5주차 3차시 프로세스와 쉘

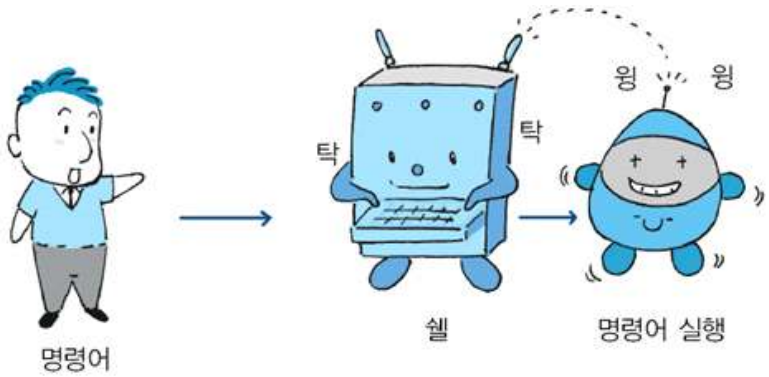
【학습목표】

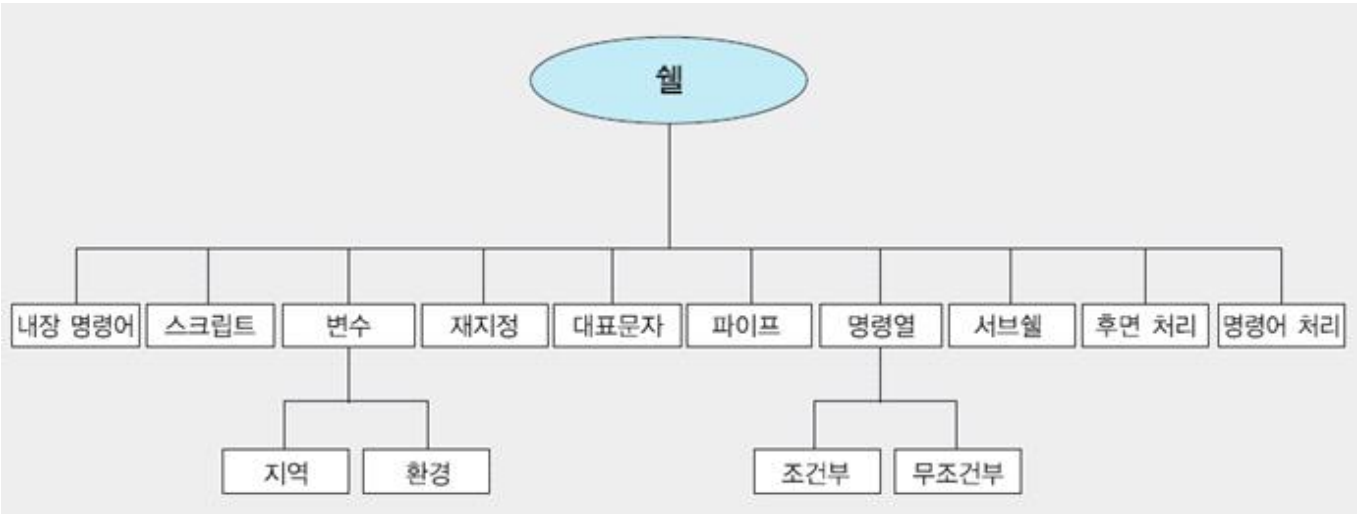
1. 리눅스 쉘을 설명할 수 있다.
2. 리눅스 프로세스 관련 함수를 사용할 수 있다.

학습내용1 : 리눅스 쉘

1. 쉘의 역할

쉘은 사용자와 운영체제 사이에 창구 역할을 하는 소프트웨어
명령어 처리기(command processor) : 사용자가 입력한 명령을 해석하고 적절한 프로그램을 실행
사용자로부터 명령어를 입력받아 이를 처리한다
시작 파일 : 로그인할 때 실행되어 사용자별로 맞춤형 사용 환경 설정
스크립트 : 쉘 자체 내의 프로그래밍 기능





2. 유닉스/리눅스에서 사용 가능한 쉘의 종류

쉘의 종류	쉘 실행 파일
본 쉘	/bin/sh
콘 쉘	/bin/ksh
C 쉘	/bin/csh
Bash	/bin/bash
tcsh	/bin/tcsh

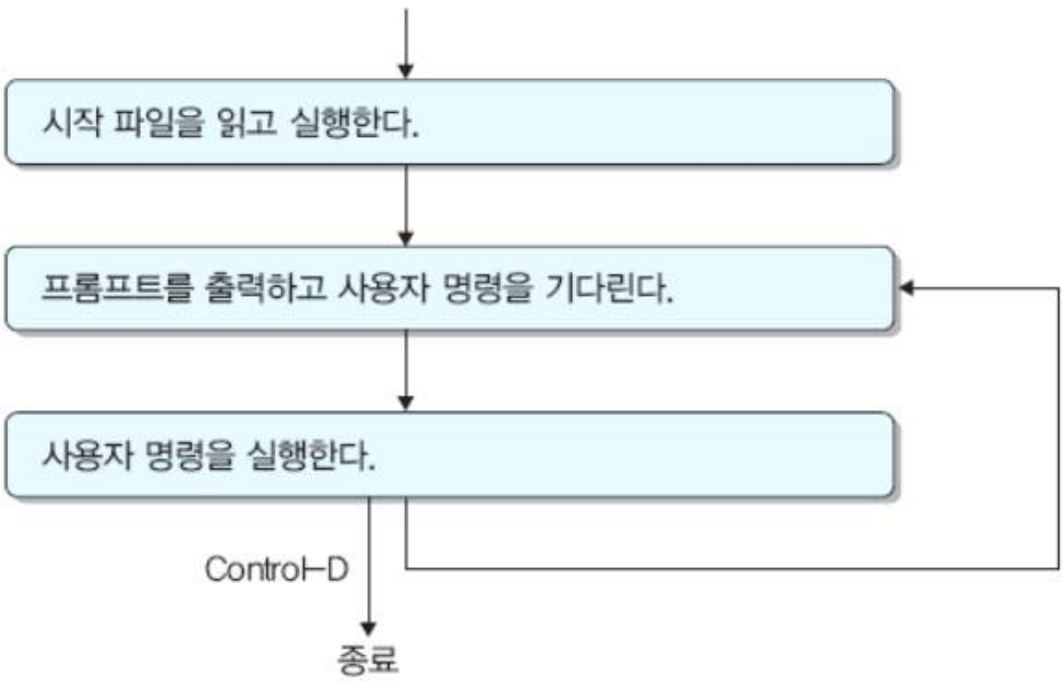
① 본 쉘(Bourne shell)
- 벨연구소의 스티븐 본(Stephen Bourne)에 의해 개발됨
- 유닉스에서 기본 쉘로 사용됨

② 콘 쉘(Korn shell)
- 1980년대에는 역시 벨연구소에서 본 쉘을 확장해서 만듬.

③ Bash(Bourne again shell)
- GNU에서 본 쉘을 확장하여 개발한 쉘
- 리눅스 및 맥 OS X에서 기본 쉘로 사용되면서 널리 보급됨
- Bash 명령어의 구문은 본 쉘 명령어 구문을 확장함

④ C 쉘(C shell)
- 버클리대학의 빌 조이(Bill Joy)
- 쉘의 핵심 기능 위에 C 언어의 특징을 많이 포함함
- BSD 계열의 유닉스에서 많이 사용됨
- 최근에 이를 개선한 tcsh이 개발되어 사용됨

3. 쉘의 실행 절차



학습내용2 : 리눅스 프로세스 처리 함수

1. 프로세스 실행 시간 측정

① 프로세스 실행 시간의 구성

```
프로세스 실행시간 = 시스템 실행시간 + 사용자 실행시간
```

시스템 실행시간 : 커널 코드를 수행한 시간(시스템 호출로 소비한 시간)
사용자 실행시간 : 사용자 모드에서 프로세스를 실행한 시간

② 프로세스 실행 시간 측정
사용자 실행시간과 시스템 실행시간으로 나누어 tms 구조체에 저장
시간 단위는 클록틱(sysconf 함수에서 _SC_CLK_TCK로 검색한 값)

```
#include <sys/times.h>
#include <limits.h>

clock_t times(struct tms *buffer);
```

buffer : 실행 시간을 저장할 tms 구조체의 주소

tms 구조체

```
struct tms {
    clock_t tms_utime;
    clock_t tms_stime;
    clock_t tms_cutime;
    clock_t tms_cstime;
};
```

utime : 사용자 모드실행시간
stime : 시스템 모드실행시간
cutime : 자식프로세스의 사용자 모드 실행 시간
cstime : 자식프로스세의 시스템 모드 실행 시간

times 함수 사용하기

```
...
08  int main(void) {
09      int i;
10      time_t t;
11      struct tms mytms;
12      clock_t t1, t2;
13
14      if ((t1 = times(&mytms)) == -1) {
15          perror("times 1");
16          exit(1);
17      }
18
19      for (i = 0; i < 999999; i++)
20          time(&t);
21
22      if ((t2 = times(&mytms)) == -1) {
23          perror("times 2");
24          exit(1);
25      }
26
27      printf("Real time : %.1f sec\n“,(double)(t2 - t1) / CLK_TCK);
28      printf("User time : %.1f sec\n“,(double)mytms.tms_utime / CLK_TCK);
29      printf("System time : %.1f sec\n“,(double)mytms.tms_stime / CLK_TCK);
30
31      return 0;
32  }
```

사용자 모드에서 시간을 소비하기 위한 반복문 처리

```
# ex5_4.out
Real time : 0.4 sec
User time : 0.2 sec
System time : 0.1 sec
```

```
<limits.h>
#define CLK_TCK ((clock_t) _sysconf(3)) /* 3 is _SC_CLK_TCK */
```

2. 환경변수의 이해

① 환경변수
프로세스가 실행되는 기본 환경을 설정하는 변수
로그인명, 로그인 쉘, 터미널에 설정된 언어, 경로명 등
환경변수는 “환경변수=값”의 형태로 구성되며 관례적으로 대문자로 사용
현재 쉘의 환경 설정을 보려면 env 명령을 사용

```
# env
_=/usr/bin/env
LANG=ko
HZ=100
PATH=/usr/sbin:/usr/bin:/usr/local/bin:.
LOGNAME=jw
MAIL=/usr/mail/jw
SHELL=/bin/ksh
HOME=/export/home/jw
TERM=ansi
PWD=/export/home/jw/syspro/ch5
TZ=ROK
...
```

3. 환경변수의 사용

① 전역변수 사용 : environ

```
#include <stdlib.h>

extern char **environ;
```

[예제 5-5] environ 전역 변수사용하기 ex5_5.c

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  extern char **environ;
05
06  int main(void) {
07      char **env;
08
09      env = environ;
10      while (*env) {
11          printf("%s\n", *env);
12          env++;
13      }
14
15      return 0;
16  }
```

```
# ex5_5.out
_=ex5_5.out
LANG=ko
HZ=100
PATH=/usr/sbin:/usr/bin:/usr/local/bin:.
LOGNAME=jw
MAIL=/usr/mail/jw
SHELL=/bin/ksh
HOME=/export/home/jw
TERM=ansi
PWD=/export/home/jw/syspro/ch5
TZ=ROK`
```

② 환경변수의 사용 : main 함수 인자 사용

```
int main(int argc, char **argv, char **envp) { ... }
```

[예제 5-6] main 함수 인자 ex5_6.c

```
01  #include <stdio.h>
02
03  int main(int argc, char **argv, char **envp) {
04      char **env;
05
06      env = envp;
07      while (*env) {
08          printf("%s\n", *env);
09          env++;
10      }
11
12      return 0;
13  }
```

```
# ex5_6.out
_=ex5_6.out
LANG=ko
HZ=100
PATH=/usr/sbin:/usr/bin:/usr/local/bin:.
LOGNAME=jw
MAIL=/usr/mail/jw
SHELL=/bin/ksh
HOME=/export/home/jw
TERM=ansi
PWD=/export/home/jw/syspro/ch5
TZ=ROK
```

4. 환경변수 검색: getenv(3)

```
#include <stdlib.h>

char *getenv(const char *name);
```

name : 환경 변수명

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      char *val;
06
07      val = getenv("SHELL");
08      if (val == NULL)
09          printf("SHELL not defined\n");
10      else
11          printf("SHELL = %s\n", val);
12
13      return 0;
14  }
```

```
# ex5_7.out
SHELL = /bin/ksh
```

5. 환경변수 설정: putenv(3)

```
#include <stdlib.h>

int putenv(char *string);
```

string : 설정할 환경 변수와 값으로 구성한 문자열

```
...
04  int main(void) {
05      char *val;
06
07      val = getenv("SHELL");
08      if (val == NULL)
09          printf("SHELL not defined\n");
10      else
11          printf("1. SHELL = %s\n", val);
12
13      putenv("SHELL=/usr/bin/csh");
14
15      val = getenv("SHELL");
16      printf("2. SHELL = %s\n", val);
18      return 0;
19  }
```

```
# ex5_8.out
1. SHELL = /usr/bin/ksh
2. SHELL = /usr/bin/csh
```

설정하려는 환경변수를 "환경변수=값"형태로 지정

6. 환경변수 설정: setenv(3)

```
#include <stdlib.h>

int setenv(const char *envname, const char *envval, int  overwrite);
```

envname : 환경변수명 지정
envval : 환경변수 값 지정
overwrite : 덮어쓰기 여부 지정, 0이 아니면 덮어쓰고, 0이면 덮어쓰지 않음

7. 환경변수 설정 삭제: unsetenv(3)

```
#include <stdlib.h>

int unsetenv(const char *name);
```

name : 환경 변수명

① setenv 함수 사용하기

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      char *val;
06
07      val = getenv("SHELL");
08      if (val == NULL)
09          printf("SHELL not defined\n");
10      else
11          printf("1. SHELL = %s\n", val);
12
13      setenv("SHELL","/usr/bin/csh", 0);
14      val = getenv("SHELL");
15      printf("2. SHELL = %s\n", val);
16
17      setenv("SHELL","/usr/bin/csh", 1);
18      val = getenv("SHELL");
19      printf("3. SHELL = %s\n", val);
20
21      return 0;
22  }
```

환경변수의 덮어쓰기가 되지 않음

환경변수의 덮어쓰기 설정

```
# ex5_9.out
1. SHELL = /usr/bin/ksh
2. SHELL = /usr/bin/ksh
3. SHELL = /usr/bin/csh
```

【학습정리】

1. 프로세스 실행 시간 측정
- 프로세스 실행 시간 = 시스템 실행시간 + 사용자 실행시간
- 프로세스 실행 시간 측정 함수 : clock_t times(struct tms *buffer);

2. 환경변수 활용
- 프로세스가 실행되는 기본 환경은 사용자의 로그인명, 로그인 쉘, 경로명 등을 포함하며, 환경 변수로 정의되어 있다.
- 환경변수는 '환경변수명=값' 형태로 구성되며, 환경 변수명은 관례적으로 대문자를 사용한다.
- 전역변수 사용 : extern char **environ;
- main() 함수의 인자 : int main(int argc, char **argv, char **envp){.....}