Unix(Linux) 환경에서의 컴퓨터 프로그래밍

목 적 : 본 단원에서는 Unix 또는 Linux 환경에서 C 또는 C++ 언어를 사용하여 프로그래밍하는 방법을 다룬다. 먼저 Unix 시스템을 간단히 소개하고, 기본적인 Unix 명령, 편집기 사용, shell programming 등을 다룬다. 다음, make 작성법, 컴파일러와 디버거 사용법 등을 통하여 C 또는 C++ 언어로 프로그래밍 하는 법을 소개한다.

목차

Unix System 소개

1. 개요

컴퓨터공학과 학생이라면 뉴스나 잡지 등 여러 매체를 통해 Linux란 용어를 흔히 접해 보았을 것이다. “리눅스, MS를 제치고 IT시장 석권”과 같은 뉴스 기사를 한번쯤은 접해보았을는지도 모른다. 하지만, Unix란 단어는 Linux에 비하여 쉽게 접하지 못하였을 것이다. 그렇다면, 과연 이 장에서 공부하고자 하는 Unix란 무엇이며, Unix와 Linux 사이의 관계는 어떻게 되는 것일까?

Unix란 컴퓨터 운영체제의 하나로, 주로 중대형 컴퓨터 및 워크스테이션에서 많이 사용되는 OS(Operating System, 운영체제)이다. 하지만 요즘에는 PC 환경에서도 Unix의 기능을 구현한 OS를 사용할 수 있게 되었는데, 이것이 바로 Linux이다.

Unix는 1969년에 개발된 이래 여러 다른 갈래로 발전을 거듭하여, 현재에는 다양한 종류의 Unix가 존재한다. 대표적인 것으로는 Sun의 Solaris와 SunOS, IBM의 IRIX, HP의 HP-UX, 버클리대학의 FreeBSD 등이 있다. 이들 Unix는 고유의 개발 환경에 맞게 조금씩 변형되어 발전되어왔다. 이들의 발전 흐름은 다음 장에서 살펴보기로 한다.

2. Unix 역사

유닉스 운영체제가 어떻게 시작하여 지금의 유닉스가 되었을까? 처음 시작은 1960년대 AT&T의 Bell 연구소에서 개발한 Multics 운영체제가 너무 복잡하여, 유닉스 운영체제를 개발한 계기가 되었다. Multics는 GE 컴퓨터를 본체로 사용한 대규모의 컴퓨팅 서비스를 제공하는 시스템이다.

1969년 Bell 연구소는 Multics 개발을 중단했다. 이후 Multics 개발에 참여했던 사람들 중 많은 사람들이 유닉스 개발에 참여하였으며, 유닉스라는 이름이 Multics로부터 파생되었을 정도로 Multics는 유닉스에 많은 영향을 미쳤다.

Bell연구소가 Multics 개발을 중단한 후 거의 사용되지 않던 PDP-7 미니컴퓨터를 사용하여 프로그래밍 환경 개선 연구를 함과 동시에 운영체제를 만들려고 노력하였다. 그리고 유닉스의 파일 시스템으로 진화되는 파일 시스템을 만들었다. Bell연구소의 특허 부서에서는 문서 작성 도구를 제공하는 PDP-11을 이용한 유닉스 시스템의 최초의 버전이 1971년에 나오게 되었다. 그후 프로그래밍의 표준인 C 언어가 유닉스에 쓰이기 위해 개발되고, 유닉스 운영 자체도 C로 다시 프로그램되었으며, 더불어 유닉스의 문서 작성 기능은 유닉스 시스템을 확산시키는 계기가 되었다.

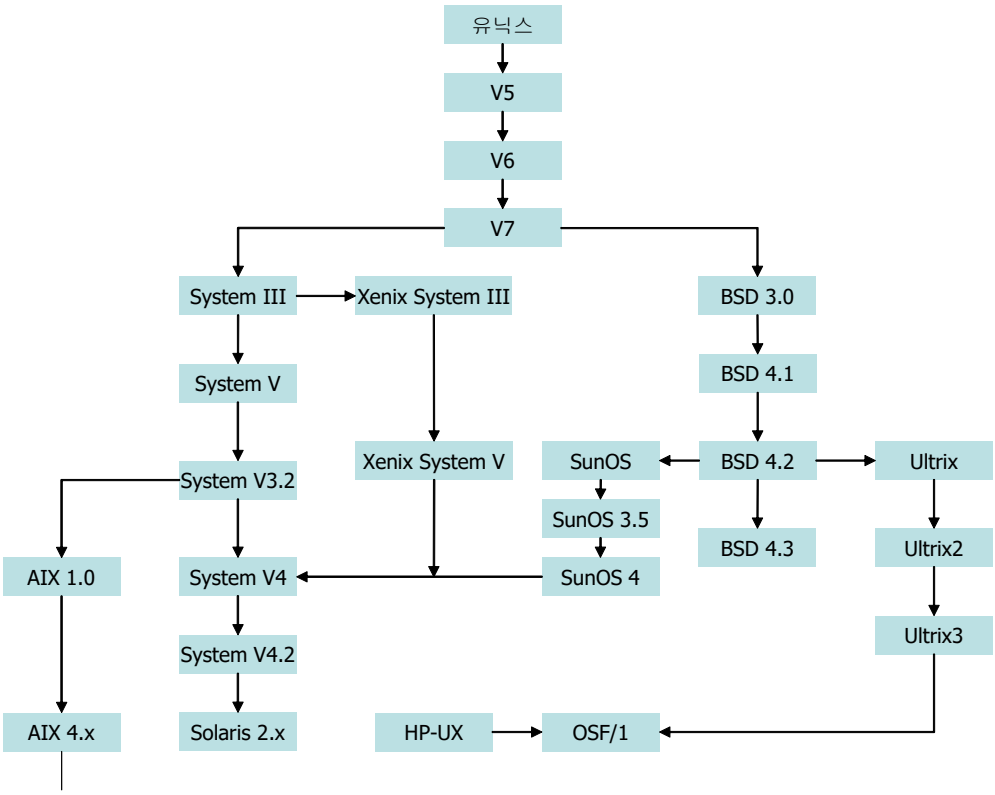
1976년 유닉스 시스템이 다른 컴퓨터 시스템인 Interdata 8/32로의 이식이 이루어졌다. 그후로 Zilog Z-80과 Z8000, Motorola MC 68000, 68010, 68020, 그리고 Intel x86과 같은 단일 칩 마이크로프로세서로부터 메인 프레임 컴퓨터에 이르기까지 실질적으로 모든 종류의 컴퓨터에 이식되었다.

AT&T Bell연구소 내에서 유닉스 시스템이 사용되기 시작하고, 운영체제에 대한 관심이 고조되면서, 유닉스 시스템에 대하여 대학들에서 관심을 갖기 시작하였다. 그러나 AT&T에서는 유닉스 시스템을 파는 것을 허용하지 않았으나, 점점 유닉스의 인기가 높아지자, 교육용으로 대학교에서 쓰이는 것을 허락하였다. 유닉스 시스템의 원시 프로그램이 공개됨에 따라, 대학에서는 이것을 이용하여 여러 응용 소프트웨어를 개발하고 연구하여 그 결과를 공개함으로써 여러 사람들이 공유하고 발전시켜나갔다. 이는 유닉스 소프트웨어의 폭발

적인 성장을 가져오게 되었다.

그후 인터랙티브 시스템(Interactive System Coperation)사가 1977년 유닉스를 사무자동화를 위한 상업용으로 제일 먼저 팔기 시작하게 되었다. 이로 인하여 유닉스가 상업적 가치를 얻게 되면서부터 지금까지 관련된 모든 분야에 큰 산업이 되었다.

오늘날 유닉스는 크게 두 가지의 버전으로 나뉘어진다. 하나는 AT&T에서 발전시킨 System V 계열이고, 다른 하나는 버클리대학에서 공급하는 BSD 계열이다. 자세한 변천 과정은 다음과 같다.



[그림 1] UNIX 변천 과정

유닉스 변천 과정에 대하여 좀더 자세히 알고 싶다면, http://www.levenez.com/unix/를 참고하기 바란다.

3. Unix System 기초

Unix System에 대한 기초적인 내용들을 기술한다.

3-1. 접속 방법

Unix에 접속하는 방법에는 원격 접속 방법과 터미널 접속 방법이 있다. 원격 접속 방법은 telnet이나 ssh 프로토콜을 이용해 원격에서 네트워크를 통해 접속하는 방법이다. 이에 반하여 터미널 접속 방법은 서버에 직접 연결되어 있는 모니터와 키보드, 마우스를 통해 연결하는 방법을 말한다. 과거에는 하나의 서버에 여러 터미널이 존재하여 이러한 터미널에서 Unix system에 접속하는 방법을 터미널 접속 방법이라고 하였으나, 여기에서는 Unix System에 직접 접근하여 접속하는 방법을 의미하므로 주의하기 바란다.

개인용 PC에 설치되는 Windows와는 달리, Unix는 보통 Server급 컴퓨터에 설치되므로, 일반 사용자들은 Unix System에 접속하기 위하여 보통 원격 접속 프로그램을 사용한

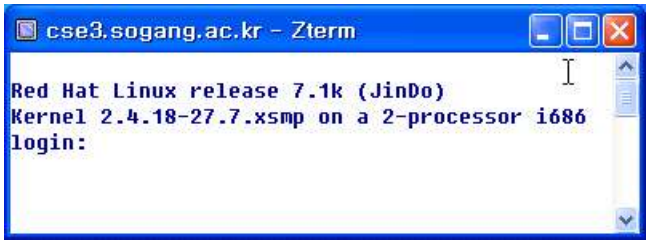
다. 과거에는 telnet 프로토콜을 이용하여 원격 접속을 하였으나, 보안상의 문제로 인하여 요즘에는 ssh를 많이 이용한다.

telnet이나 ssh 접속 프로그램은 zterm, putty, SSH Secure Shell Client, SecureCRT, Xmanager 등 여러 가지가 있다. 여기에서는 콘솔(console)용 접속 프로그램으로 zterm과 X 환경의 접속 프로그램으로 Xmanager를 사용하여 접속하는 방법에 대하여 알아보도록 하겠다. zterm은 http://www.brainz.co.kr/products/products4_2.php에서 구할 수 있다.

3-1-1. zterm을 이용한 접속

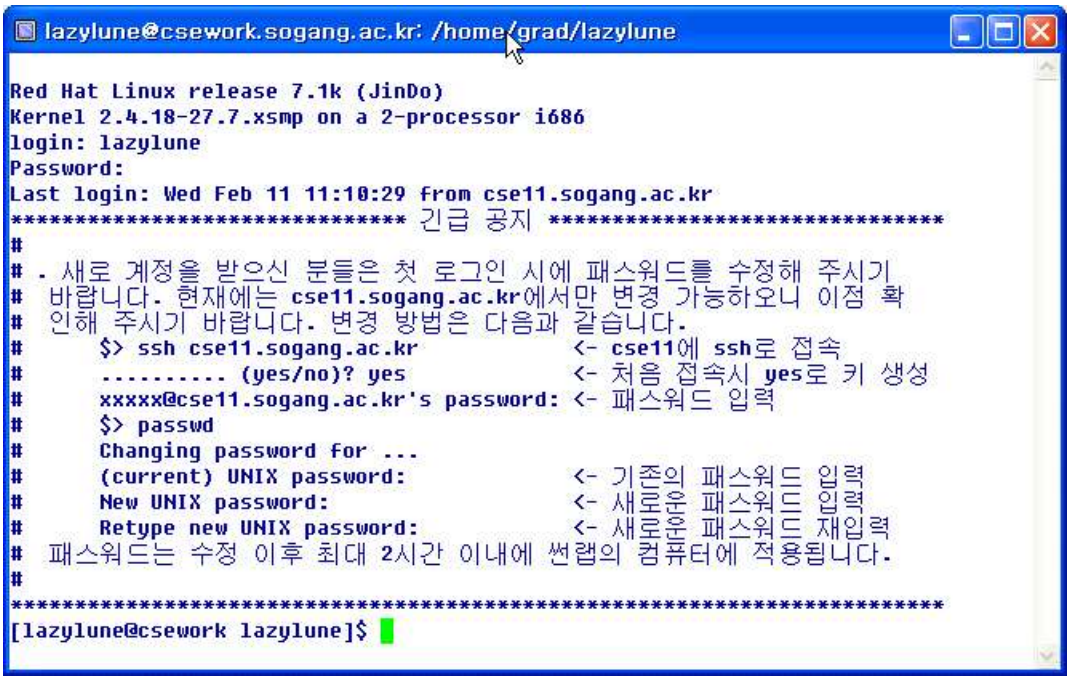
먼저 zterm을 실행시키면 다음과 같은 화면이 나타난다. Host는 접속하고자 하는 Unix System의 도메인 명을 나타낸다. 여기에서는 컴퓨터공학과 실습 서버인 csework.sogang.ac.kr에 접속해보도록 한다. Port는 접속 시 사용할 프로토콜을 가리킨다. 각각의 프로토콜에 따라서 사용하는 Port 번호가 틀리다. telnet은 23번 포트, ssh는 22번 포트를 사용한다. ssh 프로토콜을 이용하여 서버에 접속하고자 한다면, Port에서 23을 22로 고쳐주면 된다. 현재 csework.sogang.ac.kr은 교내에서는 telnet과 ssh를 이용한 접근이 모두 허용되고, 학교 밖에서는 ssh를 통한 접근만이 허용된다. 참고로 컴퓨터공학과 웹 서버인 cs.sogang.ac.kr은 telnet을 통한 접근만이 허용되며, cspro.sogang.ac.kr은 ssh를 통한 접근만이 허용된다. Login은 사용할 계정 이름을 가리키며, Pass는 해당 계정의 암호를 나타낸다. Login명과 Pass는 입력 시 자동으로 저장이 되므로, 학교 실습실과 같은 공공장소에서는 Login명과 Pass를 입력하지 않고 그냥 Connect를 누른다. Connect를 누르면 다음과 같은 화면이 나타난다.

[그림 2] zterm 실행 화면



[그림 3] zterm 로그인 화면

Sunlab에서 발급 받은 계정(accout)을 login 항목에 입력하고 비밀번호를 password 항목에 입력하면 Linux에서 기본적으로 설정되어 있는 Shell 환경을 접할 수 있다.



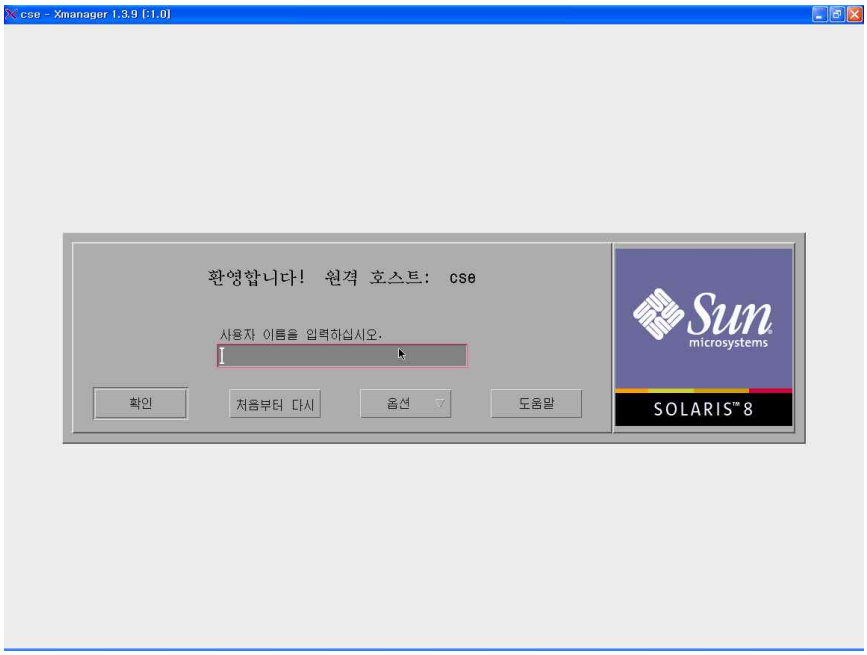
[그림 4] zterm의 작업 화면

3-1-2. Xmanager를 이용한 접속

Xmanager는 원격에서 Unix System에 GUI(Graphic User Interface)모드로 접속할 수 있게 해주는 프로그램이다. Xmanager는 상용 프로그램으로 개인이 사용하려면 직접 구매를 해야 한다. 하지만 현재 학과 내에서 Xmanager에 대한 라이센스를 보유하고 있기 때문에, PCLab이나 Sunlab 에서 Xmanager를 실습해볼 수 있다.

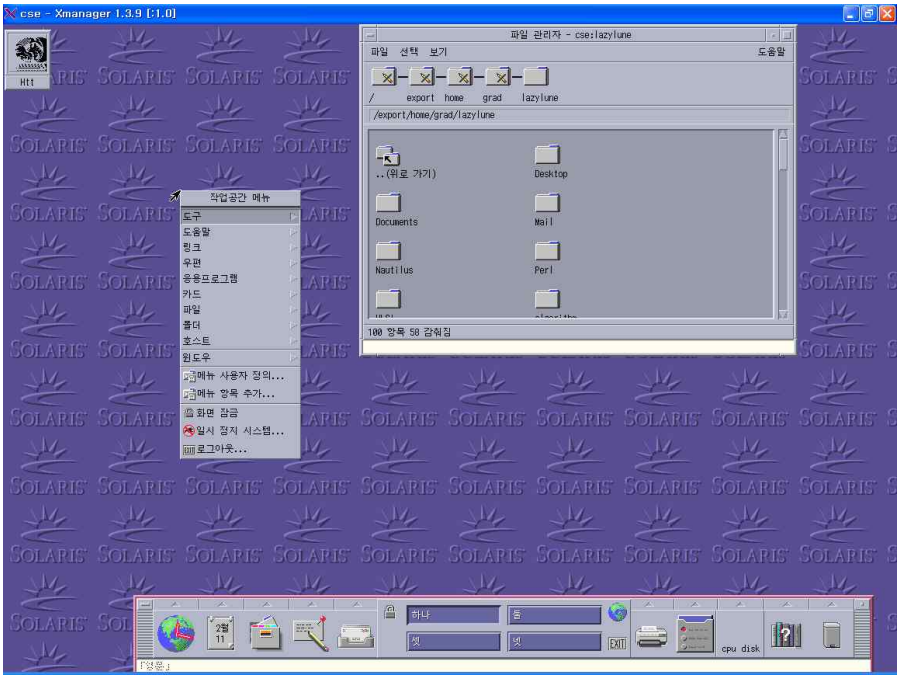
Xmanager 내의 프로그램인 Xbrowser를 실행하면 다음과 같은 화면이 나타난다. 기본적으로 화면의 밑 부분에 나오는 호스트들은 현재 Xbrowser를 실행시키고 있는 호스트와 같은 랜 상에 있는 접속 가능한 호스트들을 나타낸다. 랜 외부에 있는 호스트에 접속하고자 하면 주소 란에 호스트명을 써주면 된다. 여기에서는 학과 서버인 cs.sogang.ac.kr에 접속해보도록 한다. 주소란에 cs.sogang.ac.kr을 써주고 옆에 있는 Go 버튼을 클릭하면 다음과 같은 로그인 화면이 나타난다.

[그림 5] Xbrouser 실행 결과



[그림 6] Xmanager 로그인 화면

여기에 사용자 이름과 암호를 차례로 입력하면, Solaris의 GUI 환경인 CDE(Common Desktop Environment)가 나타난다.



[그림 7] Xmanager 작업 환경

이제 본격적으로 Unix System에 대한 공부를 시작해보도록 하겠다. 여기에서는 GUI보다는 콘솔 상에서의 작업 위주로 실습해보기로 한다.

3-1-3. 접속 종료

접속을 종료하기 위해서는 콘솔에서는 *logout* 명령을 사용한다. 쉘 상에서 단지 logout 만 쳐주면 접속이 종료된다. Xmanager를 통하여 접속했을 때에는, 마우스 오른쪽 버튼을 클릭하여 작업 공간 메뉴를 띄운 뒤에, 로그아웃이란 메뉴를 선택하면 접속 종료된다.

3-2. Shell

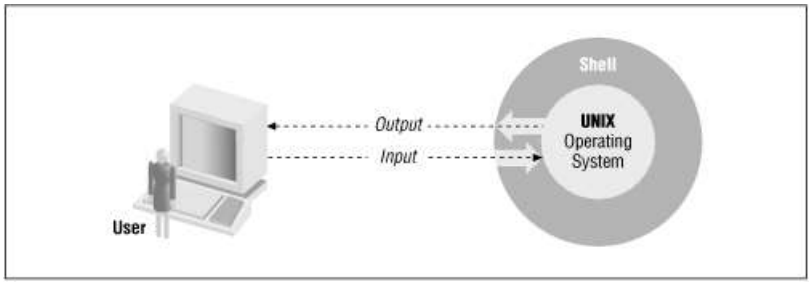
3-1-1장에서 콘솔용 접속 프로그램으로 Unix System에 접속하면 Shell 환경을 접할 수 있다고 배웠다. 그렇다면 과연 Shell이란 무엇일까? 이 장에서는 Unix System의 핵심 중에 하나인 Shell에 대하여 공부해보기로 하자.

3-2-1. 개요

O'Reilly의 "Learning the Bash Shell"을 보면 Shell에 대한 다음과 같은 설명이 있다.

> *"The shell's job, then, is to translate the user's command lines into operating system instruction."*

한마디로 Shell은 Unix OS와 사용자 간에 대화 소통을 원활하게 해주는 역할을 한다.



[그림 8] shell의 역할

다음과 같은 명령이 있다고 하자.

$> sort -n phonelist > phonelist.sorted

이 명령은 phonelist 파일 내의 라인들을 숫자 순서대로 정렬하여 그 결과를 phonelist.sorted에 저장하는 명령이다. 하지만 이러한 명령은 사용자가 이해하기 쉬운 명령이지, Unix OS가 직접적으로 이해할 수 있는 명령이 아니다. Shell은 이러한 명령을 다음과 같은 순서로 처리한다.

1. 명령행을 sort, -n, phonelist, >, phonelist.sorted와 같은 조각으로 나눈다. 이러한 조각들을 워드(word)라고 한다.
2. 각 워드가 의미하는 바를 파악한다(sort는 명령, -n, phonelist는 인자, >와 phonelist.sorted는 I/O 명령).
3. > phonelist.sorted에 따라 I/O를 설정한다. 즉 sort에 대한 결과가 phonelist.sorted 파일에 저장되도록 설정한다.
4. sort 명령을 -n 옵션으로 하고 phonelist를 인자로 하여 실행한다.

즉, OS는 단순히 일련의 명령을 수행하는 명령어들만을 제공한다. 이를 바탕으로 한 pipe나 I/O Redirection, Command History 등과 같은 사용자 편의를 위한 기능들은 쉘의 몫이다. 그러면, 이러한 기능들에 대하여 살펴보도록 하자.

3-2-2. 종류

Shell은 Unix System에 독립적인 프로그램이다. 어떠한 Unix System에서든지 자기의 취향에 맞는 Shell을 선택하여 사용할 수 있다. 같은 Shell을 사용한다면 어느 Unix System에서든지 똑같은 사용자 인터페이스를 이용할 수 있다.

Shell은 처음으로 사용된 Bourne Shell(*sh; shell 실행 파일 명*)에서부터 다양하게 개발되어 현재에는 다수의 Shell들이 존재한다. Bourne Shell을 대체할 수 있도록 개발되어 널리 사용된 C shell *(csh)*, C shell의 향상된 기능을 Bourne Shell에 접목하여 개발된 Bourne Again Shell *(bash)* 등과 함께 tcsh *(tcsh)*, korn shell *(ksh)* 등이 널리 쓰이고 있다.

여기에서는 현재 많은 Linux에서 기본 Shell로 채택하고 있는 Bash에 대하여 다루기로 한다. 하지만 각 Shell이 많은 부분 중복된 기능을 가지고 있으므로, 하나의 Shell에 대하여 공부한다면 다른 Shell도 쉽게 이해할 수 있을 것이다.

3-2-3. 명령어 실행

Shell은 입력 받은 명령어를 해석하고 실행하여 그 결과를 돌려주는 역할을 수행한다고 앞서 언급하였다. 일반적으로 하나의 명령어를 입력 받아서 실행하지만, 한 줄에 여러 개의 명령어를 입력받아서 순차적으로 실행할 수도 있다. 이때 사용하는 것이 Semicolon(;)이다.

☞ **예제**

```
$ who ; ps
lazylune pts/0       Feb 11 20:20
  PID TTY          TIME CMD
17281 pts/0    00:00:00 bash
17332 pts/0    00:00:00 ps
$
```

Unix는 기본적으로 Multi Processing을 지원한다. 즉 여러 프로그램을 동시에 실행시킬 수 있는 환경을 제공해준다. 일반적으로 사용자가 여러 프로그램을 동시에 실행시키기 위해서는 프로세스를 background로 수행시켜야 한다. 프로세스(process)란 실행되고 있는 프로그램을 말하며, background로 수행시키면 프로세스의 실행과 동시에 다른 작업을 할 수 있다. 이와 관련된 명령어들은 뒤에 부록에서 다루기로 하겠다.

현재 수행중인 process를 확인하기 위해서는 ps 명령어를 사용한다.

☞ **예제**

```
$ ps
  PID TTY          TIME CMD
17281 pts/0    00:00:00 bash
17388 pts/0    00:00:00 ps
$
```

모든 유닉스 명령들은 맨페이지를 통해 사용법을 확인할 수 있는데. 맨페이지에 대한 자세한 내용은 뒤의 부록을 참조하기 바란다.

3-2-4. Standard Input/Output/Error, Redirection, Pipe

Unix에서 명령어는 Standard Input(일반적으로 키보드를 가리킴)을 통해서 입력받고, 처

리된 결과는 Standard Output(일반적으로 모니터를 가리킴)을 통해서 출력이 된다. 또한 발생된 error는 Standard Error(기본적으로 모니터를 가리킴)를 통해 출력이 된다. 하지만, 경우에 따라서 키보드가 아니라 파일을 통하여 입력이 되거나 파일로 출력이 되어야 하는 경우가 있다. 가령 프로그램의 출력된 결과를 관찰할 때 그 양이 매우 많고 화면상에서 순식간에 지나쳐버린다면 출력된 결과를 파일로 저장한 이후에 살펴보는 것이 매우 편할 것이다. 이런 경우 Redirection(<, >)을 이용하여 입력과 출력을 바꾸게 된다.

아래의 화면은 ps 명령을 통한 결과를 test라는 파일로 저장시키는 명령을 나타낸다.

☞ **예제**

```
$ ps > test
$
```

>>는 기존의 파일에 결과를 덧붙여 저장할 때 사용된다.

☞ **예제**

```
$ ps >> test
$
```

cat 명령어를 통해 text file의 내용을 살펴보면 ps 명령에 대한 결과가 파일에 저장되었음을 살펴볼 수 있다.

☞ **예제**

```
$ cat test
  PID TTY          TIME CMD
17281 pts/0    00:00:00 bash
17422 pts/0    00:00:00 ps
  PID TTY          TIME CMD
17281 pts/0    00:00:00 bash
17433 pts/0    00:00:00 ps
$
```

다음 쪽의 화면은 standard input을 file로 대체하는 명령을 나타내고 있다. wall은 Unix System에 접속한 모든 사용자들에게 메시지를 전달하는 프로그램으로, 메시지는 standard input으로 입력받게 되어 있다.

☞ 예제

```
$ wall
Hello
^d
Broadcast message from lazylune (pts/0) (Wed Feb 11 21:50:49 2004):

Hello
$ cat > test
hello
^d
$ wall < test
$
Broadcast message from lazylune (Wed Feb 11 21:51:10 2004):

hello

$
```

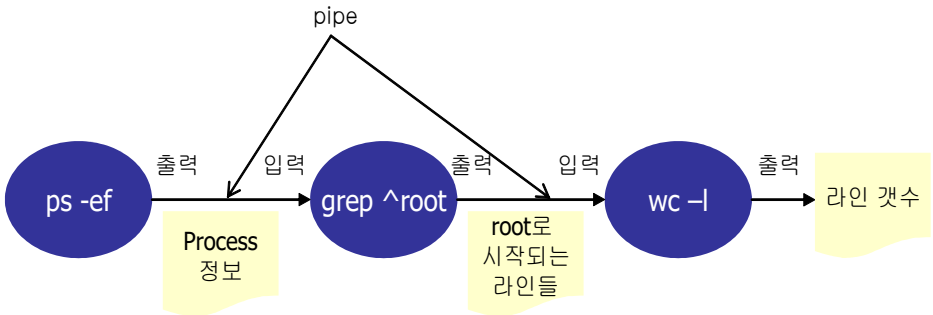
*^d*는 Ctrl+ d를 나타내며, Standard Input의 종료를 나타낸다. 즉 입력을 모두 다 마쳤으니 입력에 대한 처리를 수행하라고 명령하는 것과 같다. *cat > filename*은 Standard Input으로부터 입력을 받아들여 해당 파일명에 저장한다. 이 명령도 마찬가지로 입력이 모두 끝났으면 ^d키를 눌러 종료한다. *wall < test* 명령 결과 standard input과 동일한 결과를 출력함을 확인할 수 있다.

Pipe(|, shift + ₩)를 이용하면 한 명령의 표준 출력을 다른 명령의 표준 입력으로 보낼 수 있다. 다음의 예제는 현재 root 권한으로 실행되고 있는 프로세스의 개수를 구하는 명령이다.

☞ 예제

```
$ ps -ef | grep ^root | wc -l
    41
$
```

위의 과정을 도식화하면 다음과 같다.



[그림 9] pipe 처리 과정

pipe를 통하여 명령어의 출력은 입력으로 연결되어 우리가 구하고자 하는 root 프로세스의 개수를 구할 수 있다. 여기서 잠재적으로 존재하는 가정은 다음과 같다.

1. root 소유 프로세스의 정보는 처음에 "root"란 단어로 시작한다.

2. 하나의 프로세스에 대한 정보는 하나의 라인을 차지한다.

이를 통하여 우리는 여러 개의 명령을 pipe를 통해 간단히 조합함으로써 원하는 바를 얻었다. 이러한 pipe와 I/O redirection은 뒤에서 언급할 UNIX의 철학과도 관계가 있다.

3-2-5. Meta Character

Shell에서 특수한 목적을 위하여 미리 정의된 문자는 다음과 같다.

[표 1] Meta Character

특수문자	용도
cmd ;	command terminator
cmd &	run preceding command in the background
> *file*	output redirection
>> *file*	appending output redirection
< *file*	input redirection
<< *word*	input from "here" document
cmd | *cmd*	frame a pipeline between preceding command and the following command
*	the * is used in file name generating to match any sequence of characters
?	the ? is used in file name generation to match any single character in a file name
[*set*]	the [introduces a character set for file name generation and] closes the set 예: [0-9] -> 1 digit of numeric character
-	indicates a character range in a character class
$*word*	the *word* following the $ will be treated as a parameter and will be replaced by its value
c	the character following the backslash will be quoted
'*text*'	no substitutions will occur
"*text*"	parameter and command substitution occur
(*list*)	execute a command list in a subshell
{*list*}	execute a command list in the current shell
cmd && *cmd*	execute the second command only if the first completes with a zero exit status
cmd || *cmd*	execute the second command only if the first completes with a non-zero exit status

☞ 예제

```
$ ls
1  2  2.txt  3  4  4.txt  5  7.txt  a.dat  b.bat  c.txt
$ ls
1  2  2.txt  3  4  4.txt  5  7.txt  a.dat  b.bat  c.txt
$ ls [0-9]
1  2  3  4  5
$ ls [0-9]*
1  2  2.txt  3  4  4.txt  5  7.txt
$ ls [a-z]*
a.dat  b.bat  c.txt
$ ls [a-z].?at
a.dat  b.bat
$ FOOD="noddle"
$ echo "I like a $FOOD"
I like a noddle
$ echo 'I like a $FOOD'
I like a $FOOD
$
```

3-3. 파일 시스템

Unix에서 사용하는 파일 시스템에 대하여 설명한다.

3-3-1. path name

UNIX에는 디렉토리를 나타내는 특수한 문자가 존재한다. "."은 현재 디렉토리를 나타내며 ".."은 현재 디렉토리의 상위 디렉토리를 나타내고, "~"는 home 디렉토리를 나타낸다.

UNIX에서 경로를 나타내는 방법에는 크게 절대 경로(absolute pathname)와 상대 경로(relative pathname)가 있다. 절대 경로 방법은 root(/)로부터 시작하는 전체 경로 명을 사용하는 것이고, 상대 경로는 현재 user가 위치한 디렉토리를 기준으로 하여 경로 설정하는 방법이다. 가령 현재 위치가 /home/lazylune/project/sp/hw1/data이고 lazylune으로 로그인 한 경우 다음의 경로들은 모두 같은 경로를 가리킨다.

절대 경로: /home/lazylune/project/os/hw3/
상대 경로: ../../../os/hw3/
~/project/os/hw3
~lazylune/project/os/hw3

여기서 ~은 현재 로그인해 있는 계정의 홈 디렉토리를 나타내는 상대 경로이고, ~username은 해당 username의 홈 디렉토리를 나타내는 상대 경로이다.

3-3-2. Permission

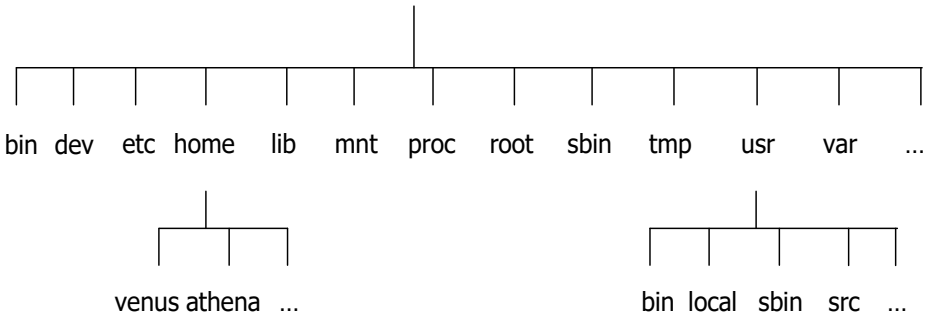
UNIX는 multi-user System이다. 따라서 다른 사용자가 자신의 Directory와 File에 접근하는 위험이 항시 존재한다. 따라서 다른 사용자가 자신의 파일이나 디렉토리에 접근하는 것을 막기 위하여 Permission이란 것이 존재한다. Permission이란 해당 파일에 대한 접근 권한을 뜻한다. 파일에 접근하는 사용자를 크게 소유자, 그룹, 다른 사용자의 세 가지

로 분류하여 각각에 대하여 읽기, 쓰기, 실행 권한을 줄 수 있다. 이러한 permission 정보는 ls -l 명령을 통해 살펴볼 수 있다.

Permission 정보는 10bit로 이루어져 있다. 가장 앞에 있는 bit는 파일의 종류 등을 나타낸다. 그 다음 3bit는 소유자에 대한 읽기, 쓰기, 실행 권한을 나타난다. 그 다음의 3bit는 그룹에 대한 권한을, 또 그 다음의 3bit는 다른 사용자에 대한 권한을 나타낸다. 가령 어느 파일의 권한 정보가 drwxr-xr-x라고 한다면 이 파일은 디렉토리이며, 파일의 소유자는 읽기, 쓰기, 실행의 모든 권한을 갖고 있고, 그룹과 다른 사용자는 읽기와 실행의 권한을 갖고 있다는 의미이다. 파일에 대한 permission은 chmod 명령을 통해 파일의 소유자나 root가 변경시킬 수 있다. chmod에 대한 자세한 설명은 뒤의 부록을 참조하기 바란다.

3-4. Unix System Directory 구조

Unix 시스템의 디렉토리 구조를 [그림 10]에 보이고 각 디렉토리에 관한 설명을 [표 2]에 보인다.



[그림 10] UNIX 디렉토리 구조

[표 2] UNIX 디렉토리

디렉토리	내용
/	최상위 디렉토리로 "/"로 표시하고 root(루트)로 불린다. 루트 디렉토리를 중심으로 유닉스 시스템의 중요한 파일들이 계층 구조를 이루고 있다.
/bin	유닉스 시스템의 명령어 중에서 중요하고도 기본적인 실행 파일들이 모여 있는 곳이다.
/dev	시스템에 관련된 장치들을 모아놓은 곳이다. 유닉스는 모든 장치를 다 파일로 인식한다.
/etc	암호 파일과 시스템 설정에 관련된 중요한 파일들이 모여 있는 곳이다. 일반 유저보다는 시스템 관리자가 더 잘 알아야 하는 곳이다.
/home	사용자들의 홈 디렉토리가 위치하는 곳이다. 가령 계정명이 sogang인 사용자의 홈 디렉토리는 /home/sogang/으로 설정된다.
/lib	UNIX 시스템에서 사용되는 라이브러리 파일들을 모아놓은 곳이다.
/mnt	기본적인 파티션 외의 다른 저장 장치가 마운트 되는 곳이다. UNIX에서는 기본적으로 파티션(디스크의 일정 공간)을 사용하기 위해서는 마운트 작업을 수행해야 하는데, cdrom이나 Windows의 파티션이 여기에 마운트 될 수 있다.
/proc	UNIX 시스템의 정보가 들어 있는 곳이다. 이곳에는 현재 실행되고 있는 프로세스의 정보라든지 사용되고 있는 장치명 등 UNIX 커널에 관련된 정보들이 저장되어 있다.

/root	슈퍼유저(root)의 홈 디렉토리이다. 유닉스에서 슈퍼유저는 권한을 무시하고 무엇이든지 할 수 있는 유저라고 보면 된다. 따라서 많은 해커들이 슈퍼유저의 권한을 얻기 위해 애를 쓴다.
/sbin	bin이 일반 사용자들이 사용하는 명령들을 저장하고 있다면, 이곳에서는 시스템 관리를 위한 명령들을 저장하고 있다.
/tmp	임시 저장 공간으로 사용자라면 누구나 읽고 쓸 수 있는 디렉토리이다.
/usr	/usr 파일 시스템은 실제 사용자들과의 관계가 없어도 user file system이라고 부른다. /usr 디렉토리는 실행가능한 명령, 프로그램에 사용되는 헤더 파일, 라이브러리 그리고 시스템 관련 유틸리티 등이 들어 있는 곳이다. /bin과 /sbin의 전형적인 UNIX 프로그램들이 저장되는 공간이라면 /usr은 기본 프로그램들 외에 추가적으로 설치되는 프로그램들이 저장되는 공간이라고 보면 된다.
/var	시스템 접속 상황, 프린터 기록, 인스톨 결과, 패키지 정보, 패치 정보 등 시스템에 관한 모든 기록들이 남겨지는 곳이다.