Slide 0

2부 프로세스 관리
(4장. 프로세스)

프로세스 개념

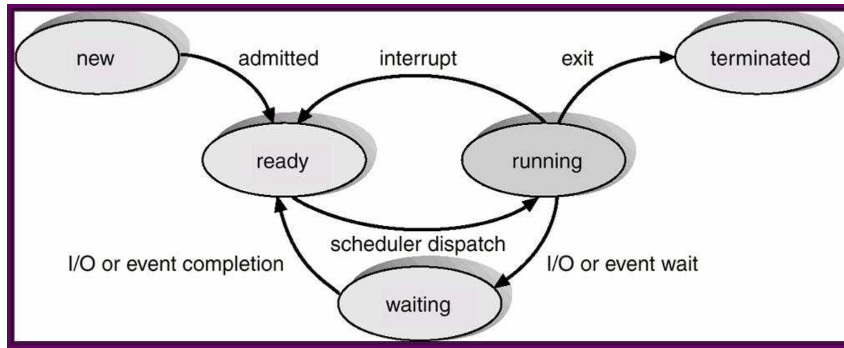
Slide 1

- 프로세스란?
  - 실행중인 프로그램
  - 하나의 프로그램이 여러개의 프로세스가 될 수 있음
    * 이 경우 프로세스의 텍스트 (text) 부분은 동일하더라도
    * 데이터나 스택의 내용은 다를 수 있음
    * 각 프로세스는 독리적으로 수행됨
- 프로세스의 구성
  - 프로그램 코드
  - 현재상태: PC (Program Counter)를 포함한 모든 레지스터의 값
  - 스택(stack): 일시적인 데이터(파라미터, 복귀주소, 지역변수)를 저장하는 곳
  - 데이터: 광역변수를 저장하는 곳

프로세스 개념 (Cont'd)

Slide 2

- 일반적인 프로세스의 상태



프로세스 개념 (Cont'd)

Slide 3

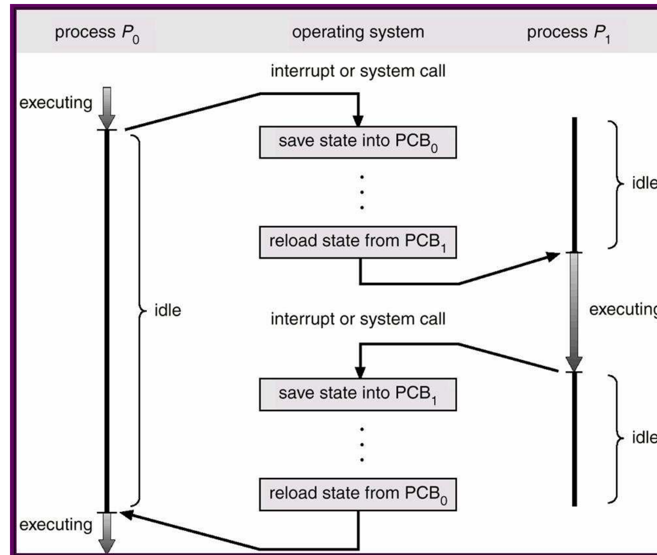
- 프로세스 제어 블록 (Process Control Block)
  - 프로세스의 모든 정보를 가지는 데이터 블록
  - 프로세스 상태
  - 프로그램 카운터
  - CPU 레지스터들
  - CPU 스케줄 정보: 우선순위등
  - 기억장치정보
    - ∗ base register 와 limit register 의 값
    - ∗ 페이지테이블, 세그먼트 테이블
  - account 정보: CPU 사용시간등의 정보
  - 입출력 상태 정보: 입출력 요구, 오픈 파일 목록등

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

프로세스 개념 (Cont'd)

Slide 4

– 프로세스의 전환 (context switching)시 PCB 사용



프로세스 개념 (Cont'd)

Slide 5

- 스레드 (Threads)
  – 단일 실행 흐름을 갖는 프로세스를 말함
  – 하나의 프로세스는 여러 실행 흐름을 가질 수 있음
    ➥하나의 프로세스는 여러개의 스레드를 가질 수 있음
    ➥하나의 프로세스는 동시에 여러개의 작업을 할 수 있음

프로세스 스케줄링

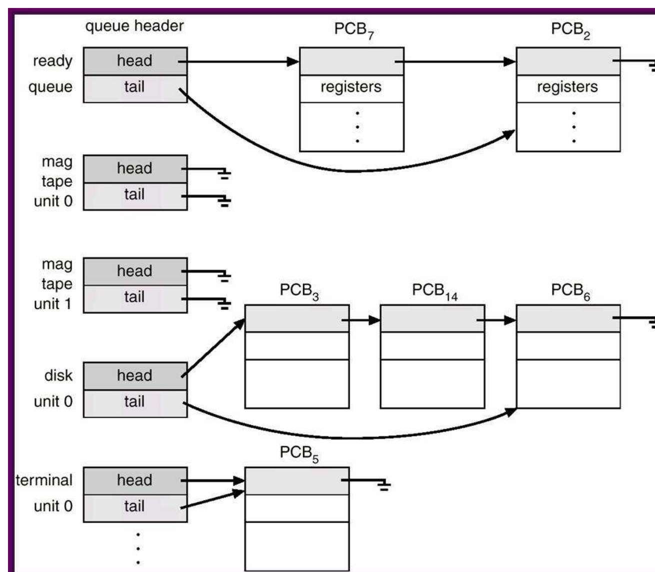
Slide 6

- 프로세스의 작업에 따라 프로세스 별로 순서나 일정을 조정하고 관리하는 작업
  - 목표: 모든 장치 (CPU 및 기타장치)의 이용률을 최대화하는 방향으로
- 스케줄링 큐 (scheduling queues)
  - 단일 프로세서에서는 한 순간에 하나의 프로세스만 실행 됨
  - 다중 프로세싱에서는 여러개의 프로세스가 동시에 수행됨
  - 이러한 프로세스를 관리하기 위해 여러개의 큐를 사용
  - 큐의 종류
    * job queue: 프로세스 시작 후 들어가는 큐로서 모든 프로세스들이 위치
    * ready queue: 주메모리에 적재되어 실행을 기다리는 프로세스들이 위치
    * device queue: 장치 사용을 기다리는 프로세스들이 위치
    * waiting queue: 사건을 기다리는 프로세스들이 위치

프로세스 스케줄링 (Cont'd)

Slide 7

- 여러가지 큐들



프로세스 스케줄링 (Cont'd)

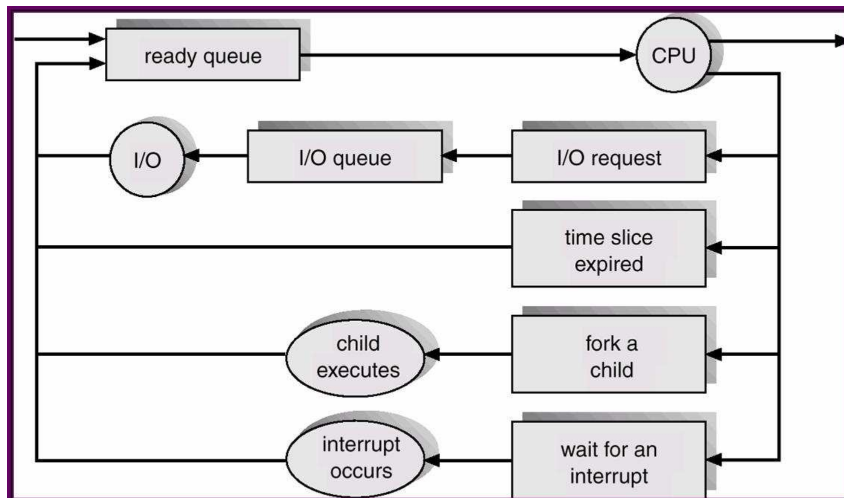
Slide 8

– 프로세스 실행 중 발생하는 상황
  * 해당 프로세스가 입출력을 요청하면 device queue 로 이동
  * 자식 프로세스를 생성하면
    · 동시수행시: ready queue 로
    · 자식 프로세스 종료후 수행시: waiting queue 로
  * 인터럽트가 발생하면
    · 현재 프로세스 중단
    · 인터럽트 서비스 루틴의 수행
    · 인터럽트 서비스 종료후 프로세스로 복귀 혹은 현재 프로세스를 ready queue 로

프로세스 스케줄링 (Cont'd)

Slide 9

– 스케줄링의 큐잉도표



프로세스 스케줄링 (Cont'd)

Slide 10

- 스케줄러
  - 종류
    - 단기 스케줄러 (short-term scheduler)
      - ready queue 에 있는 프로세스중에서 선택하여 CPU를 할당하는 스케줄러
      - 보통 하나의 프로세스는 수십밀리초 동안만 수행되므로
      - 단기 스케줄러는 수십밀리초마다 실행됨 (자주 실행됨)
      - 그러므로 처리가 빨라야함
    - 장기 스케줄러 (long-term scheduler)
      - 동시에 수행될 수 있는 작업의 수가 제한되어 있을 경우에 사용
      - disk 에 있는 프로세스중에서 선택하여 기억장치로 적재하는 스케줄러
      - 한 프로세스가 종료될 경우에만 수행됨 (가끔 실행됨)
      - 단기 스케줄러보다는 더 많은 시간을 사용해도 됨

프로세스 스케줄링 (Cont'd)

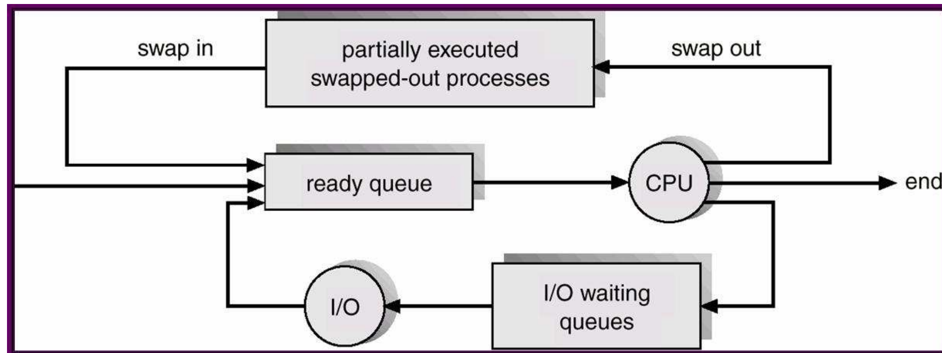
Slide 11

- 프로세스 분류
  - 입출력 중심 프로세스 (I/O-bound process)
    - 계산보다는 입출력을 많이 하는 프로세스
  - 계산 중심 프로세스 (CPU-bound process)
    - 입출력 보다는 계산을 많이 하는 프로세스
- 장기 스케줄러의 전략
  - 입출력 중심 프로세스와 계산중심 프로세스가 적절한 비율로 섞는 것이 좋음
  - 입출력 중심 프로세스가 많을 경우
    - CPU 는 놀고 입출력 큐에는 많은 프로세스가 대기함
  - 계산 중심 프로세스가 많을 경우
    - 입출력 장치는 놀고 ready queue 에는 많은 프로세스가 대기함
- 장기 스케줄러의 사용
  - 보통 시분할 시스템은 장기 스케줄러를 사용하지 않음
  - 생성되는 모든 프로세스를 ready queue 에 넣음
  - 추가적으로 중기 스케줄러 (medium-term scheduler)를 사용

프로세스 스케줄링 (Cont'd)

– 중기 스케줄러 (swapping)
  * 물리 메모리가 부족한 경우 오랫동안 사용되지 않는 프로세스등을 디스크로 보냄 (스왑아웃)
  * 실행이 필요할 경우 다시 주 메모리로 (스왑인)

Slide 12



프로세스 스케줄링 (Cont'd)

Slide 13

- 문맥 교환 (Context Switch)
  – 설명
    * 스케줄러가 새로운 프로세스의 실행을 스케줄한 경우
    * 지금까지의 프로세스 상태를 보관하고 (다음에 다시 실행하기 위하여) 새로운 프로세스의 상태를 CPU에 적재하는 과정을 문맥교환이라함
  – 프로세스의 문맥은 PCB에 보관됨
  – 문맥 교환의 오버헤드를 가능한한 줄여야함
    * 특수 명령어 사용: 하나의 명령으로 모든 레지스터를 저장 복구함
    * 레지스터 집합 사용
      · 여러개의 레지스터 집합을 두고 point만 변경
      · 레지스터 집합이 모두 사용되면 스택사용
      · 보통 RISC 에서 많이 사용하는 방법
    * 스레드 사용
      · 경량의 스레드 단위로 문맥교환함으로 오버헤드를 줄임

프로세스 수행

Slide 14

- 프로세스 생성
  - 프로세스는 실행 도중 다른 프로세스를 생성할 수 있음
  - 생성하는 프로세스 →부모 프로세스 (parent process)
    생성되는 프로세스 →자식 프로세스 (child process)
  - 실행
    * 부모는 자식과 함께 실행 됨
    * 부모는 모든 자식 프로세스가 종료될 때까지 기다림
  - 자식 프로세스의 주소공간
    * 부모 프로세스와 중복시킴
    * 새로운 프로그램을 적재함

프로세스 수행 (Cont'd)

Slide 15

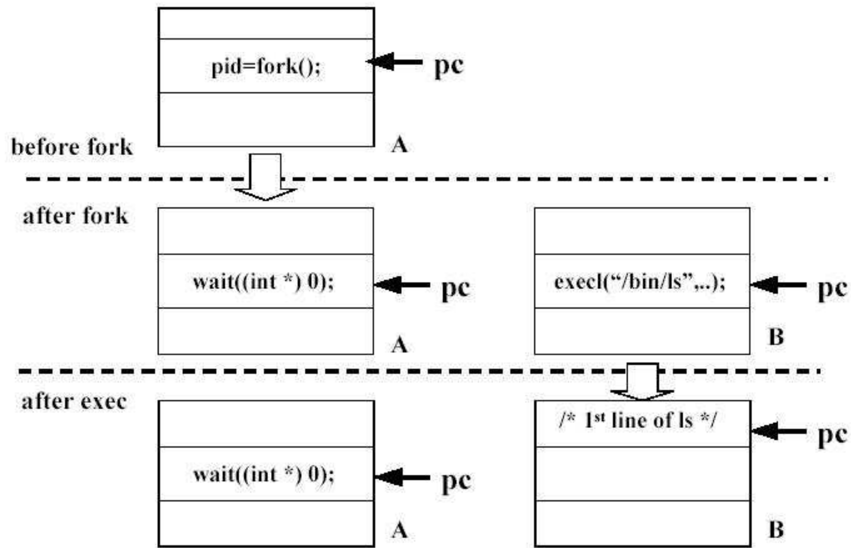
- Unix 의 예)
  * fork() 시스템 호출로 자식 프로세스 생성
    · 부모 프로세스와 동일한 주소공간 갖음 (단 하나의 차이는 fork() 리턴 값)

```
#include <stdio.h>
main()
{
    int pid,status,childPid;
    printf("I'm the parent and my PID is %d\n", getpid());
    pid = fork();
    if(pid != 0){
        printf("I'm the parent process with PID %d and PPID %d\n",
                getpid(), getppid());
        childPid = wait(&status);
    }else{
        printf("I'm the child process with PID %d and PPID %d\n",
                getpid(), getppid());
        execl("/bin/ls", "ls", "-l",NULL);
        printf("This line should not be executed !!\n");
    }
    printf("PID %d terminated\n", getpid());
}
```

프로세스 수행 (Cont'd)

Slide 16

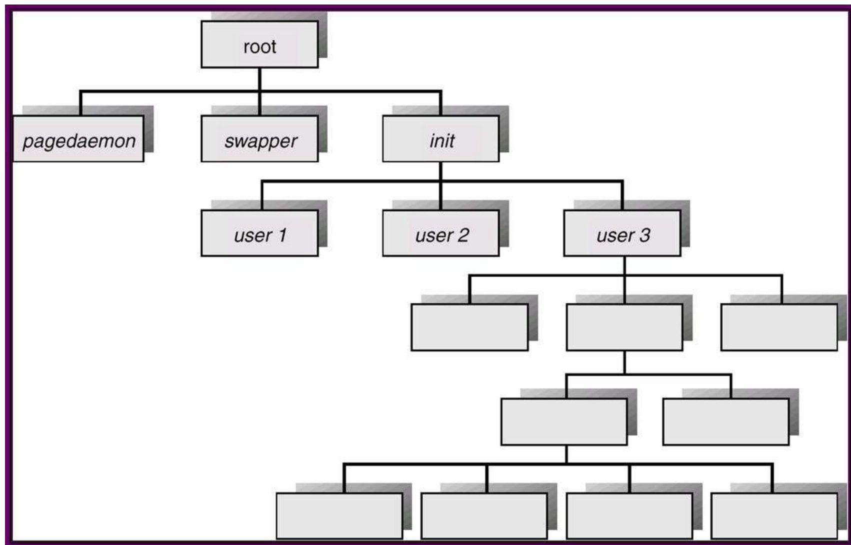
∗ fork() 시스템 호출 전후의 상황



프로세스 수행 (Cont'd)

Slide 17

– 전형적인 Unix 프로세스 트리



프로세스 수행 (Cont'd)

Slide 18

- 프로세스 종료
  - 마지막 문장을 수행후 종료
  - exit 시스템 호출을 사용하여 운영체제에게 프로세스의 삭제를 요청
  - 부모는 abort 시스템 호출을 통해 강제로 자식을 종료시킬 수 있음
  - 강제 종료해야하는 경우
    * 자식 프로세스가 할당된 자원을 초과해서 사용하는 경우
    * 자식에게 할당된 작업이 더이상 필요없을 경우
    * 부모 프로세스가 종료되는 경우
      · VMS등은 부모가 종료되면 자식을 모두 종료시킴 (cascading termination)
    * Unix 의 경우
      · cascading termination 하지 않고
      · 부모가 종료되면 고아 프로세스 (orphan process)는 init를 새부모로 함

프로세스 수행 (Cont'd)

Slide 19

- 프로세스 간 협력
  - 프로세스 종류
    * 독립 프로세스: 다른 프로세스에게 영향을 주거나 받지 않는 프로세스
    * 협조 프로세스: 다른 프로세스에게 영향을 주거나 받는 프로세스
  - 협조 프로세스가 필요한 이유
    * 정보 공유: 같은 정보를 참조할 경우
    * 연산 속도 향상
      · 태스크를 여러개의 부태스크 (subtask)로 나누고
      · 이를 다중 처리기를 통해 실행
      · 단, 복수개의 처리기 (CPU 혹은 입출력 채널) 이 있어야함
    * 모듈화
      · 모듈화를 위해 시스템 기능을 여러개의 프로세스와 스레드로 나눔
    * 편의성: 사용자가 동시에 여러가지 일을 처리하기를 원함

프로세스 수행 (Cont'd)

Slide 20

- 프로세스 간 협력의 필요조건
  - 프로세스간 통신 (IPC)
  - 동기화 메커니즘
- 상호협조의 예 (생산자 소비자 문제)
  - 한 프로세스는 생산하고 다른 한 프로세스는 이를 소비함
  - 예) 프린터 명령시
    - 프린트할 내용을 생성 (생산자)
    - 프핀트할 내용을 프린터로 보냄 (소비자)
  - 데이터의 공유를 위하여 버퍼(buffer) 필요
    - 무한버퍼
      - ☛ 생산자는 언제나 생산할 수 있음
      - ☛ 소비자는 생산을 기다릴 수 있음
    - 유한버퍼
      - ☛ 생산자는 버퍼가 모두 차 있으면 기다려야함
      - ☛ 소비자는 버퍼가 비어 있으면 기다려야함

프로세스 수행 (Cont'd)

Slide 21

- 실제 예)
  - 프로세스간 통신: 공유메모리 사용
  - 동기화 (in, out 변수와 루프로)

➥ 공유 메모리

```
#define BUFFER_SIZE 10
Typedef struct {
. . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

➥ 생산자

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

➥ 소비자

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

프로세스간 통신 (IPC)

Slide 22

- 방법
  - 공유 메모리 (위에서 설명)
  - 메시지 전달
- 메시지 전달
  - 데이터의 공유없이 프로세스간 통신
  - 두가지 연산
    * send(message): 메시지를 보냄
    * receive(message): 메시지를 받음
    * message 는 고정길이 이거나 가변길이