Chap. 3) Processes

경희대학교 컴퓨터공학과

조 진 성

Program vs. Process

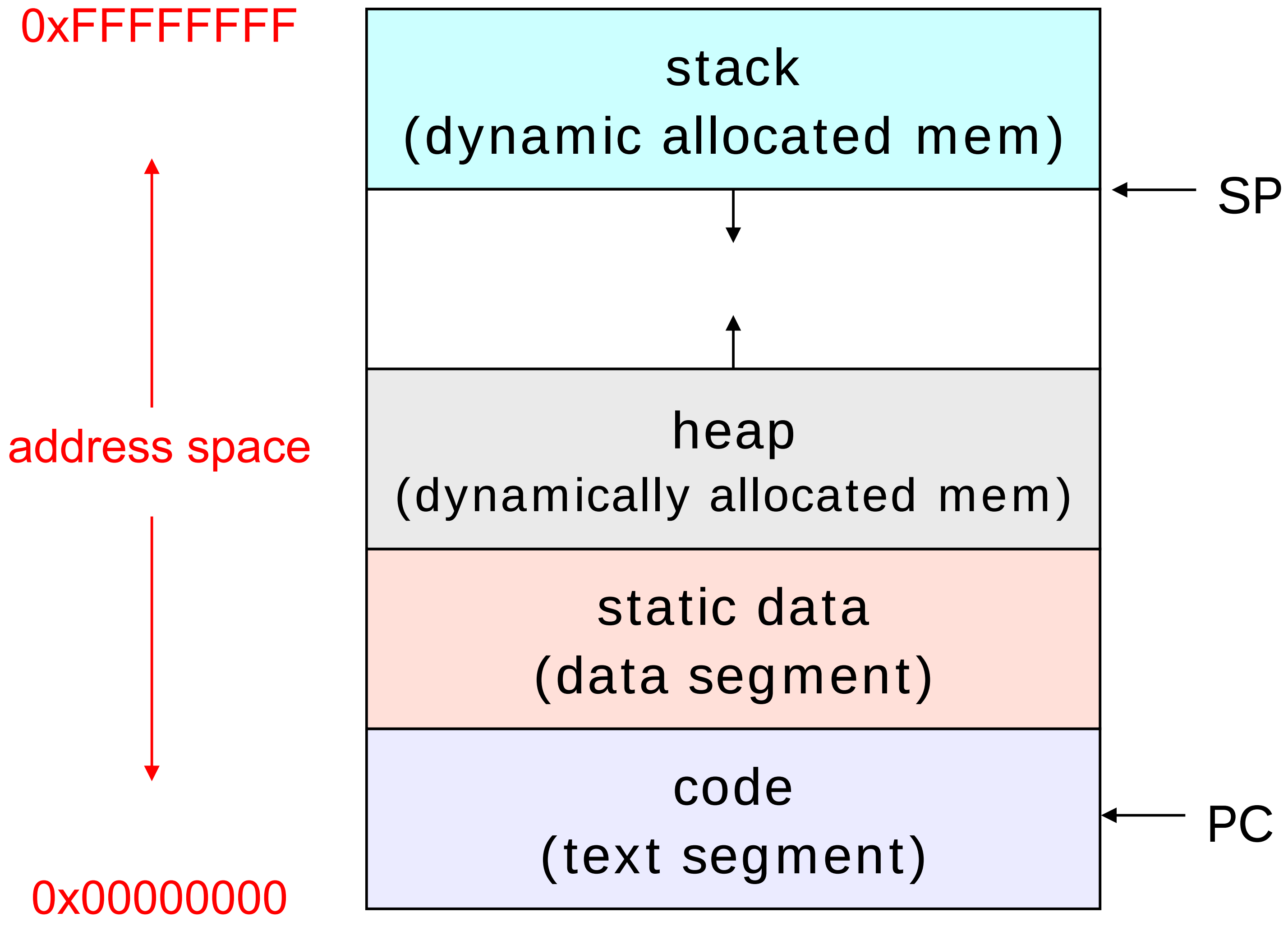
- Program
  - ✓ Executable file on a disk
  - ✓ Loaded into memory and executed by the kernel

- Process
  - ✓ Executing instance of a program
  - ✓ The basic unit of execution and scheduling
  - ✓ A process is named using its process ID (PID)

Process Concept

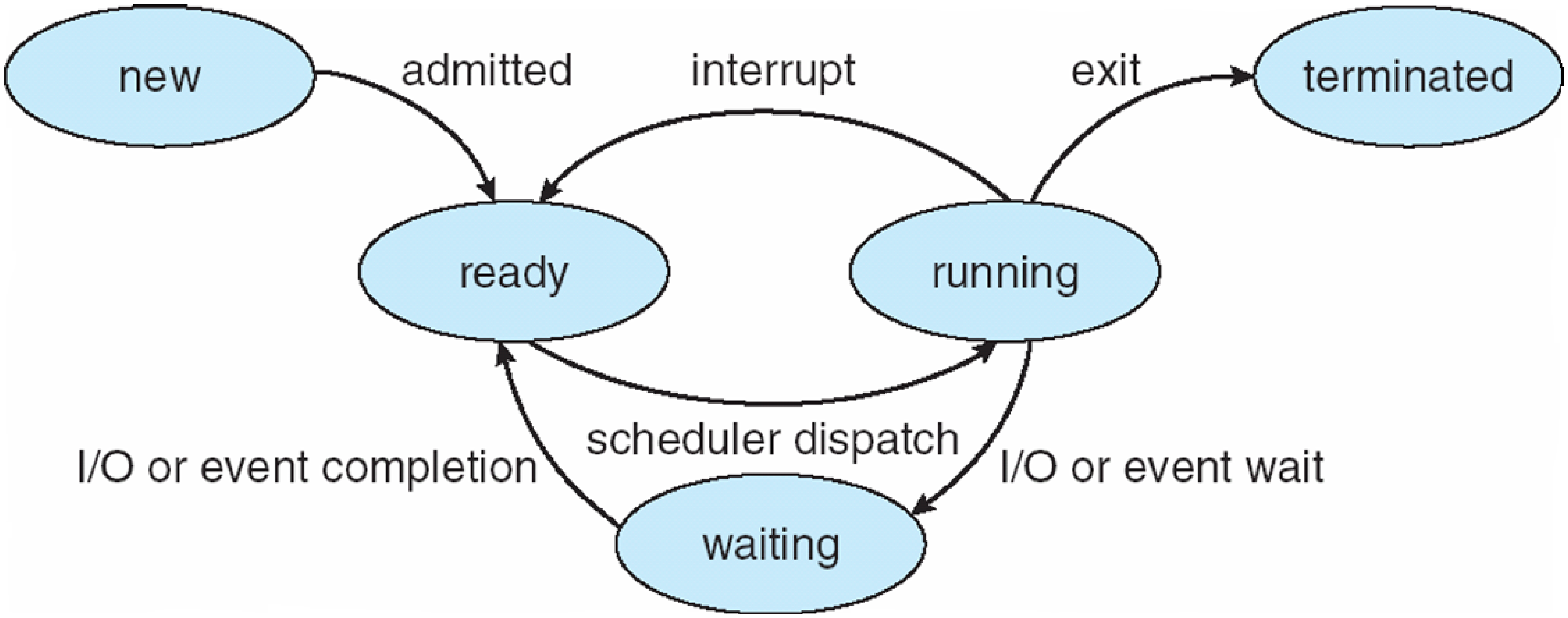
- What is the process?
  - ✓ An instance of a program in execution
  - ✓ An encapsulation of the flow of control in a program
  - ✓ A dynamic and active entity
  - ✓ The basic unit of execution and scheduling
  - ✓ A process is named using its process ID (PID)

Process Address Space



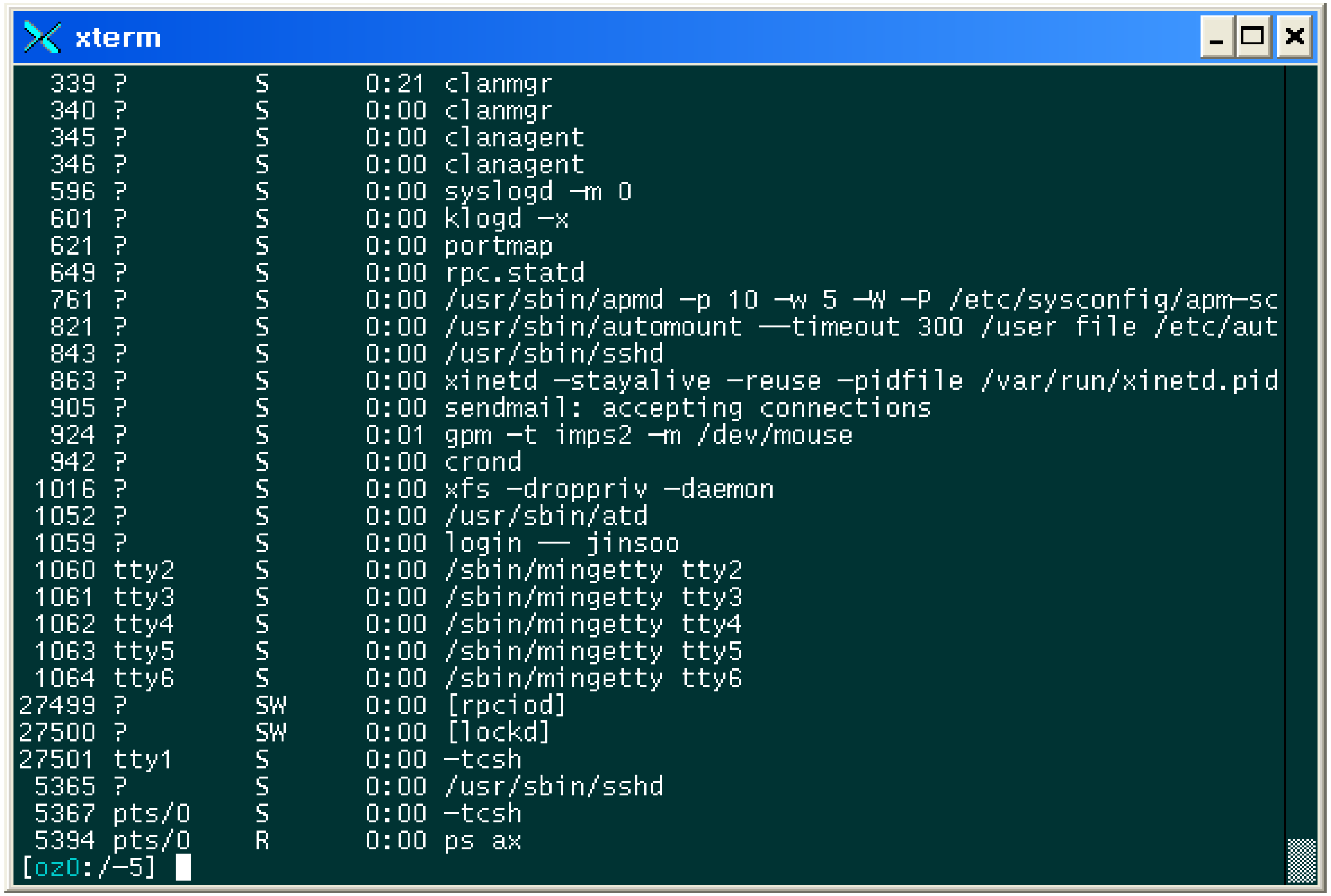
Process State

- State diagram



Process State Transition

- Linux example



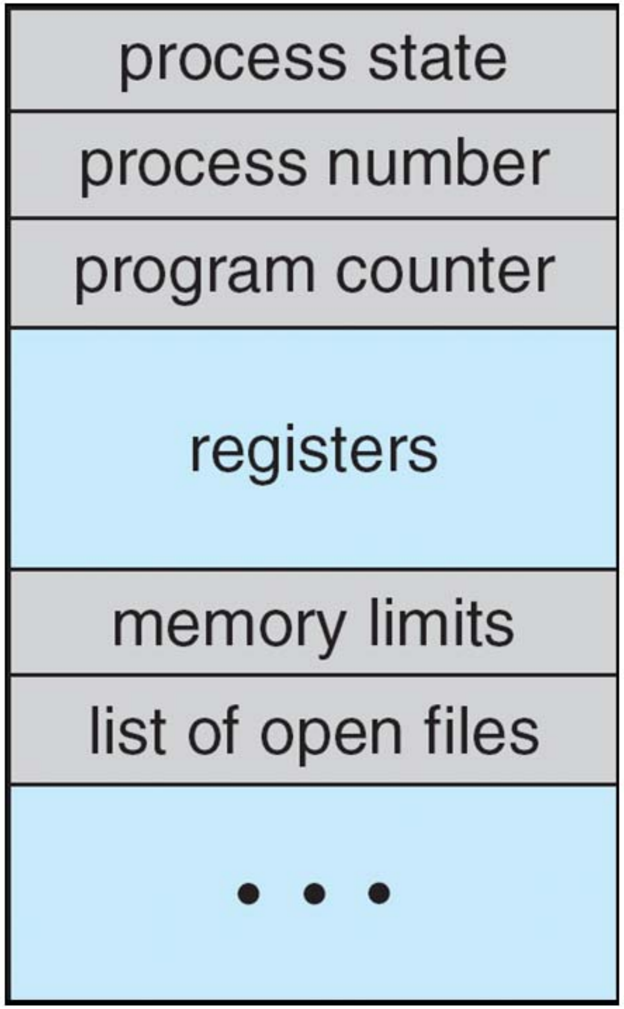
```
   339 ?        S      0:21 clanmgr
   340 ?        S      0:00 clanmgr
   345 ?        S      0:00 clanagent
   346 ?        S      0:00 clanagent
   596 ?        S      0:00 syslogd -m 0
   601 ?        S      0:00 klogd -x
   621 ?        S      0:00 portmap
   649 ?        S      0:00 rpc.statd
   761 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-sc
   821 ?        S      0:00 /usr/sbin/automount --timeout 300 /user file /etc/aut
   843 ?        S      0:00 /usr/sbin/sshd
   863 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pid
   905 ?        S      0:00 sendmail: accepting connections
   924 ?        S      0:01 gpm -t imps2 -m /dev/mouse
   942 ?        S      0:00 crond
  1016 ?        S      0:00 xfs -droppriv -daemon
  1052 ?        S      0:00 /usr/sbin/atd
  1059 ?        S      0:00 login -- jinsoo
  1060 tty2     S      0:00 /sbin/mingetty tty2
  1061 tty3     S      0:00 /sbin/mingetty tty3
  1062 tty4     S      0:00 /sbin/mingetty tty4
  1063 tty5     S      0:00 /sbin/mingetty tty5
  1064 tty6     S      0:00 /sbin/mingetty tty6
 27499 ?        SW     0:00 [rpciod]
 27500 ?        SW     0:00 [lockd]
 27501 tty1     S      0:00 -tcsh
  5365 ?        S      0:00 /usr/sbin/sshd
  5367 pts/0    S      0:00 -tcsh
  5394 pts/0    R      0:00 ps ax
[oz0:/-5] 
```

R: Runnable
S: Sleeping
T: Traced or Stopped
D: Uninterruptible Sleep
Z: Zombie

W: No resident pages
<: High-priority task
N: Low-priority task
L: Has pages locked into memory

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

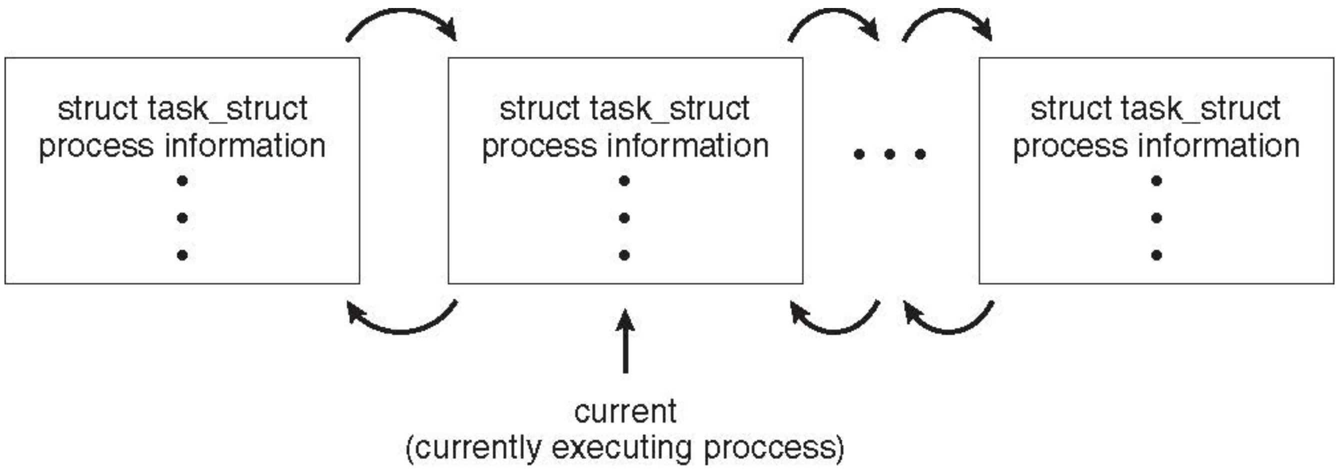


- *Cf) task_struct in Linux*
  - ✓ 1456 bytes as of Linux 2.4.18

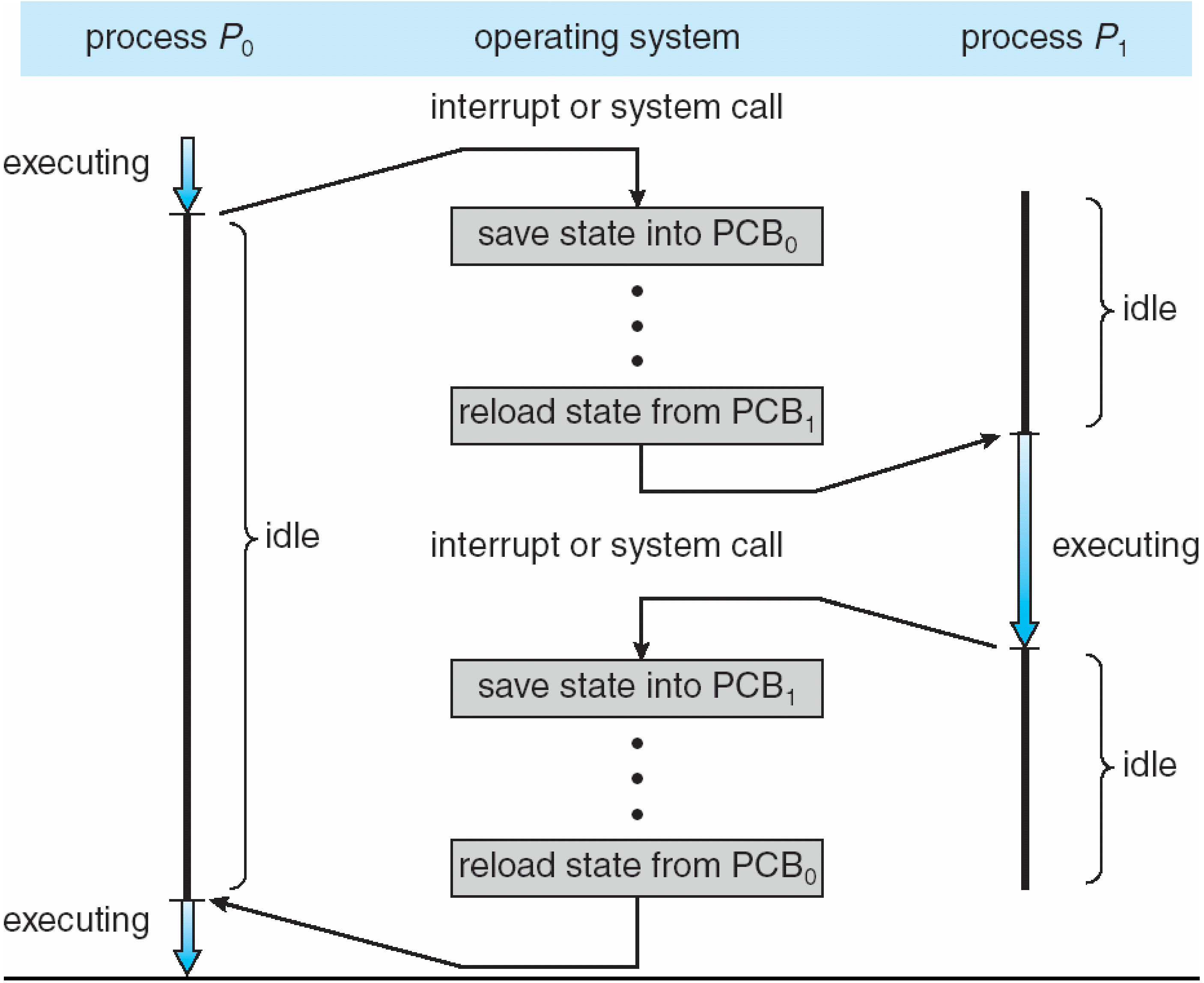
Process Control Block (PCB)

Represented by the C structure `task_struct`

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```



Context Switch (CPU Switch)



Context Switch

- The act of switching the CPU from one process to another

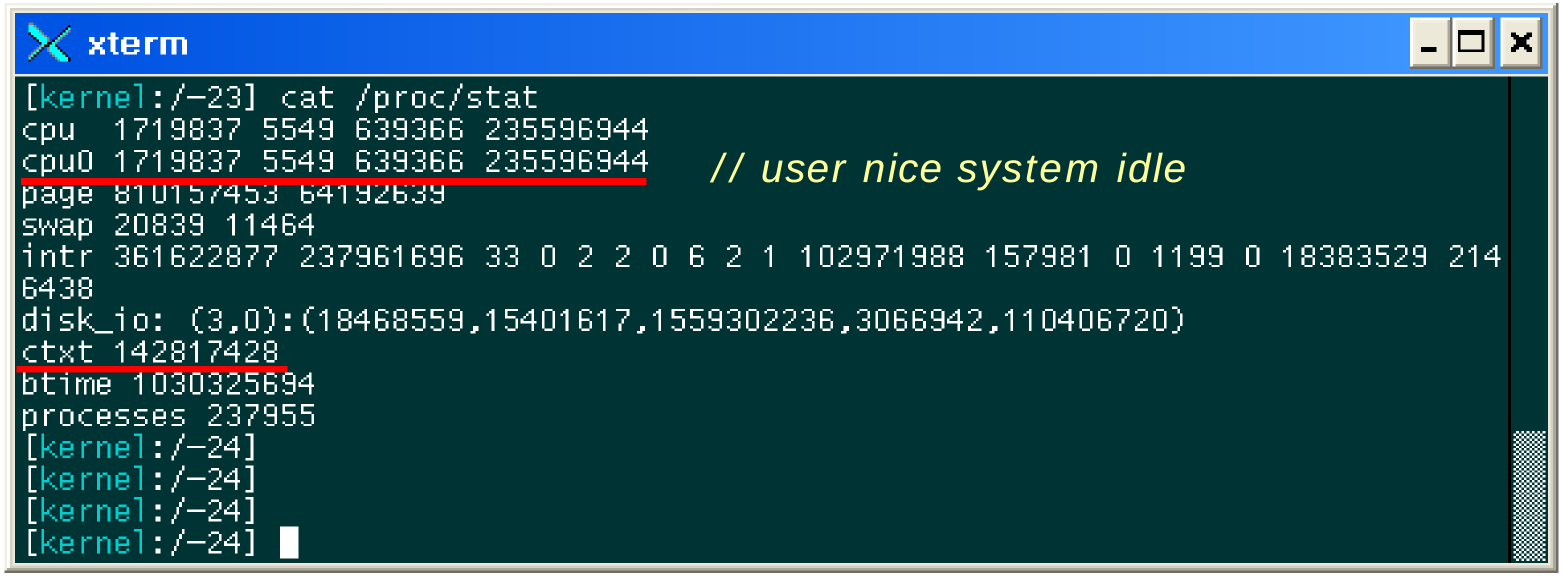
- Administrative overhead
  - ✓ saving and loading registers and memory maps
  - ✓ flushing and reloading the memory cache
  - ✓ updating various tables and lists, etc.

- Context switch overhead is dependent on hardware support
  - ✓ Multiple register sets in UltraSPARC
  - ✓ Advanced memory management techniques may require extra data to be switched with each context

- 100s or 1000s of switches/s typically

Context Switch

- Linux example
  - ✓ Total 237,961,696 ticks = 661 hours = 27.5 days
  - ✓ Total 142,817,428 context switches
  - ✓ Roughly 60 context switches / sec

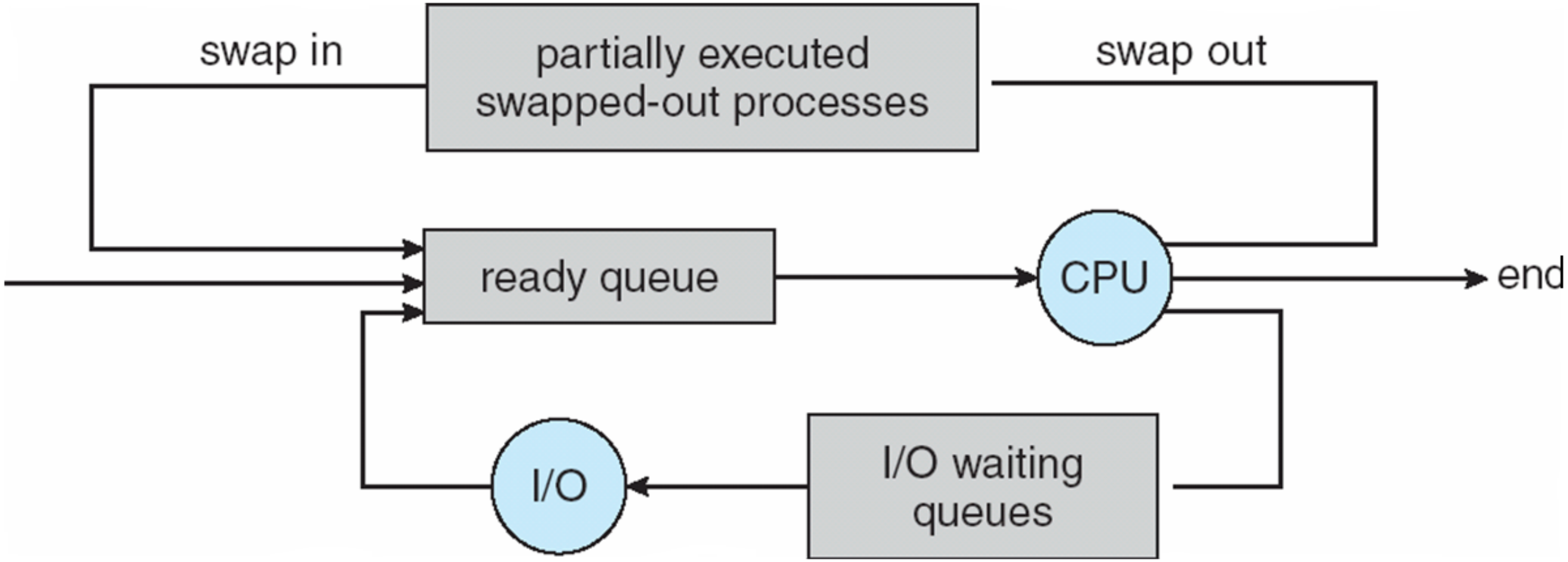


```
[kernel:/-23] cat /proc/stat
cpu  1719837 5549 639366 235596944
cpu0 1719837 5549 639366 235596944
page 810157453 64192639
swap 20839 11464
intr 361622877 237961696 33 0 2 2 0 6 2 1 102971988 157981 0 1199 0 18383529 214
6438
disk_io: (3,0):(18468559,15401617,1559302236,3066942,110406720)
ctxt 142817428
btime 1030325694
processes 237955
[kernel:/-24]
[kernel:/-24]
[kernel:/-24]
[kernel:/-24]
```

// user nice system idle

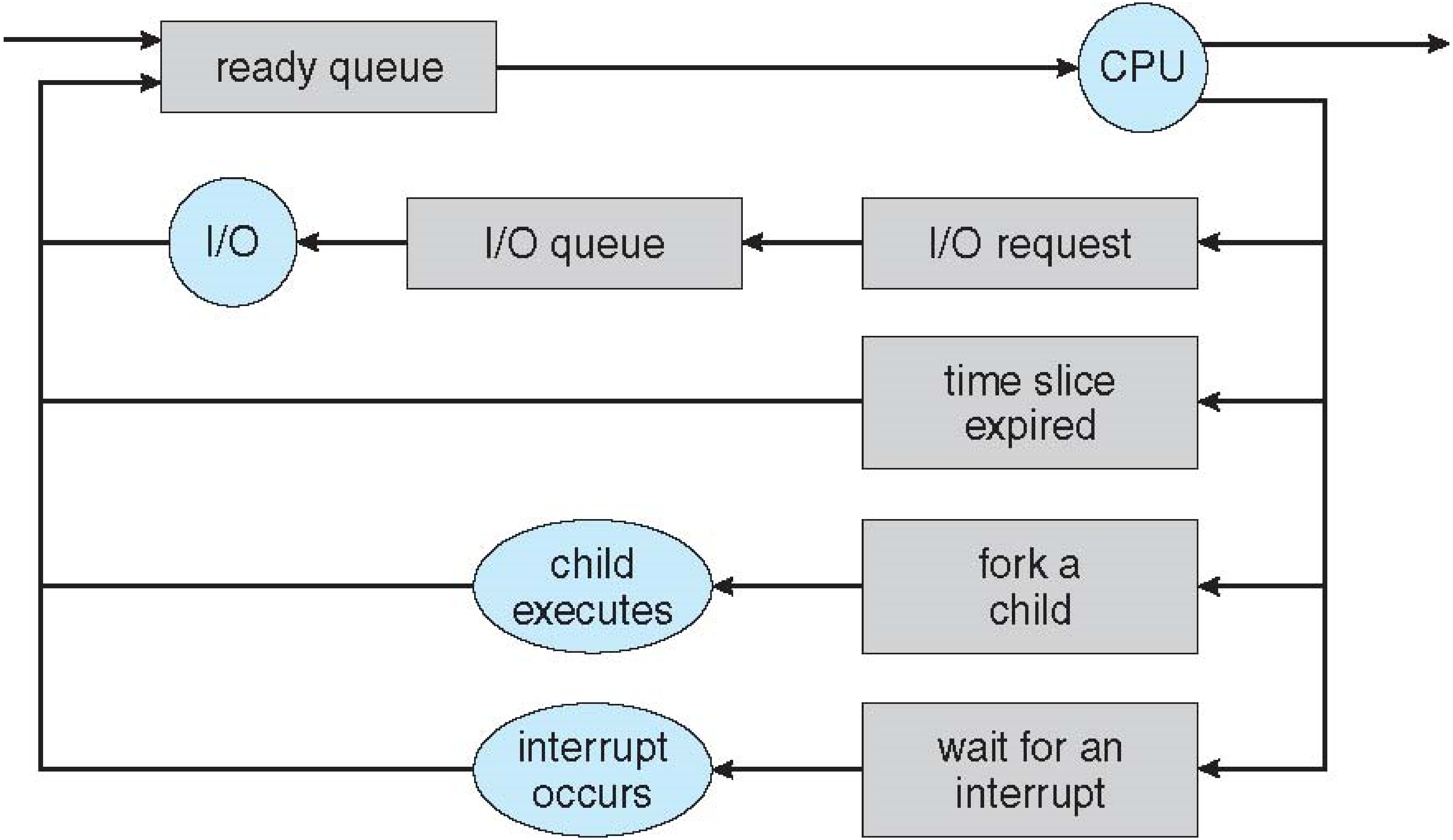
Schedulers

- 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
- Medium-term scheduler (or swapper)



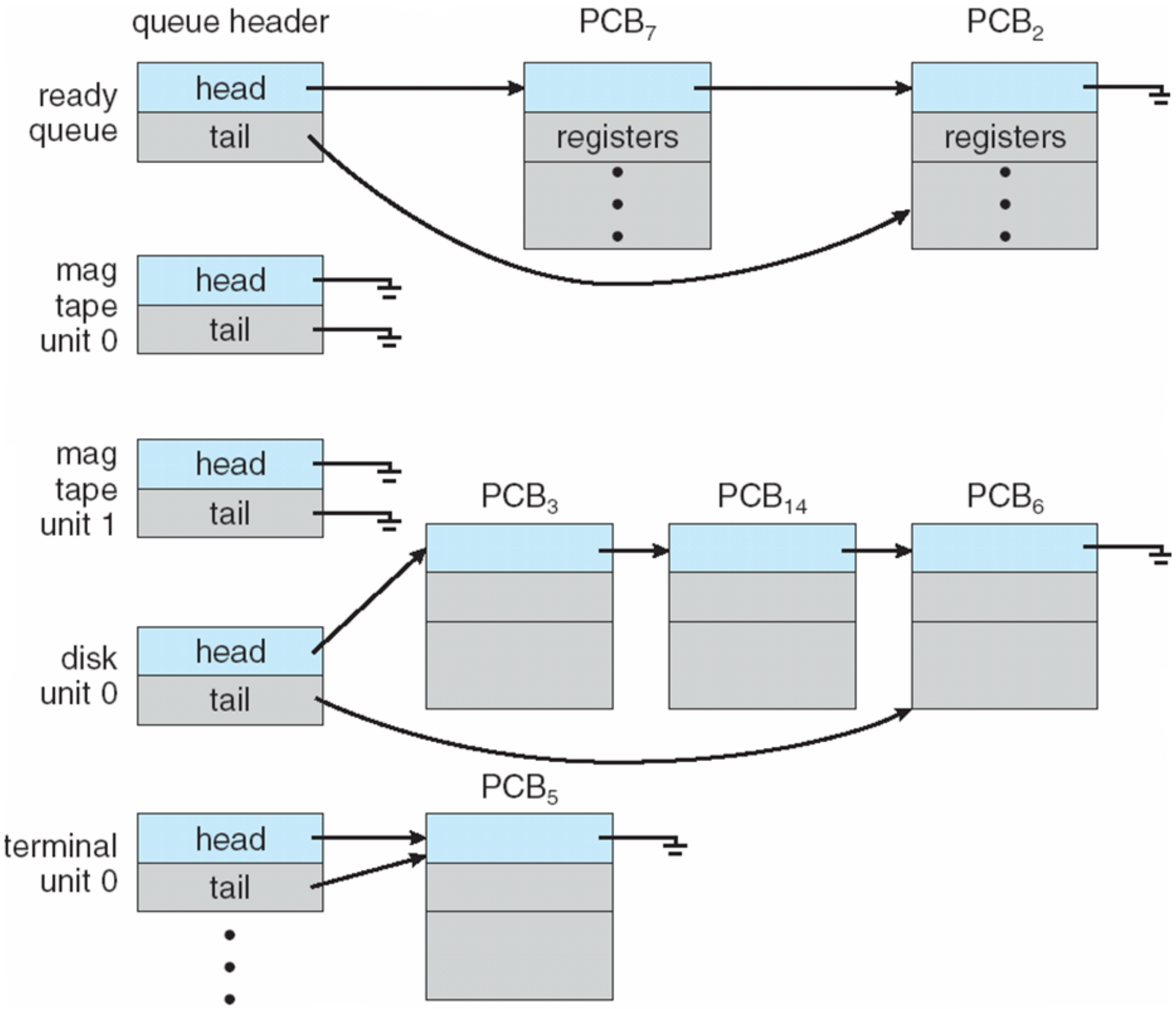
Representation of Process Scheduling

- Queueing diagram



Representation of Process Scheduling

- Ready queue and various I/O device queues



Operations on Processes

- Process creation
  - ✓ fork()
- Process execution
  - ✓ exec()
- Process termination
  - ✓ exit()
  - ✓ _exit()
  - ✓ abort()
  - ✓ wait()
- Cooperating processes
  - ✓ Inter-Process Communication (IPC)

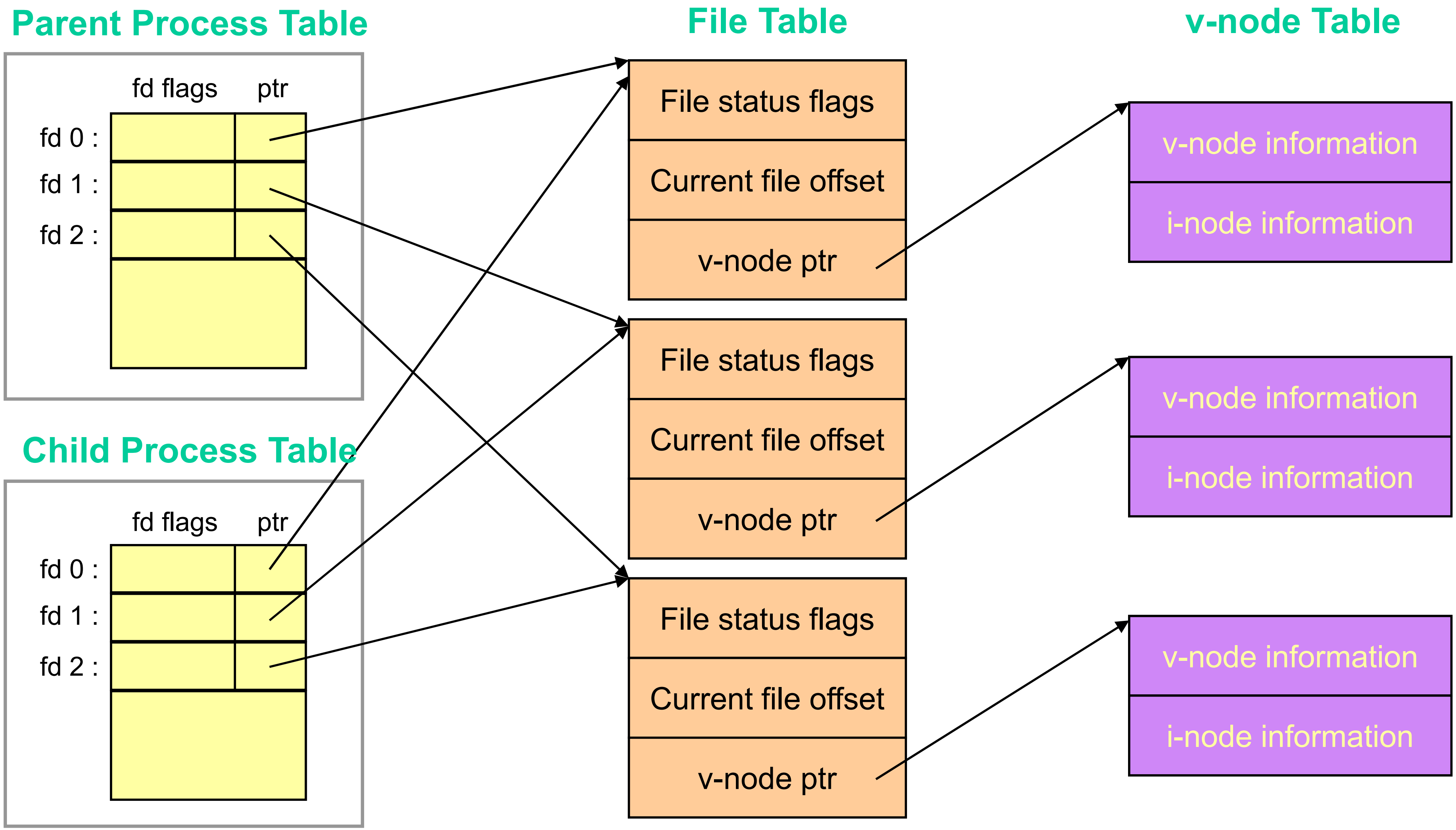
Process Creation: Unix/Linux

```
int fork()
```

- fork()
  - ✓ Creates and initializes a new PCB
  - ✓ Creates and initializes a new address space
  - ✓ Initializes the address space with a copy of the entire contents of the address space of the parent
  - ✓ Initializes the kernel resources to point to the resources used by parent (e.g., open files)
  - ✓ Places the PCB on the ready queue
  - ✓ Returns the child's PID to the parent, and zero to the child

Process Creation: Unix/Linux

- Sharing of open files between parent and child after **`fork`**



fork()

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

int main()
{
    int pid;

    if ((pid = fork()) == 0)
        /* child */
        printf (“Child of %d is %d\n”, getppid(), getpid());
    else
        /* parent */
        printf (“I am %d. My child is %d\n”, getpid(), pid);
}
```

fork(): Example Output

```
% ./a.out
I am 31098. My child is 31099.
Child of 31098 is 31099.

% ./a.out
Child of 31100 is 31101.
I am 31100. My child is 31101.
```

Why fork()?

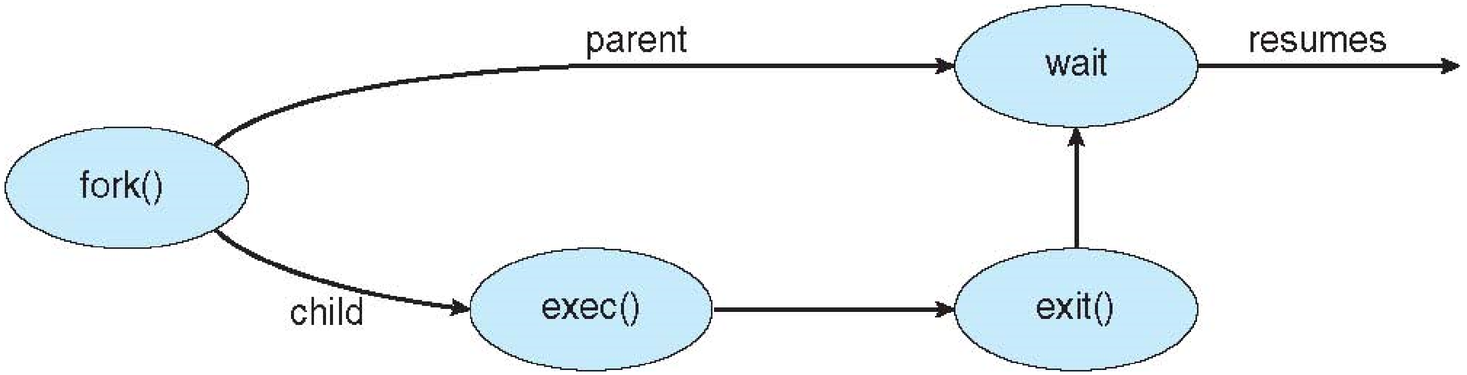
- Very useful when the child…
  - ✓ is cooperating with the parent
  - ✓ relies upon the parent’s data to accomplish its task
  - ✓ Example: Web server

```
While (1) {
    int sock = accept();
    if ((pid = fork()) == 0) {
        /* Handle client request */
    } else {
        /* Close socket */
    }
}
```

Process Execution: Unix/Linux

```
int exec (char *prog, char *argv[])
```

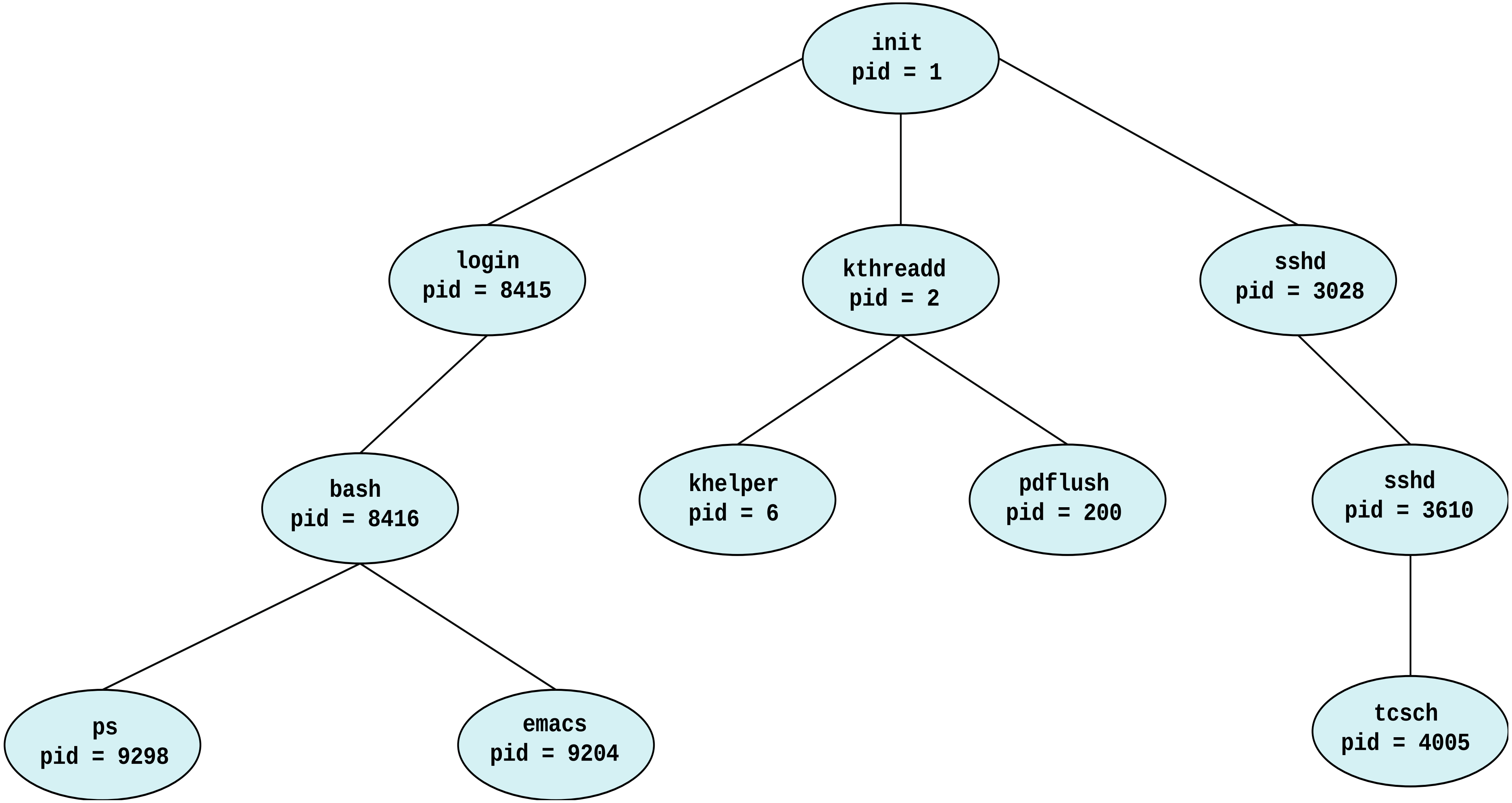
- exec()
  - ✓ Stops the current process
  - ✓ Loads the program “prog” into the process’ address space
  - ✓ Initializes hardware context and args for the new program
  - ✓ Places the PCB on the ready queue
    - Note: exec() does not create a new process
  - ✓ What does it mean for exec() to return?



Simplified Unix/Linux Shell

```
int main()
{
    while (1) {
        char *cmd = read_command();
        int pid;
        if ((pid = fork()) == 0) {
            /* Manipulate stdin/stdout/stderr for
               pipes and redirections, etc. */
            exec(cmd);
            panic(“exec failed!”);
        } else {
            wait (pid);
        }
    }
}
```

A Process Tree in Linux



Process Creation/Execution: Windows

```
BOOL CreateProcess (char *prog, char *args, ...)
```

- CreateProcess()
  - ✓ Creates and initializes a new PCB
  - ✓ Creates and initializes a new address space
  - ✓ Loads the program specified by “prog” into the address space
  - ✓ Copies “args” into memory allocated in address space
  - ✓ Initializes the hardware context to start execution at main
  - ✓ Places the PCB on the ready queue

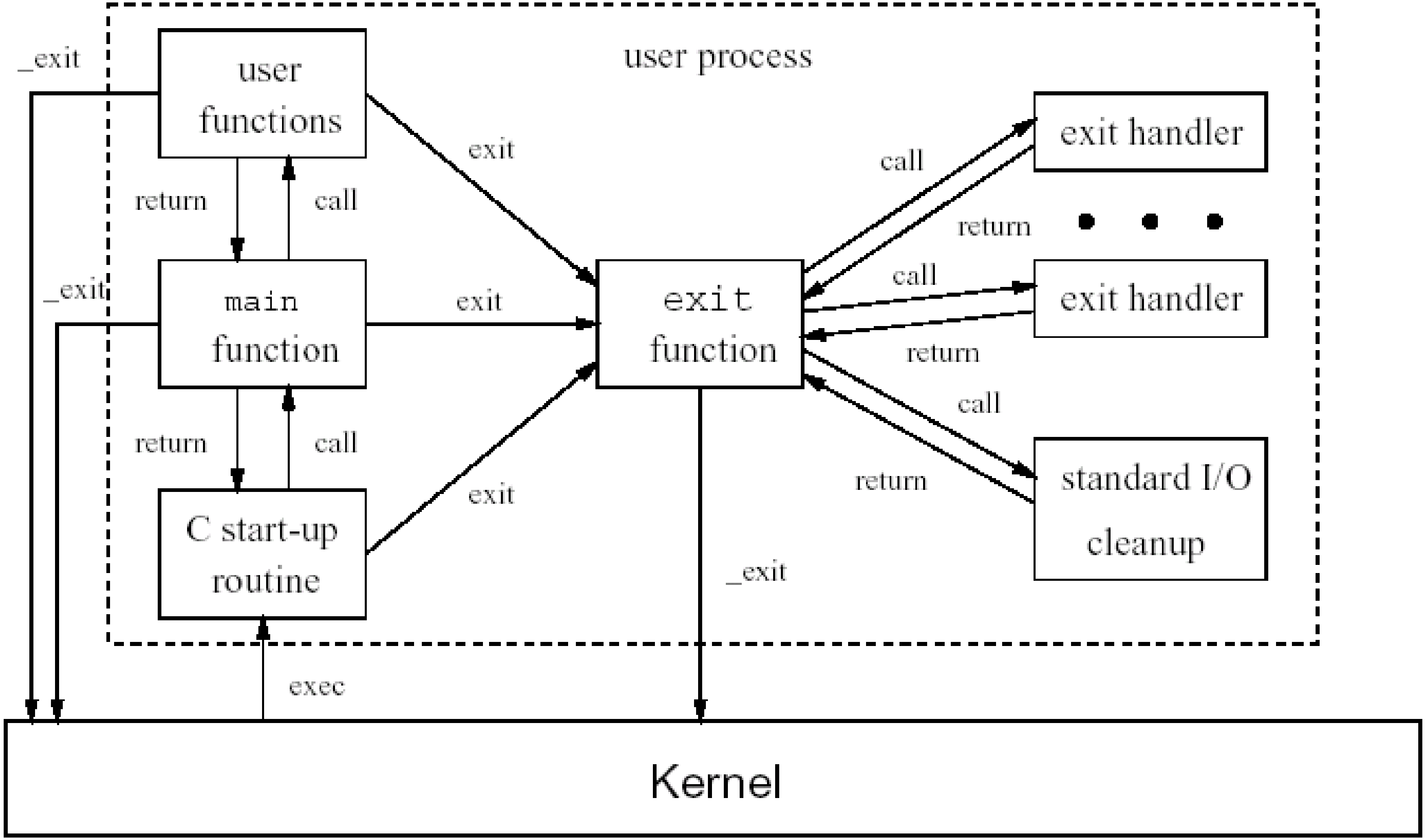
Process Termination

- Normal termination
  - ✓ return from **`main()`**
  - ✓ calling **`exit()`**
  - ✓ calling **`_exit()`**

- Abnormal termination
  - ✓ calling **`abort()`**
  - ✓ terminated by a signal

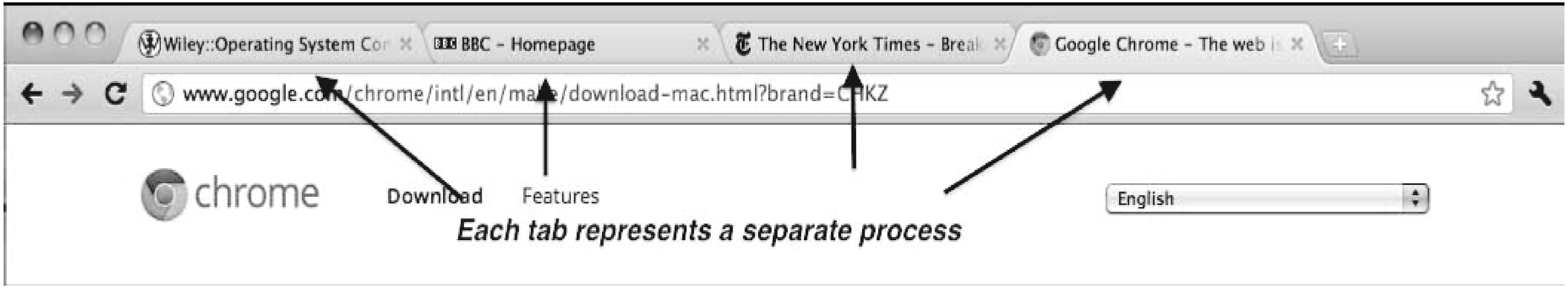
- Wait for termination of a child process
  - ✓ calling **`wait()`**
  - ✓ If no parent waiting (did not invoke wait()), process is a zombie
  - ✓ If parent terminated without invoking wait, process is an orphan

Start and Termination of a C Program



Multiprocess in Application Program

- Google Chrome Browser is multiprocess with 3 different types of processes:
  - ✓ Browser process manages user interface, disk and network I/O
  - ✓ Renderer process renders web pages, deals with HTML, Javascript
    - A new renderer created for each website opened
    - Runs in sandbox restricting disk and network I/O, minimizing effect of security exploits
  - ✓ Plug-in process for each type of plug-in


Wiley::Operating System Cor
BBC - Homepage
The New York Times - Break
Google Chrome - The web i
www.google.com/chrome/intl/en/make/download-mac.html?brand=CHKZ
chrome
Download
Features
English

Each tab represents a separate process

Multiprocess in Mobile Systems

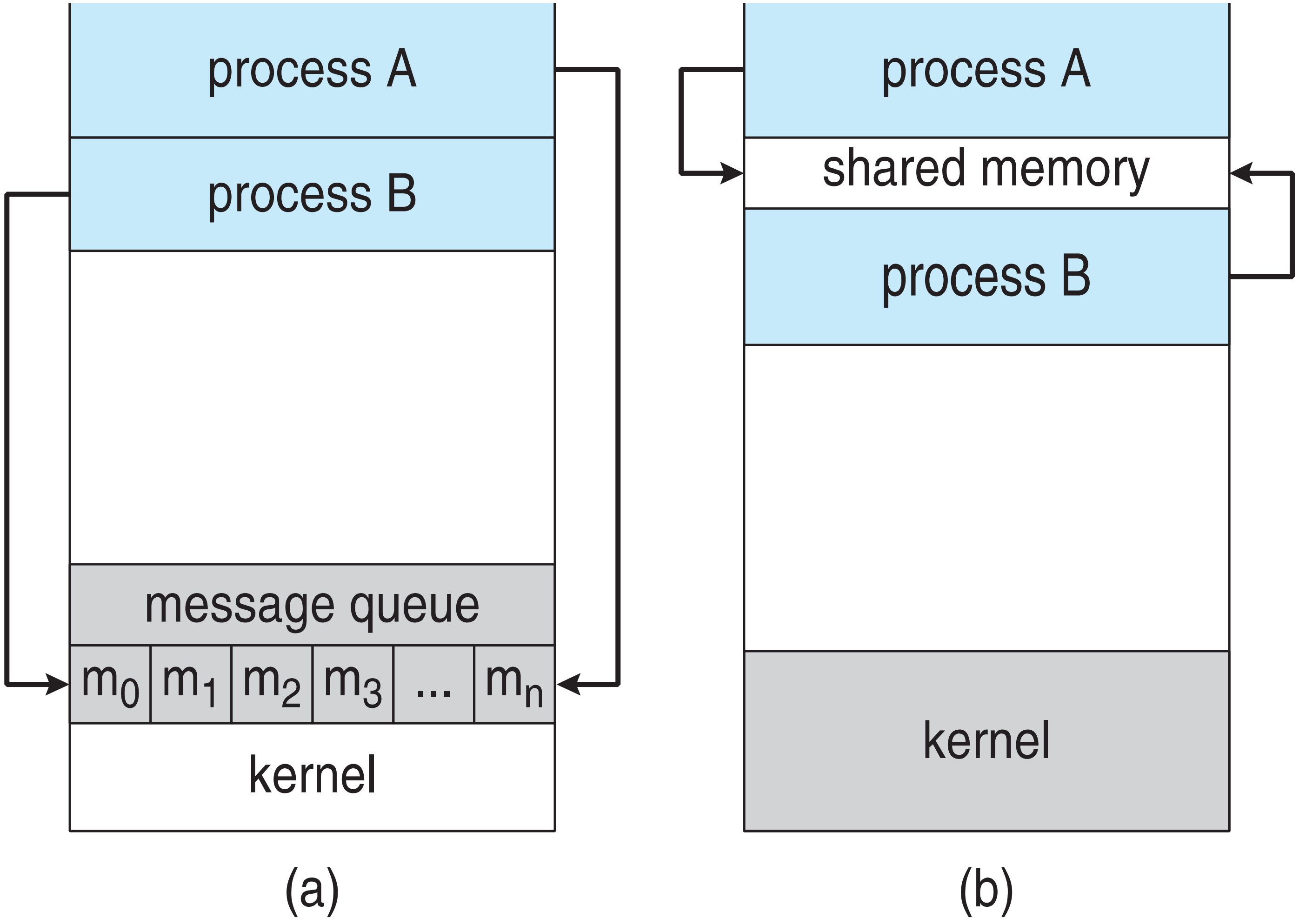
- Some mobile systems (e.g., early version of iOS) allow only one process to run, others suspended

- Due to screen real estate, user interface limits iOS provides for a
  - ✓ Single foreground process- controlled via user interface
  - ✓ Multiple background processes– in memory, running, but not on the display, and with limits
  - ✓ Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback

- Android runs foreground and background, with fewer limits
  - ✓ Background process uses a service to perform tasks
  - ✓ Service can keep running even if background process is suspended
  - ✓ Service has no user interface, small memory use

Inter-Process Communication (IPC)

- Communication models
  - ✓ (a) message passing vs. (b) shared memory



Inter-Process Communication (IPC)

- Cooperating processes
  - ✓ Example: Bounded buffer problem (Producer-Consumer problem)

Producer

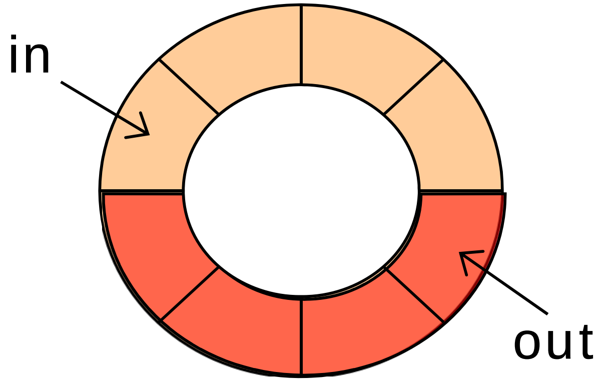
```
void producer(data)
{

 while (count==N) ;
 buffer[in] = data;
 in = (in+1) % N;
 count++;

}
```

int count;

struct item buffer[N];
int in, out;



Consumer

```
void consumer(data)
{

 while (count==0) ;
 data = buffer[out];
 out = (out+1) % N;
 count--;

}
```

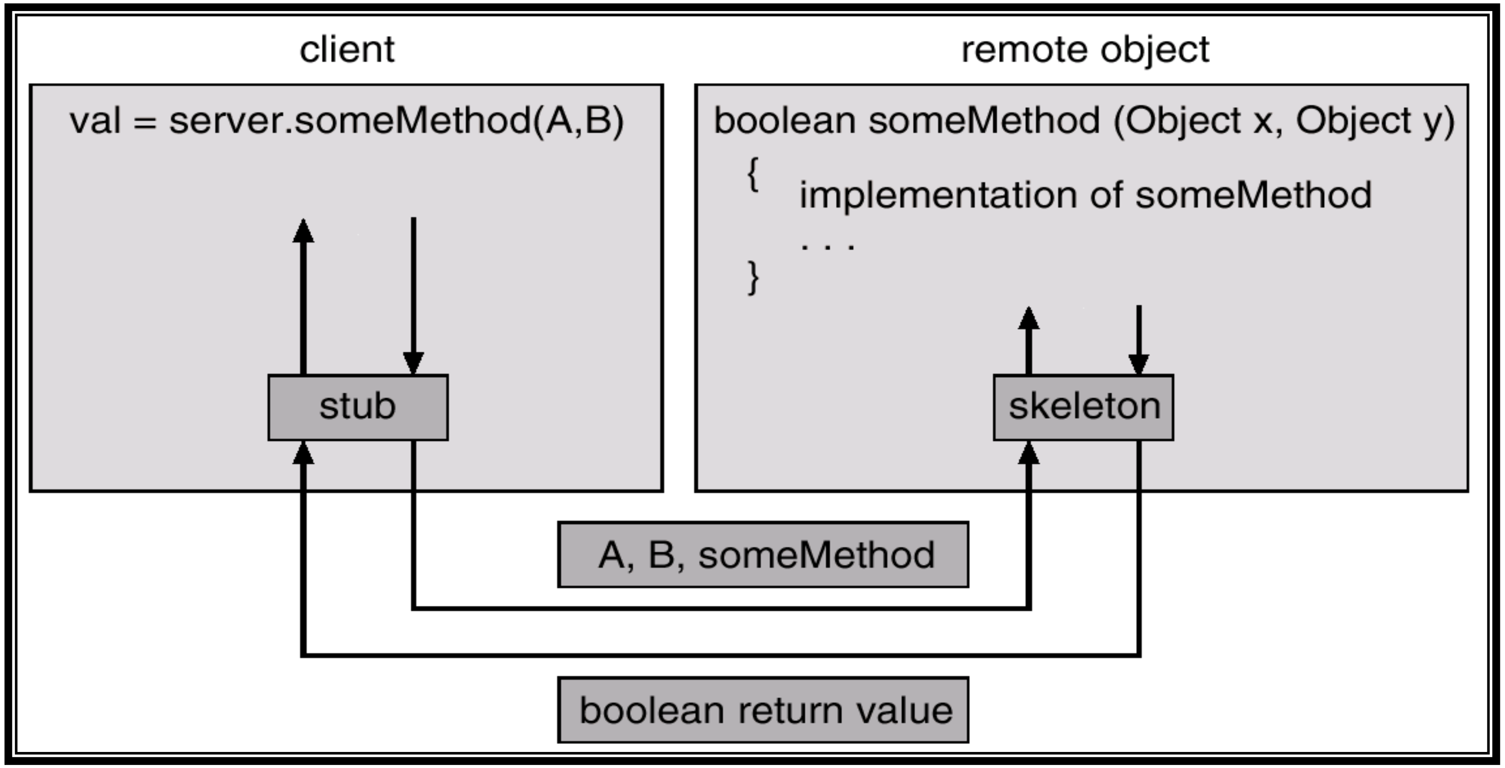
Inter-Process Communication (IPC)

- Unix/Linux IPC
  - ✓ pipes
  - ✓ FIFOs
  - ✓ message queue
  - ✓ shared memory
  - ✓ sockets

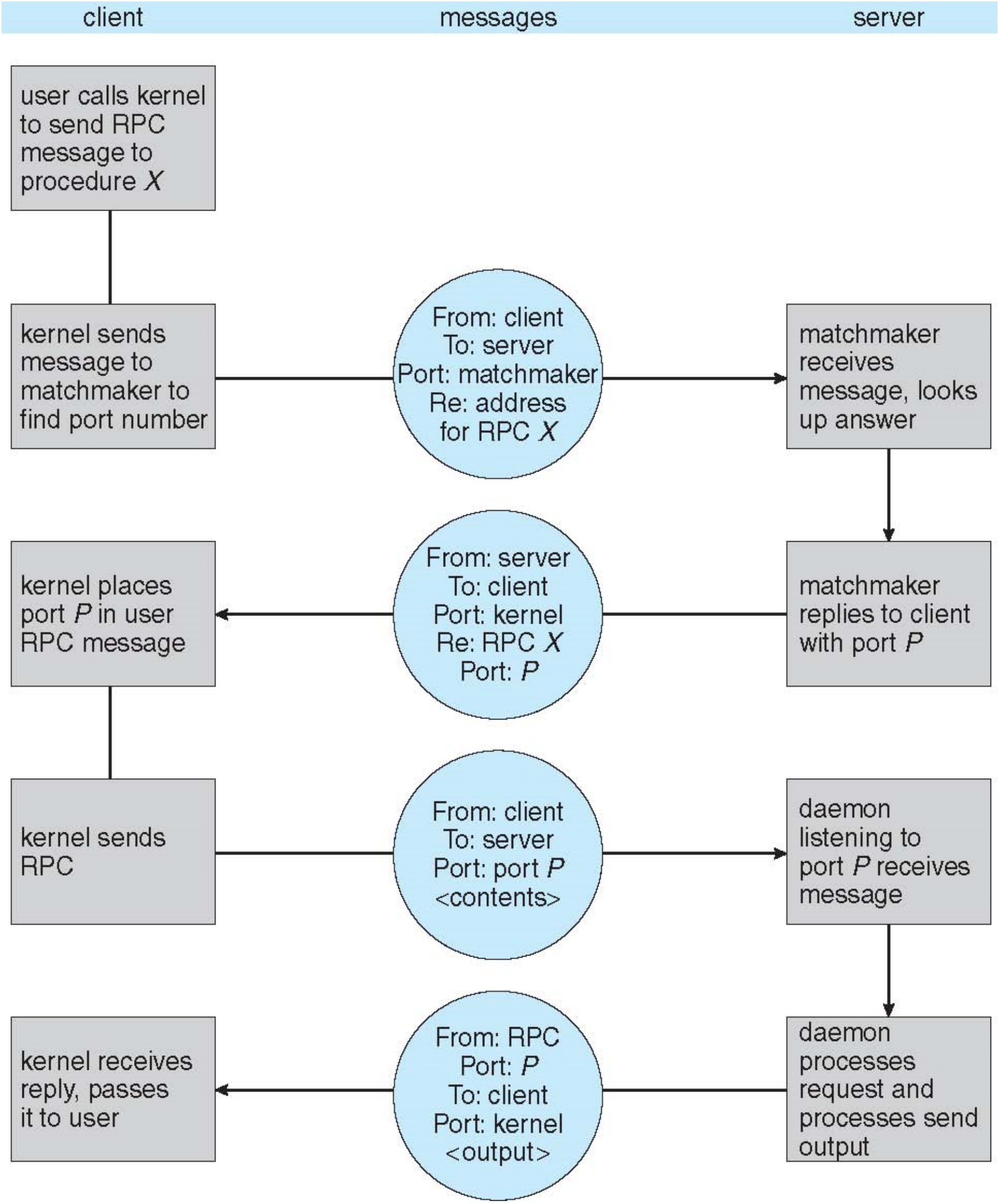
Client-Server Communication

- Sockets
- Remote Procedure Call (RPC)
- Remote Method Invocation (RMI in Java)

- Marshalling parameters


client
remote object
val = server.someMethod(A,B)
boolean someMethod (Object x, Object y)
{
implementation of someMethod
. . .
}
stub
skeleton
A, B, someMethod
boolean return value

Execution of RPC



client

messages

server

user calls kernel to send RPC message to procedure *X*

kernel sends message to matchmaker to find port number

From: client
To: server
Port: matchmaker
Re: address for RPC *X*

matchmaker receives message, looks up answer

matchmaker replies to client with port *P*

From: server
To: client
Port: kernel
Re: RPC *X*
Port: *P*

kernel places port *P* in user RPC message

kernel sends RPC

From: client
To: server
Port: port *P*
<contents>

daemon listening to port *P* receives message

daemon processes request and processes send output

From: RPC
Port: *P*
To: client
Port: kernel
<output>

kernel receives reply, passes it to user