

1. 프로세스, 데몬

1) 프로세스

프로세스(process)

컴퓨터 내에서 실행 중인 프로그램

데몬(daemon)

주기적인 서비스 요청을 처리하기 위해 계속 실행되는 특별한 프로그램
(telnetd, ftpd...)

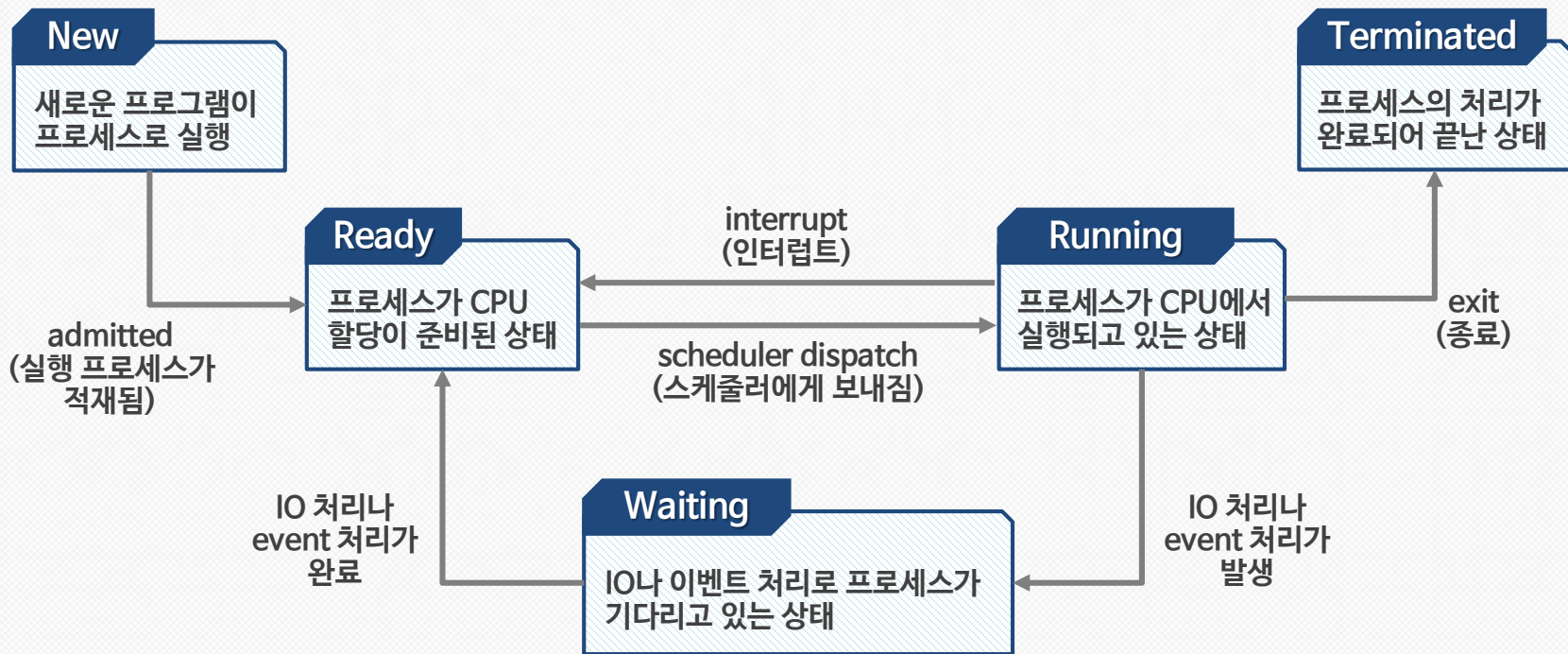


프로세스 관련

: 프로세스 명, 프로세스 번호(process id , PID), 부모프로세스, 자식프로세스

1) 프로세스

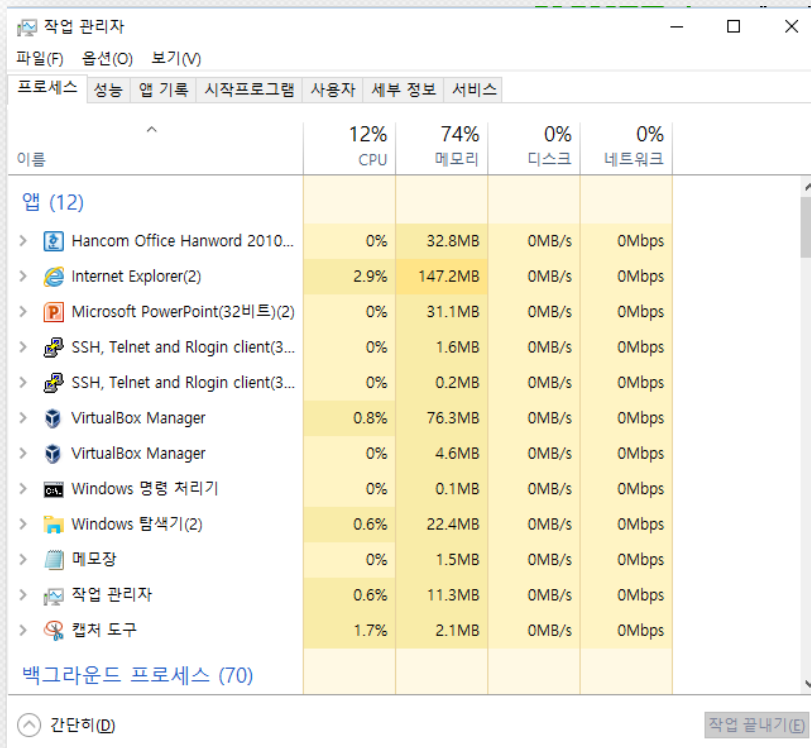
➤ 프로세스 스케줄링



1) 프로세스

✓ 프로세스 실행 중에 자신에게
종속적인 프로세스를 두어 보조업무를
처리하는데, 이 두 프로세스를
부모 프로세스, 자식 프로세스라고 함

✓ 윈도우에서도 작업관리자를 통하여
프로세스의 실행 상황을 볼 수 있는데,
이러한 개념은 유닉스, 리눅스에서의
개념으로부터 시작된 사항임



The screenshot shows the Windows Task Manager window titled '작업 관리자' (Task Manager). The '프로세스' (Processes) tab is selected. The table lists running applications with their CPU, Memory, Disk, and Network usage. The '앱 (12)' (Apps) section is expanded, showing 12 processes. The '백그라운드 프로세스 (70)' (Background processes) section is also visible at the bottom.

이름	12% CPU	74% 메모리	0% 디스크	0% 네트워크
앱 (12)				
> Hancom Office Hanword 2010...	0%	32.8MB	0MB/s	0Mbps
> Internet Explorer(2)	2.9%	147.2MB	0MB/s	0Mbps
> Microsoft PowerPoint(32비트)(2)	0%	31.1MB	0MB/s	0Mbps
> SSH, Telnet and Rlogin client(3...	0%	1.6MB	0MB/s	0Mbps
> SSH, Telnet and Rlogin client(3...	0%	0.2MB	0MB/s	0Mbps
> VirtualBox Manager	0.8%	76.3MB	0MB/s	0Mbps
> VirtualBox Manager	0%	4.6MB	0MB/s	0Mbps
> Windows 명령 처리기	0%	0.1MB	0MB/s	0Mbps
> Windows 탐색기(2)	0.6%	22.4MB	0MB/s	0Mbps
> 메모장	0%	1.5MB	0MB/s	0Mbps
> 작업 관리자	0.6%	11.3MB	0MB/s	0Mbps
> 캡처 도구	1.7%	2.1MB	0MB/s	0Mbps
백그라운드 프로세스 (70)				

〈출처 : 교수자 제작물〉

2) 프로세스 조회

/proc/ 내의 파일 (Linux)

프로세스들은 파일로 상태가 나타남

/proc/cpuinfo (Linux)

cpu 관련 상황을 알 수 있는 파일

[ps]

(proess status) 프로세스를 조회하는 유용한 명령

a	전체 사용자의 모든 프로세스 출력
e	명령문 실행 후, 프로세스 환경 변수 출력
i	자세히 출력
u	사용자 이름과 프로세스 시작 시간을 출력
w	출력 결과가 한 줄이 넘어도 모두 출력
x	제어하는 터미널이 없는 프로세스 출력

2) 프로세스 조회

〈프로세스 상태 STAT〉

구분	설명
R	실행 대기(Ready)
S	잠든 상태(Sleep)
D	입 · 출력을 기다리는 인터럽트가 불가능한 상태
T	멈춰있거나 흔적이 남아있는 상태
Z	zombie 상태, 완전히 죽어있는 상태

2) 프로세스 조회

〈ps 명령 출력 필드〉

구분	설명	구분	설명
USER	프로세스 소유자의 계정 이름	VSZ	가상 메모리
PPID	부모 프로세스의 PID	RSS	사용된 실제 메모리
%CPU	프로세스의 CPU 사용량	TIME	총 CPU 사용시간
%MEM	프로세스가 사용하는 시스템 메모리 자원량	CMD	실행된 프로세스 명령어

```
root@sjcu:/proc# ps -al
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
4 S 0 2289 2246 0 80 0 - 1577 wait pts/0 00:00:00 su
4 S 0 2290 2289 0 80 0 - 1654 wait pts/0 00:00:00 bash
0 R 0 2306 2290 0 80 0 - 1246 - pts/0 00:00:00 ps
root@sjcu:/proc#
```

〈출처 : 교수자 저작물〉

3) 프로세스 관련 시스템 현황 조회



실습을 통해 알아봄

`ps tree`

트리 구조로 시스템 프로세스를 보여줌 (Linux)

`top`

프로세스들의 실시간 구동 상황을 보여줌

`ulimit -a`

프로세스 자원한도를 보여줌

`vmstat`

cpu 활동상황을 보여줌

3) 프로세스 관련 시스템 현황 조회

```
root@sjcu:/proc# pstree
init─acpid
  ├apache2──5*[apache2]
  ├atd
  ├cron
  ├dbus-daemon
  ├dhclient
  ├6*[getty]
  ├java──14*[{java}]
  ├mysqld──18*[{mysqld}]
  ├rsyslogd──3*[{rsyslogd}]
  ├sshd──sshd──sshd──bash──su──bash──pstree
  ├systemd-logind
  ├systemd-udevd
  ├upstart-file-br
  ├upstart-socket-
  ├upstart-udev-br
  ├whoopsie──2*[{whoopsie}]
  └xinetd
root@sjcu:/proc#
```

〈출처 : 교수자 제작물〉



2. foreground, background

1) foreground, background

✓ 기본적으로 셸에서 프로그램을 실행하면 종료시까지 셸을 사용할 수 없도록 하고 실행

✓ 실제로는 하나의 셸 수행명령 상태에서 여러 개의 프로그램(프로세스)를 실행할 수 있음

1) foreground, background



foreground 작업

셸 전면 상황에서 실행한 프로세스,
프로그램 명을 입력 후 실행



background 작업

- 셸 후면 상황에서 작업을 실행,
백그라운드로 프로세스를 실행하면
프로세스는 셸 뒷단에서 실행되므로
셸을 사용 가능
- 프로그램 뒤에 &를 붙여서 실행함



참고

‘%숫자’는 백그라운드 작업을 지칭 : %1(%), %2, %3, ...

1) foreground, background

〈foreground 실행 예〉

```
root@sjcu:~# ./a.sh 1
[ 1 ]: Mon Jan 18 18:03:28 KST 2016
[ 1 ]: Mon Jan 18 18:03:31 KST 2016
[ 1 ]: Mon Jan 18 18:03:34 KST 2016
[ 1 ]: Mon Jan 18 18:03:37 KST 2016
```

〈background 실행 예〉

```
root@sjcu:~# ./a.sh 1&
[1] 1633
root@sjcu:~# ./a.sh 2&
[2] 1638
root@sjcu:~# ./a.sh 3&
[3] 1645
root@sjcu:~# [ 3 ]: Mon Jan 18 18:06:50 KST 2016
[ 1 ]: Mon Jan 18 18:06:51 KST 2016
[ 2 ]: Mon Jan 18 18:06:52 KST 2016
[ 3 ]: Mon Jan 18 18:06:53 KST 2016
[ 1 ]: Mon Jan 18 18:06:54 KST 2016
[ 2 ]: Mon Jan 18 18:06:55 KST 2016
[ 3 ]: Mon Jan 18 18:06:56 KST 2016
[ 1 ]: Mon Jan 18 18:06:57 KST 2016
```

〈출처 : 교수자 저작물〉

2) 실행 제어

jobs, ps

실행되는 작업을 알 수 있음

fg % 작업번호,
bg % 작업번호

포그라운드와 백그라운드 전환

kill %, kill PID

프로세스를 강제로 끝냄

2) 실행 제어

〈jobs 실행 예〉

```
root@sjcu:~# j[ 2 ]:Jobs
[1] Running ./a.sh 1 &
[2]- Running ./a.sh 2 &
[3]+ Running ./a.sh 3 &
root@sjcu:~#
```

〈출처 : 교수자 저작물〉

〈kill %1 %2 %3 실행 예〉

```
root@sjcu:~#kill %1 %2 %3
[1] Terminated ./a.sh 1
[2]- Terminated ./a.sh 2
[3]+ Terminated ./a.sh 3
root@sjcu:~#
```

〈출처 : 교수자 저작물〉

3. 프로세스 스케줄러

1) crontab

“ 서버 내의 특정 프로세스를 주기적으로 사용하려고 할 때 사용하는 스케줄 설정 파일과 실행을 처리하는 데몬 ”

〈crontab의 편집 및 관리〉

구분	설명
crontab -l	예약된 작업 리스트 출력
crontab -e	예약된 작업 수정
crontab -r	예약된 작업 삭제

1) crontab

crontab -e

크론템의 편집

- 분, 시, 일, 월, 요일, 실행명령 (전체경로)

분	시	일	월	요일
0~59	0~23	1~31	1~12	1~7(월~일)

1) crontab

➤ 필드 다중 설정방법

구분	표현식	내용
여러 값	10,30,50****	매 10, 30, 50분마다 실행
범위 값	01-3***	매일 1~3시에 진행
시간간격	1*/5***	매일 5시간 간격으로 진행
모든 값	*****	매 분마다 실행

예

- 매주 **일요일마다** my.sh를 실행하고 싶을 경우
➡ ****7 /home/kim/my.sh
- 매일, **5시간 간격**으로 my.sh를 실행하고 싶을 경우
➡ 1*/5*** /home/kim/my.sh

2) 시작프로그램 역할

➤ Inittab의 변경이력

- ✓ 일반적인 unix에서는 /etc/inittab 설정파일에 unix부팅 이후 자동으로 실행될 프로그램을 등록하며 지금도 사용함
- ✓ 추후 linux에서 initab의 기능을 /etc/event.d의 sh파일이 담당했음
- ✓ 리눅스(우분투) 표준 방식에 의거 /etc/rc.local파일 맨 마지막 행에 시작 시 구동될 프로세스를 등록함

2) 시작프로그램 역할

➤ Inittab의 변경이력



- 유닉스 시스템에서는 아직 inittab이 사용되고 있음
- 리눅스 버전에서도 event.d가 아직 사용되고 있는 경우도 있음을 참고할 것
- 하지만 우분투에서는 /etc/rc.local 파일임

2) 시작프로그램 역할

➤ rc.local

✓ /etc/rc.local 파일 맨 마지막 행에 시작 시 구동될 프로세스를 등록

✓ 마지막 행에 [exit 0] 즉, 강제 종료 명령이 있기 때문에, 그 윗줄에 기록

✓ 실습 예시

- 부팅 완료 시, 시간을 매번 /home/boot.log에 기록함
- 마지막 행 기록 문구 : [date >> /home/boot.log]

2) 시작프로그램 역할

➤ rc.local

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
date > /home/boot.log
exit 0

~~~~~
:wq
```

〈출처 : 교수자 저작물〉

4. 프로세스, 데몬 실습

1) 실습하기

실습내용

(1) 프로세스 관리

(2) foreground, background

5. 프로세스 스케줄링 실습

1) 실습하기

실습내용

- (1) 프로세스 관리
- (2) 시작프로그램의 역할

* 일시정지 버튼을 클릭하고 학습활동에 참여해 보세요.

Q 오늘은 리눅스에서 프로세스, foreground, background, 스케줄러 등을 배웠습니다. 윈도우OS에서 동일한 기능으로는 무엇이 있는지 정리해 보세요.

※ 학습활동에 대한 해설

Q

오늘은 리눅스에서 프로세스, foreground, background, 스케줄러 등을 배웠습니다. 윈도우OS에서 동일한 기능으로는 무엇이 있는지 정리해 보세요.

A

- 리눅스 시스템은 우리가 많이 사용하는 PC 윈도우의 근본 사상이기도 합니다. 윈도우에서 프로세스도 리눅스와 마찬가지로의 개념입니다.
- 프로세스의 처리 상황을 윈도우의 작업관리자에서 확인해 볼 수 있습니다.
- 리눅스의 foreground, background 개념은 여러 개의 job을 한번의 실행시킬 수 있는 개념입니다. 윈도우에서는 여러 개의 윈도우 창을 실행시킬 수 있습니다.
- 윈도우 창이 활성화 된 것이 foreground의 개념이고, 비활성화 된 상태에서 일을 수행하고 있는 윈도우 창이 background의 개념입니다.
- 또한 윈도우의 “작업 스케줄러”가 스케줄러와 동일한 역할을 하고 있습니다.