

Chapter 3

Process Description and Control

Contents

- 3.1 What is a Process?
- 3.2 Process States
- 3.3 Process Description
- 3.4 Process Control
- 3.5 운영체제의 실행
- 3.6 UNIX SVR4 프로세스 관리
- 3.7 Summary

What is a Process?

■ Definitions

- ✓ A program in execution
- ✓ An instance of a program running on a computer
- ✓ The entity that can be assigned to and executed on a processor
- ✓ 명령들의 순차 수행, 현재상태, 연계된 시스템 자원들의 집합 등에 의해 특징지어지는 활동단위 (A unit of activity)

■ Structures

- ✓ program code
- ✓ a set of data
- ✓ process information

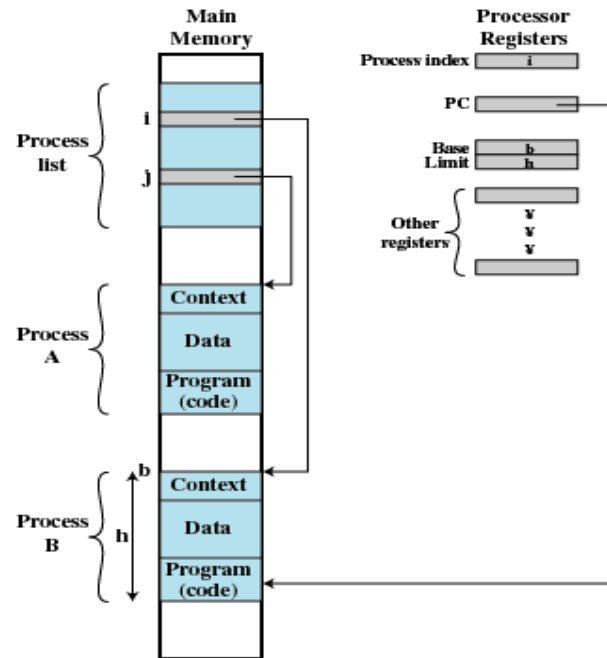


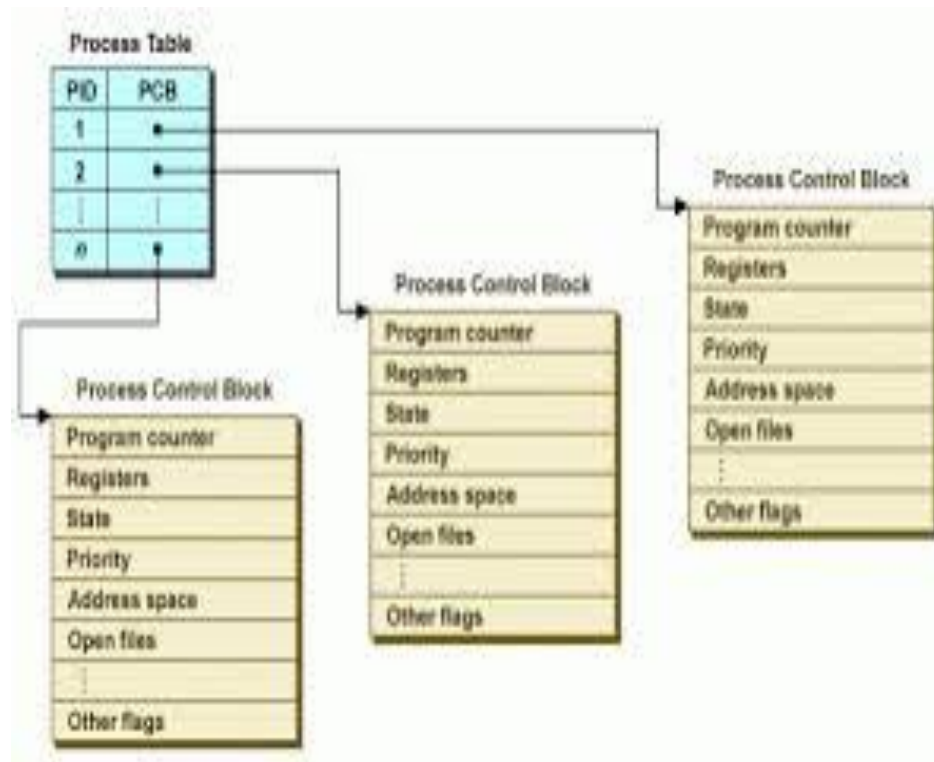
Figure 2.8 Typical Process Implementation

What is a Process?

■ PCB (Process Control Block)

- ✓ managed by OS
- ✓ 수행 프로세스를 인터럽트한 후, 프로세스 수행을 재개할 수 있도록 정보 유지
- ✓ Process = program code + 관련 data + PCB

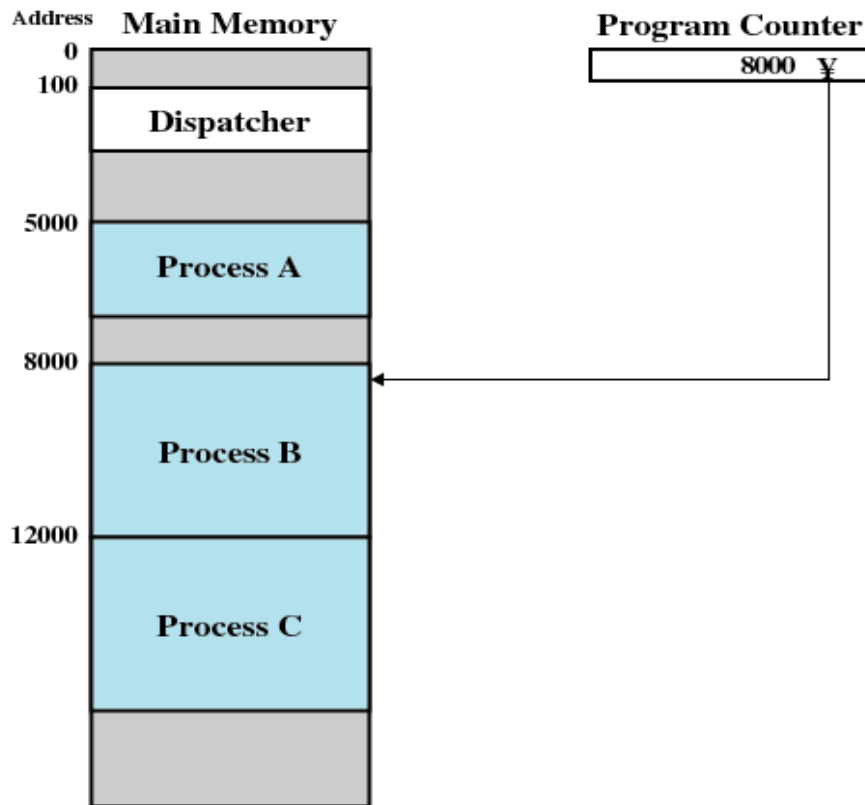
Process Id
Process state
Program counter
Register information
Scheduling information
Memory related information
Accounting information
Status information related to I/O



3.2 Process States: Traces of Processes

■ Process level

- ✓ 프로세스 궤적(trace)이란 프로세스를 위해 수행되는 명령어 리스트
- ✓ CPU 행위의 특징은 trace를 통해 파악가능



5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011
(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.2 Snapshot of Example Execution (Figure 3.4)
at Instruction Cycle 13

Figure 3.3 Traces of Processes of Figure 3.2

Process States: Traces of Processes

- System level

A	1	5000		27	12004	
	2	5001		28	12005	
	3	5002				-----Time out
	4	5003		29	100	
	5	5004		30	101	
	6	5005		31	102	
		-----Time out				
	7	100		32	103	
	8	101		33	104	
	9	102		34	105	
	10	103		35	5006	
	11	104		36	5007	
	12	105		37	5008	
	13	8000		38	5009	
B	14	8001		39	5010	
	15	8002		40	5011	
	16	8003				-----Time out
				41	100	
		-----I/O request				
	17	100		42	101	
	18	101		43	102	
	19	102		44	103	
	20	103		45	104	
	21	104		46	105	
	22	105		47	12006	
	23	12000		48	12007	
	24	12001		49	12008	
C	25	12002		50	12009	
	26	12003		51	12010	
				52	12011	
						-----Time out

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2

Process States: Traces of Processes

- Timing Diagram: need process state

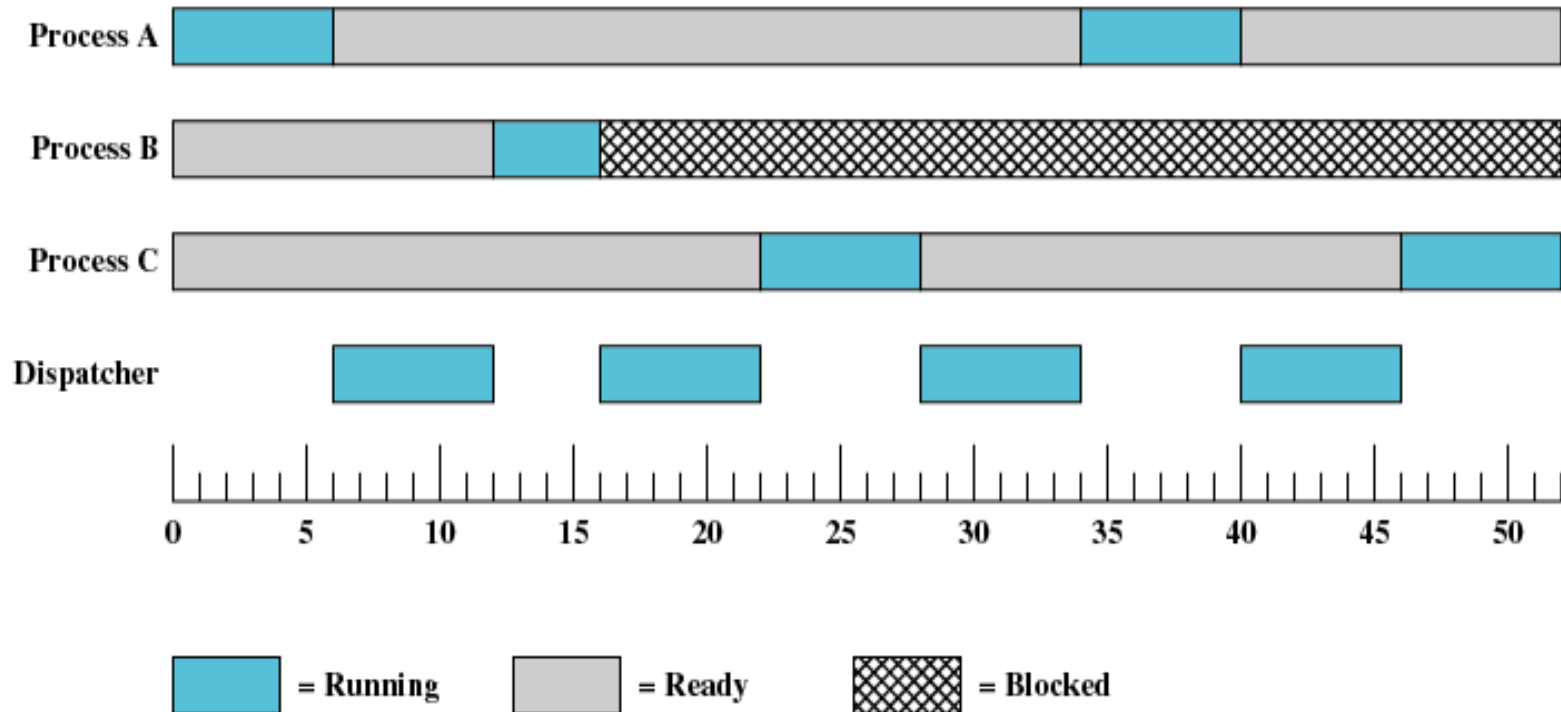
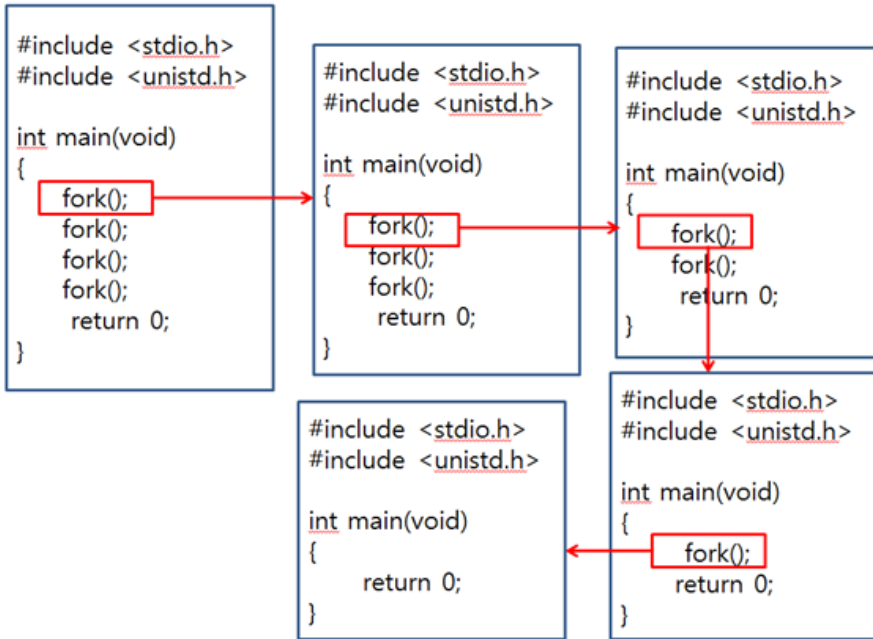


Figure 3.7 Process States for Trace of Figure 3.4

fork() & exec()



printf("# of children\n");

Operating System © click2study.net

A process executes the code

```

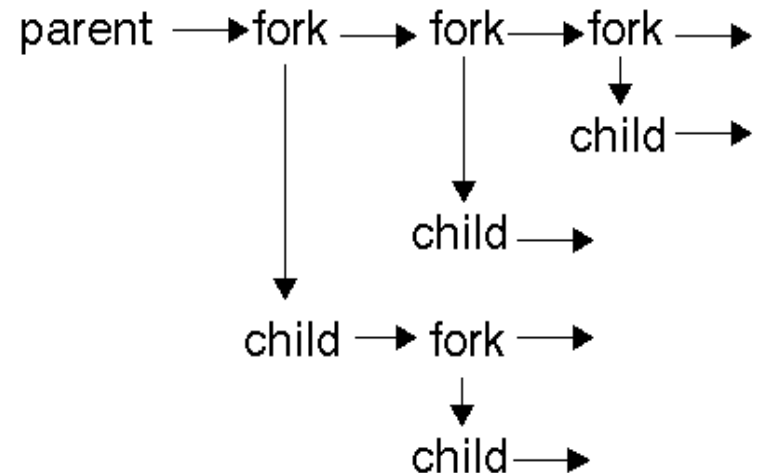
fork();
fork();
fork();

```

The total number of **child** processes created is

(A) 3 (B) 4 (C) 7 (D) 8 **GATE 2012**

UNIX fork()



fork() & exec()

Exec System Call Family

- The exec system call family has the following members:
 - execl, execlp, execl, execl, execl, and execl.
- Naming convention:

exec	x	y
------	---	---

For x,

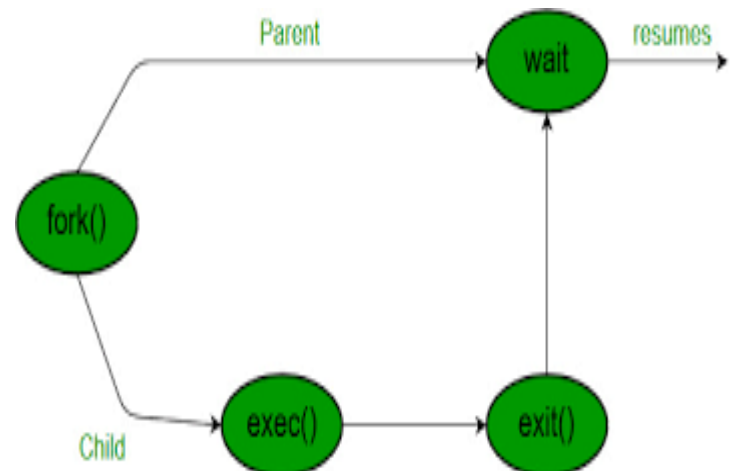
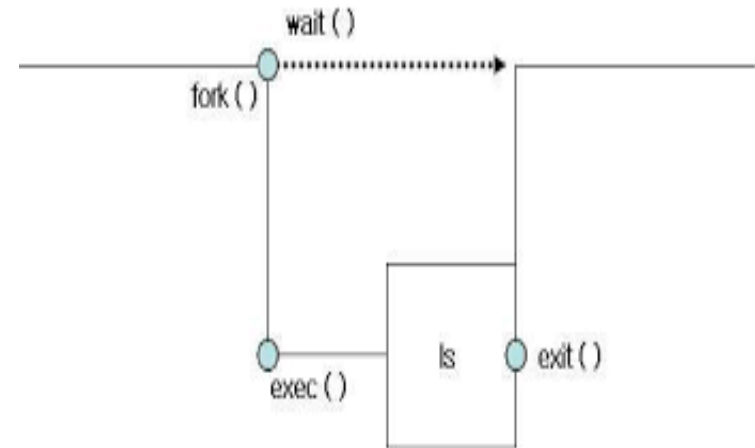
- l – provide a **list** of program arguments, terminated by NULL;
- x – provide a **vector** (array) of program arguments.

For y,

- p – **PATH** environment variable will be searched;
- e – a vector of **environment variables**^{*} should be supplied.

^{*} See supplementary notes.

Source: T.Y. Wang, Chinese University of Hong Kong



fork() & exec()

■ Examples

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

int glob =6 ; /* 외부 변수 초기화 */
int main() {
    pid_t pid;
    int var =88;
    printf(" Before fork() \n");
    if ( (pid = fork()) <0 )
        printf(" fork() error! \n");
    else if (pid ==0) { /* child */
        glob++;
        var++;
    }
    else
        sleep(2); /* parent */
    printf(" pid= %d, glob = %d, var =%d \n", getpid(), glob, var);
    exit(0);
}
```

⇒ **pid=430, glob=7, var=89 /*child process */**
pid=429, glob=6, var=88 /* parent process는 변화 없음 */

fork() & exec()

■ Examples

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

int main() {
    pid_t pid;
    char *message;
    int n;
    printf (" fork program starting ! \n");
    pid = fork();
    switch (pid) {
        case -1 :
            printf (" fork failed \n");
            exit(1);
        case 0 :
            message= "This is child process";
            n= 5;
            break;
        default :
            message= "This is parent process";
            n= 3;
            break; }
    for ( ; n>0; n-- ) {
        puts(message);
        sleep(1); }
    exit(0);
}
```

⇒ **child process** 는 메시지 **5번** 출력하고 **parent process**는 **3번** 출력함

fork() & exec()

- Examples - "system" library function을 이용하여
새로운 프로세스 생성 확인

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

int main() {
    printf (" Running ps with system \n");
    system ("ls -l");
    printf (" I am done ~~~\n");
    exit(0);
}
```

⇒ 실행결과 ???

⇒ `system ("ls -l");` ==> `system ("ls -l & ");` 실행결과 확인?

fork() & exec()

- Examples - "exec" 함수는 현재 프로세스를 path 나 file 인수에 지정된 new process로 replace 확인

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

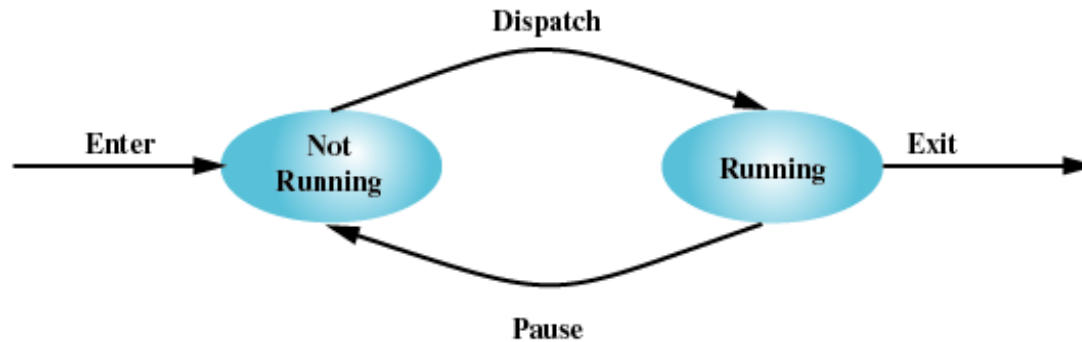
int main() {
    printf (" Running ps with execlp \n");
    execlp ("ls", "ls", "-l", 0);
    printf (" I am done ~~~\n");
    exit(0);
}
```

⇒ 실행결과 (특이점) ???

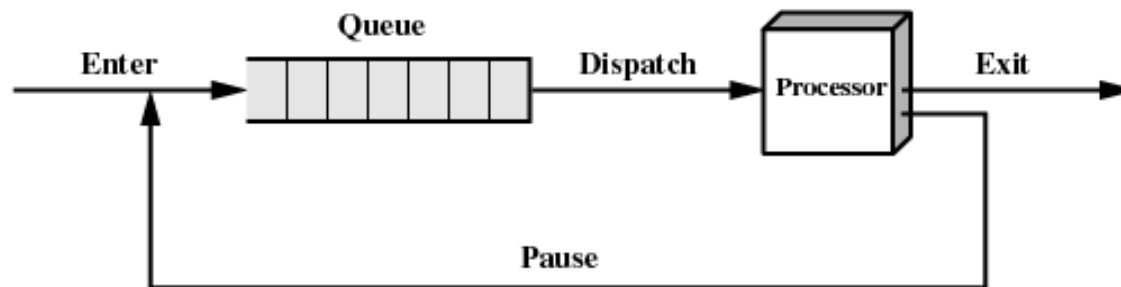
Process States: Basic

■ Two-State Process Model

- ✓ Running
- ✓ Not-running



(a) State transition diagram



(b) Queuing diagram

☞ each entry is a pointer to the PCB or linked list of PCBs

Process States: Extend1

■ Process Creation and Termination

- ✓ Creation: enter into process model
 - Spawned by existing process -> `fork()` system call
 - Created by OS
 - ...

표 3.1 | 프로세스 생성 이유

새로운 일괄처리 작업 (new batch job)	보통 테이프나 디스크를 통해 운영체제에 일괄 작업 제어 스트림이 제공된다. 운영체제가 새로운 작업을 처리할 준비가 되면, 다음에 수행할 일련의 작업 제어 명령을 읽어들이는 것이다
대화형 로그인	사용자가 터미널에서 시스템에 로그인한다.
서비스를 제공하기 위해 운영체제가 생성	사용자가 대기할 필요 없도록, 운영체제는 사용자 프로그램을 대신하여 어떤 기능을 수행할 프로세스(예를 들어, 프린트 작업을 제어하는 프로세스)를 생성할 수 있다.
기존 프로세스에 의한 생성(spawn)	모듈화를 위해서나 병렬성을 활용하기 위해, 사용자 프로그램은 많은 프로세스의 생성을 명령할 수 있다.

Process States: Extend1

■ Process Creation and Termination

- ✓ Process Termination: exit from process model
 - Normal completion
 - Bound violation, protection error, arithmetic error, ...
 - Time limit exceeded, Time overrun
 - I/O failure, Invalid instruction, Data misuse
 - Operator or OS intervention
 - Parent termination or request

표 3.2 | 프로세스 종료 이유

정상 완료	프로세스가 수행 완료를 알리는 운영체제 서비스 호출 수행
시간 한도 초과	프로세스가 명시된 전체 시간 한도보다 더 오래 수행되었다. 측정될 수 있는 시간의 종류에는 전체 경과시간(실제 시간, wall clock time), 수행에 사용된 시간, 그리고 대화형 프로세스의 경우 마지막 사용자 입력이 제공된 이후 시간 등 여러 가지가 있을 수 있다.
메모리 부족	프로세스가 시스템이 제공할 수 있는 메모리보다 더 많은 용량 요청
경계범위 위반	프로세스가 접근이 허용되지 않는 메모리 위치에 접근하려 시도한다.
보호 오류	프로세스가 사용 금지된 파일과 같은 자원을 사용하려 하거나, 읽기전용 파일에 쓰기를 시도하는 등 부적절한 방식으로 자원을 사용하려고 한다.
산술 오류	프로세스가 0으로 나누기와 같은 금지된 계산을 하거나, 하드웨어가 수용할 수 있는 것보다 큰 숫자의 저장을 시도한다.
시간 초과	프로세스가 어떤 이벤트 발생을 명시된 최대 시간을 초과하여 기다렸다.

Process States: Extend1

■ Five-State Process Model

- ✓ Running
- ✓ Ready
- ✓ 블록/대기 (Blocked/Waiting)
 - IO 완료 등과 같은 어떤 이벤트가 발생할 때까지, 수행될 수 없는 프로세스
- ✓ New
- ✓ Exit

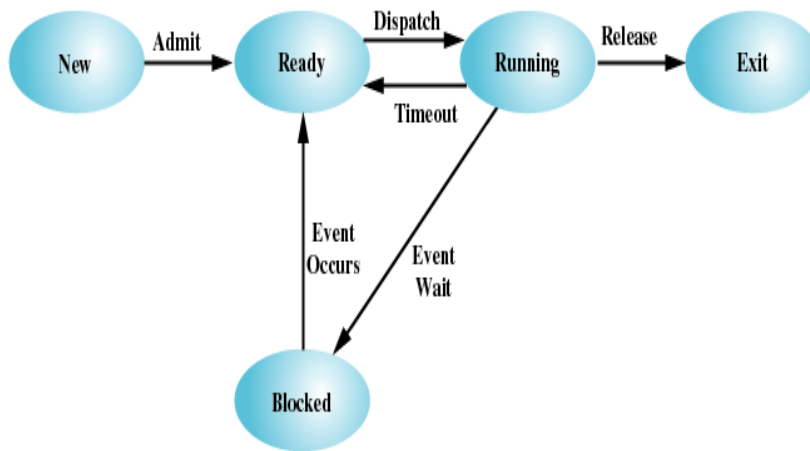
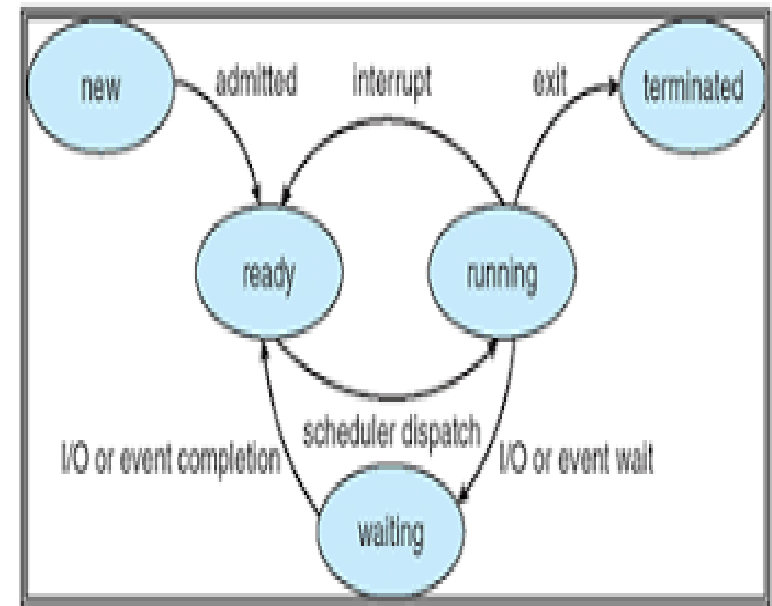


Figure 3.6 Five-State Process Model

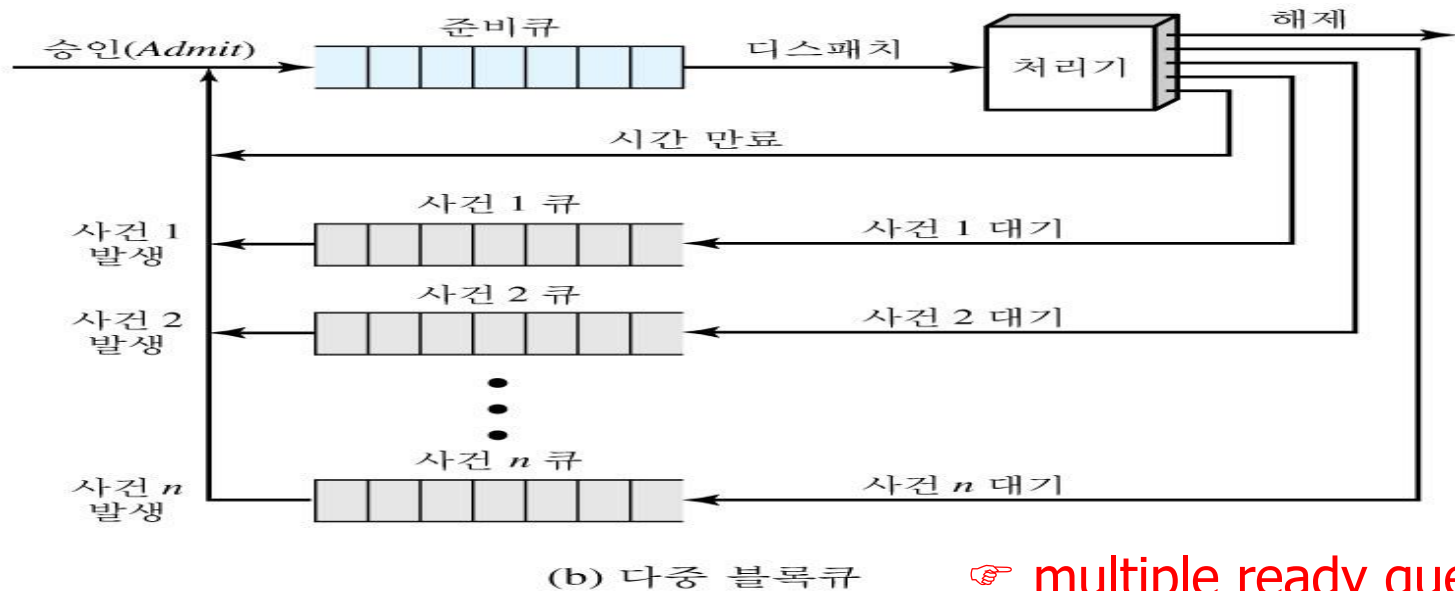
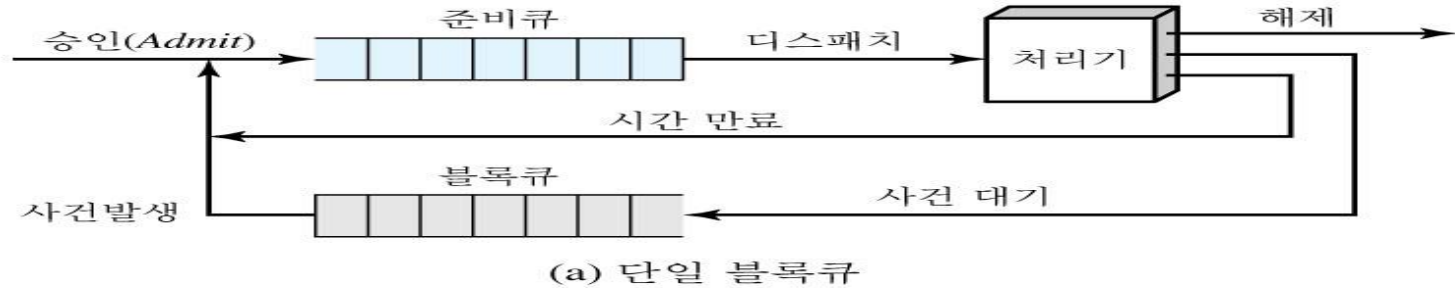


Process States: Extend1

- Five-State Process Model: transition
 - ✓ Null → New: create
 - ✓ New → Ready: admit
 - ✓ Ready → Running: **dispatch**
 - ✓ Running → Ready: **preempt** (timeout or higher priority)
 - ✓ Running → Blocked: event wait (sleep)
 - ✓ Blocked → Ready: event occur (wakeup)
 - ✓ Running → Exit: terminate
 - ✓ Ready → Exit
 - ✓ Blocked → Exit

Process States: Extend1

■ Five-State Process Model: 여러 큐를 사용



👉 multiple ready queue?

Process States: Extend2

■ Suspension

표 3.3 | 프로세스 보류의 이유

스와핑	수행할 준비가 된 프로세스를 불러 들여오기 위해 운영체제는 충분한 주기억장치를 방출할 필요가 있다.
운영체제의 다른 이유	운영체제는 후면 프로세스, 유틸리티 프로세스, 또는 문제를 야기한다고 의심되는 프로세스를 보류시킬 수 있다.
대화식 사용자의 요청	사용자는 디버깅을 위해서 또는 자원 사용과 관련해서 프로그램의 수행을 보류시킬 수 있다.
타이밍	주기적으로 수행되는 프로세스(예, 과금 프로세스나 시스템 모니터링 프로세스)가 다음 주기까지 대기할 때 보류될 수 있다.
부모 프로세스 요청	부모 프로세스는 보류된 자식 프로세스를 검사하거나 수정하기 위해, 또는 다양한 자손들의 활동을 조정하기 위해 자식 프로세스의 수행을 보류시킬 수 있다.

Process States: Extend2

✓ Swapping

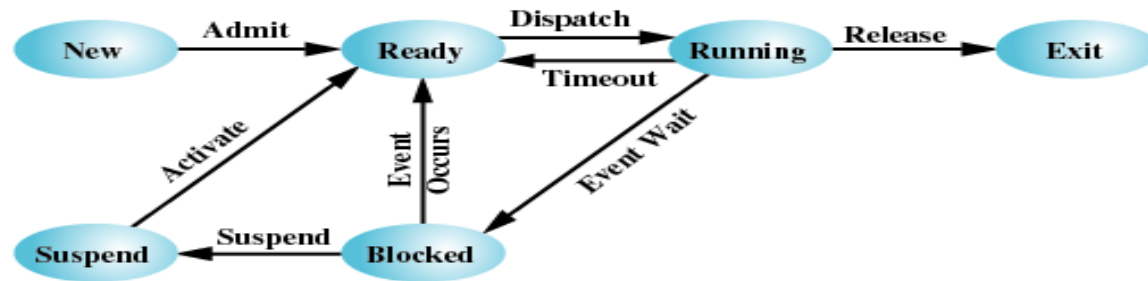
- moving part or all of a process from main memory to disk
- 주기억장치에 있는 프로세스 중에서 준비상태에 있는 프로세스가 하나도 없다면, OS는 블록된 프로세스중 1개를 디스크로 내보내고, 보류 큐(Suspended Q)에 넣음

✓ The need for swapping

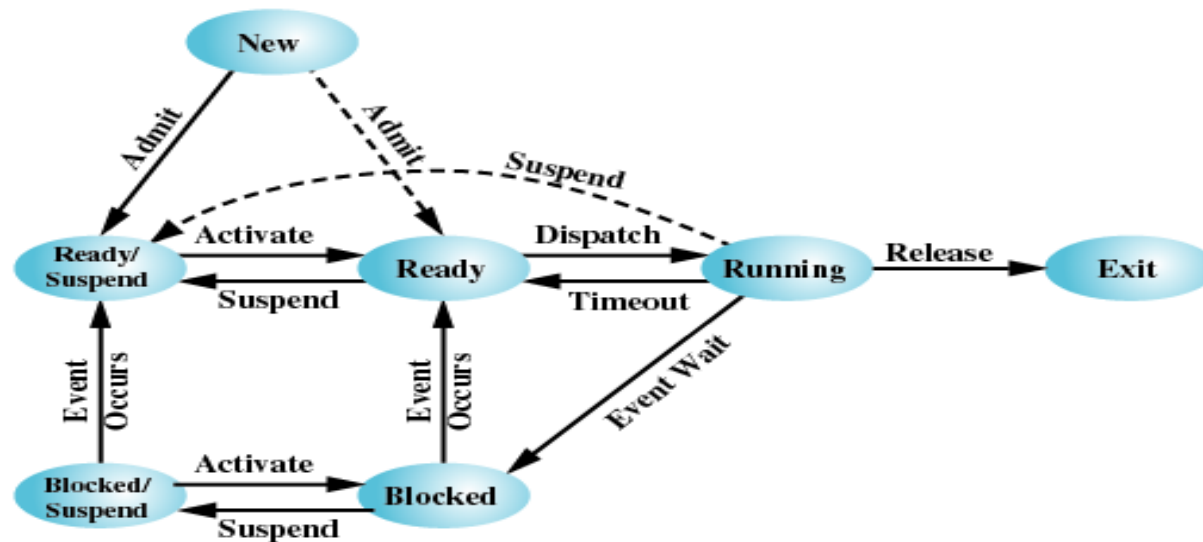
- To free up more memory
- When none of the processes in main memory is in the Ready state

Process States: Extend2

■ Suspended Processes



(a) With One Suspend State



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

Process States: Extend2

- Swapping을 사용하려면, 또 다른 상태인 보류상태 (Suspended state) 추가 필요
 - ✓ OS 설계 시에는 프로세스가 사건을 기다리고 있는지 (블록된 것인지, 아닌지)와 주기억장치로부터 swap-out 되었는지 (보류된 것인지, 아닌지) 를 고려한 2x2 상태 정의
 - 준비(Ready)
 - 블록(Blocked)
 - 프로세스가 주기억장치에서 사건을 기다리고 있는 상태
 - 블록/보류(Blocked/Suspend)
 - 프로세스가 보조기억장치에서 사건을 기다리고 있는 상태
 - 준비/보류(Ready/Suspend)
 - 프로세스가 보조기억장치에 있지만, 주기억장치에 로드되면 즉시 수행 가능

Process States: Extend2

- New transition for Suspended Processes
 - ✓ Blocked → Blocked/Suspend: **suspend** or swap out
 - ✓ Blocked/Suspend → Ready/Suspend: wakeup in suspended state
 - ✓ Ready/Suspend → Ready: **activate** or swap in
 - ✓ Ready → Ready/Suspend: suspend or swap out

 - ✓ New → Ready/Suspended
 - ✓ Blocked/Suspend → Blocked
 - ✓ Running → Ready/Suspend
 - ✓ Any State → Exit

3.3 Process Description: OS

■ OS: Management of Processes and Resources

- ✓ 멀티프로그래밍 환경에서 여러 프로세스 (P_1, P_2, \dots, P_n) 가 있을 때, P_1 은 일부는 주기억장치에 있고 2개 I/O장치를 제어하고 있음
- ✓ P_2 는 주기억장치에 있지만 P_1 에 할당된 입출력장치를 얻기 위해 기다리면서 블럭되어 있음
- ✓ P_n 은 주기억장치로부터 swap-out되어 보류상태에 있음

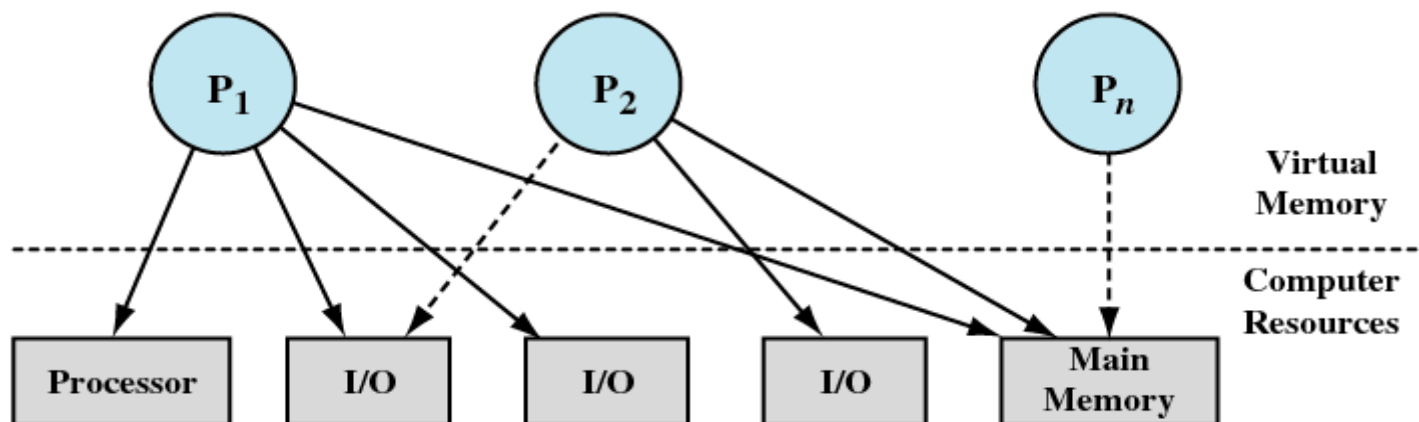


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

Process Description: OS

■ Operating System Control Structures

- ✓ Information about the current status of each process and resource
- ✓ Tables are constructed for each entity the operating system manages

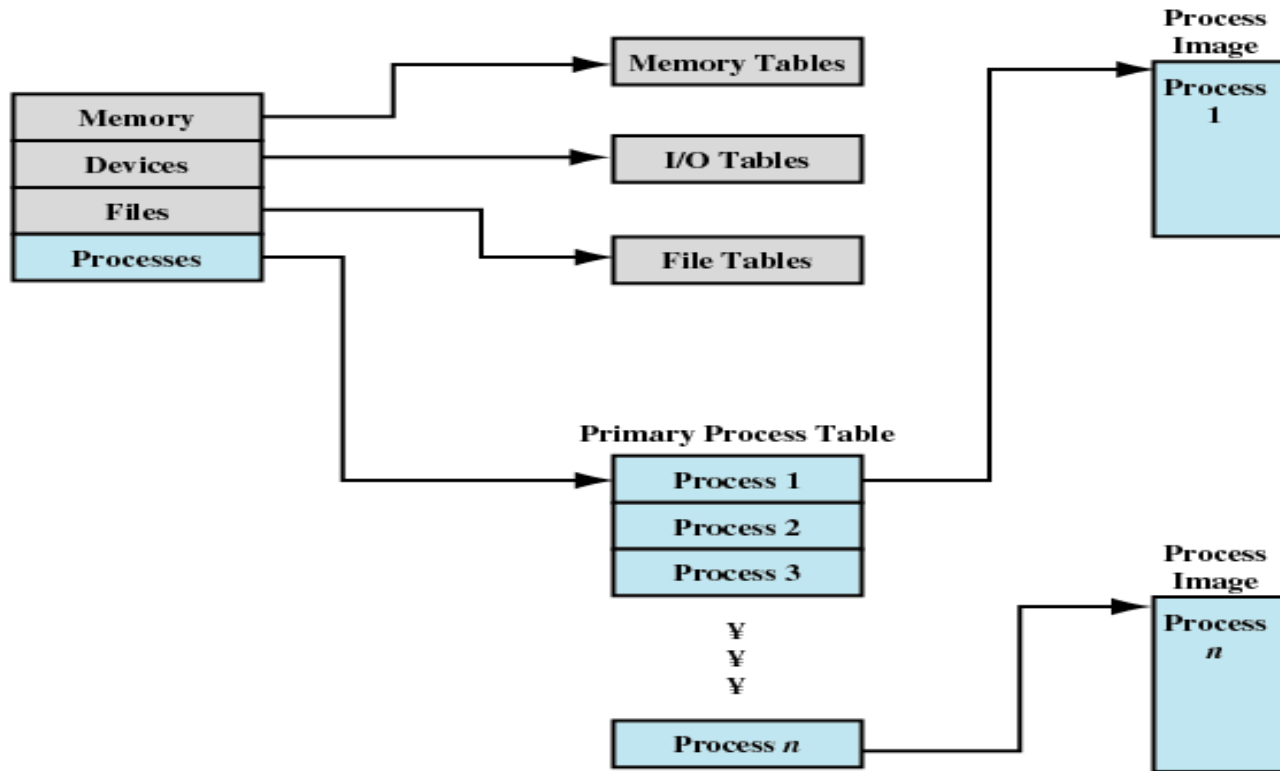



Figure 3.11 General Structure of Operating System Control Tables

Process Description: OS

■ Operating System Control Structures

✓ Memory table

- Allocation of main memory to processes  buddy (chapter 7)
- Allocation of secondary memory to processes
- 어떤 프로세스가 특정 공유메모리 영역에 접근이 가능한지를 나타내는 보호 속성
- Information needed to manage virtual memory

✓ I/O table

 drivers (chapter 11)

- Status of I/O operation: available or assigned to a particular process
- Location in main memory being used as the source or destination of the I/O transfer

✓ File table

 file system (chapter 12)

- Location on secondary memory
- Current Status, Attributes
- Information needed to manage file system

✓ Process table

- Where process is located
- Attributes in the process control block: Program, Data, Stack

Process Description: Process Control Structure

■ PCB (Process Control Block)

- ✓ Per each process
- ✓ OS에 의해 관리
- ✓ 다수의 프로세스를 지원하고 다중처리를 제공할 수 있게 지원하는 주요 자료구조

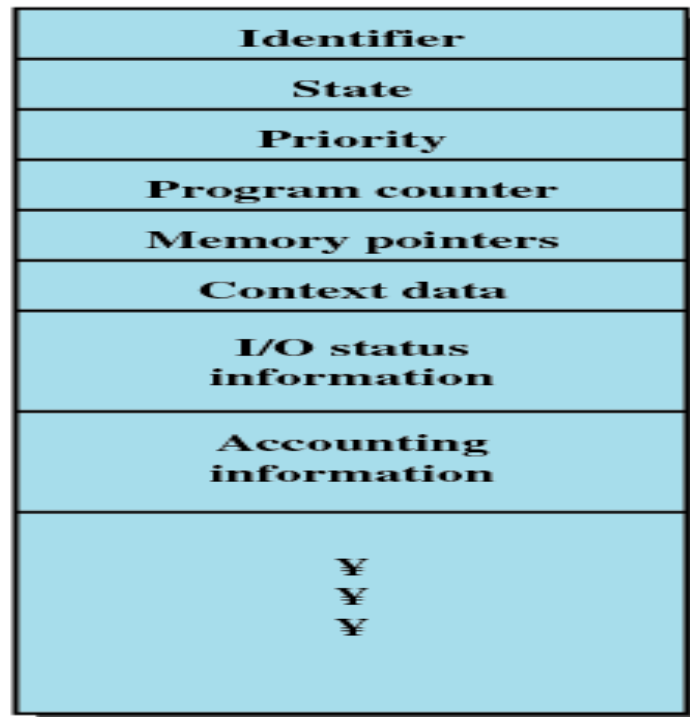


Figure 3.1 Simplified Process Control Block

프로세스 기술: 프로세스 제어 구조

- **PCB (계속)**

- 프로세스 식별: 숫자로 된 식별자
 - PID, PPID, UID, ...
- 처리기 상태 정보
 - User-Visible Registers :사용자가 이용 가능한 8~32개 레지스터
 - 제어 및 상태 레지스터
 - *Program counter, Condition codes, Status information*
 - 스택 포인터
 - PSW
- 프로세스 제어 정보
 - 스케줄링 및 상태 정보
 - *Process state, Priority, Scheduling-related information, Event*
 - 다른 프로세스들 간의 포인터 정보
 - 프로세스간 통신
 - 프로세스 권한
 - 메모리 관리
 - 자원 소유권 및 이용률

프로세스 기술: 프로세스 제어 구조

- 프로그램 상태 워드 (PSW)
 - ✓ 예: x86 상에서 EFLAGS 레지스터



X ID	=	식별 플래그	C DF	=	방향 플래그
X VIP	=	가상 인터럽트 대기(pending)	X IF	=	인터럽트 가능 플래그
X VIF	=	가상 인터럽트 플래그	X TF	=	트랩 플래그
X AC	=	정렬 검사(Alignment check)	S SF	=	부호 플래그
X VM	=	가상 8086 모드	S ZF	=	제로 플래그
X RF	=	재개 플래그	S AF	=	보조 자리올림 플래그
X NT	=	중첩 태스크 플래그	S PF	=	패리티 플래그
X IOPL	=	입출력 특권 레벨	S CF	=	자리올림 플래그
S OF	=	오버플로우 플래그			

S 는 상태 (Status) 플래그를 나타냄
C 는 제어 (Control) 플래그를 나타냄
X 는 제어 (System) 플래그를 나타냄
음영으로 표시된 비트들은 나중을 위해 예약되어 있음

그림 3.12 x86 EFLAGS 레지스터

Process Description: Process Control Structure

■ PCB의 전형적인 구성 요소

표 3.5 | 프로세스 제어블록의 일반적인 구성요소

프로세스 식별

식별자

프로세스 제어블록에는 숫자로 된 다음과 같은 식별자가 있다.

- 이 프로세스의 식별자
- 이 프로세스를 생성한 프로세스(부모 프로세스)의 식별자
- 사용자 식별자

처리기 상태 정보

사용자 가용 레지스터

이것은 처리기가 사용자 모드에서 수행하는 기계 언어에 의해 참조될 수 있는 레지스터이다. 어떤 RISC 구현에서는 100개 이상 있을 수 있지만, 대개 8개에서 32개의 레지스터가 있다.

제어 레지스터 및 상태 레지스터

다음과 같이 처리기의 동작을 제어하기 위해 사용되는 다양한 처리기 레지스터들이 있다.

- 프로그램 카운터: 다음에 반입(fetch)할 명령어의 주소를 가짐
- 조건 코드(condition codes): 가장 최근에 수행된 산술 또는 논리 연산의 결과(즉, 부호, 0, 자리올림, 같음, 오버플로)
- 상태 정보: 인터럽트 가능/불가 플래그들과 수행 모드를 가짐

스택 포인터

각 프로세스는 하나 이상의 후입선출(LIFO) 시스템 스택을 가진다. 스택은 프로시저와 시스템 호출의 매개변수와 호출 주소를 저장하는 데 사용된다. 스택 포인터는 스택의 정상(top)을 가리킨다.

Process Description: Process Control Structure

■ PCB의 전형적인 구성 요소

프로세스 제어 정보

스케줄링과 상태 정보

운영체제가 스케줄링 기능을 수행하기 위해 필요한 정보들이다. 전형적인 항목들은 다음과 같다.

- 프로세스 상태: 수행되기 위해 스케줄될 프로세스의 준비 상황을 정의한다(예, 수행, 준비, 블록, 중단).
- 우선순위: 프로세스의 스케줄링 우선순위를 나타내기 위해서 하나 이상의 필드가 사용될 수 있다. 어떤 시스템에서는 여러 개의 값이 요구된다(즉, 기본값, 현재값, 허용가능 최대치).
- 스케줄링과 관련된 정보: 이 값은 사용되는 스케줄링 알고리즘에 따라 다르다. 프로세스가 대기하고 있었던 시간, 마지막에 수행되었던 시간 등이 그 예가 될 수 있다.
- 이벤트: 프로세스가 재시작될 수 있기 전에 기다리고 있는 사건을 알려준다.

자료구조화(data structuring)

프로세스는 큐, 링, 또는 다른 자료구조로 다른 프로세스와 연결될 수 있다. 예를 들어, 특정 우선순위를 가지고 대기 상태에 있는 모든 프로세스들은 하나의 큐에 연결될 수 있다. 프로세스는 다른 프로세스와 부모-자식(생성한 자와 생성된 자) 관계가 될 수도 있다. 그러한 구조를 지원하기 위해 프로세스 제어블록에는 다른 프로세스들을 가리키는 포인터를 포함할 수 있다.

프로세스 간 통신(IPC)

여러 플래그와 시그널, 메시지 등이 두 개의 독립된 프로세스 사이의 통신과 연관될 수가 있다. 이 정보의 전부 또는 일부가 프로세스 제어블록 내에 유지된다.

프로세스 권한(process privileges)

프로세스는 접근할 수 있는 메모리 조건 및 수행할 수 있는 명령어 유형 등에 대해 어떤 권리를 가진다. 덧붙여, 이 권한은 시스템 유틸리티와 서비스를 사용하는 경우에도 적용될 수 있다.

메모리 관리

이 부분은 프로세스에 할당된 가상 메모리를 나타내는 세그먼트나 페이지 테이블로의 포인터를 가질 수 있다.

자원의 소유권과 이용률

프로세스에 의해 제어되는 자원들은 개방된(opened) 파일처럼 표시될 수 있다. 처리기 및 자원들의 이용률에 대한 이력 정보도 포함될 수 있는데, 이 정보는 스케줄러에 의해 사용된다.

프로세스 기술: 프로세스 제어 구조

■ 가상 메모리에서 사용자 프로세스 (그림 3.13)

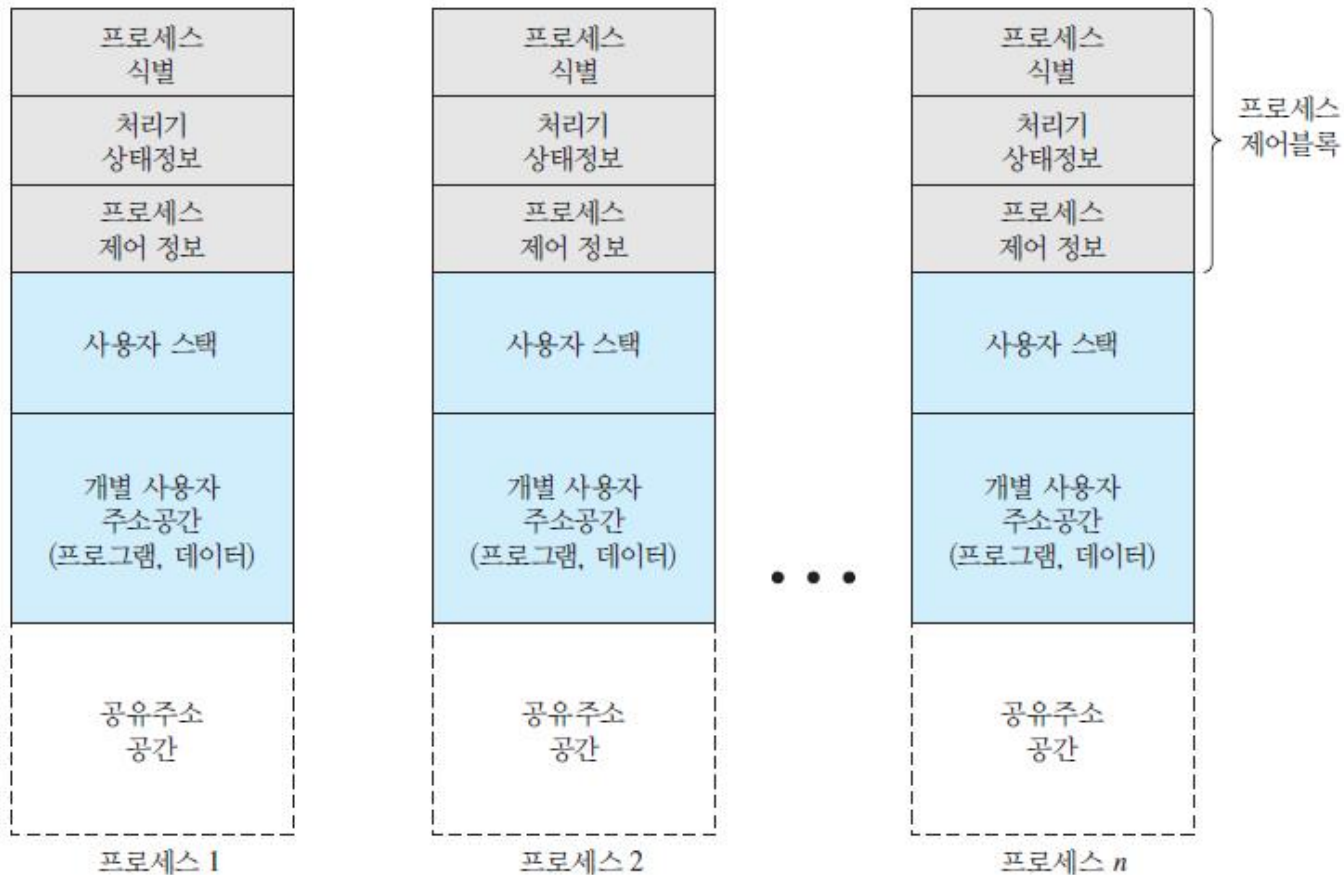


그림 3.13 가상메모리에서의 사용자 프로세스들

Process Description: Process Control Structure

■ User Processes and List structure

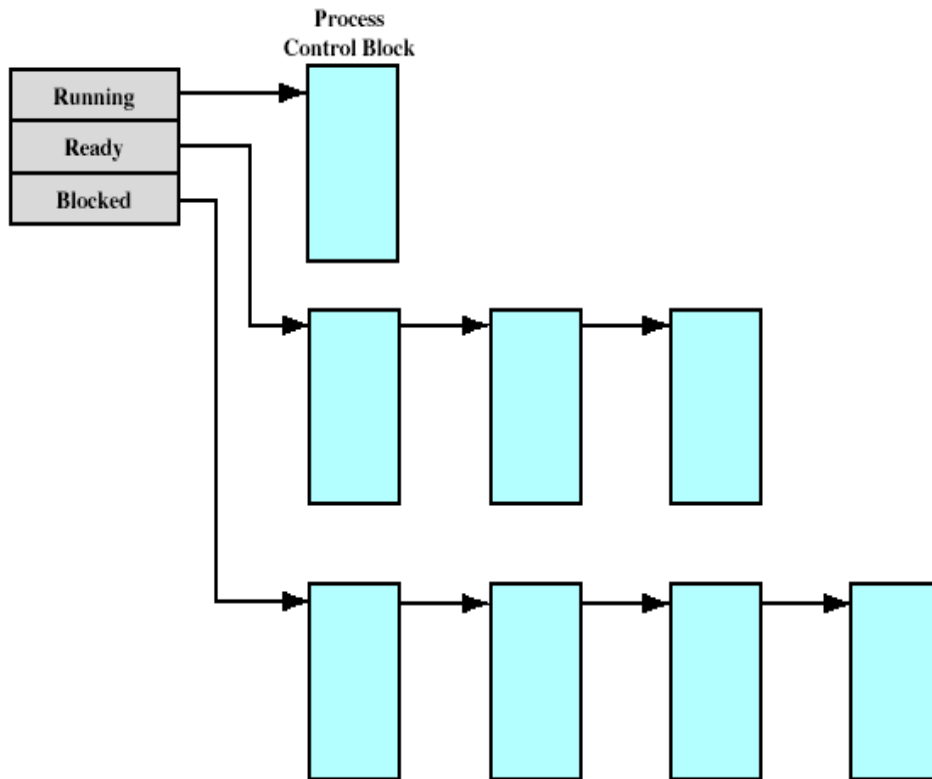
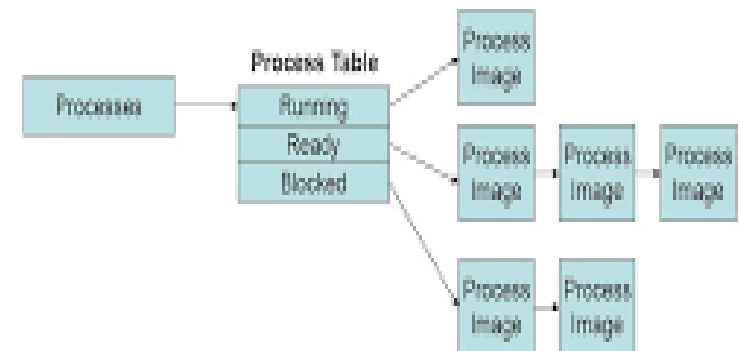


Figure 3.14 Process List Structures

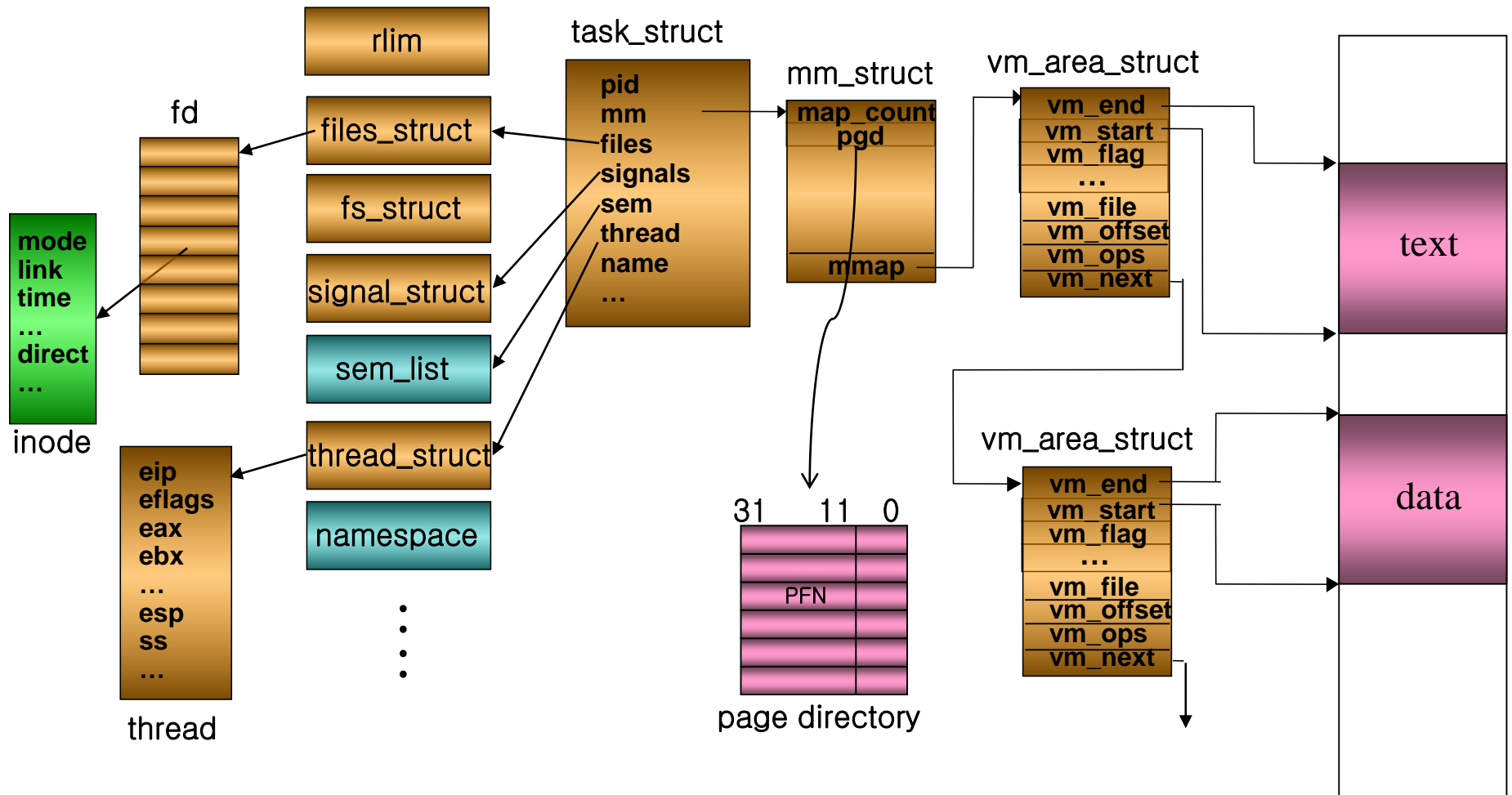
Revised Design of Process Table



Linked List or array structure

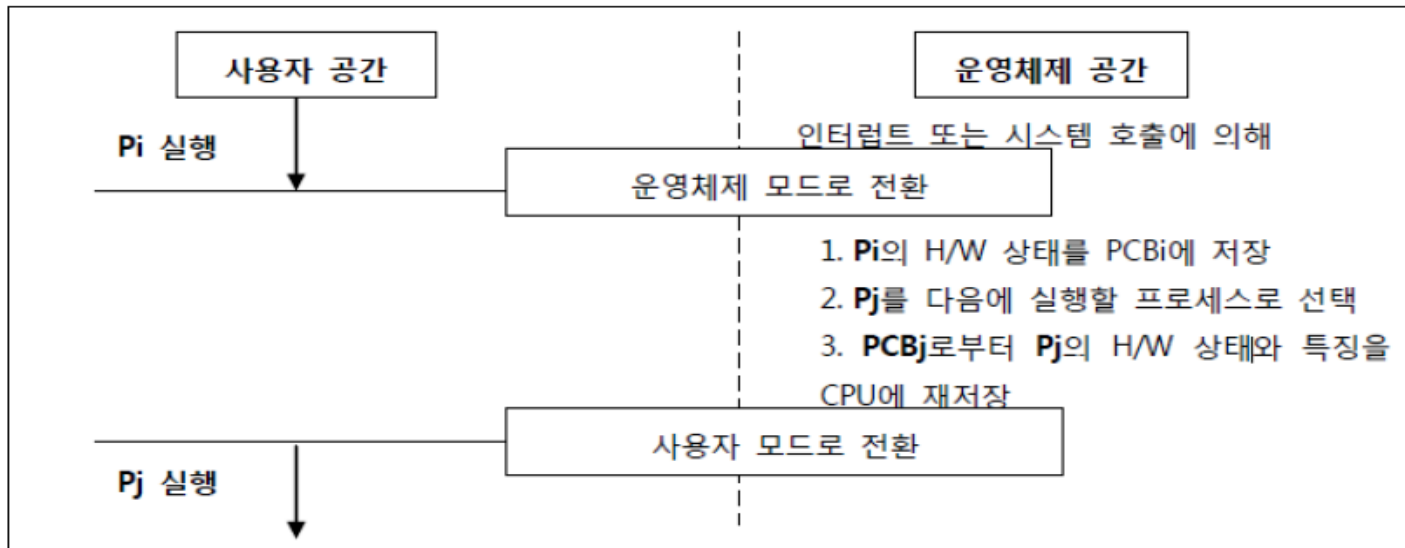
Process Description: Process Control Structure

■ Process description in Linux



3.4 Process Control: Mode Switch

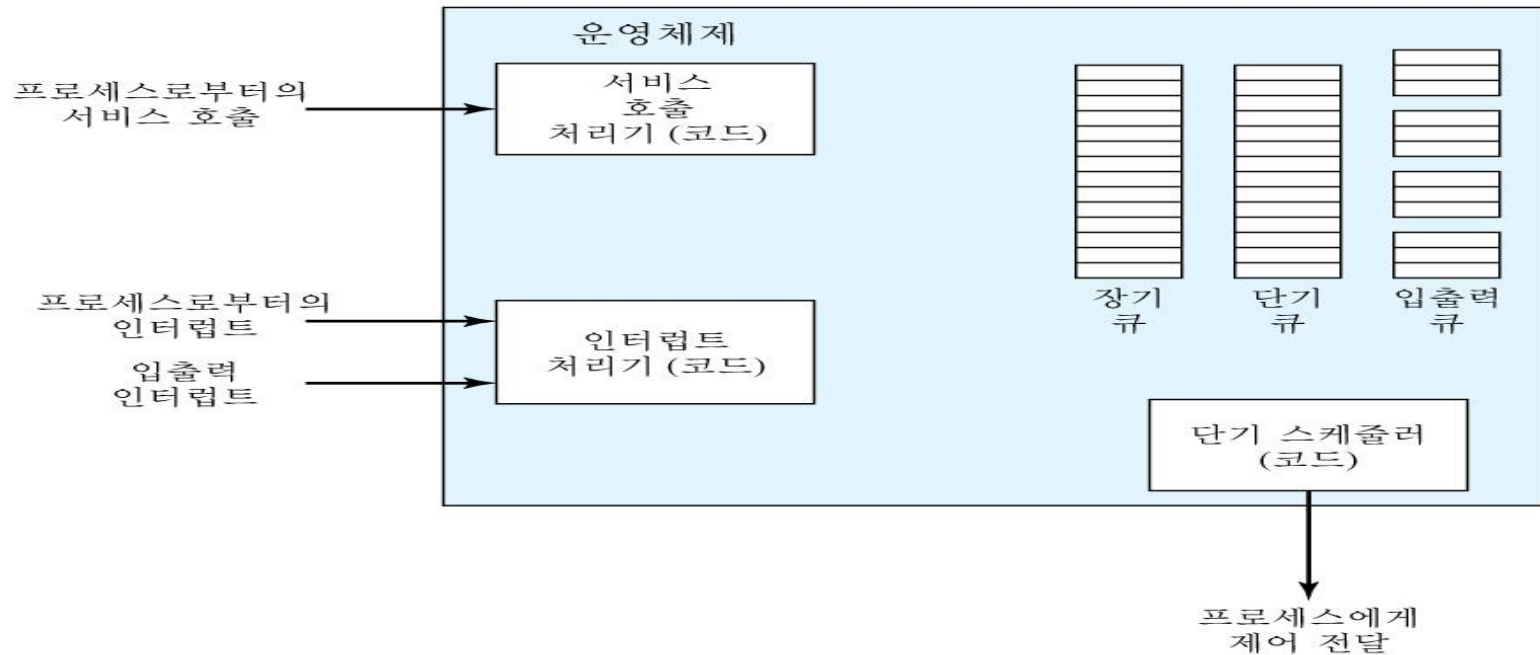
- Mode of Processor Execution
 - ✓ User mode
 - ✓ Kernel mode (control or system or supervisor mode)
- 모드 전환(mode switching) ≡ 모드 전이
- 처리기 실행 모드
 - ✓ 대부분의 처리기는 최소한 두 가지의 수행 모드를 제공
 - ✓ 사용자 모드(user mode)
 - ✓ 시스템 모드(system mode, OS mode, kernel mode)



3.4 Process Control: Mode Switch

■ When to switch mode?

- ✓ 사용자 프로그램으로부터 OS 및 PCB 와 같은주요 OS Table 들을 보호하기 위해 2가지 모드 사용



프로세스 제어: 프로세스 생성

- 유일한 프로세스 식별자 할당 (unique PID)
- 프로세스의 주소공간 할당
 - ✓ 사용자 주소공간 및 사용자 스택
 - ✓ 공유 영역 (shared area)
- PCB 초기화
 - ✓ PC, SP, 초기 상태, 우선순위, ...
- 적절한 연결 설정
 - ✓ 새로운 프로세스를 준비큐 또는 준비/보류 큐에 삽입
- 다른 자료구조를 생성하거나 확장
 - ✓ OS may maintain an accounting file

프로세스 제어: 프로세스 교환

- 프로세스 교환을 유발하는 사건들
 - ✓ Clock interrupt: 최대 허용된 시간단위(time slice)가 지나면 발생
 - 수행 → 준비
 - ✓ I/O interrupt
 - 블록 → 준비
 - ✓ 메모리 폴트 (페이지 부재)
 - 수행 → 블록
 - ✓ 트랩 (Trap)
 - 수행 중에 발생한 오류 및 예외상황(exception)
 - 해당 프로세스가 종료될 수도 있음 ← 복구 불가능한 상황
 - ✓ 슈퍼바이저 호출 (Supervisor call)
 - file open 등
 - 이 때 모드 전환 발생, 필요하면 프로세스 교환

프로세스 제어: 프로세스 교환

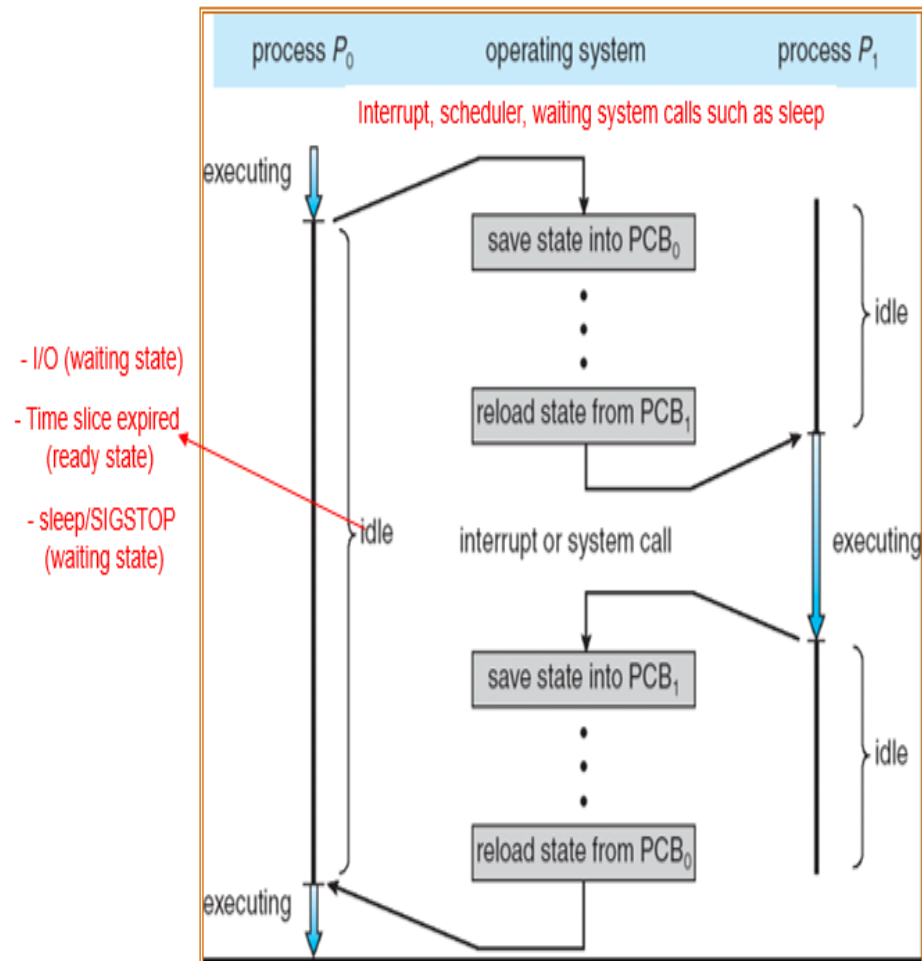
■ 프로세스 상태 변경(change)

1. 프로그램 카운터 및 다른 레지스터들을 포함하여 처리기의 문맥을 저장
2. 현재 수행 상태에 있는 프로세스(예로 P_A)의 PCB를 갱신
3. 그 PCB를 준비큐, 블록큐, 또는 준비/보류큐 중의 하나에 삽입
4. 실행할 다른 프로세스(예로 P_B)를 선택
5. 새로 선택된 프로세스(P_B)의 PCB를 갱신
6. 메모리 관리 자료구조를 갱신
7. 선택된 프로세스(P_B)의 문맥을 복원(restore)

☞ **context switch (문맥 교환)**

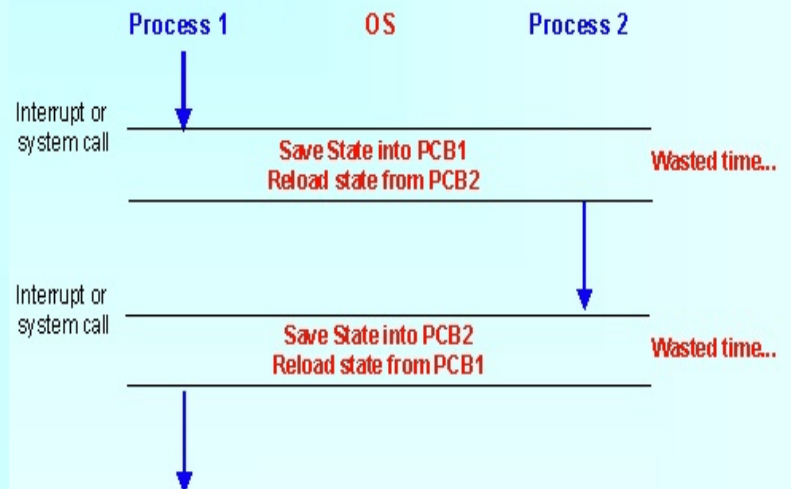
☞ **mode switch may occur without changing the state of the current running process**

프로세스 제어: 프로세스 교환



Context Switch

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.

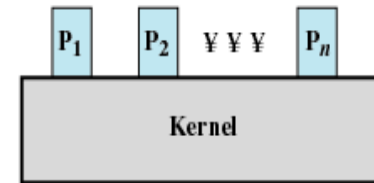


3.5 Process Control: 운영체제의 수행

■ Execution of OS

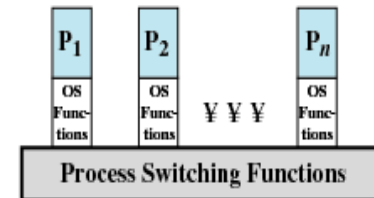
✓ 비 프로세스 커널 (Non-process Kernel, 분리된 커널) -> (a)

- 모든 프로세스의 외부에서 OS 커널 수행
- OS 코드는 분리된 개체로서 수행이 되고, 특권모드(privileged mode)에서 동작
- 프로세스 개념이 단지 user programs에만 적용됨

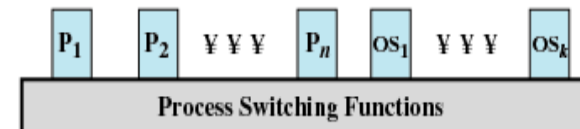


(a) Separate kernel

* 기능 = function = 함수



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

Figure 3.15 Relationship Between Operating System and User Processes

3.5 Process Control: 운영체제의 수행

■ Execution of OS

- ✓ 사용자 프로세스 내에서 실행(Execution within User Processes) -> (b)
 - 사용자 프로세스 문맥에서 모든 운영체제 SW 수행
 - Need mode switch only
- ✓ 프로세스 기반 OS (Process-Based OS) -> (c)
 - 시스템 프로세스들의 집합으로 OS를 구현
 - 다중 처리기(multi-processor) 또는 다중 컴퓨터 환경에 유용

Process Control: Execution of the Operating System

■ Execution of the OS within User Processes

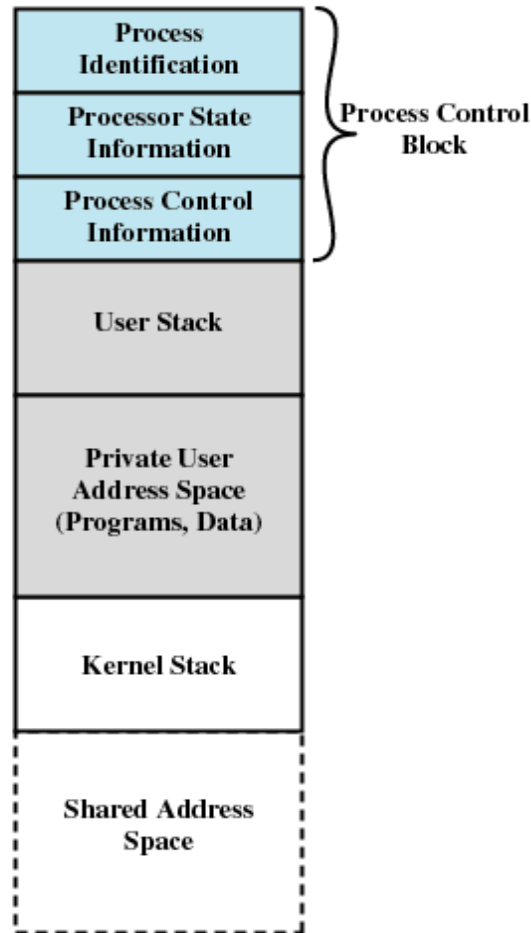


Figure 3.16 Process Image: Operating System Executes Within User Space

3.6 UNIX SVR4 Process Management

- Model: operating system executes within user processes
- Process states: 9

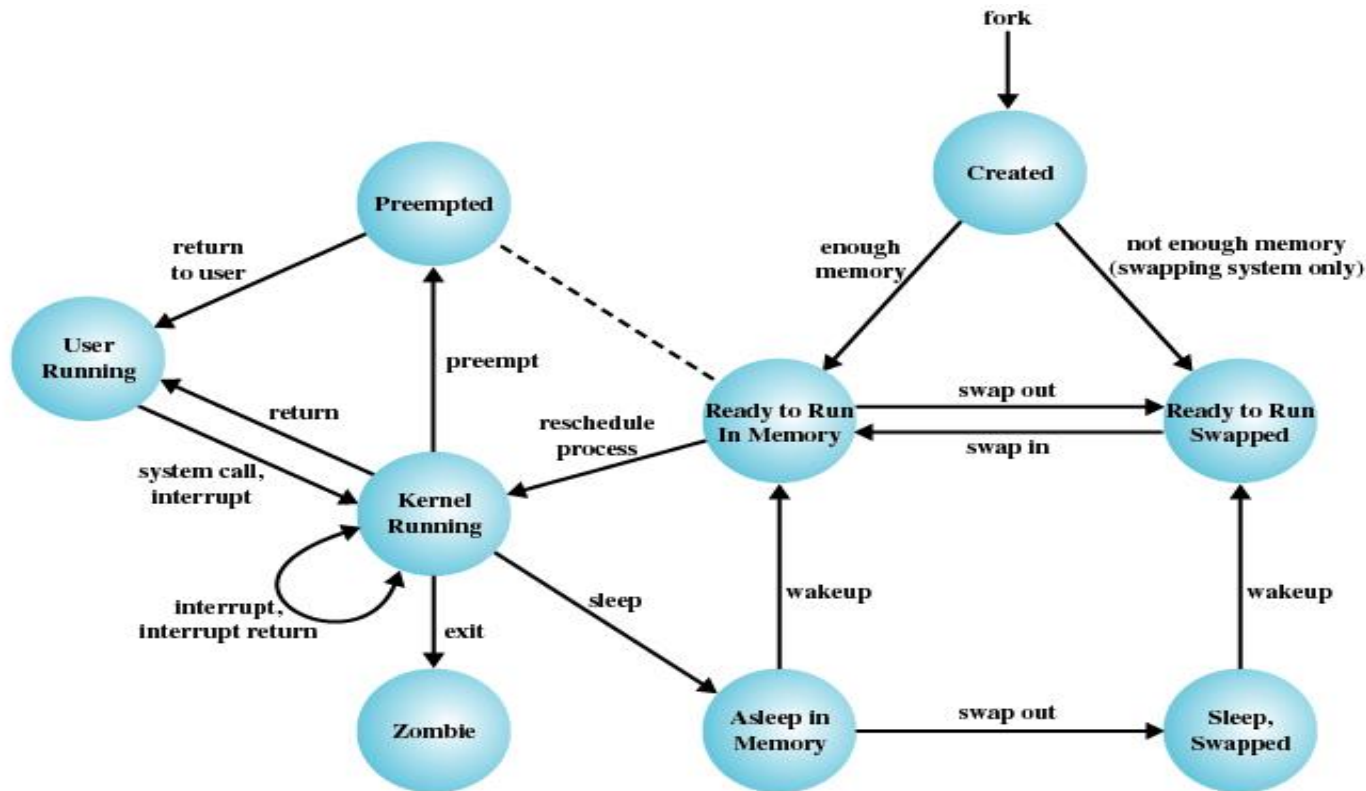


Figure 3.17 UNIX Process State Transition Diagram

UNIX SVR4의 프로세스 관리

■ 프로세스 기술 (Process Description)

표 3.10 | UNIX 프로세스 이미지

	사용자 수준 문맥
프로세스 텍스트 프로세스 데이터 사용자 스택 공유 메모리	프로그램에서 수행가능한 기계 명령어 이 프로세스의 프로그램이 접근할 수 있는 데이터 사용자 모드에서 수행되는 함수의 인자와 지역 변수, 포인터를 보유 다른 프로세스와 공유하는 메모리로 프로세스 간 통신에 사용
	레지스터 문맥
프로그램 카운터 처리기 상태 레지스터 스택 포인터 범용 레지스터	다음에 수행될 명령어의 주소로, 해당 프로세스의 커널 또는 사용자 메모리 공간 내의 주소 선점될 때의 하드웨어 상태를 보유함. 내용과 형식은 하드웨어에 따라 다름. 특정 시점의 연산 모드에 따라, 커널이나 사용자 스택의 정상을 가리킴. 하드웨어에 따라 다름.
	시스템 수준 문맥
프로세스 테이블 항목 사용자 영역 프로세스당 영역 테이블 커널 스택	프로세스 상태를 정의하며, 운영체제는 항상 이 정보에 접근가능 프로세스 문맥에서만 접근할 필요가 있는 프로세스 제어 정보 가상 주소를 물리 주소로 사상하는 방식을 정의한다. 또한 읽기 전용, 읽기와 쓰기, 읽기와 수행 등 프로세스에 허용된 접근 종류를 나타내는 허가 필드도 포함하고 있음. 프로세스가 커널 모드에서 수행될 때, 커널 프로시저의 스택 프레임(frame)을 포함.

UNIX SVR4의 프로세스 관리

■ 프로세스 테이블

표 3.11 | UNIX 프로세스 테이블 항목

프로세스 상태	프로세스의 현재 상태
포인터	U 영역과 프로세스 메모리 영역(텍스트, 데이터, 스택)을 가리킴.
프로세스 크기	프로세스에 할당된 공간의 크기를 운영체제가 알 수 있도록 함.
사용자 식별자	실제 사용자 ID(real user ID) 는 현재 수행 중인 프로세스를 책임지고 있는 사용자를 식별한다. 유효 사용자 ID(effective user ID) 는 프로세스가 특정 프로그램과 관련된 임시 권한을 얻기 위해 사용될 수도 있다. 그 프로그램이 프로세스의 일부로서 수행되는 동안, 프로세스는 해당 프로그램의 유효 사용자 ID를 가지고 동작한다.
프로세스 식별자	이 프로세스의 ID와 부모 프로세스의 ID. 이 값들은 fork 시스템 호출 수행 중 프로세스가 생성(Created) 상태로 진입할 때 설정된다.
사건 디스크립터	프로세스가 수면 상태에 있을 때 유효하다. 사건(event)이 발생되면, 프로세스는 준비 상태로 전이된다.
우선순위	프로세스 스케줄링을 위해 사용된다.
시그널(signal)	프로세스에 보내졌으나 아직 처리되지 않은 시그널들을 열거한다.
타이머	프로세스 수행시간과 커널의 자원 이용률, 그리고 프로세스에 알람 신호(alarm signal)를 보내기 위해 사용되는 사용자 설정 타이머가 포함된다.
P_link	준비 큐에서 다음에 연결된 프로세스를 가리키는 포인터(프로세스가 수행될 준비가 되어 있을 경우에만 유효하다)
메모리 상태	프로세스 이미지가 주기억장치에 있는지 스왑아웃되어 보조기억장치에 있는지를 알려준다. 주기억장치에 있다면, 이 필드는 또한 프로세스 이미지가 스왑아웃될 수 있는지 또는 주기억장치에서 일시적으로 락(lock)이 걸려 있는지도 알려준다.

■ 프로세스 제어

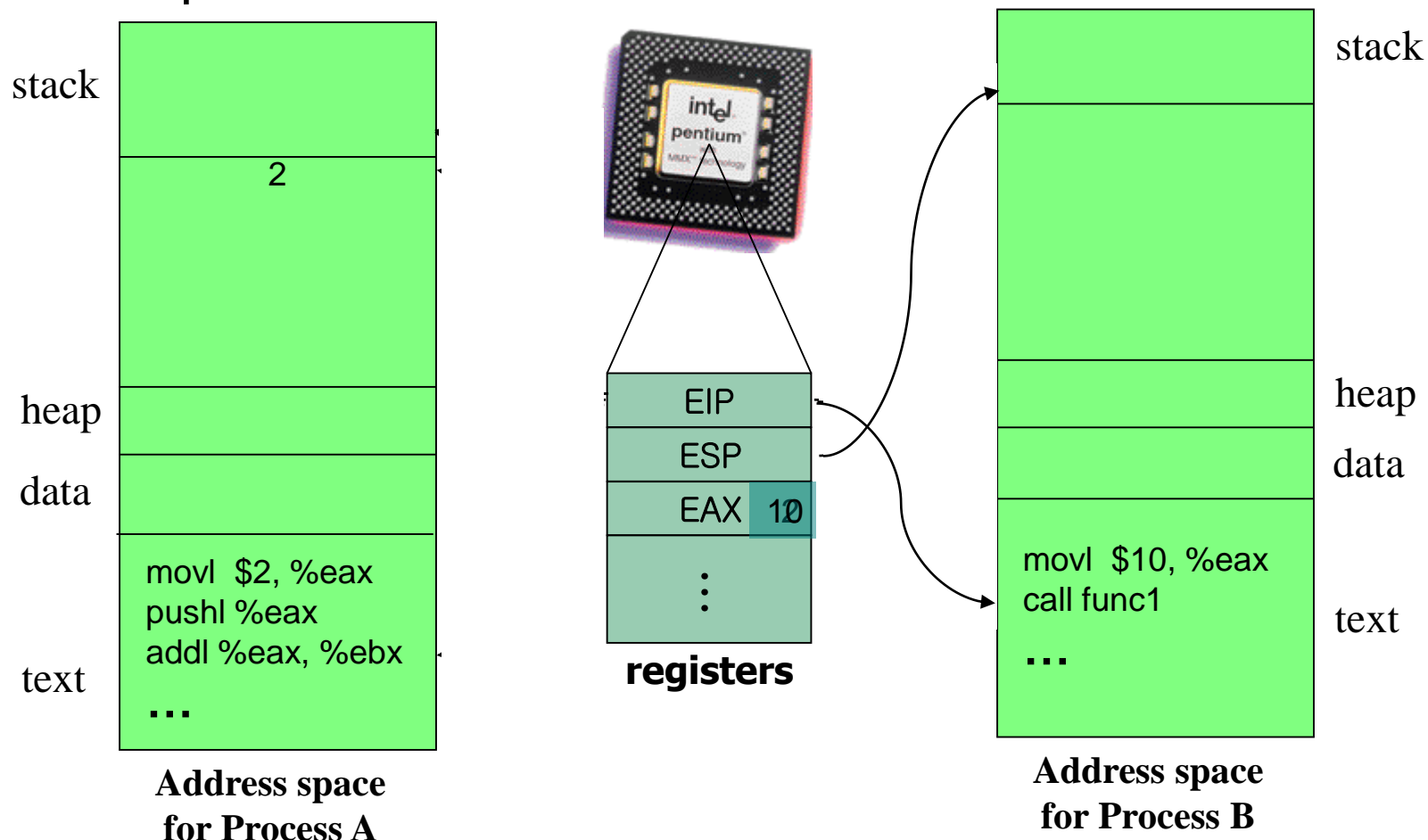
- fork()
 1. 프로세스 테이블에 한 슬롯(slot, 엔트리)를 할당
 2. 유일한 프로세스 ID (PID)를 할당
 3. 부모의 프로세스 이미지를 복사
 4. 부모가 소유하고 있는 모든 파일들의 참조계수(count)를 증가
 5. 자식 프로세스를 준비 상태로 설정
 6. 부모 프로세스에는 자식의 PID를, 자식 프로세스에는 0을 반환
- 디스패처(Dispatcher)는 다음 중 하나를 수행
 1. 제어를 부모 프로세스가 유지
 2. 자식 프로세스에 제어를 넘김
 3. 다른 프로세스에게 제어를 넘김

Summary

- Process and PCB Definition
- Process States
 - ✓ two-state model
 - ✓ five-state model
 - ✓ seven-state model
- Process Description
 - ✓ Identification, State information, Control information
- Process Control
 - ✓ Mode switch
 - ✓ Process switch
- UNIX SVR4 Process Management
 - ✓ 9-state model
 - ✓ Context: user context, register context, system context

Appendix: Context switch

■ Multiple Processes



- 👉 **Scheduler decides to preempt process A and dispatch process B.**
- 👉 **Scheduler decides to preempt process B and dispatch process A. Where..?**
- 👉 **Execution information (Register context)**

Appendix: Context switch

■ CPU abstraction

