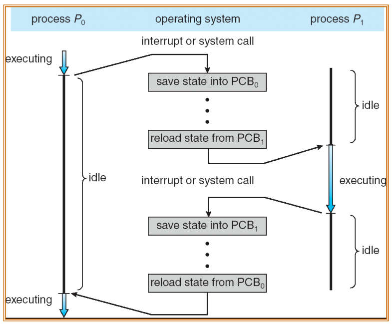
Information associated with each process

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

Process Control Block (PCB)



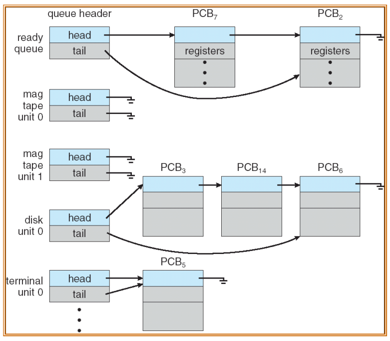
CPU Switch From Process to Process



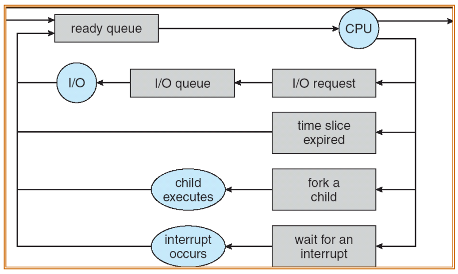
Process Scheduling Queues

- **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
- Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

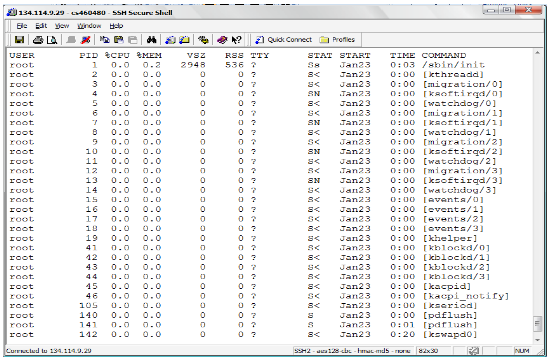
Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- 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

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Linux Process Sample



USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.2	2948	536	?	Ss	Jan23	0:03	/sbin/init
root	2	0.0	0.0	0	0	?	S<	Jan23	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S<	Jan23	0:00	[migration/0]
root	4	0.0	0.0	0	0	?	SN	Jan23	0:00	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	Jan23	0:00	[watchdog/0]
root	6	0.0	0.0	0	0	?	S<	Jan23	0:00	[migration/1]
root	7	0.0	0.0	0	0	?	SN	Jan23	0:00	[ksoftirqd/1]
root	8	0.0	0.0	0	0	?	S<	Jan23	0:00	[watchdog/1]
root	9	0.0	0.0	0	0	?	S<	Jan23	0:00	[migration/2]
root	10	0.0	0.0	0	0	?	SN	Jan23	0:00	[ksoftirqd/2]
root	11	0.0	0.0	0	0	?	S<	Jan23	0:00	[watchdog/2]
root	12	0.0	0.0	0	0	?	S<	Jan23	0:00	[migration/3]
root	13	0.0	0.0	0	0	?	SN	Jan23	0:00	[ksoftirqd/3]
root	14	0.0	0.0	0	0	?	S<	Jan23	0:00	[watchdog/3]
root	15	0.0	0.0	0	0	?	S<	Jan23	0:00	[events/0]
root	16	0.0	0.0	0	0	?	S<	Jan23	0:00	[events/1]
root	17	0.0	0.0	0	0	?	S<	Jan23	0:00	[events/2]
root	18	0.0	0.0	0	0	?	S<	Jan23	0:00	[events/3]
root	19	0.0	0.0	0	0	?	S<	Jan23	0:00	[khelper]
root	41	0.0	0.0	0	0	?	S<	Jan23	0:00	[kblockd/0]
root	42	0.0	0.0	0	0	?	S<	Jan23	0:00	[kblockd/1]
root	43	0.0	0.0	0	0	?	S<	Jan23	0:00	[kblockd/2]
root	44	0.0	0.0	0	0	?	S<	Jan23	0:00	[kblockd/3]
root	45	0.0	0.0	0	0	?	S<	Jan23	0:00	[kacpid]
root	46	0.0	0.0	0	0	?	S<	Jan23	0:00	[kacpi_notify]
root	105	0.0	0.0	0	0	?	S<	Jan23	0:00	[kseriod]
root	140	0.0	0.0	0	0	?	S	Jan23	0:00	[pdflush]
root	141	0.0	0.0	0	0	?	S	Jan23	0:01	[pdflush]
root	142	0.0	0.0	0	0	?	S<	Jan23	0:20	[kswapd0]

Connected to 134.114.9.29 SSH2 - aes128-cbc - hmac-md5 - none 82x30 NUM

Linux PROCESS STATE CODES

D uninterruptible sleep (usually IO)
R runnable (on run queue)
S sleeping
T traced or stopped
Z a defunct ("zombie") process

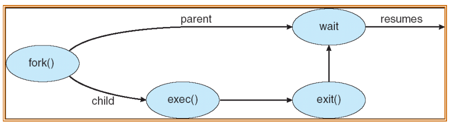
Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

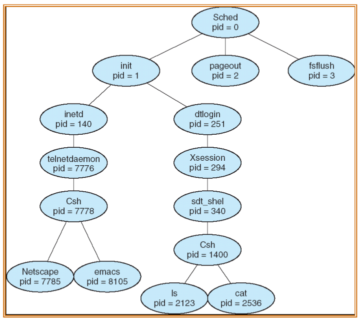
Process Creation



C Program Forking Separate Process

```
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 */
            /* parent will wait for the child to complete */
            wait (NULL);
            printf ("Child Complete");
            exit(0);
    }
}
```

A tree of processes on a typical Solaris



Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Cooperating Processes

- **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

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer
  - *bounded-buffer* assumes that there is a fixed buffer size

Bounded-Buffer – Shared-Memory Solution

- 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

Bounded-Buffer – Insert() Method

```
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;
{
```

Bounded Buffer – Remove() Method

```
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;
 {
```

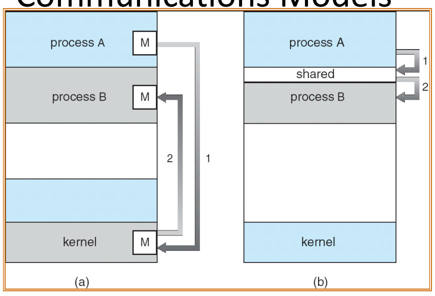
Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Communications Models



Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A

Indirect Communication

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.