# 프로세스 관리

# Process Management

양희재 교수 (hjyang@ks.ac.kr) / 경성대학교 컴퓨터공학과

# 프로세스

- 프로그램 vs 프로세스 (program vs process)
  - process, task, job …
  - program in execution: text + data + stack, pc, sp, registers, …
  - 무덤 속 *프로그램*, 살아 움직이는 *프로세스*
- 프로세스 상태
  - new, ready, running, waiting, terminated (그림)
  - 프로세스 상태 천이도 (process state transition diagram)
  - 상태 천이는 언제 발생?

메모리에 올라오면 new, 실행 준비까지 다 하면 ready, 돌고 있으면 running, 기존 프로세스 중에 다른 프로세스 실행할 때는 waiting, 끝나면 terminated
running중에 자신에게 주어진 일정 시간이 지나면 강제로 ready로 돌아간다.

# PCB

- Process Control Block (PCB)
  - Task Control Block (TCB)
  - 프로세스에 대한 모든 정보
  - process state (running, ready, waiting, …), PC, registers, MMU info (base, limit), CPU time, process id, list of open files, …
  - 사람과 비유?

프로세스의 정보들을 담고 있다. 프로세스 하나당 하나의 TCB가 할당됨

프로세스가 switch후에 다시 돌아올때 재실행시킨다든가 할 때 필요

# Queues

- Job Queue 하드디스크의 수많은 job중 메모리에 들어가려면 줄을 서야 됨 이게 job queue
  - Job scheduler 어떤 걸 먼저 메모리에 올려 줄 것인가? 이게 job scheduling
  - Long-term scheduler 메모리에 빈자리가 생길때까지 몇분정도씩 기다려야 하므로 long term
- Ready Queue cpu서비스를 이용하려 해도 줄서서 기다려야 됨
  - CPU scheduler
  - Short-term scheduler io를 만나든지, time expire되면 프로세스 변경. 1초에도 수십번씩이므로 short term
- Device Queue IO등 디바이스를 사용하려 해도 줄 서서 기다려야 됨
  - Device scheduler

# Multiprogramming

- Degree of multiprogramming 메인메모리에 몇 개의 프로그램이 올라가 있는지
- i/o-bound vs CPU-bound process io bound: 하는일이 주로 io. cpu bound: 주로 계산
  적절히 믹스해서 올려야 효율적이다. 이것도 잡 스케줄러의 역할
- Medium-term scheduler 어떤 프로세스를 swapping할것인가 결정하는 스케쥴러
  - Swapping 오래 사용안하는 프로세스를 디스크로 내보내거나, 다시 올리는걸 swapping
- Context switching (문맥전환)
  - Scheduler 이렇게 디스크에서 swapping out된 프로세스를 저장하는 부분을 backing store라 한다
  - Dispatcher context를 저장하고, switching하려는 프로세스의 TCB정보를 가져와서 restore하는 프로그램
  - Context switching overhead
    switching하는데는 당연히 overhead가 존재

# CPU 스케쥴링

# CPU Scheduling

기존 process를 강제로 쫓아낼 수 있으면 비선점, 끝나거나 io만나는등까지 기다려야 되면 선점

- Preemptive vs Non-preemptive
  - 선점 (先占) : 비선점(非先占)
- Scheduling criteria
  - CPU Utilization (CPU 이용률) cpu가 일하는 시간 비율
  - Throughput (처리율) 단위시간당 몇개의 작업? job/s
  - Turnaround time (반환시간) 작업이 시작된 후 끝날 때까지 소요되는 시간
  - Waiting time (대기시간) cpu기다리는 큐에서 얼마나 기다리는지
  - Response time (응답시간) 처음 응답이 나올 때까지 기다리는 시간
  - …

# CPU Scheduling Algorithms

- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF)
  - Shortest-Remaining-Time-First
- Priority
- Round-Robin (RR)
- Multilevel Queue
- Multilevel Feedback Queue

# First-Come, First-Served

- Simple & Fair
- Example: Find *Average Waiting Time*
  - AWT = (0+24+27)/3 = 17 msec *cf. 3 msec!*

순서 바꾸면 0,3,6 으로 평균 3ms만 기다려도 된다

- Gantt Chart

| Process | Burst Time (msec) |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Convoy Effect (호위효과)
- Nonpreemptive scheduling

# Shortest-Job-First (1)

- Example: AWT = (3+16+9+0)/4 = 7 msec
  - *cf.* 10.25 msec (FCFS)
- Provably *optimal* 최적임이 증명되어 있다.
- *Not realistic*; prediction may be needed 그러나 얼마나 사용할지 미리 알기 힘들다
  이전 기록을 하더라도 overhead가 너무 큼

| Process | Burst Time (msec) |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |

# Shortest-Job-First (2)

- Preemptive or Nonpreemtive 하던걸 쫓아낼 수 있으면 preemtive
  - *cf.* Shortest-Remaining-Time-First (최소잔여시간 우선)
- Example
  - Preemptive: AWT = (9+0+15+2)/4 = 26/4 = 6.5 msec
  - Nonpreemptive: 7.75 msec

0-1(p1)
1-5(p2)
5-10(p4)
10-17(p1)
17-26(p3)

| Process | Arrival Time | Burst Time (msec) |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

# Priority Scheduling (1)

- Priority (우선순위): typically an integer number
  - Low number represents high priority in general (Unix/Linux)
- Example
  - AWT = 8.2 msec

| Process | Burst Time | Priority |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 4 |
| $P_4$ | 1 | 5 |
| $P_5$ | 5 | 2 |

# Priority Scheduling (2)

- Priority  priority를 정하는 방법에는 internal, external이 있는데 각자 ..
  - Internal: time limit, memory requirement, i/o to CPU burst, …
  - External: amount of funds being paid, political factors, …
- Preemptive or Nonpreemptive
- Problem
  - Indefinite blocking: *starvation* (기아)  계속 우선순위가 높은 애들이 들어오면..
  - Solution: againg

# Round-Robin (1)

- Time-sharing system (시분할/시공유 시스템)
- Time *quantum* 시간양자 = time *slice* (10 ~ 100msec)
  주기적으로 프로세스를 옮겨다니며 시행
- Preemptive scheduling
- Example
  - Time Quantum = 4msec
  - AWT = 17/3 = 5.66 msec

| Process | Burst Time (msec) |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

# Round-Robin (2)

- Performance depends on the size of the time quantum
  - $\Delta \rightarrow \infty$ FCFS
  - $\Delta \rightarrow 0$ Processor sharing (* context switching overhead)
- Example: *Average turnaround time (ATT)*
  - ATT = 11.0 msec ($\Delta$ = 1), 12.25 msec ($\Delta$ = 5)

| Process | Burst Time (msec) |
|---|---|
| $P_1$ | 0 6으로수정 |
| $P_2$ | 3 |
| $P_3$ | 1 |
| $P_4$ | 7 |

# Multilevel Queue Scheduling

- Process groups
  - System processes — os안에서 작업하는(가상메모리 맵핑이나 파일 등..)
  - Interactive processes — 게임 등 사용자와 대화
  - Interactive editing processes
  - Batch processes — 컴퓨터가 일괄적으로 처리
  - Student processes
- Single ready queue → Several separate queues — 큐를 하나 말고 여러개 두자
  - 각각의 Queue 에 절대적 우선순위 존재
  - 또는 CPU time 을 각 Queue 에 차등배분
  - 각 Queue 는 독립된 scheduling 정책

# Multilevel *Feedback* Queue Scheduling

프로세스가 한 큐에만 있는게 아니고 상황에 따라 다른 큐로 옮길 수 있다

- 복수 개의 Queue
- 다른 Queue 로의 점진적 이동
  - 모든 프로세스는 하나의 입구로 진입
  - 너무 많은 CPU time 사용 시 다른 Queue 로
  - 기아 상태 우려 시 우선순위 높은 Queue 로

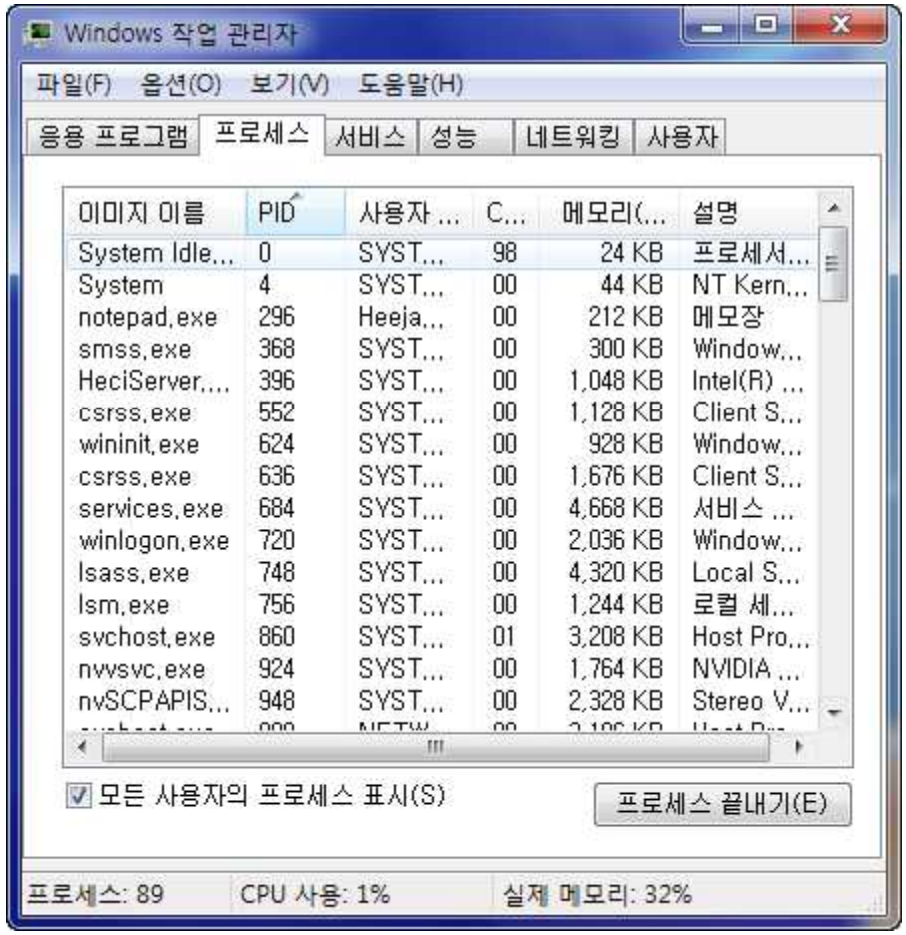
# 프로세스 생성과 종료

# Process Creation

- 프로세스는 프로세스에 의해 만들어진다!
  - 부모 프로세스 (Parent process)
  - 자식 프로세스 (Child process)
    - *cf.* Sibling processes
  - 프로세스 트리 (process tree)
- Process Identifier (PID)
  - Typically an integer number
  - *cf.* PPID
- 프로세스 생성
  - *fork()* system call – 부모 프로세스 복사
  - exec() – 실행파일을 메모리로 가져오기

컴퓨터를 키면 os가 메모리에 올라가고, 첫 번째 프로세스(unix의 경우 init)가 만들어진 후 init으로부터 프로세스들이 만들어짐

# 예제: Windows 7 프로세스 나열



# 예제: Ubuntu Linux 프로세스 나열

```
hjyang@rm303:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  2197  2189  0  80   0 -  1799 wait   pts/0    00:00:00 bash
0 R  1000  2368  2197  0  80   0 -  1177 -      pts/0    00:00:00 ps
hjyang@rm303:~$ ps -axl
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4     0     1     0  20   0   3536  1948 poll_s Ss   ?          0:00 /sbin/init
1     0     2     0  20   0      0     0 kthrea S    ?          0:00 [kthreadd]
1     0     3     2  20   0      0     0 run_ks S    ?          0:00 [ksoftirqd/0]
1     0     4     2  20   0      0     0 worker S    ?          0:00 [kworker/0:0]
5     0     5     2  20   0      0     0 worker S    ?          0:00 [kworker/u:0]
......
1  1000  1820     1  20   0  55944  3992 poll_s Sl   ?          0:00 /usr/bin/gnome-...
4  1000  1831  1658  20   0  50924  9096 poll_s Ssl  ?          0:00 gnome-session --sessio...
0  1000  2196  2189  20   0   2404   724 unix_s S    ?          0:00 gnome-pty-helper
0  1000  2197  2189  20   0   7196  3572 wait   Ss   pts/0      0:00 bash
0  1000  2370  2197  20   0   4708   708 -      R+   pts/0      0:00 ps -axl
hjyang@rm303:~$
```

# Process Termination

- 프로세스 종료
  - *exit()* system call
  - 해당 프로세스가 가졌던 모든 자원은 O/S 에게 반환

    (메모리, 파일, 입출력장치 등)