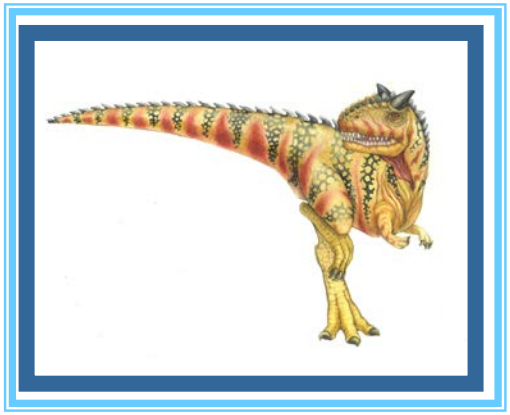
Chapter 3: Process Concept



Silberschatz, Galvin and Gagne ©2009

Chapter 3: Process-Concept

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication
- Examples of IPC Systems
- Communication in Client-Server Systems

Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To describe 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

```c
#include "stdafx.h"

int x;
int y = 1;


int _tmain(int argc, _TCHAR* argv[])
{
    int z;
    int *h = new int[100];

    printf("main => %p\n", _tmain);
    printf("x => %p\n", &x);
    printf("y => %p\n", &y);
    printf("z => %p\n", &z);
    printf("h is %p\n", h);

    getchar();

    return 0;
}
```

Process in Memory

```
     3:
     4: #include "stdafx.h"
     5:
     6: int x;
     7: int y = 1;
     8:
     9:
    10: int _tmain(int argc, _TCHAR* argv[])
    11: {
000007F7FB402E50 48 89 54 24 10       mov         qword ptr [rsp+10h],rdx
000007F7FB402E55 89 4C 24 08          mov         dword ptr [rsp+8],ecx
000007F7FB402E59 57                   push        rdi
000007F7FB402E5A 48 83 EC 50          sub         rsp,50h
000007F7FB402E5E 48 8B FC             mov         rdi,rsp
000007F7FB402E61 B9 14 00 00 00       mov         ecx,14h
000007F7FB402E66 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
000007F7FB402E6B F3 AB                rep stos    dword ptr [rdi]
000007F7FB402E6D 8B 4C 24 60          mov         ecx,dword ptr [rsp+60h]
    12:     int z;
    13:     int *h = new int[100];
000007F7FB402E71 B9 90 01 00 00       mov         ecx,190h
000007F7FB402E76 E8 DB E2 FF FF       call        operator new (7F7FB401156h)
000007F7FB402E7B 48 89 44 24 40       mov         qword ptr [rsp+40h],rax
000007F7FB402E80 48 8B 44 24 40       mov         rax,qword ptr [rsp+40h]
000007F7FB402E85 48 89 44 24 38       mov         qword ptr [h],rax
    14:
    15:     printf("main => %p\n", _tmain);
000007F7FB402E8A 48 8D 15 74 E1 FF FF lea         rdx,[@ILT+0(wmain) (7F7FB401005h)]
000007F7FB402E91 48 8D 0D F8 38 00 00 lea         rcx,[__xi_z+130h (7F7FB406790h)]
000007F7FB402E98 FF 15 72 86 00 00    call        qword ptr [__imp_printf (7F7FB40B510h)]
    16:     printf("x => %p\n", &x);
000007F7FB402E9E 48 8D 15 AB 62 00 00 lea         rdx,[x (7F7FB409150h)]
000007F7FB402EA5 48 8D 0D F4 38 00 00 lea         rcx,[__xi_z+140h (7F7FB4067A0h)]
000007F7FB402EAC FF 15 5E 86 00 00    call        qword ptr [__imp_printf (7F7FB40B510h)]
    17:     printf("y => %p\n", &y);
000007F7FB402EB2 48 8D 15 47 61 00 00 lea         rdx,[y (7F7FB409000h)]
000007F7FB402EB9 48 8D 0D F0 38 00 00 lea         rcx,[__xi_z+150h (7F7FB4067B0h)]
000007F7FB402EC0 FF 15 4A 86 00 00    call        qword ptr [__imp_printf (7F7FB40B510h)]
```

Process in Memory

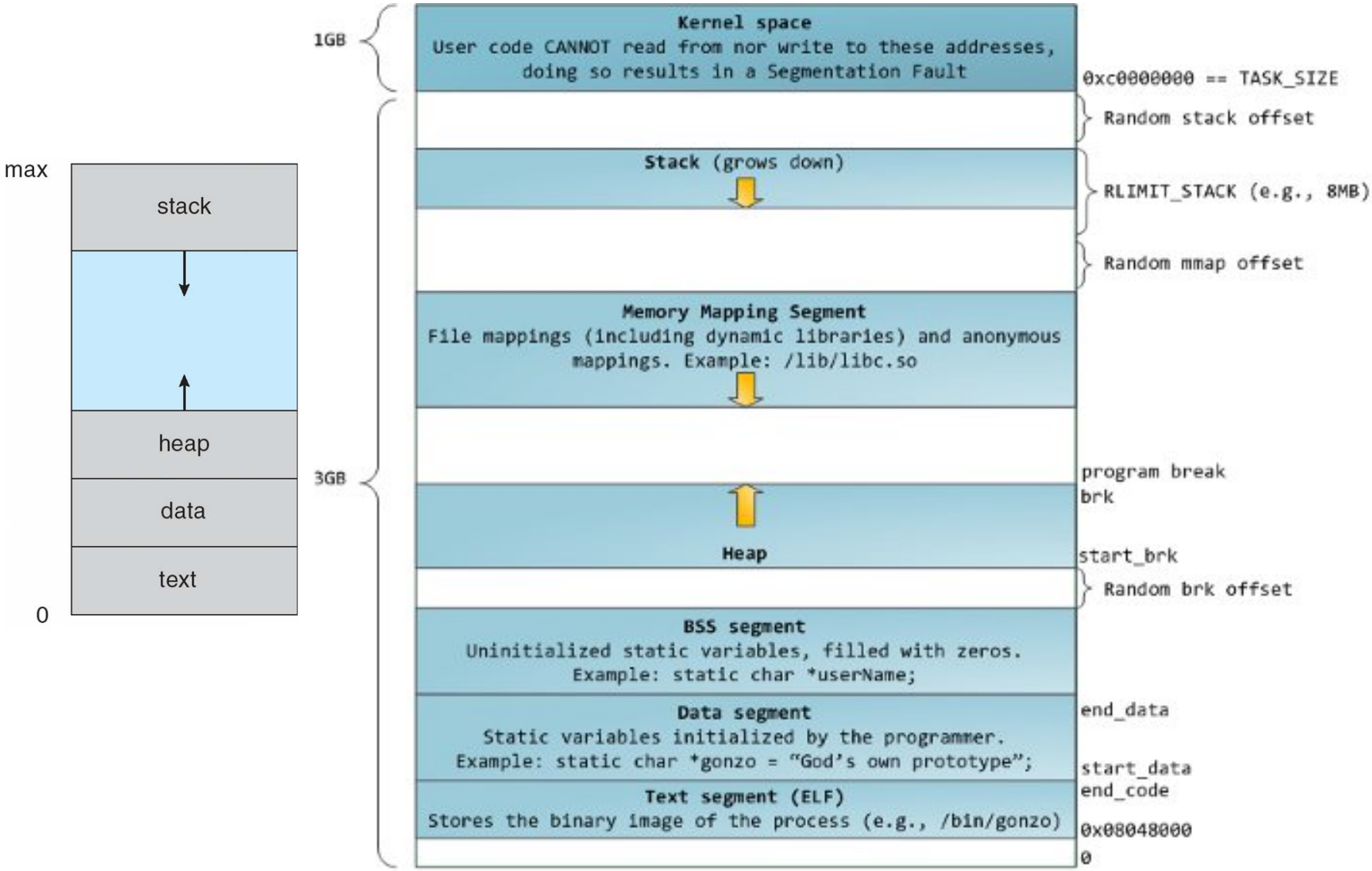
```
    18:     printf("z => %p\n", &z);
000007F7FB402EC6 48 8D 54 24 24       lea         rdx,[z]
000007F7FB402ECB 48 8D 0D EE 38 00 00 lea         rcx,[__xi_z+160h (7F7FB4067C0h)]
000007F7FB402ED2 FF 15 38 86 00 00    call        qword ptr [__imp_printf (7F7FB40B510h)]
    19:     printf("h is %p\n", h);
000007F7FB402ED8 48 8B 54 24 38       mov         rdx,qword ptr [h]
000007F7FB402EDD 48 8D 0D EC 38 00 00 lea         rcx,[__xi_z+170h (7F7FB4067D0h)]
000007F7FB402EE4 FF 15 26 86 00 00    call        qword ptr [__imp_printf (7F7FB40B510h)]
    20:
    21:     getchar();
000007F7FB402EEA FF 15 28 86 00 00    call        qword ptr [__imp_getchar (7F7FB40B518h)]
    22:
    23:     return 0;
000007F7FB402EF0 33 C0                xor         eax,eax
    24: }
000007F7FB402EF2 8B F8                mov         edi,eax
000007F7FB402EF4 48 8B CC             mov         rcx,rsp
000007F7FB402EF7 48 8D 15 22 39 00 00 lea         rdx,[__xi_z+1C0h (7F7FB406820h)]
000007F7FB402EFE E8 CD E1 FF FF       call        _RTC_CheckStackVars (7F7FB4010D0h)
000007F7FB402F03 8B C7                mov         eax,edi
000007F7FB402F05 48 83 C4 50          add         rsp,50h
000007F7FB402F09 5F                   pop         rdi
000007F7FB402F0A C3                   ret
```

c:\users\hank\documents\visual studio 2010\Projects\test_process1\x64\...

```
main => 000007F7FB401005
x => 000007F7FB409150
y => 000007F7FB409000
z => 000000000097FAD4
h is 0000000000AD83A0
```

Address	Type	Size	Protection	Details
0000000000850000	Heap (Shareable)	64 K	Read/Write	Heap ID: 2 [COMPATABILITY]
0000000000860000	Mapped File	4 K	Read	C:\Windows\Globalization\zh-TW.nlx
0000000000870000	Shareable	36 K	Read	
0000000000880000	Thread Stack	1,024 K	Read/Write/Guard	Thread ID: 1340
0000000000880000	Thread Stack	992 K	Reserved	
0000000000978000	Thread Stack	12 K	Read/Write/Guard	
000000000097B000	Thread Stack	20 K	Read/Write	
0000000000980000	Shareable	16 K	Read	
0000000000990000	Shareable	4 K	Read	
00000000009A0000	Private Data	8 K	Read/Write	
00000000009B0000	Mapped File	468 K	Read	C:\Windows\System32\locale.nls
0000000000AD0000	Heap (Private Data)	64 K	Read/Write	Heap ID: 3 [COMPATABILITY]
0000000000AD0000	Heap (Private Data)	36 K	Read/Write	Heap ID: 3 [COMPATABILITY]
0000000000AD9000	Heap (Private Data)	28 K	Reserved	Heap ID: 3 [COMPATABILITY]
0000000000B50000	Heap (Private Data)	1,024 K	Read/Write	Heap ID: 1 [COMPATABILITY]
000000005B3E0000	Image (ASLR)	1,856 K	Execute/Read	C:\Windows\System32\msvcr100d.dll
000000007FFE0000	Private Data	64 K	Read	
000007F7FB0F0000	Shareable	1,024 K	Read	
000007F7FB1F0000	Shareable	204 K	Read	
000007F7FB223000	Private Data	4 K	Read/Write	Process Environment Block
000007F7FB22E000	Private Data	8 K	Read/Write	Thread Environment Block ID: 1340
000007F7FB400000	Image (ASLR)	56 K	Execute/Read	C:\Users\Hank\Documents\Visual Studio 2010\Projects\test_pr
000007F7FB400000	Image (ASLR)	4 K	Read	Header
000007F7FB401000	Image (ASLR)	20 K	Execute/Read	.text
000007F7FB406000	Image (ASLR)	12 K	Read	.rdata
000007F7FB409000	Image (ASLR)	4 K	Read/Write	.data
000007F7FB40A000	Image (ASLR)	4 K	Read	.pdata
000007F7FB40B000	Image (ASLR)	4 K	Read/Write	.idata
000007F7FB40C000	Image (ASLR)	4 K	Read	.rsrc
000007F7FB40D000	Image (ASLR)	4 K	Read	.reloc
000007FDCCDA0000	Image (ASLR)	972 K	Execute/Read	C:\Windows\System32\KernelBase.dll

Process in Memory



Process State

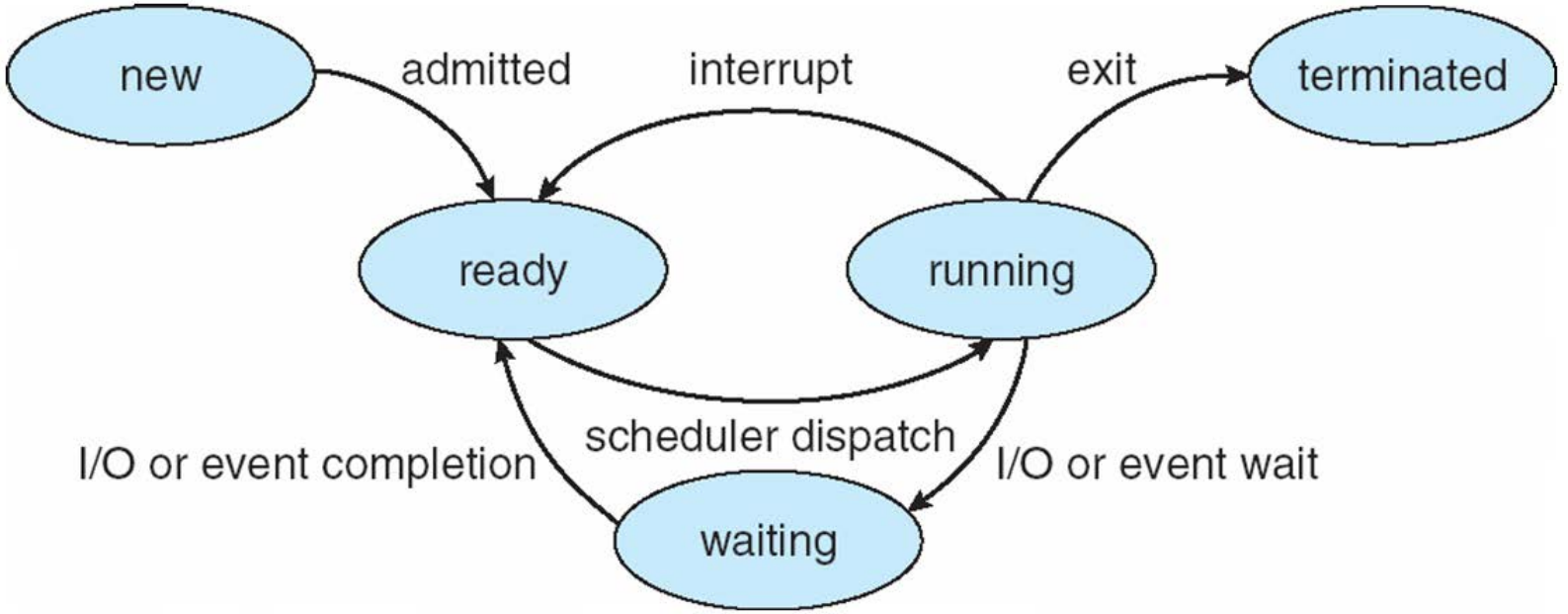
- As a process executes, it changes *state*
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has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

hank@Maestro:/home/hank

File Edit View Search Terminal Help

```
top - 23:19:48 up 3 min,  2 users,  load average: 0.56, 0.52, 0.25
Tasks: 144 total,   2 running, 142 sleeping,   0 stopped,   0 zombie
Cpu(s): 10.3%us,  7.4%sy,  0.0%ni, 81.9%id,  0.2%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:   1019704k total,   878904k used,   140800k free,    34432k buffers
Swap:  2064380k total,        0k used,  2064380k free,   318172k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2669 hank      20   0  721m 100m  31m S 19.3 10.0   0:15.16 firefox
 1936 root      20   0  303m  93m  14m S 12.0  9.4   0:07.24 Xorg
 2268 hank      20   0 1836m  94m  43m S 10.6  9.5   0:08.35 gnome-shell
 2477 hank      20   0  572m  15m 9884 S  0.3  1.5   0:00.55 gnome-terminal
 2668 root      20   0 15256 1212  900 R  0.3  0.1   0:00.21 top
    1 root      20   0 66744  24m 2080 S  0.0  2.5   0:01.20 systemd
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.01 ksoftirqd/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0
    5 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/u:0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kworker/1:0
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.04 ksoftirqd/1
   11 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
   12 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 cpuset
   13 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
```

Diagram of Process State



Process State

hank@Maestro:/home/ha

File Edit View Search Terminal Help

```
[root@Maestro hank]# ps axfj | more
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S        0   0:00 [kthreadd]
    2     3     0     0 ?           -1 S        0   0:00  \_ [ksoftirqd/0]
    2     5     0     0 ?           -1 S        0   0:00  \_ [kworker/u:0]
    2     6     0     0 ?           -1 S        0   0:00  \_ [migration/0]
    2     7     0     0 ?           -1 S        0   0:00  \_ [watchdog/0]
    2     8     0     0 ?           -1 S        0   0:00  \_ [migration/1]
    2    10     0     0 ?           -1 S        0   0:00  \_ [ksoftirqd/1]
    2    11     0     0 ?           -1 S        0   0:00  \_ [watchdog/1]
    2    12     0     0 ?           -1 S<       0   0:00  \_ [cpuset]
    2    13     0     0 ?           -1 S<       0   0:00  \_ [khelper]
    2    14     0     0 ?           -1 S        0   0:00  \_ [kdevtmpfs]
    2    15     0     0 ?           -1 S<       0   0:00  \_ [netns]
    2    16     0     0 ?           -1 S        0   0:00  \_ [sync_supers]
    2    17     0     0 ?           -1 S        0   0:00  \_ [bdi-default]
    2    18     0     0 ?           -1 S<       0   0:00  \_ [kintegrityd]
    2    19     0     0 ?           -1 S<       0   0:00  \_ [kblockd]
    2    20     0     0 ?           -1 S<       0   0:00  \_ [ata_sff]
    2    21     0     0 ?           -1 S        0   0:00  \_ [khubd]
    2    22     0     0 ?           -1 S<       0   0:00  \_ [md]
    2    23     0     0 ?           -1 S        0   0:00  \_ [kworker/1:1]
    2    25     0     0 ?           -1 S        0   0:00  \_ [kswapd0]
    2    26     0     0 ?           -1 SN       0   0:00  \_ [ksmd]
    2    27     0     0 ?           -1 SN       0   0:00  \_ [khugepaged]
    2    28     0     0 ?           -1 S        0   0:00  \_ [fsnotify_mark]
    2    29     0     0 ?           -1 S<       0   0:00  \_ [crypto]
    2    35     0     0 ?           -1 S<       0   0:00  \_ [kthrotld]
    2    38     0     0 ?           -1 S        0   0:00  \_ [scsi_eh_0]
    2    39     0     0 ?           -1 S        0   0:00  \_ [scsi_eh_1]
    2    40     0     0 ?           -1 S        0   0:00  \_ [scsi_eh_2]
    2    41     0     0 ?           -1 S        0   0:00  \_ [kworker/u:2]
    2    43     0     0 ?           -1 S<       0   0:00  \_ [kpsmoused]
    2    44     0     0 ?           -1 S<       0   0:00  \_ [deferwq]
    2    46     0     0 ?           -1 S        0   0:00  \_ [kworker/0:2]
    2   238     0     0 ?           -1 S        0   0:00  \_ [kworker/1:2]
    2   290     0     0 ?           -1 S<       0   0:00  \_ [kdmflush]
    2   291     0     0 ?           -1 S<       0   0:00  \_ [kdmflush]
    2   338     0     0 ?           -1 S        0   0:00  \_ [jbd2/dm-1-8]
    2   339     0     0 ?           -1 S<       0   0:00  \_ [ext4-dio-unwrit]
    2   375     0     0 ?           -1 S        0   0:00  \_ [kauditd]
```

UNIX-LIKE FOR LIFE

Friday, July 31, 2009

Linux process state codes

Here are the different values that the s, stat and state output specifiers (header "STAT" or "S") will display to describe the state of a process.

D Uninterruptible sleep (usually IO)
R Running or runnable (on run queue)
S Interruptible sleep (waiting for an event to complete)
T Stopped, either by a job control signal or because it is being traced.
W paging (not valid since the 2.6.xx kernel)
X dead (should never be seen)
Z Defunct ("zombie") process, terminated but not reaped by its parent.

For BSD formats and when the stat keyword is used, additional characters may be displayed:
< high-priority (not nice to other users)
N low-priority (nice to other users)
L has pages locked into memory (for real-time and custom IO)
s is a session leader
l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
+ is in the foreground process group

Posted by M.Burak Alkan at 12:01 AM

Labels linux

No comments:

Post a Comment

Newer Post Home Old

Subscribe to: Post Comments (Atom)

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)

Process Control Block (PCB)

include/linux/sched.h

C/C++ – my_kernel/include/linux/sched.h – Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer | Outline

task_struct
- state : volatile long
- stack : void*
- usage : atomic_t
- flags : unsigned int
- ptrace : unsigned int
- wake_entry : struct llist_node
- on_cpu : int
- on_rq : int
- prio : int
- static_prio : int
- normal_prio : int
- rt_priority : unsigned int
- sched_class : const struct sched_class*
- se : struct sched_entity
- rt : struct sched_rt_entity
- preempt_notifiers : struct hlist_head
- fpu_counter : unsigned char
- btrace_seq : unsigned int
- policy : unsigned int
- nr_cpus_allowed : int
- cpus_allowed : cpumask_t
- rcu_read_lock_nesting : int
- rcu_read_unlock_special : char
- rcu_node_entry : struct list_head
- rcu_blocked_node : struct rcu_node*
- rcu_boost_mutex : struct rt_mutex*
- sched_info : struct sched_info
- tasks : struct list_head

sched.h

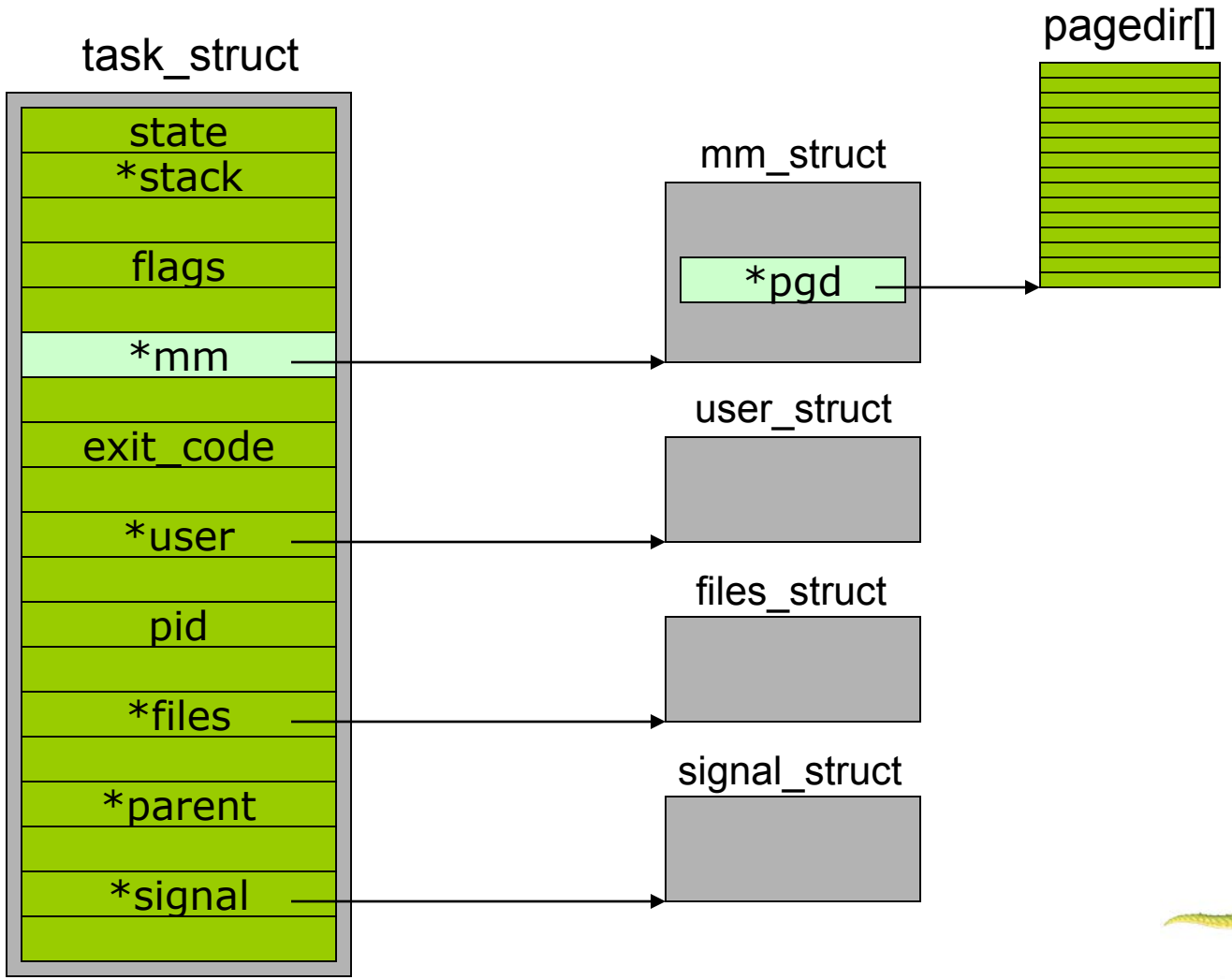
```
struct task_struct {
    volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
    void *stack;
    atomic_t usage;
    unsigned int flags; /* per process flags, defined below */
    unsigned int ptrace;

#ifdef CONFIG_SMP
    struct llist_node wake_entry;
    int on_cpu;
#endif
    int on_rq;

    int prio, static_prio, normal_prio;
    unsigned int rt_priority;
    const struct sched_class *sched_class;
    struct sched_entity se;
    struct sched_rt_entity rt;

#ifdef CONFIG_PREEMPT_NOTIFIERS
    /* list of struct preempt_notifier: */
    struct hlist_head preempt_notifiers;
#endif

    /*
     * fpu_counter contains the number of consecutive context switches
     * that the FPU is used. If this is over a threshold, the lazy fpu
     * saving becomes unlazy to save the trap. This is an unsigned char
     * so that after 256 times the counter wraps and the behavior turns
     * lazy again; this to deal with bursty apps that only use FPU for
     * a short time
     */
    unsigned char fpu_counter;
#ifdef CONFIG_BLK_DEV_IO_TRACE
    unsigned int btrace_seq;
#endif

    unsigned int policy;
    int nr_cpus_allowed;
    cpumask_t cpus_allowed;

#ifdef CONFIG_PREEMPT_RCU
    int rcu_read_lock_nesting;
    char rcu_read_unlock_special;
    struct list_head rcu_node_entry;
#endif /* #ifdef CONFIG_PREEMPT_RCU */
#ifdef CONFIG_TREE_PREEMPT_RCU
    struct rcu_node *rcu_blocked_node;
#endif /* #ifdef CONFIG_TREE_PREEMPT_RCU */
#ifdef CONFIG_RCU_BOOST
    struct rt_mutex *rcu_boost_mutex;
#endif /* #ifdef CONFIG_RCU_BOOST */
```

The Linux process descriptor

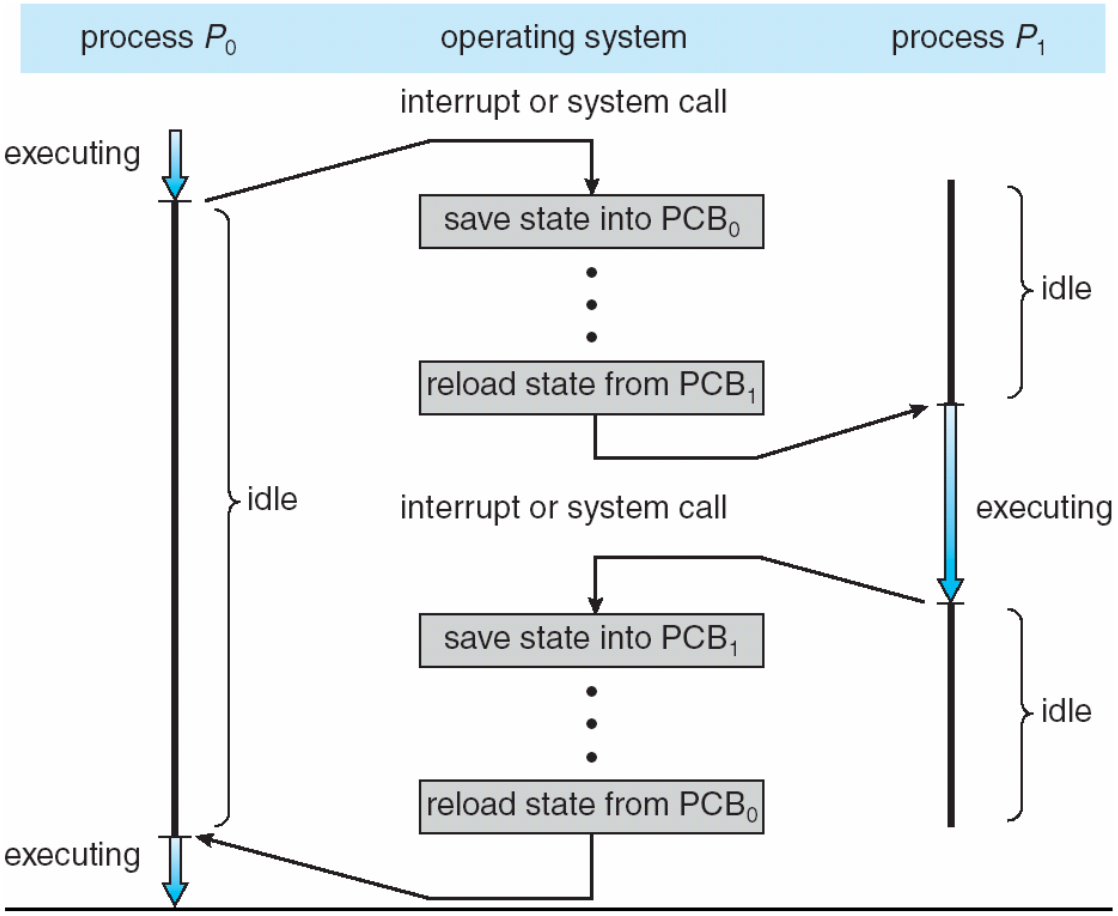
Each process descriptor contains many fields

and some are pointers to other kernel structures

which may themselves include fields that point to structures



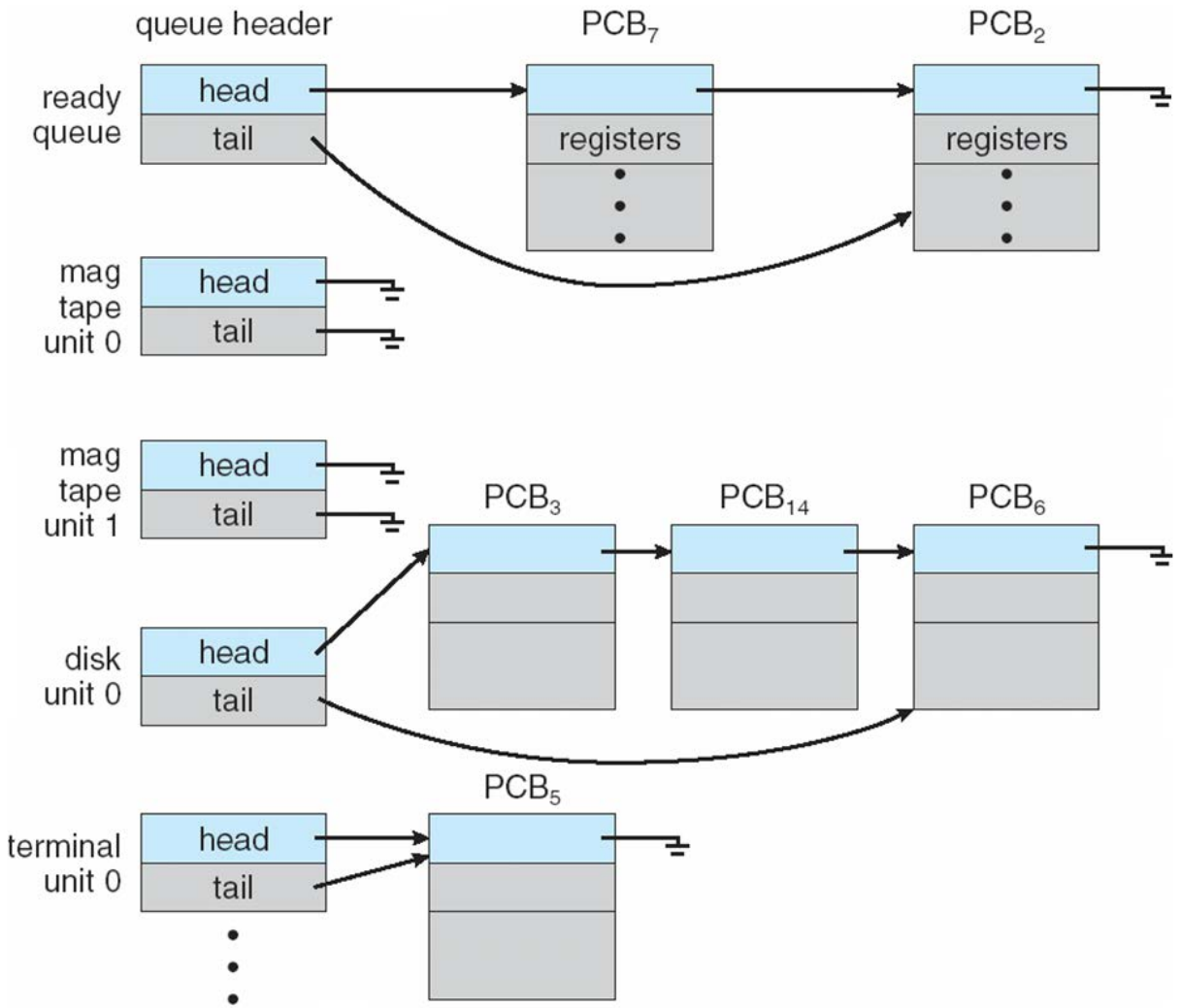
CPU Switch From Process to Process



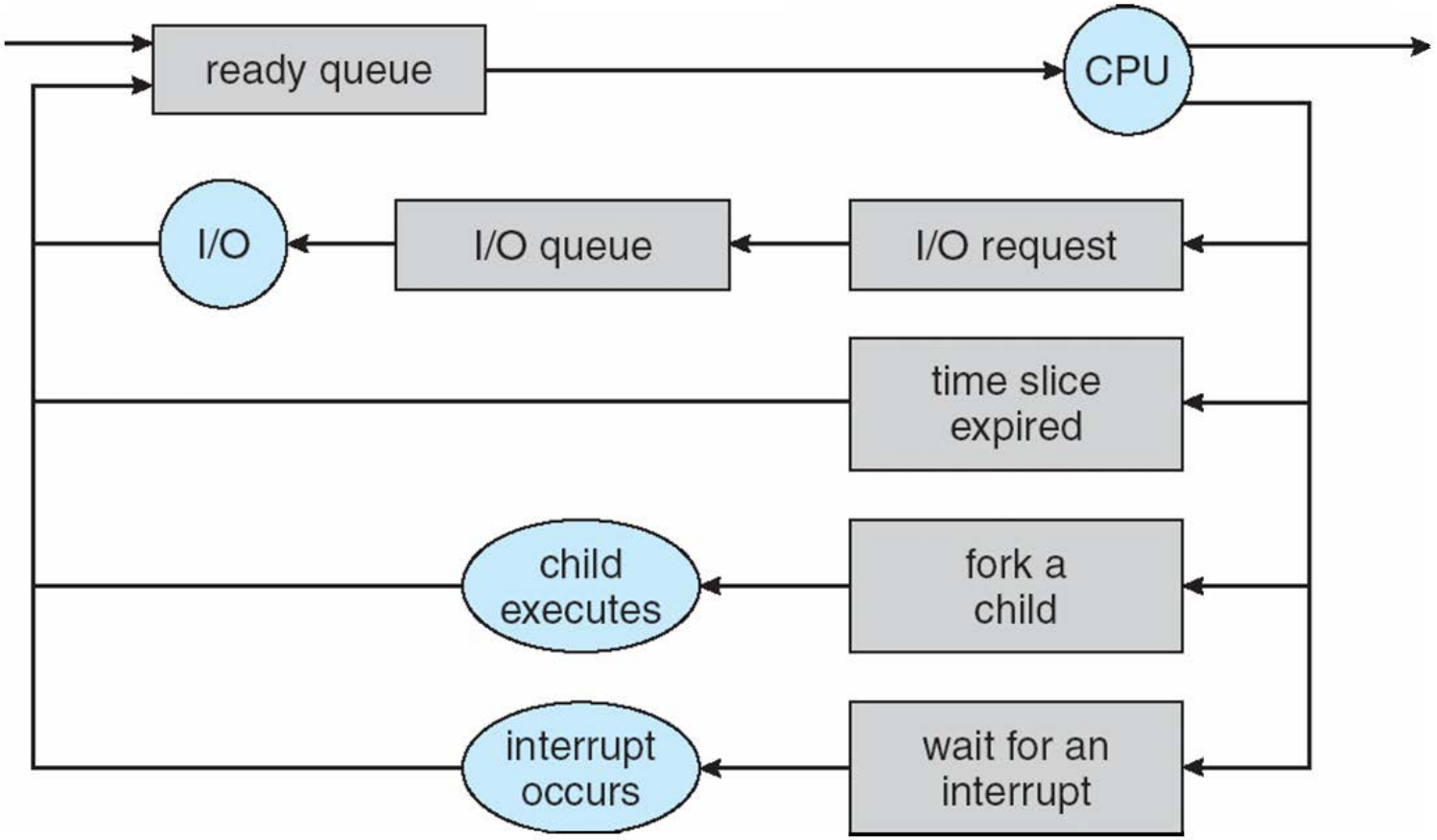
Process Scheduling Queues

- **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

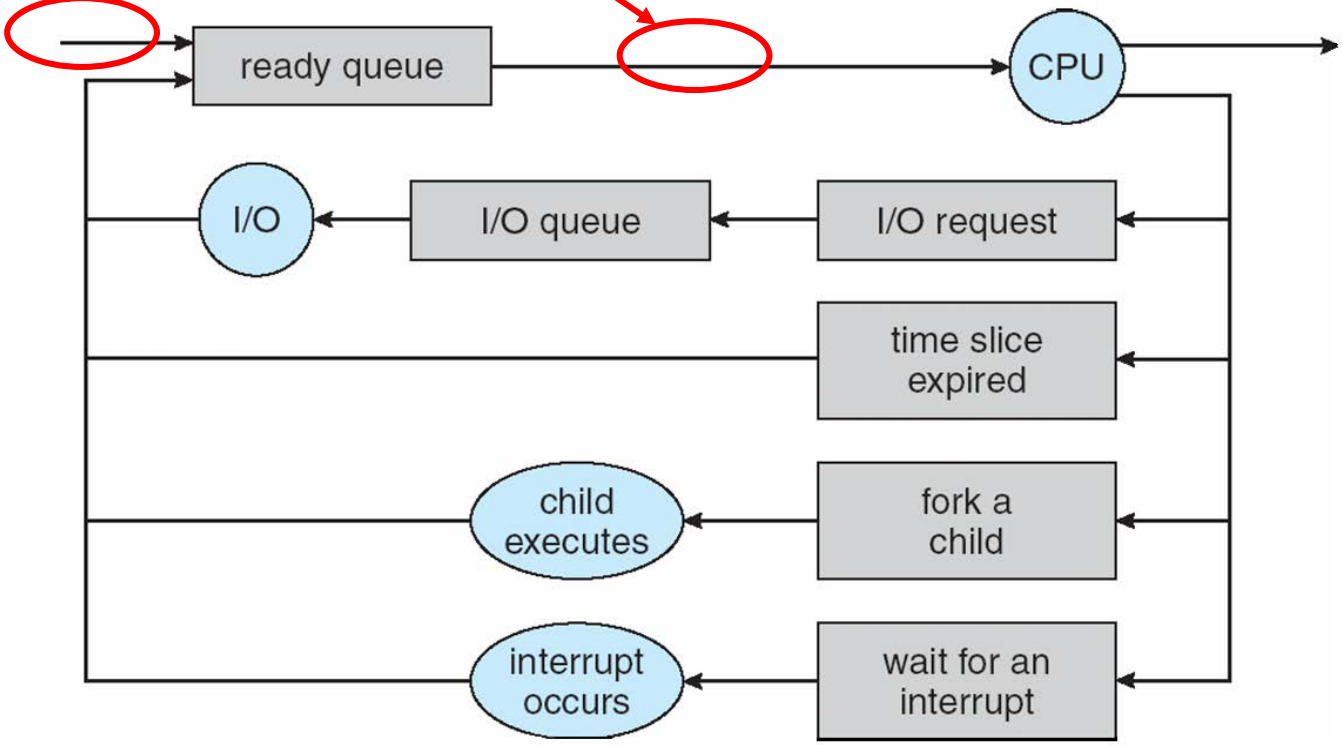


Representation of Process Scheduling

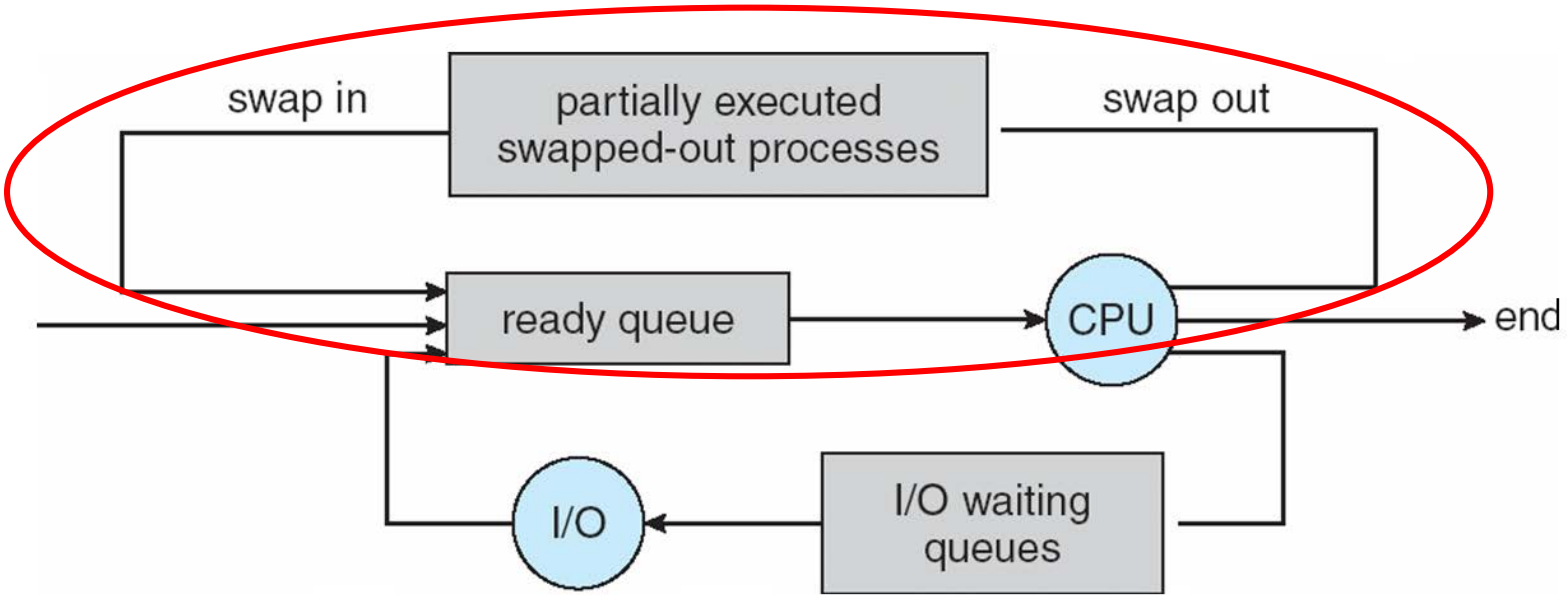


Schedulers

- **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



Addition of Medium Term Scheduling



Schedulers (Cont)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (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 via a context switch
- Context of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a tree of processes

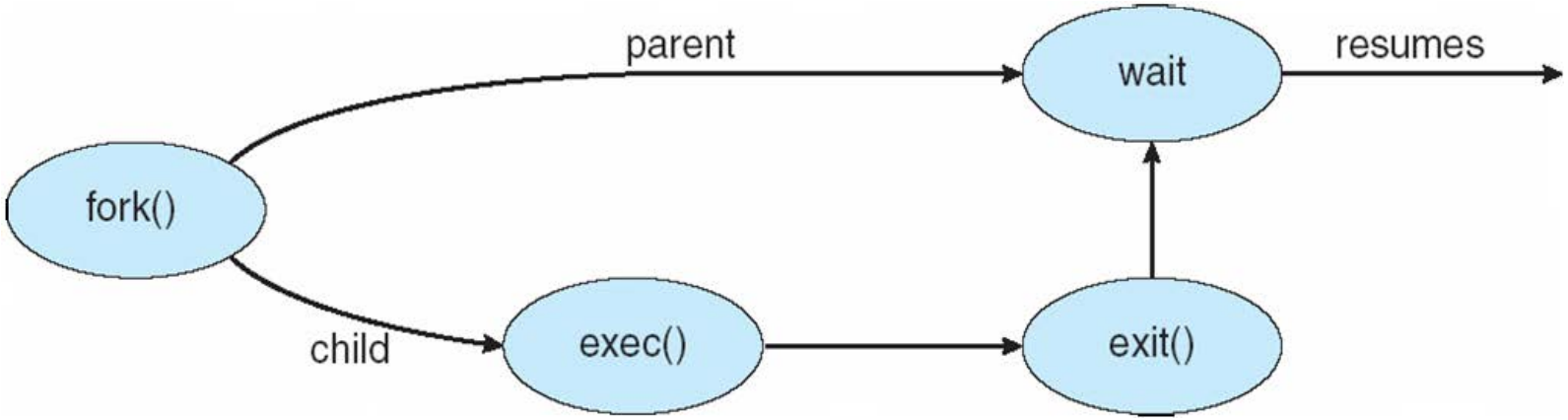
```
[hank@Maestro linux-3.5.0-2.fc17.x86_64]$ pstree
systemd─┬─NetworkManager─┬─dhclient
        │                └─3*[{NetworkManager}]
        ├─3*[VBoxClient───{VBoxClient}]
        ├─VBoxService───7*[{VBoxService}]
        ├─abrt-watch-log
        ├─abrtd
        ├─accounts-daemon───{accounts-daemon}
        ├─atd
        ├─auditd─┬─audispd─┬─sedispatch
        │        │         └─{audispd}
        │        └─{auditd}
        ├─avahi-daemon───avahi-daemon
        ├─bluetoothd
        ├─chronyd
        ├─colord───{colord}
        ├─colord-sane───{colord-sane}
        ├─crond
        ├─cupsd
        ├─2*[dbus-daemon───{dbus-daemon}]
        ├─dbus-launch
        ├─dconf-service───2*[{dconf-service}]
        ├─gconfd-2
        ├─gdm-binary─┬─gdm-simple-slav─┬─Xorg───8*[{Xorg}]
        │            │                 └─gdm-session-wor─┬─gnome-session─┬─abrt-applet
        │            │                                   │               ├─deja-dup-monito───2*[{deja-dup-monito+
        │            │                                   │               ├─evolution-alarm───2*[{evolution-alarm+
        │            │                                   │               ├─gnome-screensav───2*[{gnome-screensav+
        │            │                                   │               ├─gnome-settings-───2*[{gnome-settings-+
        │            │                                   │               ├─gnome-shell─┬─eclipse───java───23*[{j+
        │            │                                   │               │             ├─gnome-system-mo───2*[{g+
        │            │                                   │               │             └─gnome-terminal─┬─bash─+
++      │            │                                   │               │                              ├─bash─+
++      │            │                                   │               │                              ├─gnome-+
        │            │                                   │               │                              └─3*[{gn+
        │            │                                   │               └─15*[{gnome-shell}]
        │            │                                   ├─nm-applet───{nm-applet}
        │            │                                   ├─seapplet
        │            │                                   ├─tracker-miner-f───2*[{tracker-miner-f+
        │            │                                   ├─zeitgeist-datah───{zeitgeist-datah}
        │            │                                   └─3*[{gnome-session}]
        │            │                                 └─2*[{gdm-session-wor}]
        │            └─{gdm-simple-slav}
        └─{gdm-binary}
        ├─gnome-keyring-d───6*[{gnome-keyring-d}]
```

- Generally, process identified and managed via **a process identifier** (**pid**)
- 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);
        }
}
```

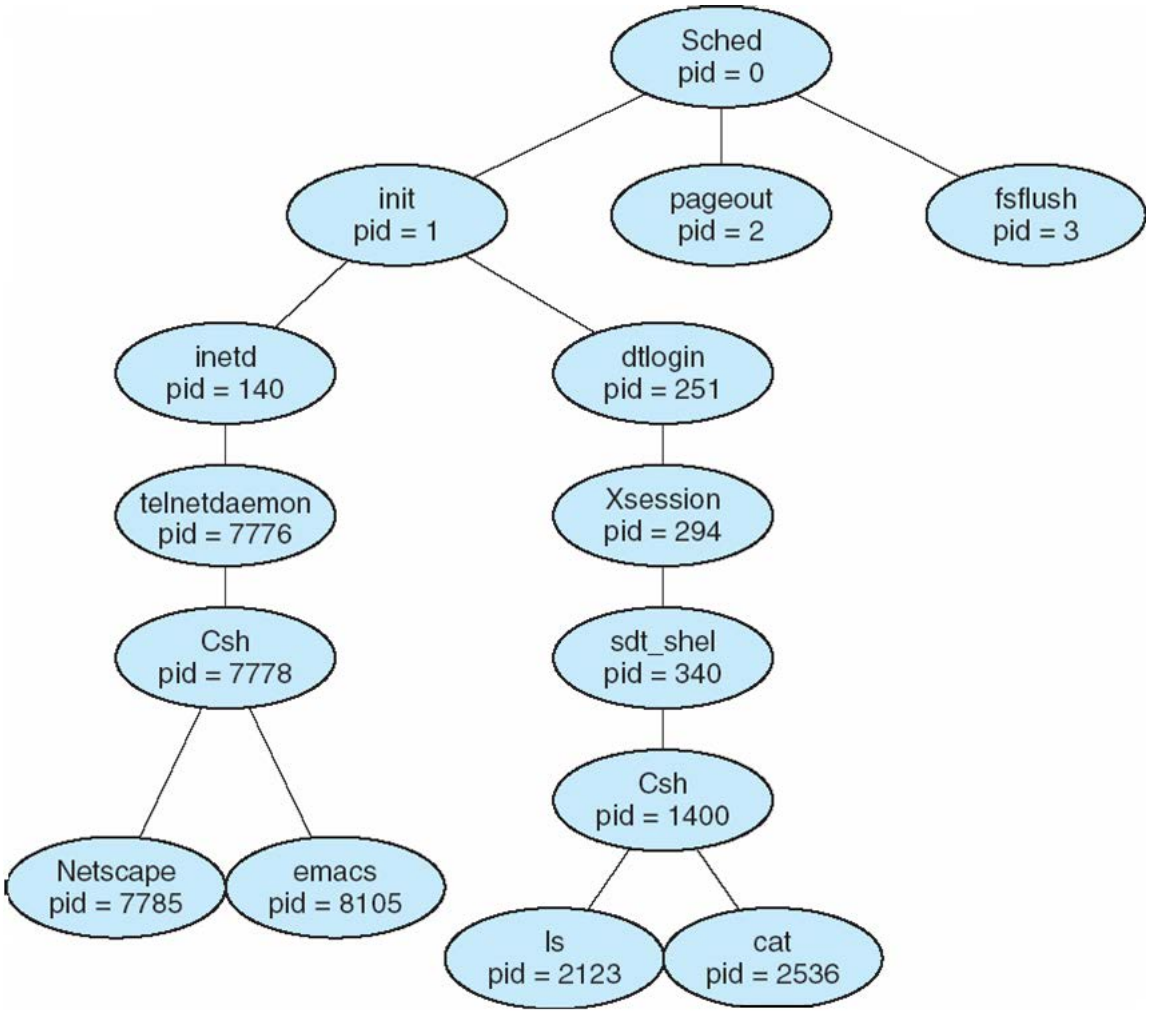
```
int main()
{
pid_t  pid;
        /* fork another process */
        pid = fork();
        if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
        }
        else if (pid == 0) { /* child process */
                execlp("/bin/ls", "ls",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

parent process memory

```
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);
}
}
```

child process memory

A tree of processes on a typical Solaris



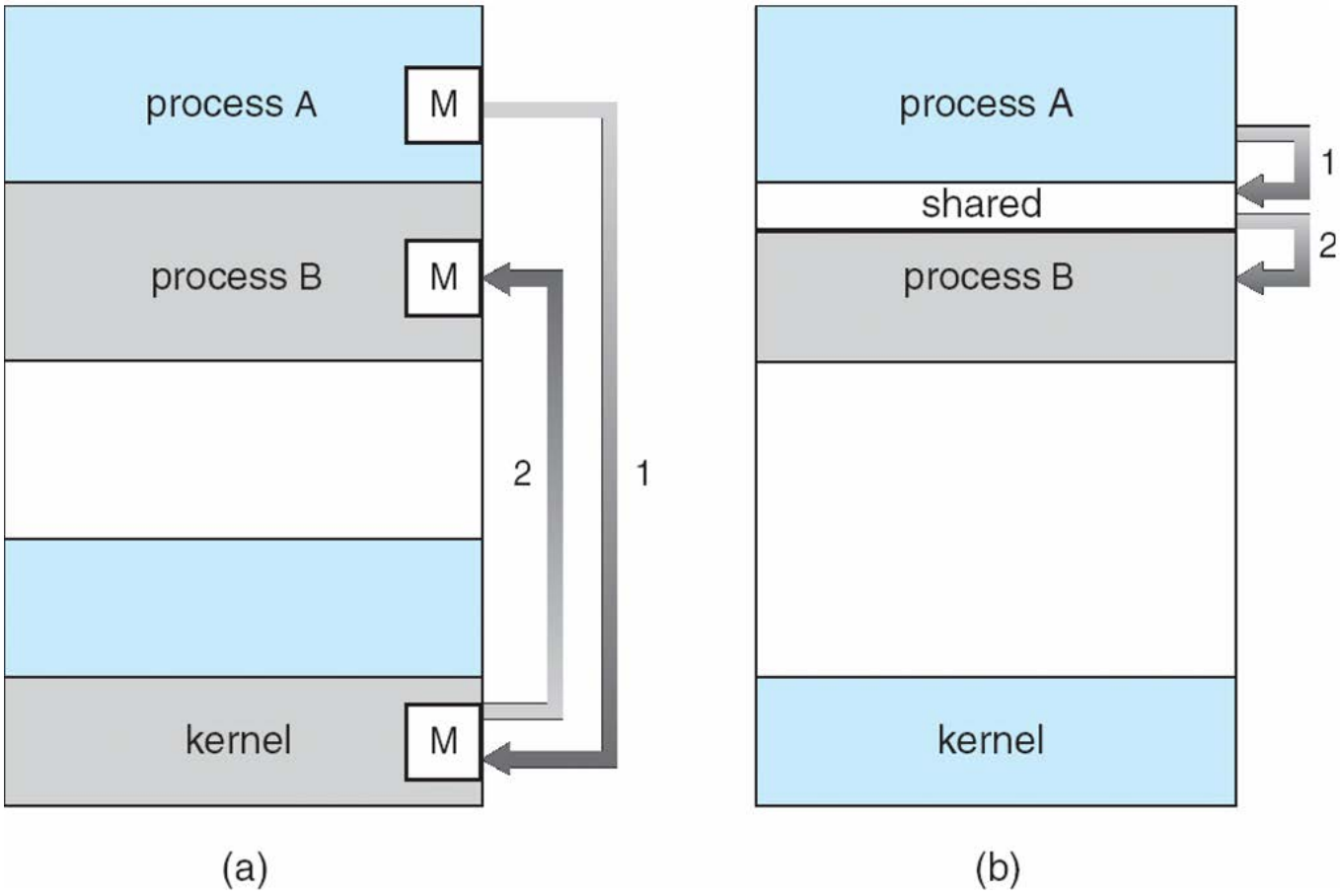
Process Termination

- 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**

Interprocess Communication

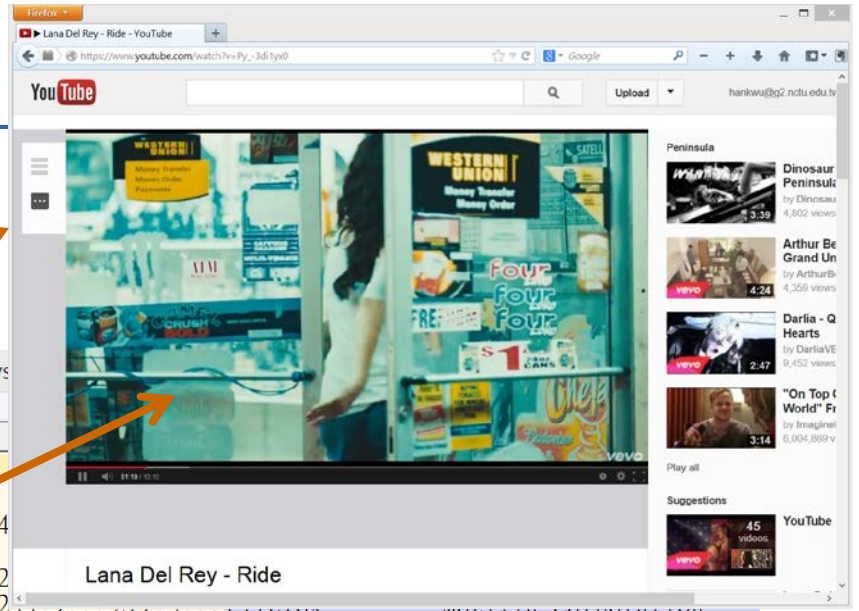
- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models



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



Process Explorer - Sysinternals: www.sys

File Options View Process Find Handle Users Help

Process	PID	CPU			
putty.exe	11228				
POWERPNT.EXE	5512	4			
procexp.exe	7760				
procexp64.exe	8224	2.09	2		
firefox.exe	3564	0.41	2		Mozilla Corporation
plugin-container.exe	5404	0.34	9,336 K	15,440 K	Plugin Conta... Mozilla Corporation
FlashPlayerPlugin_11_8_800_168.exe	9140	0.01	4,652 K	11,0	
FlashPlayerPlugin_11_8_800_168.exe	8848	2.58	100,9...	109,	
jusched.exe	8888		2,480 K	10,2	
iTunesHelper.exe	7404	< 0.01	3,444 K	11,5	
pageant.exe	8492		1,044 K	4,6	
AdobeARM.exe	2608	0.01	3,832 K	12,9	
chrome.exe	7768	0.02	175,4...	230,	
chrome.exe	2136		131,6...	127,	
chrome.exe	7500		94,724 K	99,3	

Type	Name
Key	HKLM\SYSTEM\ControlSet001\Services\WinSock2\Parameters\NameSpace_
Key	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\MMDevices\Audio\Re
Key	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\MMDevices\Audio\Re
Key	HKCU\Software\Classes
Mutant	\Sessions\8\BaseNamedObjects\MSCTF.Asm.MutexDefault8
Section	\Windows\Theme3398143603
Section	\Sessions\8\Windows\Theme3468387045
Section	\BaseNamedObjects__ComCatalogCache__
Section	\BaseNamedObjects__ComCatalogCache__
Section	\...\ASqmManifestVersion
Section	\...\ASqmManifest_27eb
Section	\Sessions\8\BaseNamedObjects\windows_shell_global_counters
Thread	FlashPlayerPlugin_11_8_800_168.exe(8848): 4504
Thread	FlashPlayerPlugin_11_8_800_168.exe(8848): 10116
Thread	FlashPlayerPlugin_11_8_800_168.exe(8848): 4504

CPU Usage: 8.17% | Commit Charge: 29.74% | Processes: 95 | Physical Usage: 29.95%

\Windows\Theme3398143603 Properties

Details | 安全性

Basic Information

Name: \Windows\Theme3398143603

Type: Section

Description: A memory mapped file or paging-file backed memory region.

Address: 0xFFFFF8A003B62E60

References

References: 13893608

Handles: 53

Quota Charges

Paged: 2272

Non-Paged: 168

Section Info

Type: COMMITED

Size: 0x00115000

確定 取消

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 – Producer

```
while (true) {
   /* Produce an item */

   while (( (in + 1) % BUFFER_SIZE)  == out)   ;   /* do nothing -- no
free buffers */

    buffer[in] = item;

    in = (in + 1) % BUFFER_SIZE;

}
```

Bounded Buffer – Consumer

```
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 – Message Passing

- 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?

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.

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full
  3. Unbounded capacity – infinite length
     Sender never waits

Examples of IPC Systems - POSIX

- POSIX Shared Memory
  - Process first creates shared memory segment

```
segment id = shmget(IPC_PRIVATE, size, S_IRUSR |
  S_IWUSR);
```

  - Process wanting access to that shared memory must attach to it

```
shared memory = (char *) shmat(id, NULL, 0);
```

  - Now the process could write to the shared memory

```
sprintf(shared memory, "Writing to shared memory");
```

  - When done a process can detach the shared memory from its address space

```
shmdt(shared memory);
```

Examples of IPC Systems - Mach

- Mach communication is message based
  - Even system calls are messages
  - Each task gets two mailboxes at creation- Kernel and Notify
  - Only three system calls needed for message transfer

`msg_send(), msg_receive(), msg_rpc()`

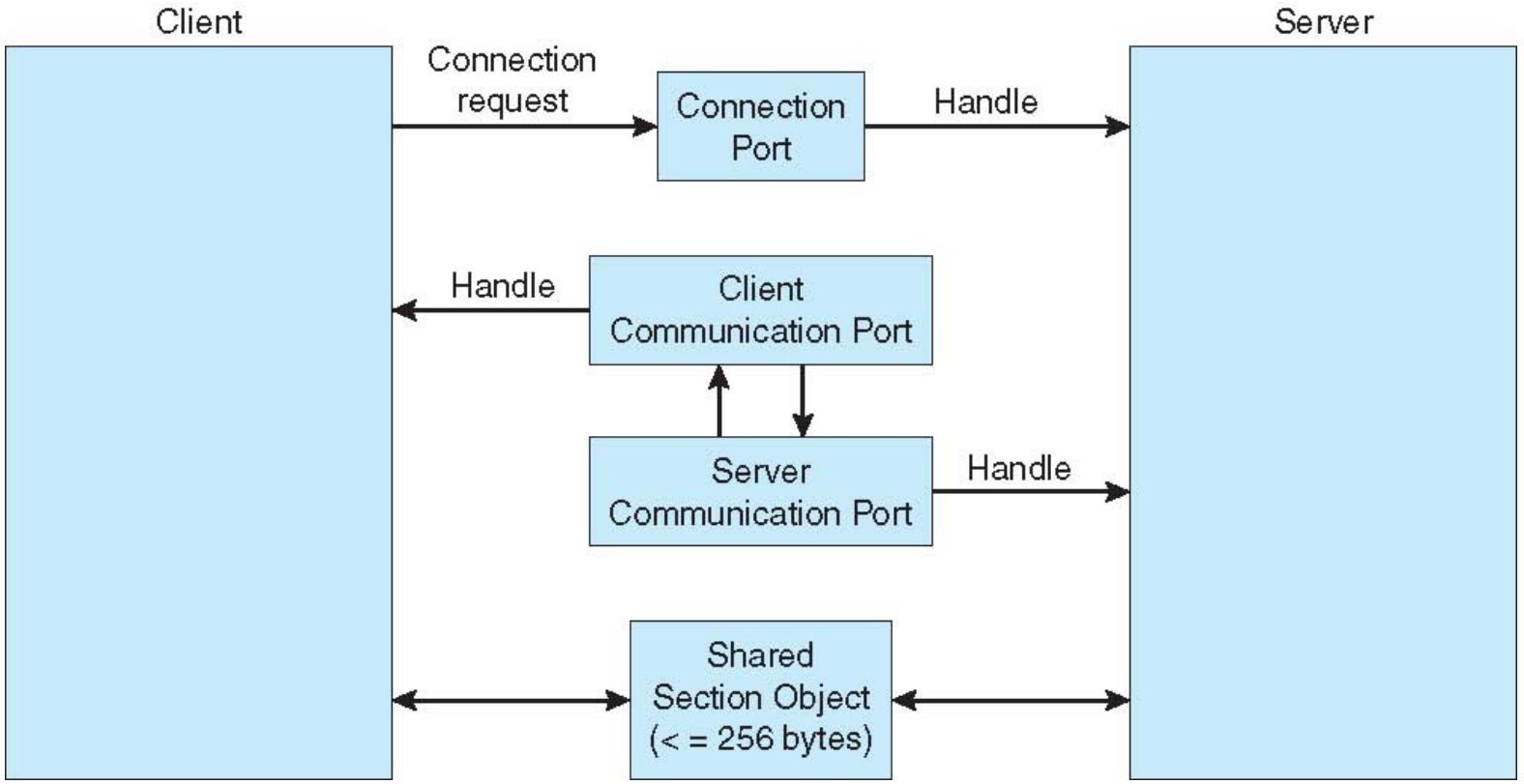
  - Mailboxes needed for commuication, created via

`port_allocate()`

Examples of IPC Systems – Windows XP

- Message-passing centric via local procedure call (LPC) facility
  - Only works between processes on the same system
  - Uses ports (like mailboxes) to establish and maintain communication channels
  - Communication works as follows:
    - The client opens a handle to the subsystem's connection port object
    - The client sends a connection request
    - The server creates two private communication ports and returns the handle to one of them to the client
    - The client and server use the corresponding port handle to send messages or callbacks and to listen for replies

Local Procedure Calls in Windows XP



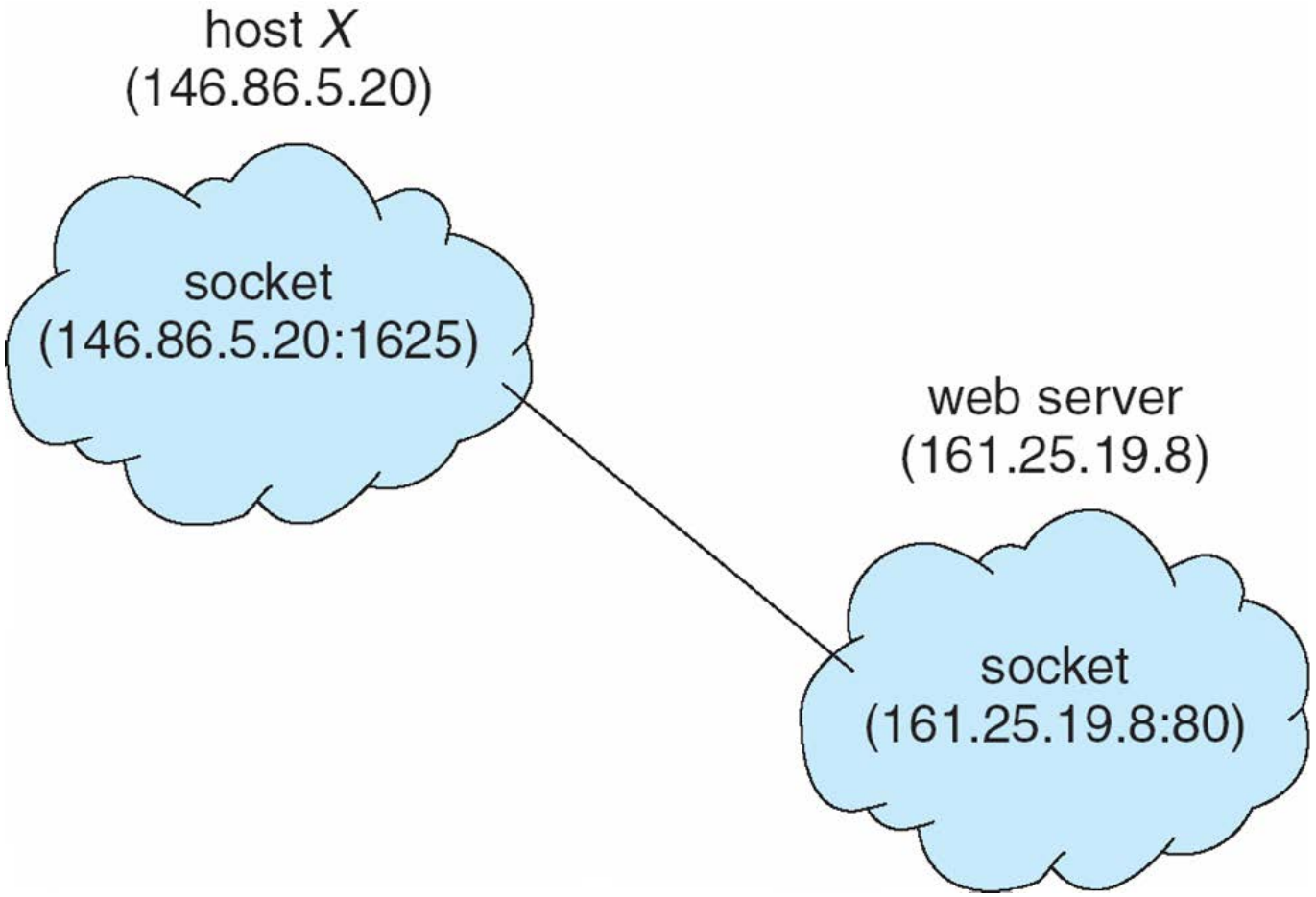
Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)

Sockets

- A socket is defined as an *endpoint for communication*
- Concatenation of IP address and port
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets

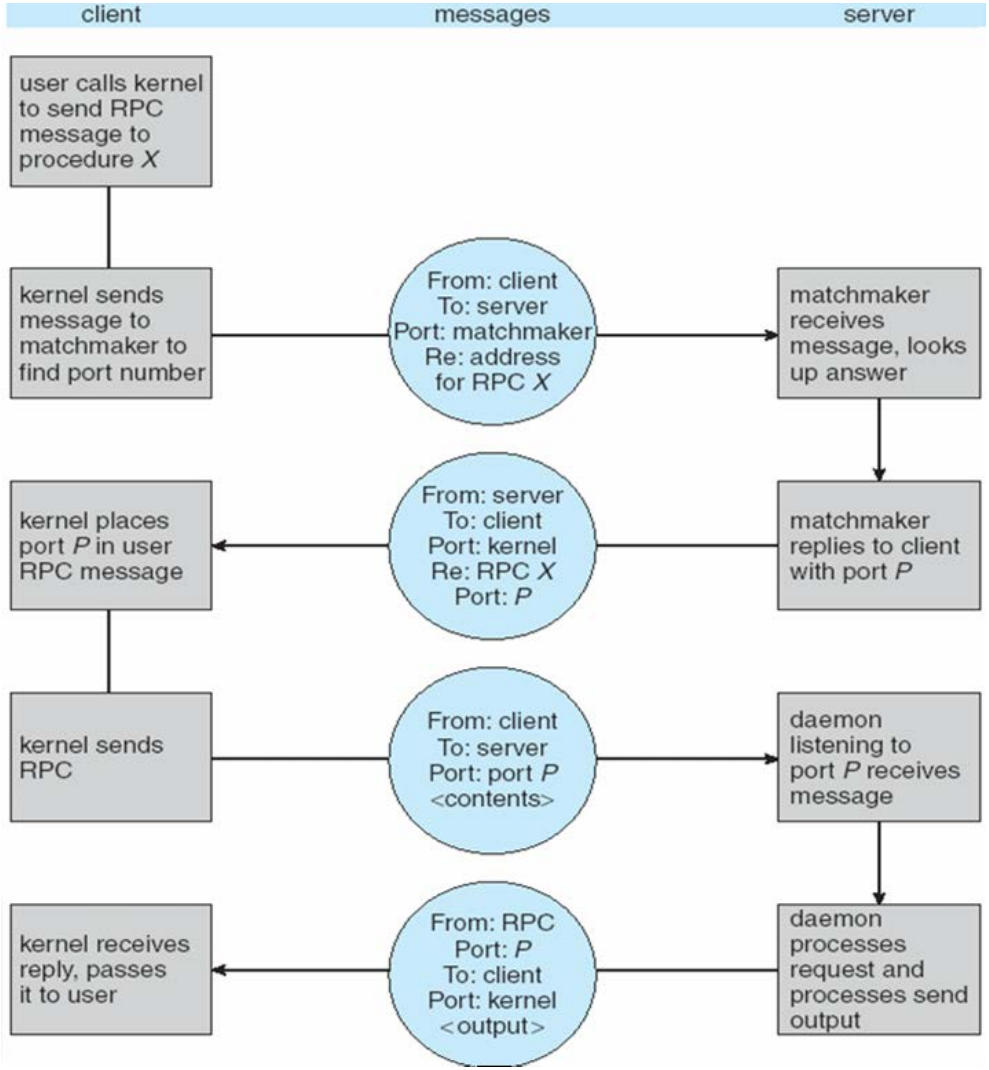
Socket Communication



Remote Procedure Calls

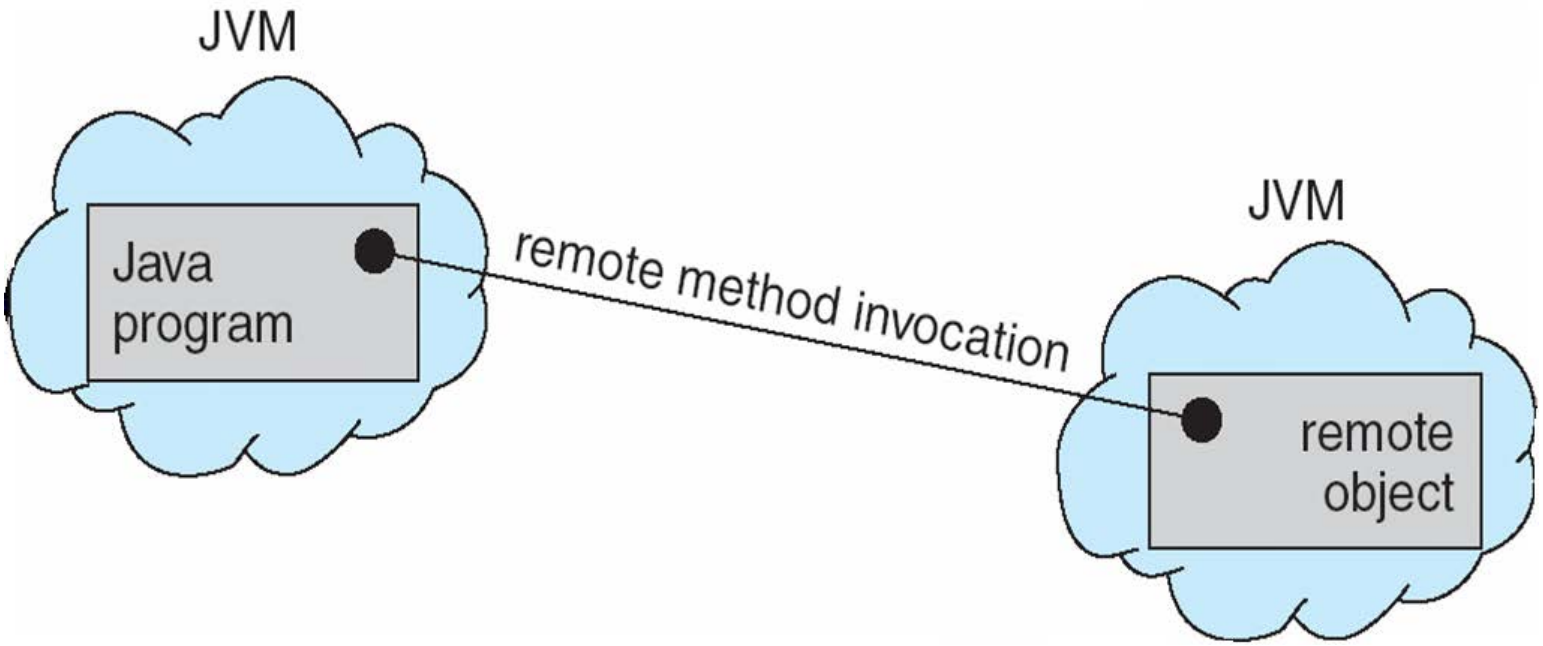
- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
- **Stubs** – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and *marshalls* the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and peforms the procedure on the server

Execution of RPC

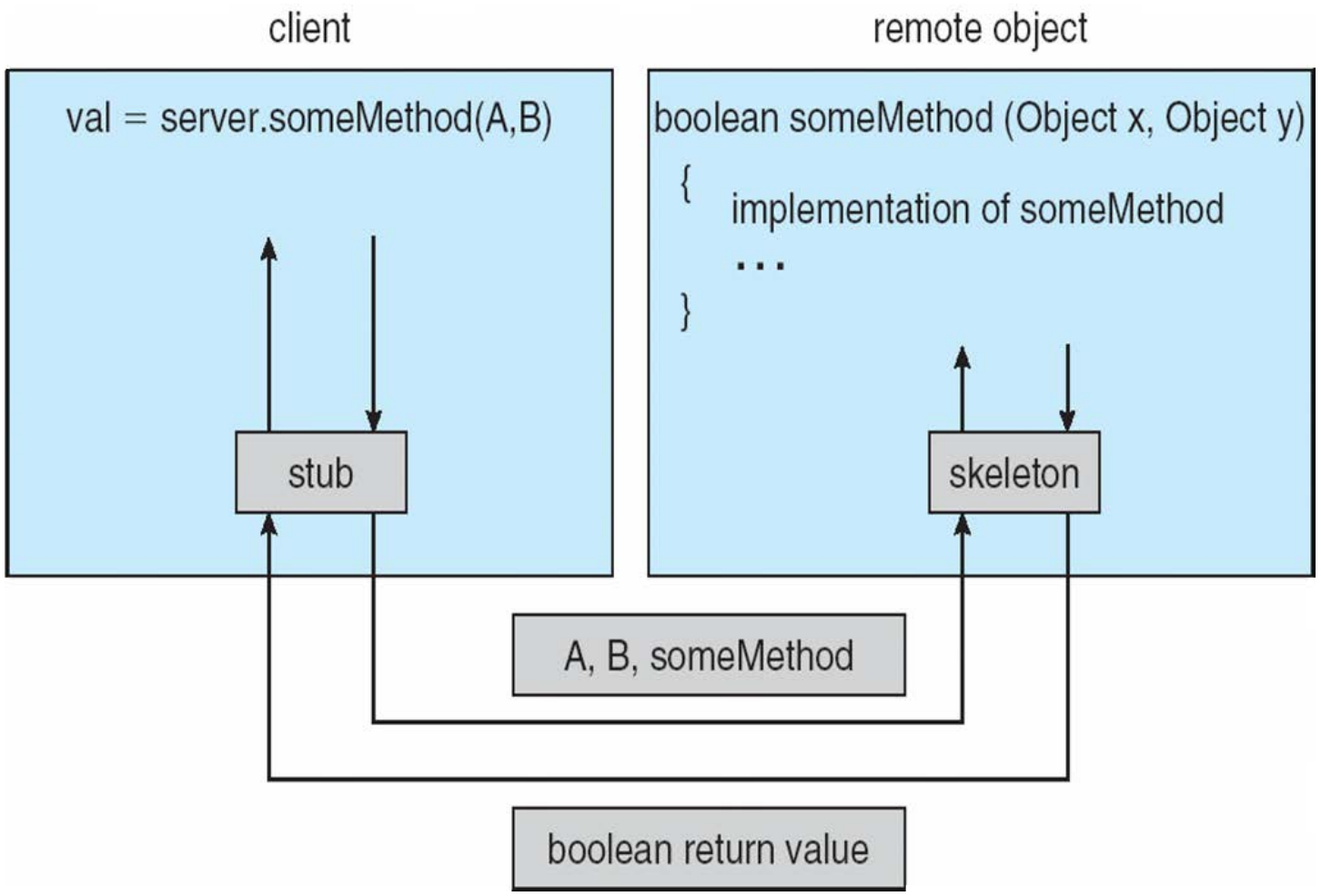


Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs
- RMI allows a Java program on one machine to invoke a method on a remote object



Marshalling Parameters

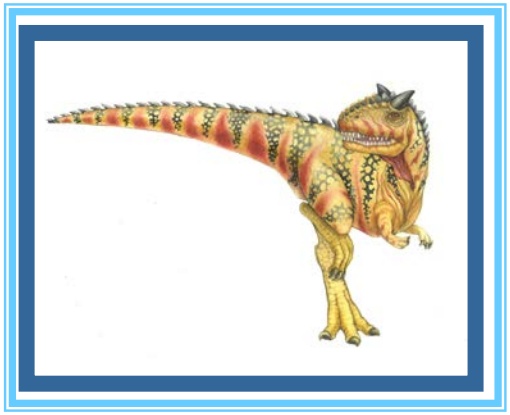


Reflection

- Emulate shared memory with message passing
- Emulate message passing with shared memory

- How to implement RPC ?
- Advantage / Disadvantage of RPC

End of Chapter 3



Silberschatz, Galvin and Gagne ©2009