

Chapter 3: Process



Chapter 3: Processes

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



Process Concept

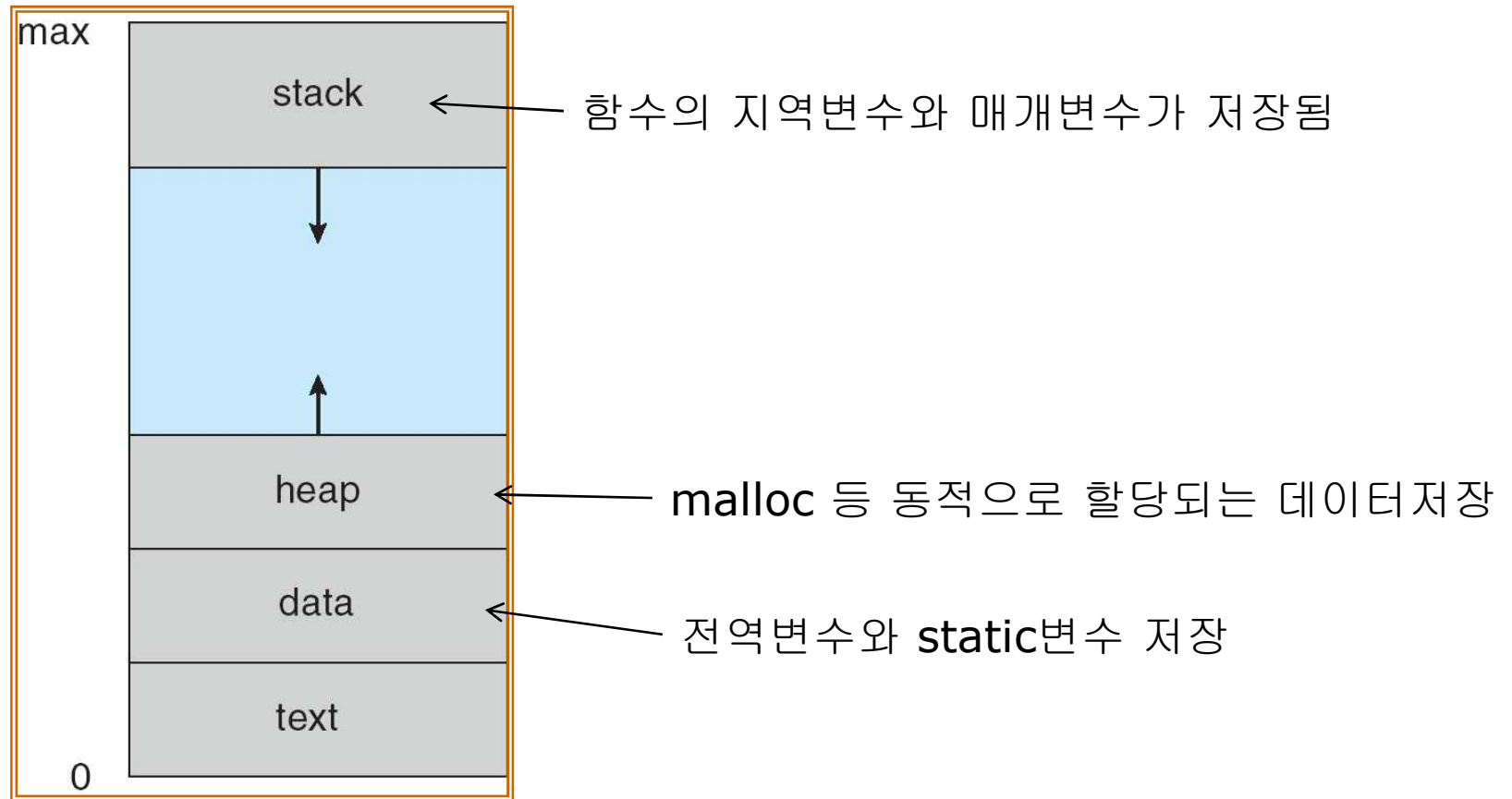
- **Process** : 수행중인 프로그램, 현대 시분할 시스템에서 작업의 단위
- An operating system executes a variety of programs:
 - Batch system – jobs
 - Time-shared systems – user programs or tasks

} 둘 다 Process
- **Process != Program**
 - Program은 디스크에 저장된 파일의 내용과 같은 **passive entity**인 반면
 - Process는 실행할 명령어를 저장하는 **program counter**와 연관된 자원의 집합을 가진 **active entity**
- A process includes:
 - program counter
 - stack
 - data section

Process vs. Thread?



Process Concept - Process in Memory



Process Concept - Process in Memory

□ Heap 과 Stack

- Heap : run-time시에 크기가 결정되는 요소들의 저장공간
 - c의 malloc() 함수나 C++의 new 연산
- Stack : 컴파일시에 크기가 결정되어있는 요소들이 저장되는 공간
 - 함수의 매개변수 지역변수

```
#include <stdio.h>
int A, B;
main()
{
    int a = 0;
    int b = 0;
    int *p1 = NULL;
    int *p2 = NULL;
    p1 = (int*)malloc(sizeof(A));
    p2 = (int*)malloc(sizeof(A));
    printf("전역 변수의 주소값 출력\n");
    printf("%d\n", &A);
    printf("%d\n", &B);
    printf("동적할당된 포인터의 주소값 출력\n");
    printf("%d\n", p1);
    printf("%d\n", p2);
    printf("지역 변수의 주소값 출력\n");
    printf("%d\n", &a);
    printf("%d\n", &b);
    free(p1);
    free(p2);
}
```

[출처] [\[C/C++\]Heap 과 Stack 영역](#) | 작성자 [우기우기](#)

```
전역 변수의 주소값 출력
4359392
4359396
동적할당된 포인터의 주소값 출력
3681776
3681824
지역 변수의 주소값 출력
1244884
1244872
Press any key to continue.
```



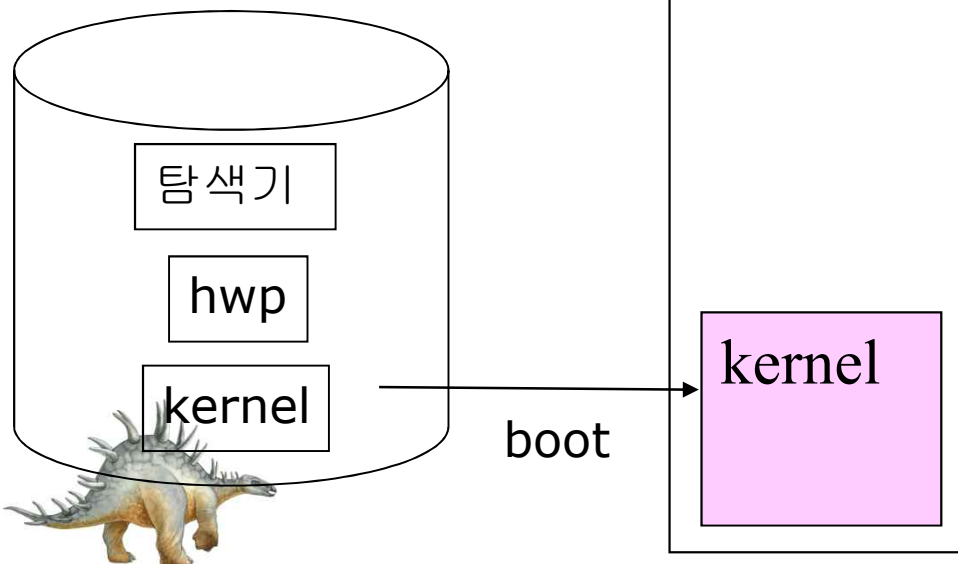
Process Concept – Process의 구동(1)

Multi-user System -- multiple shells

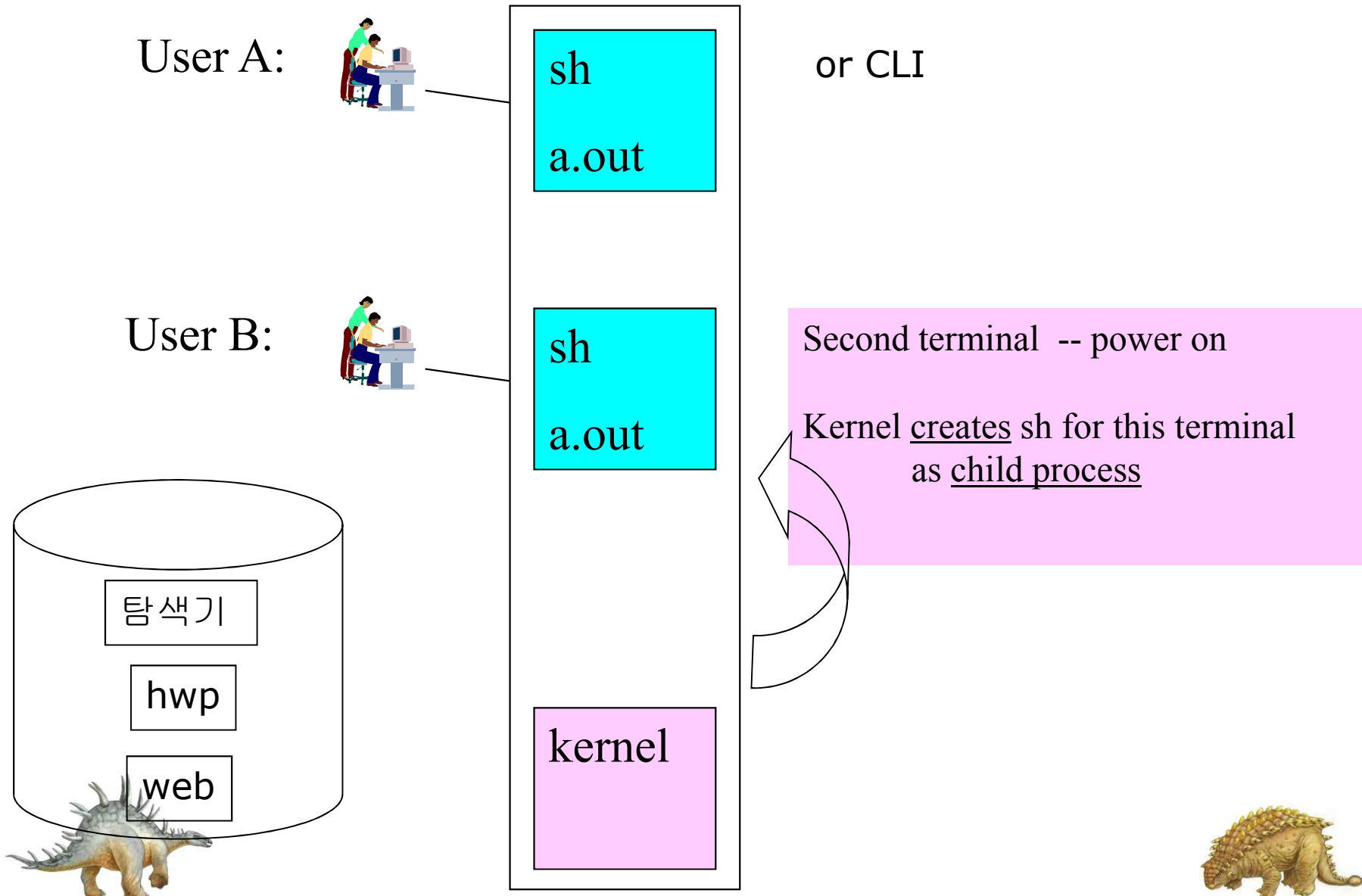
or CLI

Second terminal -- power on

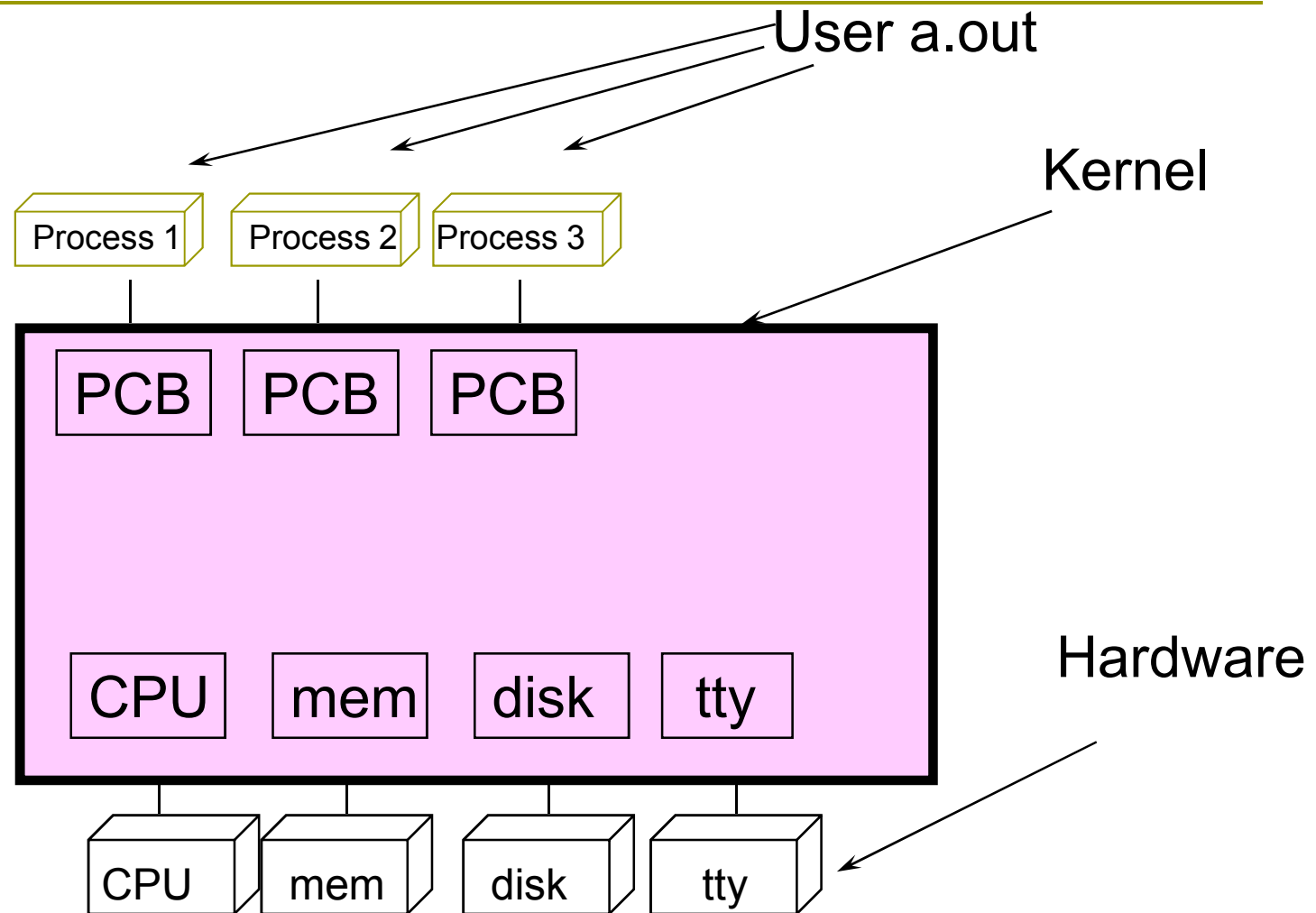
Kernel creates sh for this terminal





Process Concept – Process의 구동(2)



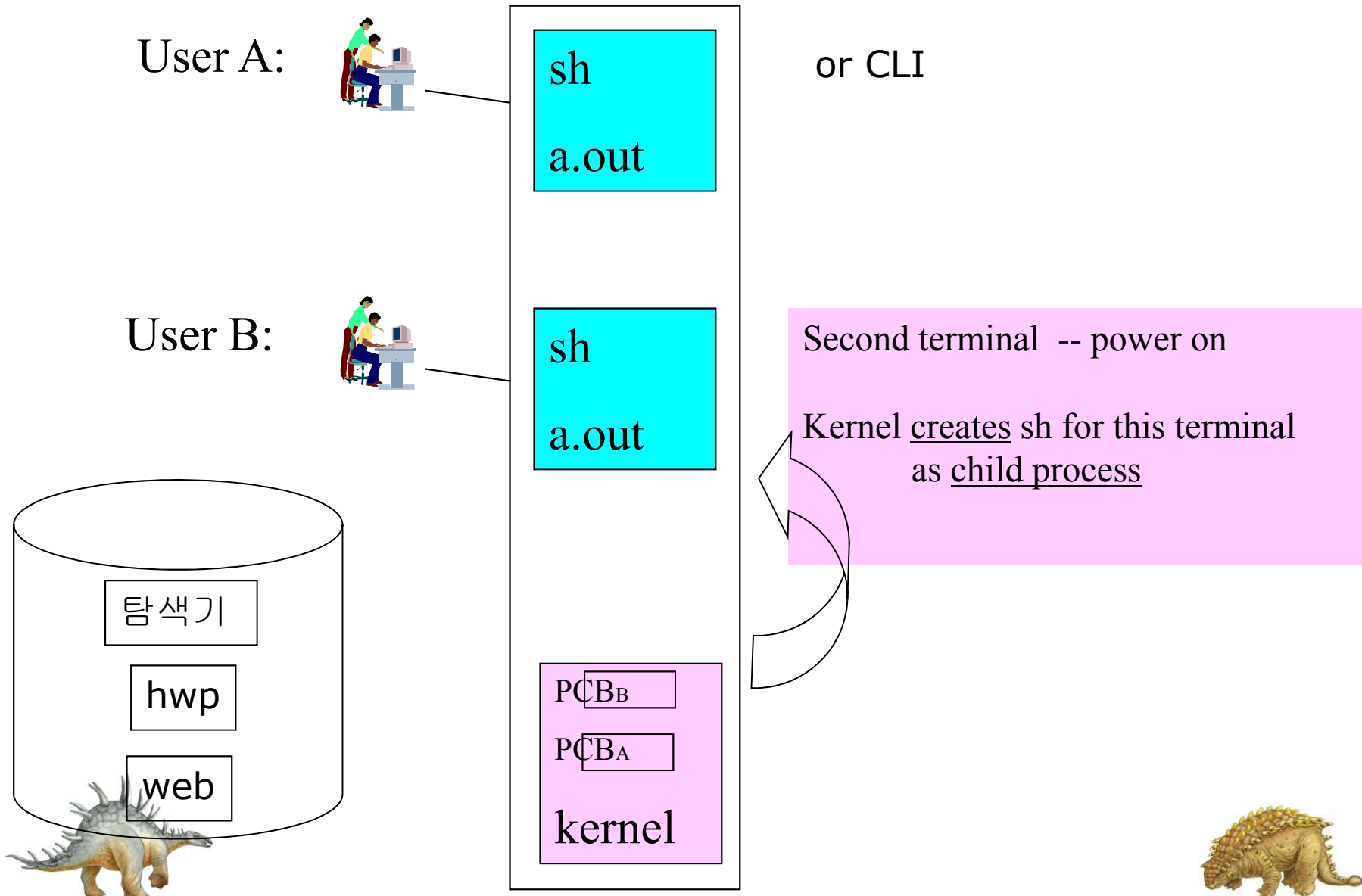
Kernel 내부에서의 Process(3)



 : Table (Data Structure)
 : Object (hardware or software)

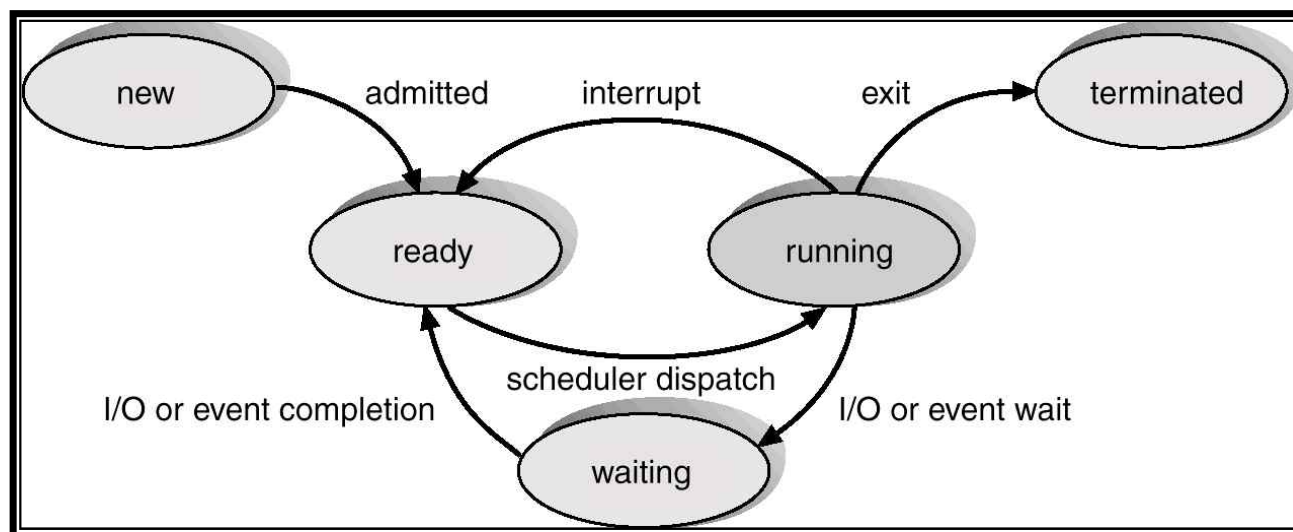


Process Concept – Process의 구동(4)



Process Concept - Process State

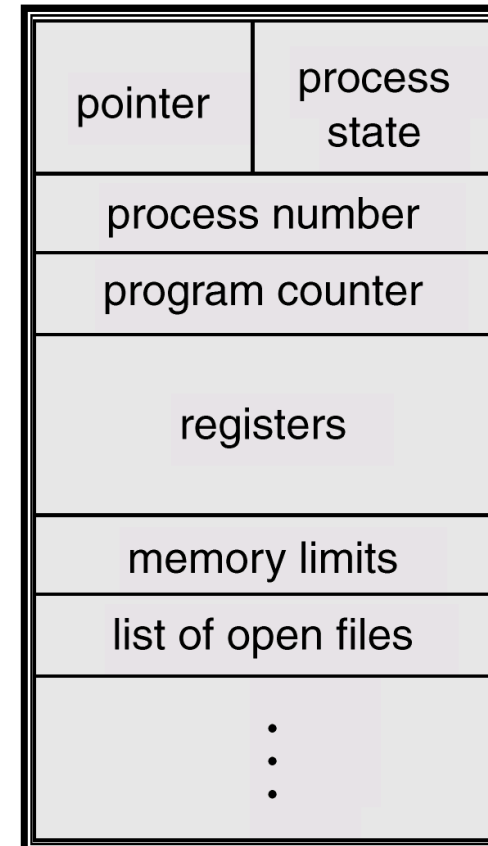
- As a process executes, it changes *state*
 - **new**: The process is being created.
 - **running**: Instructions are being executed.
 - **waiting**: The process is waiting for some event to occur.
 - **ready**: The process is waiting to be assigned to a process.
 - **terminated**: The process has finished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



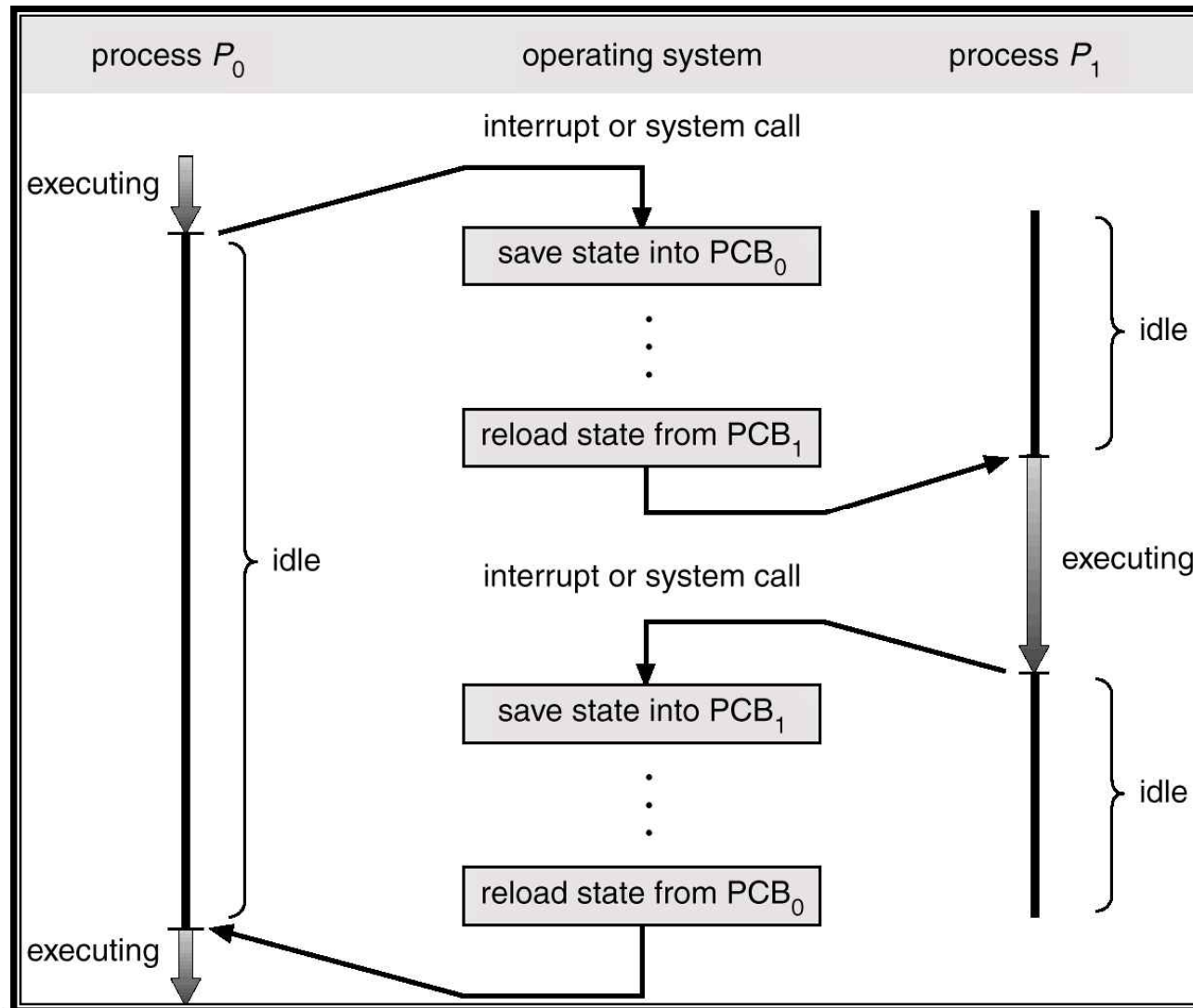
Process Concept - Process Control Block (PCB)

Information associated with each process.

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



Process Concept – CPU Switch From Process to Process



Process Concept – Linux에서의 프로세스 표현

- Linux Process : task_struct
 - <http://www.ibm.com/developerworks/kr/library/l-linux-process-management/index.html>

```
struct task_struct {  
  
    volatile long state;  
    void *stack;  
    unsigned int flags;  
  
    int prio, static_prio;  
  
    struct list_head tasks;  
  
    struct mm_struct *mm, *active_mm;  
  
    pid_t pid;  
    pid_t tgid;  
  
    struct task_struct *real_parent;  
  
    char comm[TASK_COMM_LEN];  
  
    struct thread_struct thread;  
  
    struct files_struct *files;  
  
    ...  
  
};
```



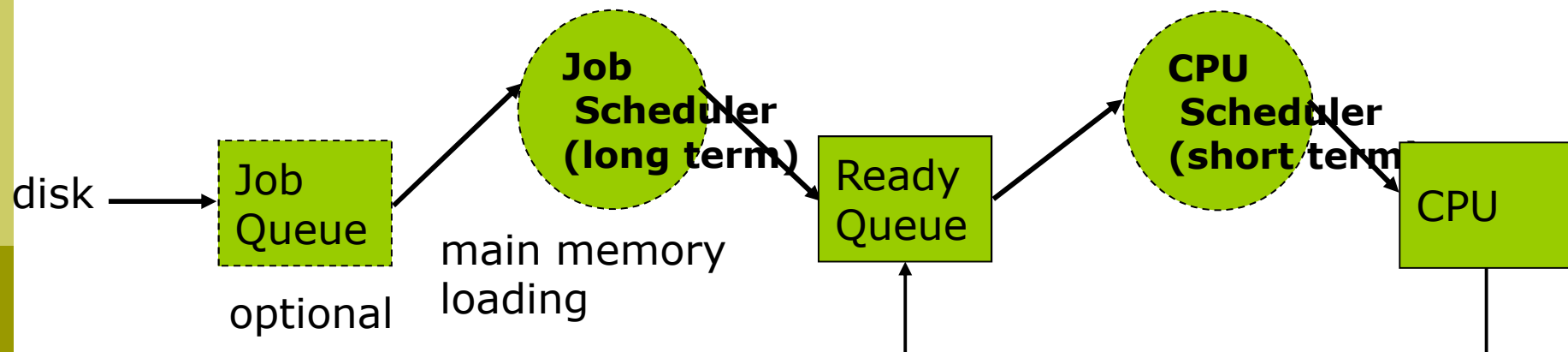
Process Scheduling – Process Scheduling Queues

- ❑ **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.
- ❑ Process migration between the various queues.



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.



Unix는 Long term Scheduler가 없음 => 물리적인 제한에 의존함

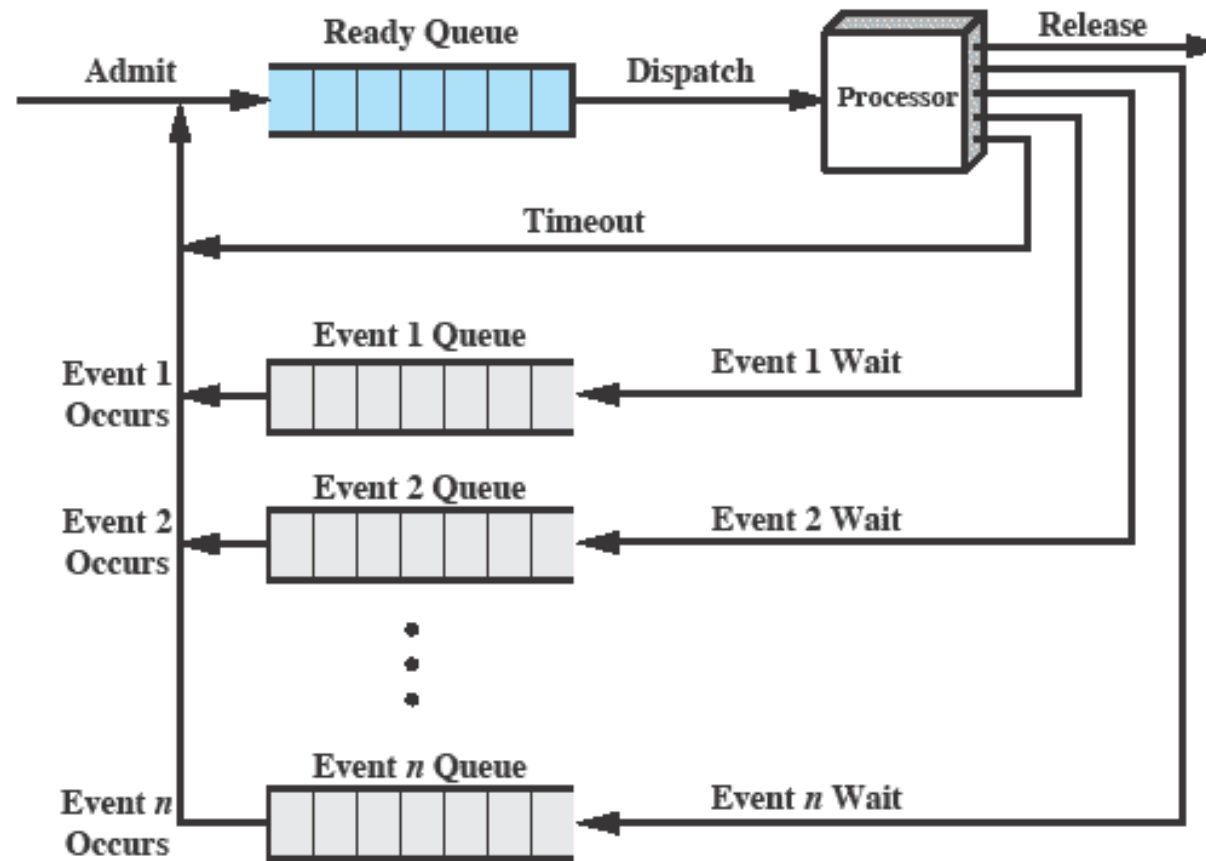


Process Scheduling –Schedulers

- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the **ready queue**.
 - ➔ Giving Memory
 - ➔ very infrequently (seconds, minutes)
⇒ (may be slow).
- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and **allocates CPU**.
 - ➔ Giving CPU
 - very frequently (milliseconds) ⇒ (must be fast).
- **Mid-term scheduler** – Swapping의 고려



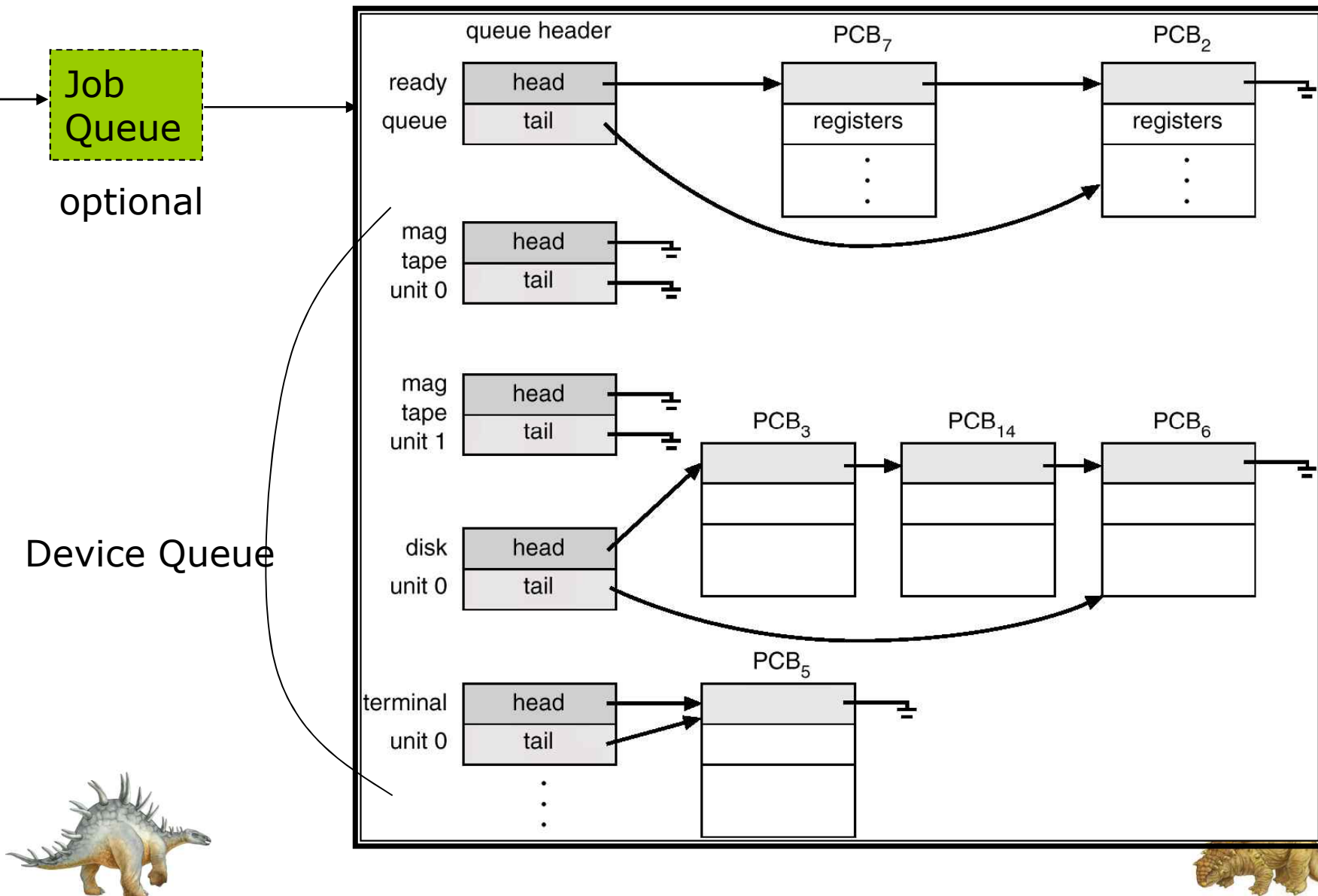
Process Scheduling



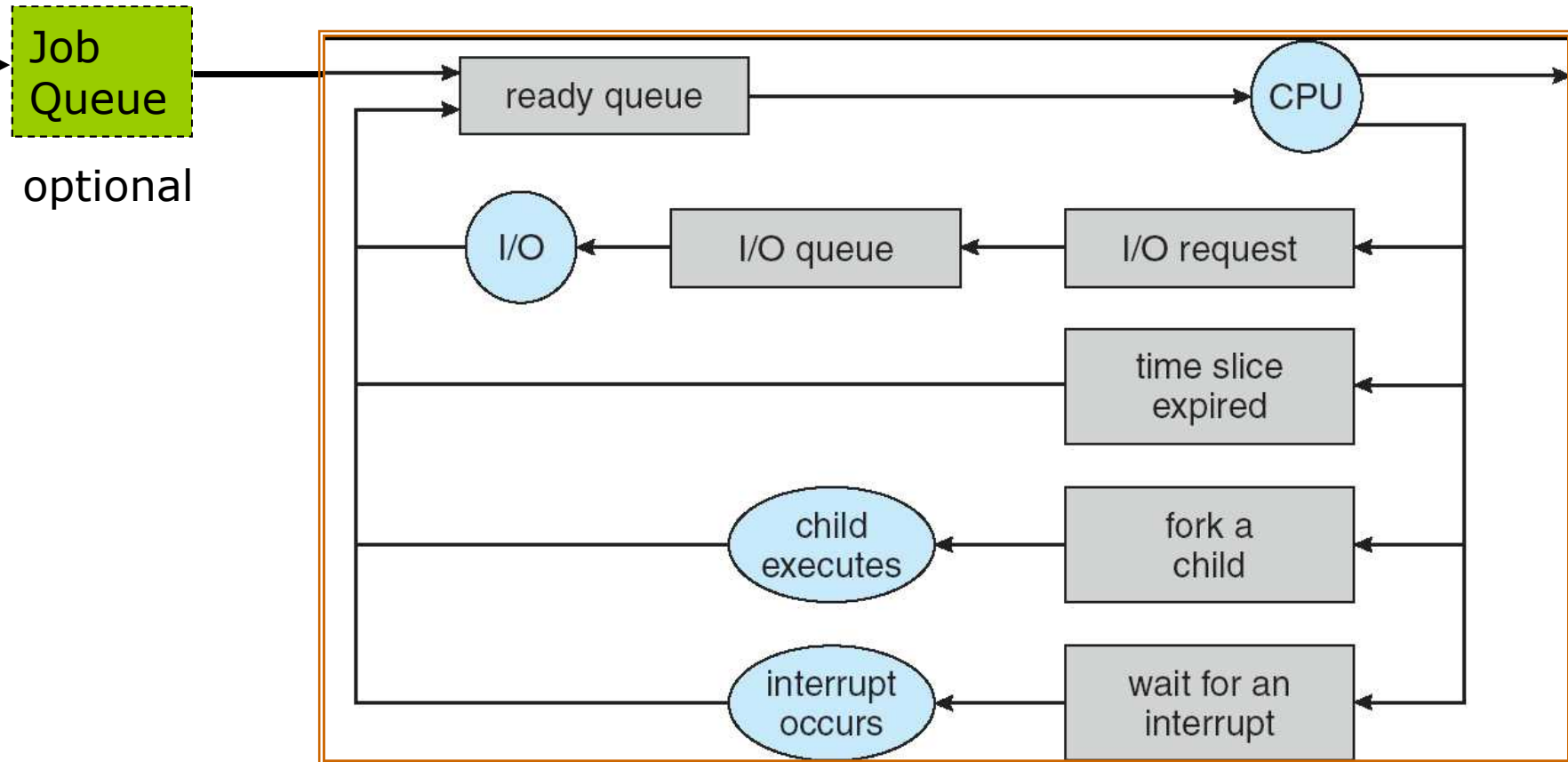
(b) Multiple blocked queues



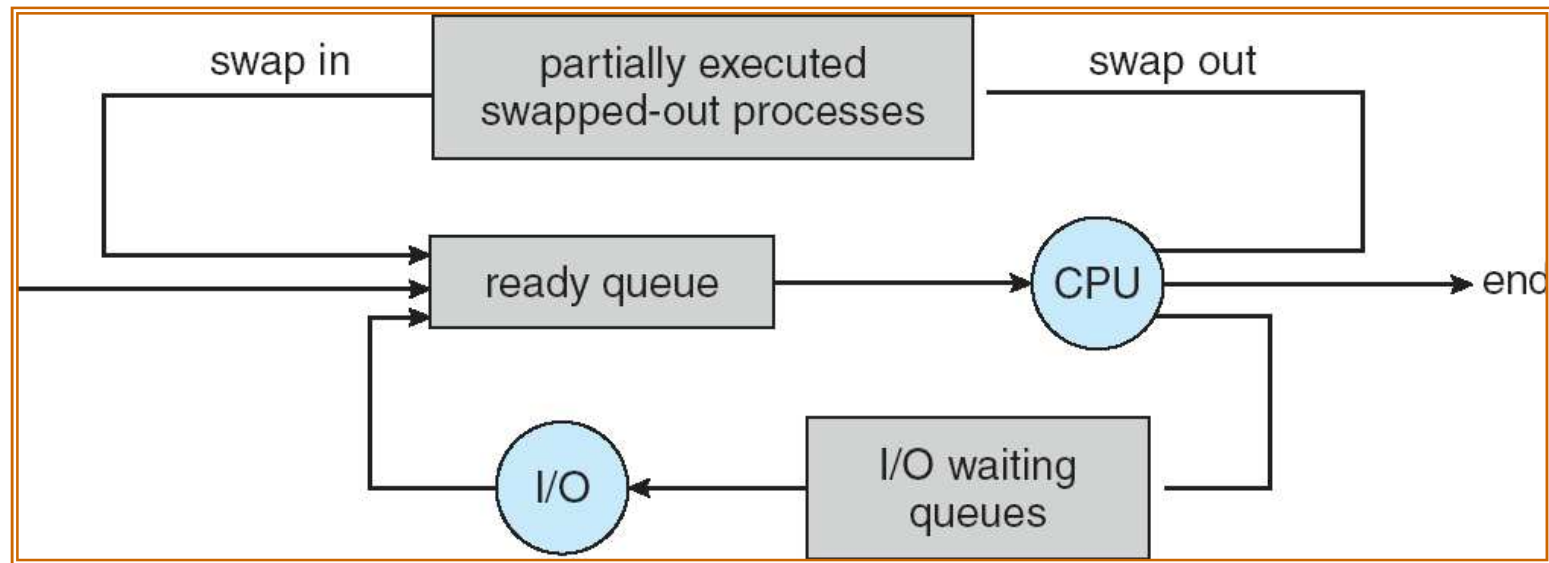
Process Scheduling – Ready Queue And Various I/O Device Queues



Process Scheduling – Representation of Process Scheduling



Process Scheduling – Addition of Medium Term Scheduling



Medium Term Scheduler : 메모리에서 CPU를 위해 적극적으로 경쟁하는 프로세스들을 일시적으로 제거하여 multiprogramming의 정도를 완화하도록 함 => Swapping



Process Scheduling –Schedulers (Cont.)

- 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.



Process Scheduling – 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.

효율적인 Context Switch

=> Sun Ultra SPARC의 경우 여러 개의 레지스터 집합을 제공하여 메모리 및 CPU의 overhead를 감소시킴



Operation on Process

- Process Creation
- Process Termination



Operation on Process – Process Creation

- ❑ Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- ❑ Resource sharing strategies
 - 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.



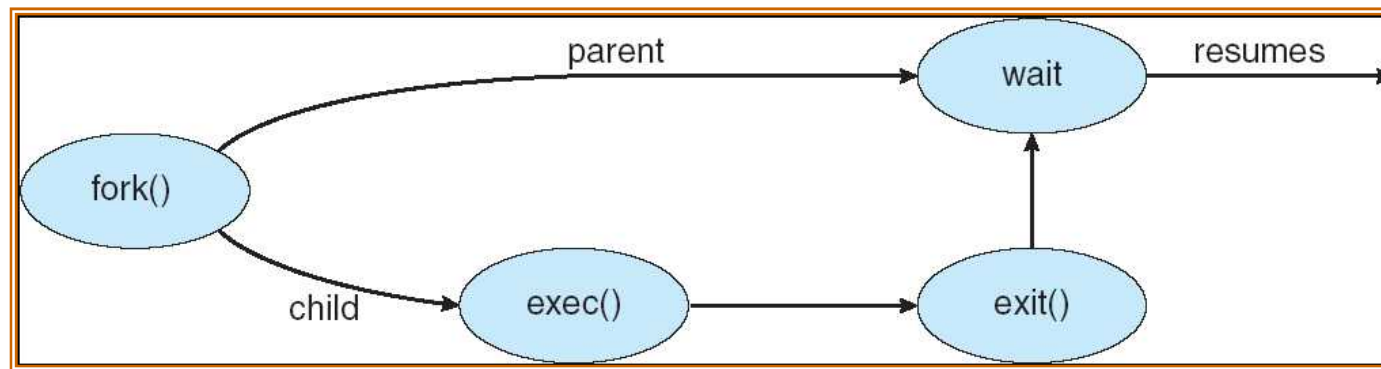
Operation on Process – Process Creation (Cont.)

- 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.



Operation on Process – Process Creation

□ Process Creation in UNIX



Operation on Process – Process Creation

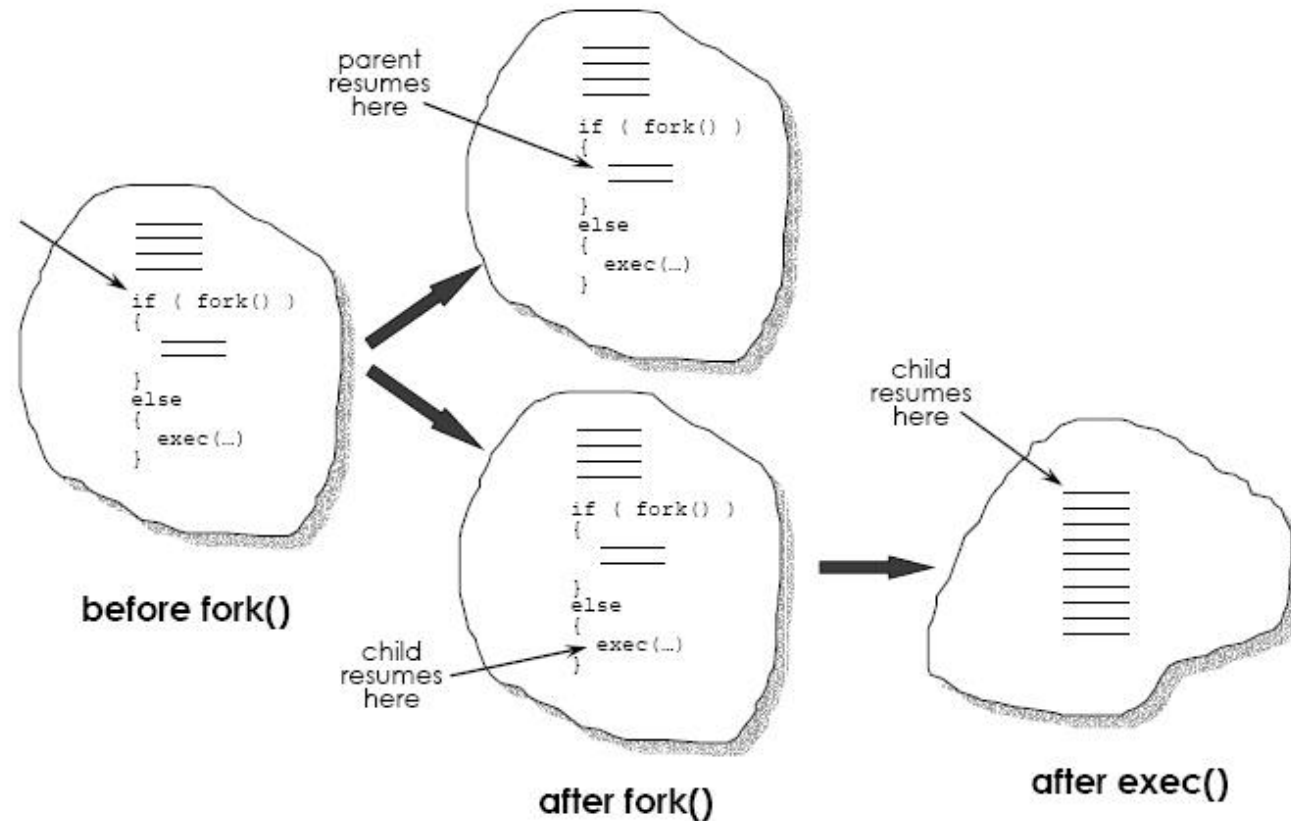
C Program Forking Separate Process

```
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);
    }
}
```



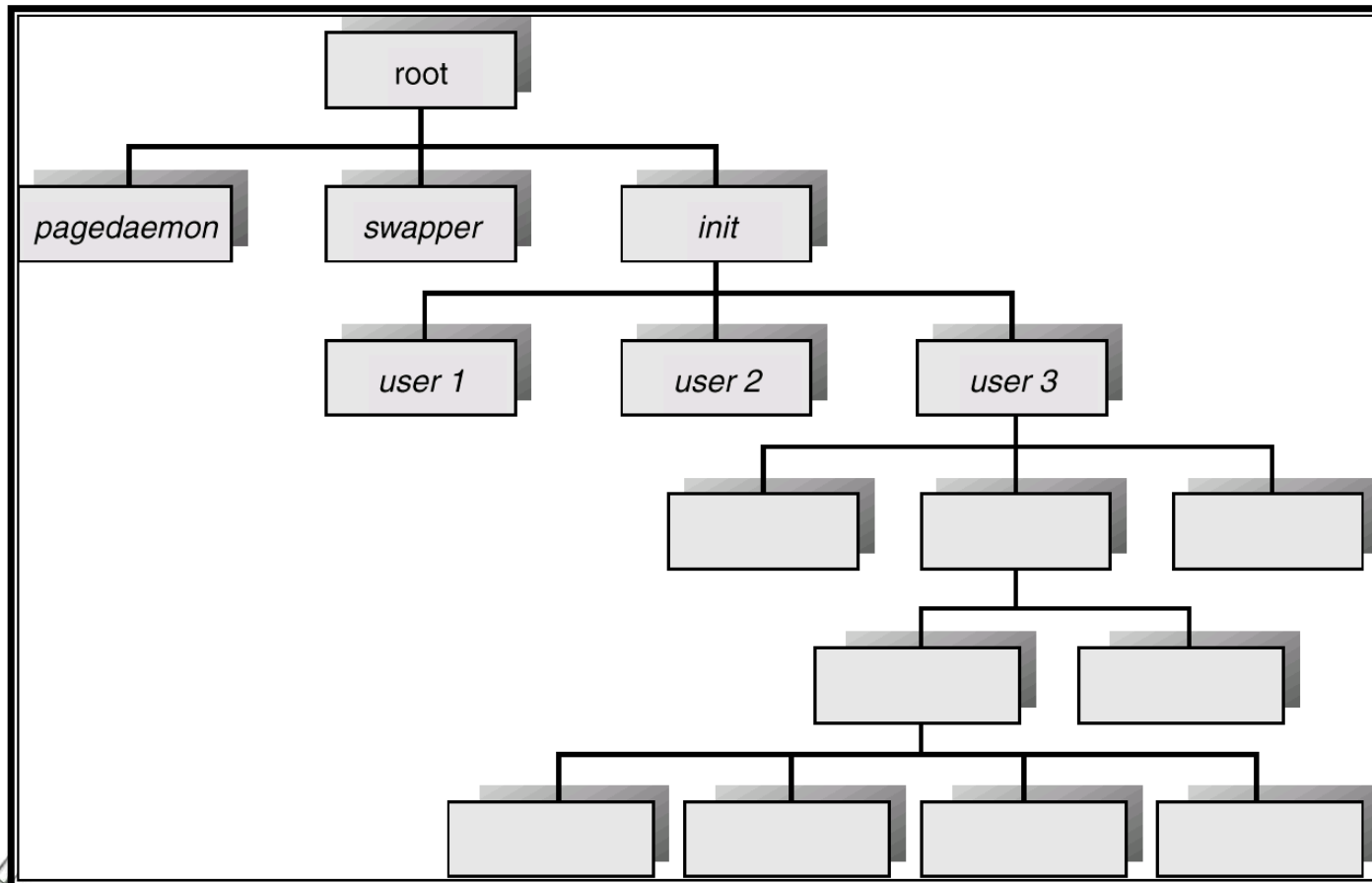
Operation on Process – Process Creation

- C Program Forking Separate Process
 - Fork – Before and After



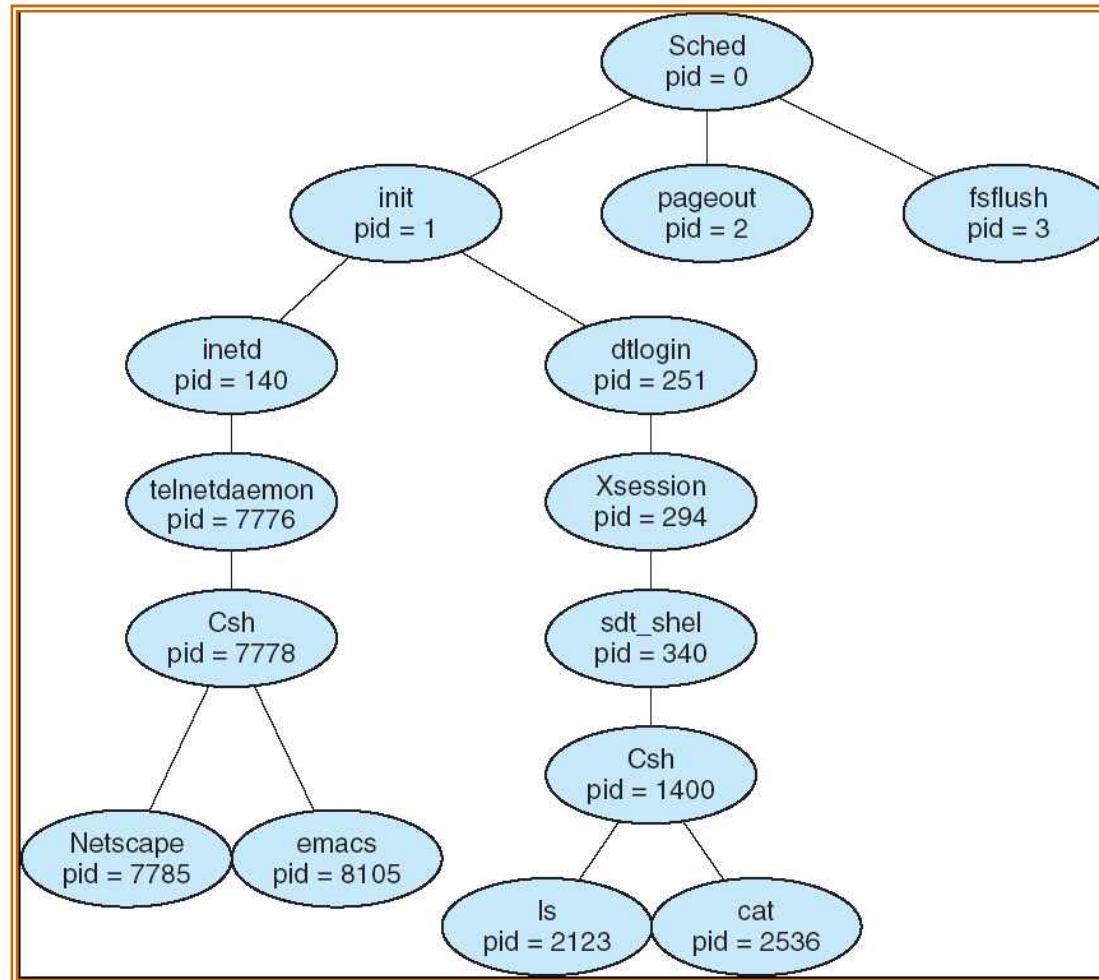
Operation on Process – Process Creation

Processes Tree on a UNIX System



Operation on Process – Process Creation

- A tree of processes on a typical Solaris



Operation on Process – Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
 - Output data from child to parent (via **wait**).
 - Process' resources are deallocated by operating system.

- Parent가 execution of children processes을 종료하는 경우(cascading termination)
 - Child has exceeded allocated resources.
 - Task assigned to child is no longer required.
 - Parent is exiting.
 - Operating system does not allow child to continue if its parent terminates.
 - Cascading termination.



프로세스간 통신(Interprocess Communication:IPC)

- 독립적 프로세스 : *Independent* process cannot affect or be affected by the execution of another process.
- 협력적 프로세스 : *Cooperating* process can affect or be affected by the execution of another process
- 프로세스간 협력을 제공하는 이유
 - 정보공유
 - 계산 가속화
 - 모듈성
 - 편의성

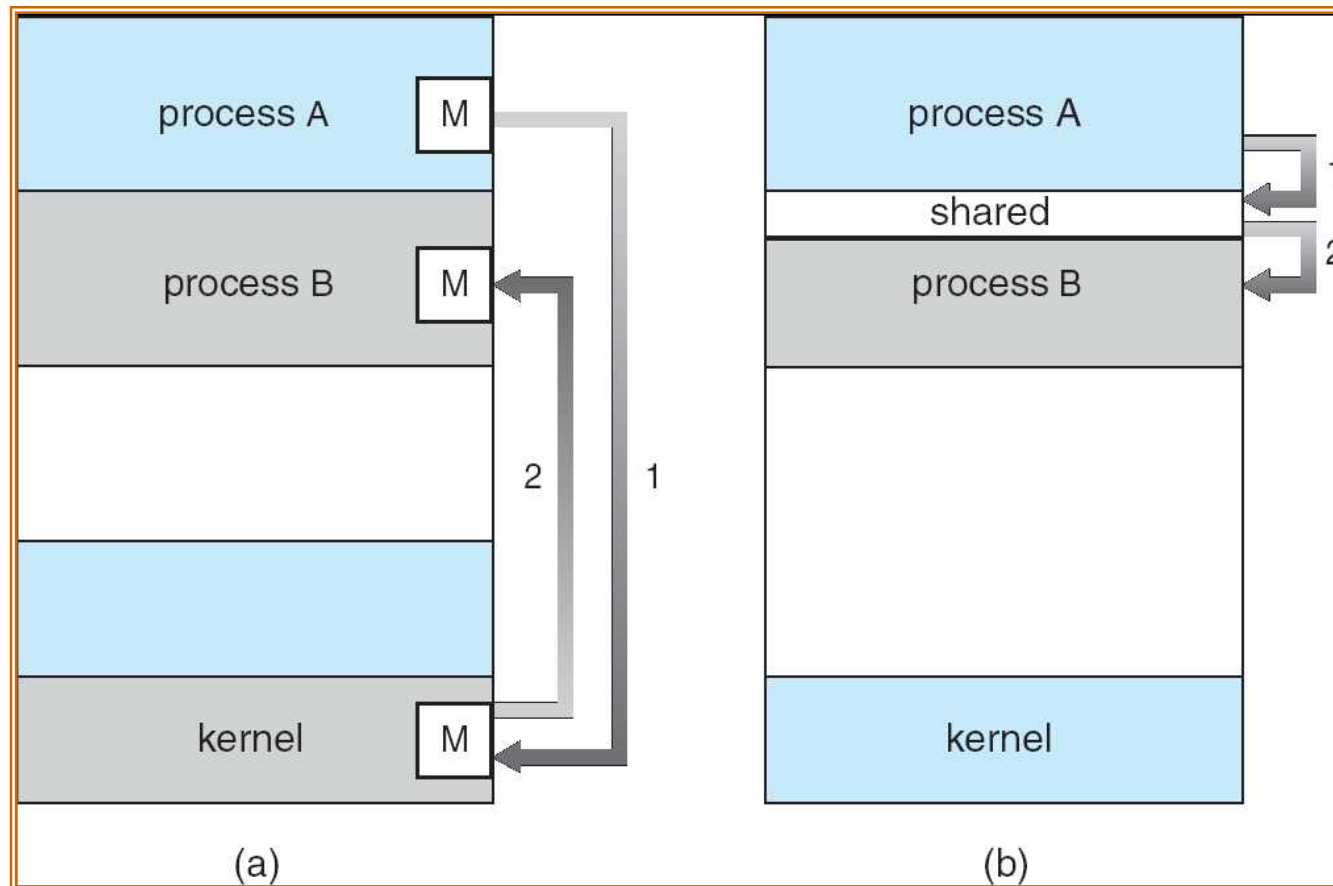


IPC

- IPC를 위한 기본적인 기법
 - 공유 메모리(shared memory)
 - 생산자와 소비자간의 공유하는 메모리를 사용하여 연동
 - 동일한 주소공간을 공유
 - 메시지 전달(message passing)
 - 동일한 주소공간을 공유하지 않고 통신하며 동기화



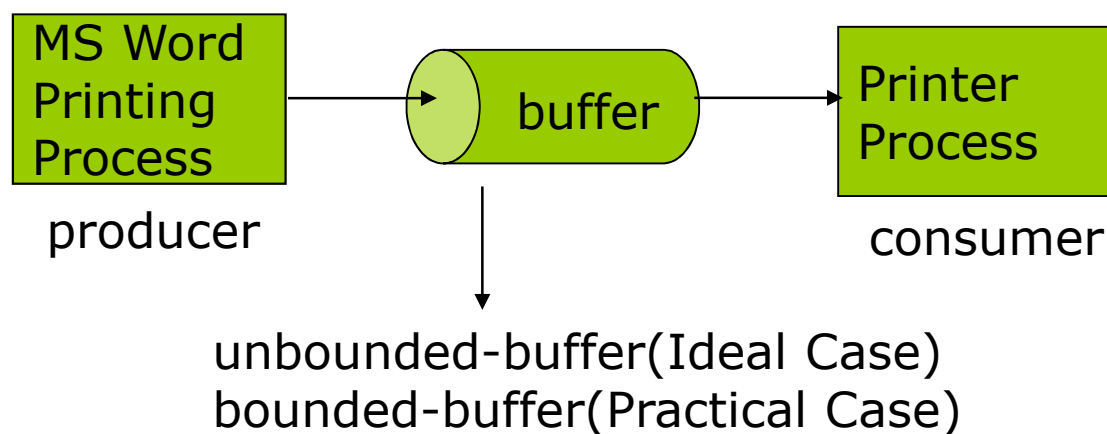
IPC - Communications Model



IPC-공유메모리 시스템

□ Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.
- 두가지 유형의 buffer
 - *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;
```



Bounded-Buffer– Producer Process

```
item nextProduced;
```

```
while (1) {  
    while (((in + 1) % BUFFER_SIZE) == out)  
        ; /* do nothing */  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
}
```



Bounded-Buffer– Consumer Process

item nextConsumed;

```
while (1) {  
    while (in == out)  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
}
```

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



IPC- Message Passing System

- ❑ IPC : Mechanism for processes to communicate and to synchronize their actions. : ex. chat program
- ❑ 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)



Message Passing System

- 직접 또는 간접 통신
- 대칭 또는 비 대칭 통신
- 자동 또는 명시적 버퍼링
- 복사에 의한 전송 또는 참조에 의한 전송
- 고정 길이 또는 가변길이 메세지



IPC- 직접 통신

- 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.
- Asymmetry도 가능
 - **send**(P , $message$)
 - **receive**(id , $message$)



IPC- 간접통신

- 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.



IPC- 간접통신

- 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



IPC- 간접통신

□ 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.



IPC- 간접통신-Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
 - 송신하는 프로세스는 메시지가 수신 프로세스 또는 메일 박스에 의해 수신될 때까지 **blocking** 됨
- **Non-blocking** is considered **asynchronous**
 - 송신하는 프로세스가 메시지를 보내고 작업을 재시작함
- **send** and **receive** primitives may be either blocking or non-blocking.



IPC- 간접통신-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.



IPC의 예 : POSIX 공유메모리

```
#include <sys/shm.h>
#include <sys/stat.h>
#include <stdio.h>

int main()
{
    /* the identifier for the shared memory segment */
    int segment_id;
    /* a pointer to the shared memory segment */
    char* shared_memory;
    /* the size (in bytes) of the shared memory segment */
    const int size = 4096;

    /* allocate a shared memory segment */
    segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);

    /* attach the shared memory segment */
    shared_memory = (char *) shmat(segment_id, NULL, 0);

    /* write a message to the shared memory segment */
    sprintf(shared_memory, "Hi there!");

    /* now print out the string from shared memory */
    printf("%s\n", shared_memory);

    /* now detach the shared memory segment */
    shmdt(shared_memory);

    /* now remove the shared memory segment */
    shmctl(segment_id, IPC_RMID, NULL);

    return 0;
}
```

C program illustrating POSIX shared-memory API.



IPC의 예 : Client-Server Communication

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



IPC의 예 : Sockets

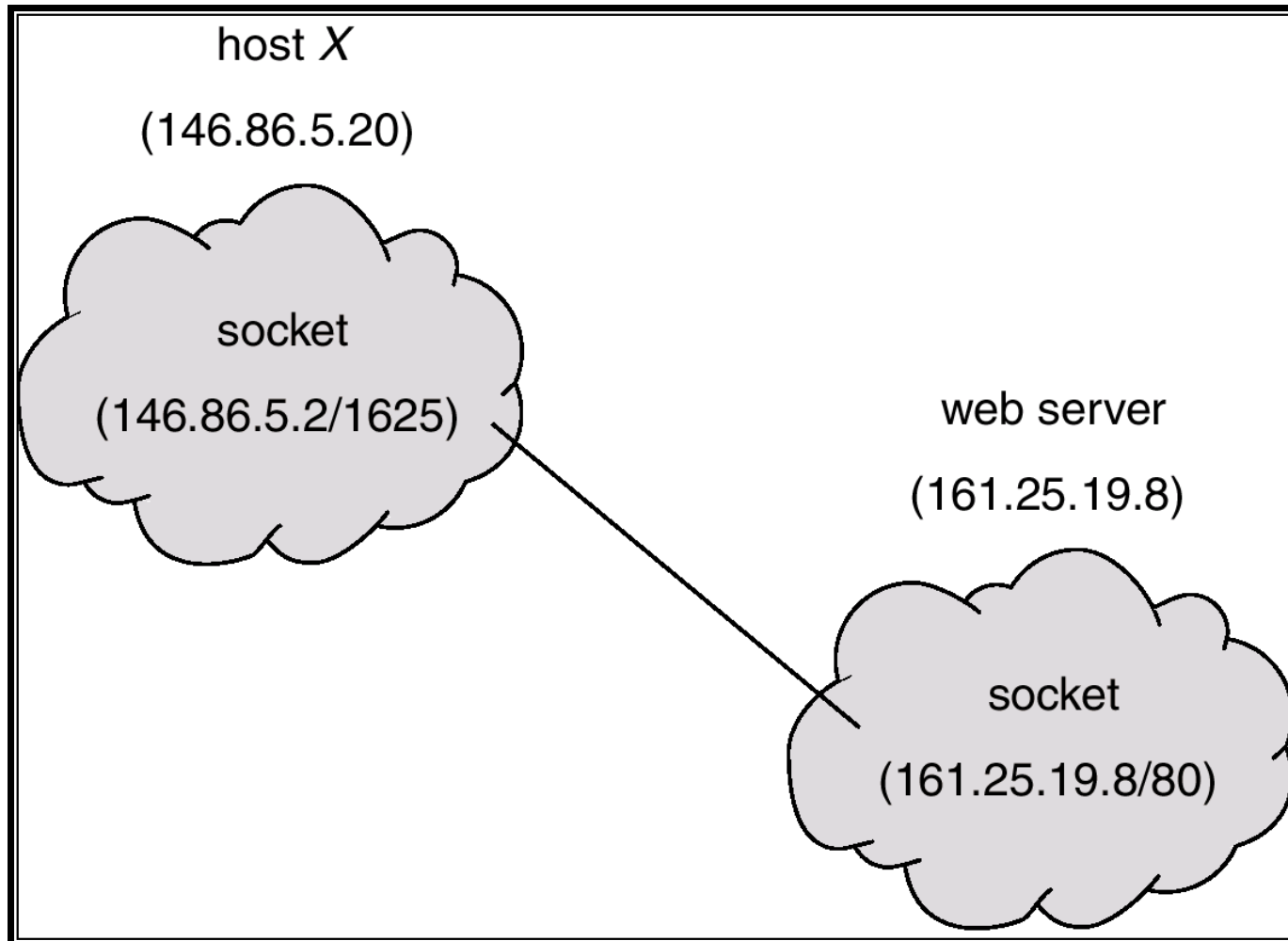
- ❑ A **socket** is defined as an *endpoint for communication*.
- ❑ Concatenation of IP address and port
- ❑ The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- ❑ **well known ports**
 - telnet server : port 23
 - ftp server : port 21
 - web(http) server : port 80
- ❑ Communication consists between a pair of sockets.

Java가 제공하는 세가지 소켓

- 1) Connection-oriented(TCP) Socket : Socket Class
- 2) Connectionless(UDP) Socket : DatagramSocket Class
- 3) MulticastSocket



IPC의 예 : Socket Communication



IPC의 예 :

```
import java.net.*;
import java.io.*;

public class DateServer
{
    public static void main(String[] args) {
        try {
            ServerSocket sock = new ServerSocket(6013);

            // now listen for connections
            while (true) {
                Socket client = sock.accept();

                PrintWriter pout = new
                    PrintWriter(client.getOutputStream(), true);

                // write the Date to the socket
                pout.println(new java.util.Date().toString());

                // close the socket and resume
                // listening for connections
                client.close();
            }
        }
        catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```



Figure 3.18 Date server.



IPC의 예 :

```
import java.net.*;
import java.io.*;

public class DateClient
{
    public static void main(String[] args) {
        try {
            //make connection to server socket. 127.0.0.1 is loopback IP addr
            Socket sock = new Socket("127.0.0.1", 6013);

            InputStream in = sock.getInputStream();
            BufferedReader bin = new
                BufferedReader(new InputStreamReader(in));

            // read the data from the socket
            String line;
            while ( (line = bin.readLine()) != null)
                System.out.println(line);

            // close the socket connection
            sock.close();
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

Figure 3.19 Date client.

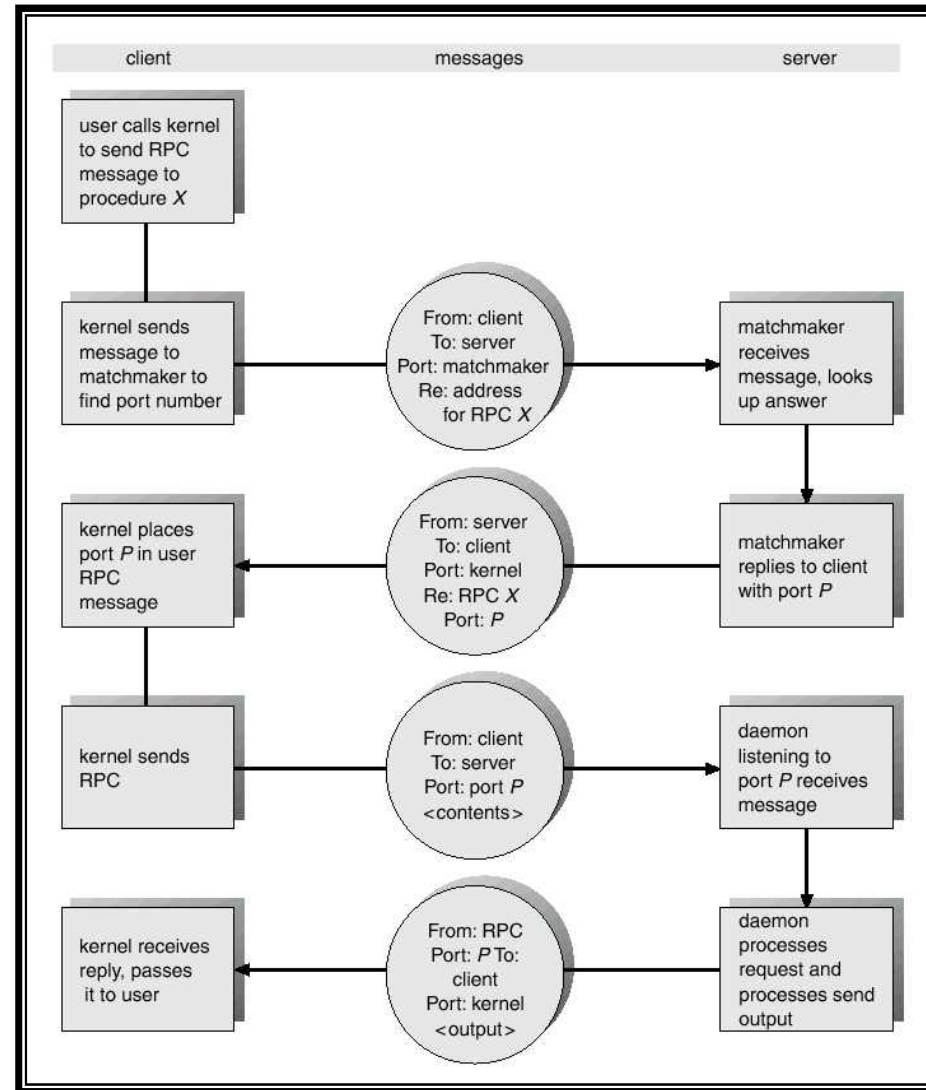


IPC의 예 : Remote Procedure Calls

- ❑ 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 performs the procedure on the server.

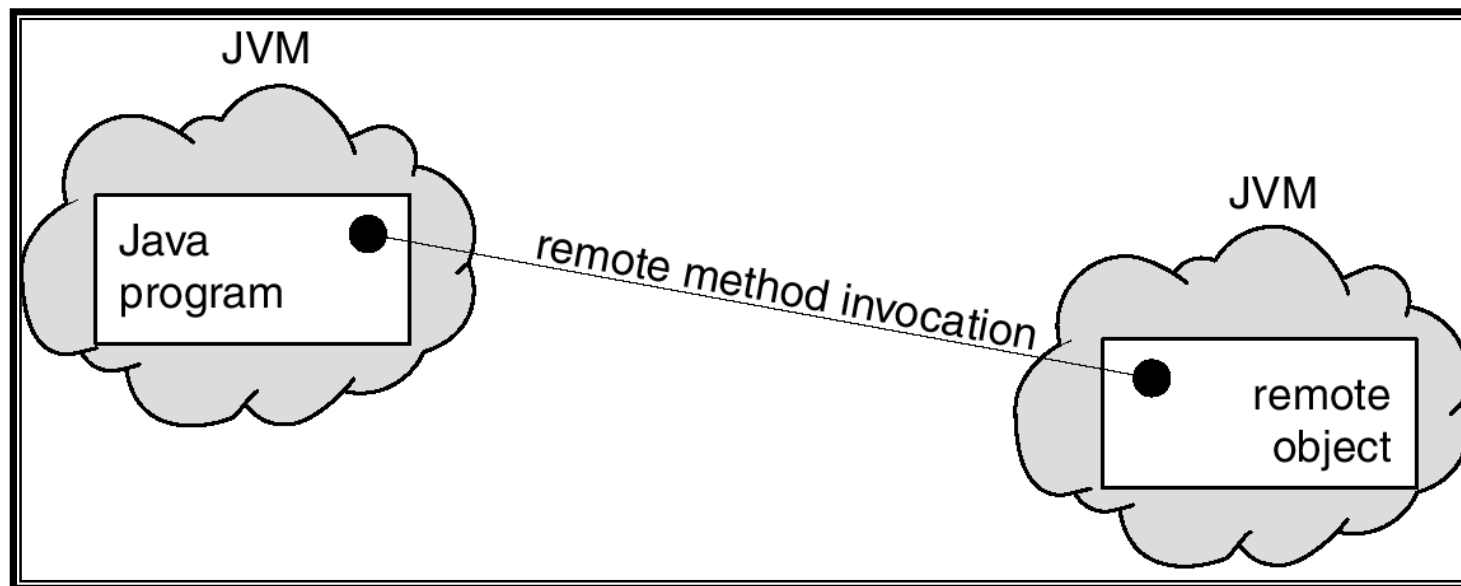


IPC의 예 : Execution of RPC



IPC의 예 : 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.



IPC의 예 : Marshalling Parameters

