

제 6 장

시스템 정보와 프로세스

ACS30021
고급 프로그래밍

나보균 (bkna@kpu.ac.kr)

컴퓨터 공학과
한국산업기술 대학교

학습목표

- 유닉스 시스템 관련 정보
 - ✓ 운영체제 기본정보 검색
 - ✓ 시스템 정보 검색과 설정
 - ✓ 시스템 자원 정보 검색
- 사용자 정보 검색
 - ✓ 로그인명 검색
 - ✓ 패스워드 파일 검색
 - ✓ 쉘도우 파일 검색
 - ✓ 그룹 파일 검색
 - ✓ 로그인 기록 정보 검색
- 시간 관리 함수
 - ✓ 초 단위 시간 검색
 - ✓ 시간대 설정
 - ✓ 시간 정보 분해 함수
 - ✓ 초 단위 시간 생성 함수
 - ✓ 형식 지정 시간 출력 함수

유닉스 시스템 관련정보

- ❑ 시스템에 설치된 운영체제에 관한 정보
- ❑ 호스트명 정보
- ❑ 하드웨어 종류에 관한 정보
- ❑ 하드웨어에 따라 사용할 수 있는 자원의 최댓값
 - ✓ 최대 프로세스 개수
 - ✓ 프로세스당 열 수 있는 최대 파일 개수
 - ✓ 메모리 페이지 크기 등

운영체제 기본 정보 검색

□ 시스템에 설치된 운영체제에 대한 기본 정보 검색

```
# uname -a
```

SunOS	hanbit	5.10	Generic_118855-33	i86pc	i386	i86pc
운영체제명	호스트명	릴리즈 레벨	버전 번호	하드웨어 형식명	CPU명	플랫폼명

➤ 시스템은 인텔PC고 솔라리스 10운영체제가 설치되어 있고, 호스트명은 hanbit

□ 운영체제 정보 검색 함수 : uname(2)

```
#include <sys/utsname.h>
```

```
int uname(struct utsname *name);
```

✓ utsname 구조체에 운영체제 정보 저장

- sysname : 현재 운영체제 이름
- nodename : 호스트명
- release : 운영체제의 릴리즈 번호
- version : 운영체제 버전 번호
- machine : 하드웨어 아키텍처 이름

```
struct utsname {  
    char sysname[_SYS_NMLN];  
    char nodename[_SYS_NMLN];  
    char release[_SYS_NMLN];  
    char version[_SYS_NMLN];  
    char machine[_SYS_NMLN];  
};
```

uname 함수 사용하기

```
01 #include <sys/utsname.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main(void) {
06     struct utsname uts;
07
08     if (uname(&uts) == -1) {
09         perror("uname");
10         exit(1);
11     }
12
13     printf("OSname : %s\n", uts.sysname);
14     printf("Nodename : %s\n", uts.nodename);
15     printf("Release : %s\n", uts.release);
16     printf("Version : %s\n", uts.version);
17     printf("Machine : %s\n", uts.machine);
18
19     return 0;
20 }
```

```
# ex4_1.out
OSname : SunOS
Nodename : hanbit
Release : 5.10
Version : Generic_118855-33
Machine : i86pc
```

시스템 정보 검색과 설정[1]

❑ 시스템 정보 검색과 설정: sysinfo(2)

```
#include <sys/systeminfo.h>
long sysinfo (int command, char *buf, long count);
```

Posix

gethostname(2)

sethostname(2)

✓ command에 검색 command 인자

상수	설명
SI_SYSNAME(1)	운영체제명을 리턴한다. uname 함수의 sysname 항목과 같은 값이다.
SI_HOSTNAME(2)	uname 함수의 nodename 항목과 같은 값으로, 현재 시스템의 호스트명을 리턴한다.
SI_VERSION(4)	uname 함수의 version 항목과 같은 값을 리턴한다.
SI_MACHINE(5)	하드웨어 형식 값을 리턴한다. uname 함수의 machine 항목과 같은 값이다.
SI_ARCHITECTURE(6)	하드웨어의 명령어 집합 아키텍처(ISA, Instruction Set Architecture) 정보를 리턴한다. 예를 들면, sparc, mc68030, i386 등이다.
SI_HW_SERIAL(7)	하드웨어 장비의 일련번호를 리턴한다. 기본적으로 이 일련번호는 중복되지 않는다. SI_HW_PROVIDER 값과 함께 사용하면 모든 SVR4 업체의 제품을 구별하는 유일한 번호가 된다.
SI_HW_PROVIDER(8)	하드웨어 제조사 정보를 리턴한다.
SI_SRPC_DOMAIN(9)	Secure RPC(Remote Procedure Call) 도메인명을 리턴한다.

✓ 유닉스 표준에서 정의한 정보 설정 command 인자

상수	설명
SI_SET_HOSTNAME(258)	호스트명을 설정한다. 이 명령은 root 사용자만 사용할 수 있다.
SI_SET_SRPC_DOMAIN(265)	Secure RPC 도메인을 설정한다.

sysinfo 함수 사용하기(검색)

```
01 #include <sys/systeminfo.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main(void) {
06     char buf[257];
07
08     if (sysinfo(SI_HW_SERIAL, buf, 257) == -1) {
09         perror("sysinfo");
10         exit(1);
11     }
12     printf("HW Serial : %s\n", buf);
13
14     if (sysinfo(SI_ISALIST, buf, 257) == -1) {
15         perror("sysinfo");
16         exit(1);
17     }
18     printf("ISA List : %s\n", buf);
19
20     return 0;
21 }
```

하드웨어 일련번호 검색

사용가능한 아키텍처 목록검색

```
# ex4_2.out
HW Serial : 545486663
ISA List : amd64 pentium_pro+mmx pentium_pro pentium+mmx pentium
i486 i386 i86
```

sysinfo 함수 사용하기(설정)

```
01 #include <sys/systeminfo.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04 #include <string.h>
05
06 int main(void) {
07     char buf[257];
08
09     if (sysinfo(SI_HOSTNAME, buf, 257) == -1) {
10         perror("sysinfo");
11         exit(1);
12     }
13     printf("Before Hostname : %s\n", buf);
14
15     strcpy(buf, "hbooks");
16     if (sysinfo(SI_SET_HOSTNAME, buf, 257) == -1) {
17         perror("sysinfo");
18         exit(1);
19     }
20
21     if (sysinfo(SI_HOSTNAME, buf, 257) == -1) {
22         perror("sysinfo");
23         exit(1);
24     }
25     printf("After Hostname : %s\n", buf);
26
27     return 0;
28 }
```

ex4_3.out

Before Hostname : hanbit

After Hostname : hbooks

호스트 이름 변경

시스템 자원 정보 검색[1]

- ❑ 프로그램의 컴파일 이나 실행 중 하드웨어에 따라 사용할 수 있는 자원들의 설정 값이나 지원 여부를 검색
- ❑ 시스템 자원 정보 검색 : pathconf ()와 유사 기능

```
#include <unistd.h>
```

```
long sysconf(int name);
```

✓ 검색할 정보를 나타내는 상수를 사용해야 한다.

상수	설명
_SC_ARG_MAX(1)	argv[]와 envp[]를 합한 최대 크기로, 바이트 단위로 표시한다.
_SC_CHILD_MAX(2)	한 UID에 허용되는 최대 프로세스 개수를 나타낸다.
_SC_CLK_TCK(3)	초당 클럭 틱 수를 나타낸다.
_SC_OPEN_MAX(5)	프로세스당 열 수 있는 최대 파일 개수를 나타낸다.
_SC_VERSION(8)	시스템이 지원하는 POSIX.1의 버전을 나타낸다.

시스템 자원 정보 검색

✓ SVR4에서 정의한 상수

상수	설명
_SC_PASS_MAX(9)	패스워드의 최대 길이를 나타낸다.
_SC_LOGNAME_MAX(10)	로그인명의 최대 길이를 나타낸다.
_SC_PAGESIZE(11)	시스템 메모리의 페이지 크기를 나타낸다.

상수	설명
_SC_MEMLOCK(25)	프로세스 메모리 잠금 기능을 제공하는지 여부를 나타낸다.
_SC_MQ_OPEN_MAX(29)	한 프로세스가 열 수 있는 최대 메시지 큐 개수를 나타낸다.
_SC_SEMAPHORES(35)	시스템에서 세마포어를 지원하는지 여부를 나타낸다.

상수	설명
_SC_2_C_BIND(45)	C 언어의 바인딩 옵션을 지원하는지 여부를 알려준다.
_SC_2_C_VERSION(47)	ISO POSIX-2 표준의 버전을 나타낸다.

✓ XPG.4에서 정의한 상수

sysconf 함수 사용하기

```
01 #include <unistd.h>
02 #include <stdio.h>
03
04 int main(void) {
05     printf("Clock Tick : %ld\n", sysconf(_SC_CLK_TCK));
06     printf("Max Open File : %ld\n", sysconf(_SC_OPEN_MAX));
07     printf("Max Login Name Length : %ld\n", sysconf(_SC_LOGNAME_MAX));
08
09     return 0;
10 }
```

```
# ex4_4.out
Clock Tick : 100
Max Open File : 256
Max Login Name Length : 8
```

시스템 자원 정보 검색[3]

- 파일과 디렉토리 관련 자원 검색 : pathconf(3), fpathconf(3)

```
#include <unistd.h>
```

```
long pathconf(const char *path, int name);  
long fpathconf(int fildes, int name);
```

- ✓ 경로(path)나 파일기숀자에 지정된 파일에 설정된 자원값이나 옵션값 리턴
- ✓ name 사용할 상수

상수	설명
_PC_LINK_MAX(1)	디렉토리 혹은 파일 하나에 가능한 최대 링크 수를 나타낸다.
_PC_NAME_MAX(4)	파일명의 최대 길이를 바이트 크기로 나타낸다.
_PC_PATH_MAX(5)	경로명의 최대 길이를 바이트 크기로 나타낸다.

pathconf 함수 사용하기

```
01  #include <unistd.h>
02  #include <stdio.h>
03
04  int main(void) {
05      printf("Link Max : %ld\n", pathconf(".", _PC_LINK_MAX));
06      printf("Name Max : %ld\n", pathconf(".", _PC_NAME_MAX));
07      printf("Path Max : %ld\n", pathconf(".", _PC_PATH_MAX));
08
09      return 0;
10  }
```

```
# ex4_5.out
Link Max : 32767
Name Max : 255
Path Max : 1024
```

사용자 정보 검색

❑ 사용자 정보, 그룹정보, 로그인 기록 검색

✓ /etc/passwd, /etc/shadow, /etc/group, /var/adm/utmpx

❑ 로그인명 검색 : getlogin(3), cuserid(3)

```
#include <unistd.h>
char *getlogin(void);
```

✓ /var/adm/utmpx 파일을 검색해 현재 프로세스를 실행한 사용자의 로그인명을 리턴

```
#include <stdio.h>
char *cuserid(char *s);
```

✓ 현재 프로세스의 소유자 정보로 로그인명을 찾아 리턴

❑ UID검색

```
#include <sys/types.h>
#include <unistd.h>

uid_t getuid(void);
uid_t geteuid(void);
```

getuid, geteuid 함수 사용하기

```
01 #include <sys/types.h>
02 #include <unistd.h>
03 #include <stdio.h>
04
05 int main(void) {
06     uid_t uid, euid;
07     char *name, *cname;
08
09     uid = getuid();
10     euid = geteuid();
11
12     name = getlogin();
13     cname = cuserid(NULL);
14
15     printf("Login Name=%s,%s UID=%d, EUID=%d\n", name, cname,
16           (int)uid, (int)euid);
17
18     return 0;
19 }
```

ex4_6.out

Login Name=root,root UID=0, EUID=0

```
# chmod 4755 ex4_6.out
```

```
# ls -l ex4_6.out
```

```
-rwsr-xr-x  1 root    other
```

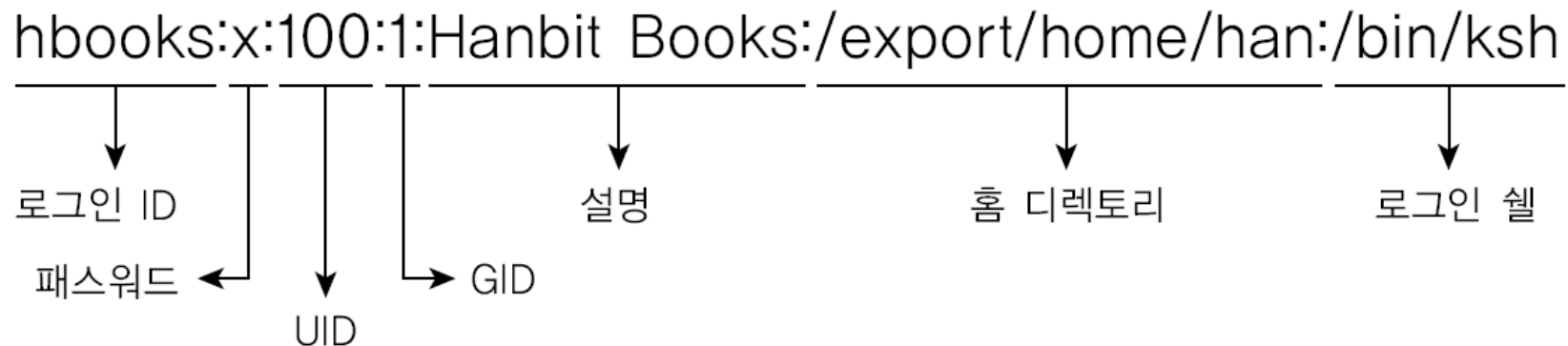
setuid 설정 후 일반사용자가 이 파일을 실행하면?

```
5964  1월 29일  15:11 ex4_6.out
```

패스워드 파일 검색[1]

- ❑ /etc/passwd 파일의 구조
- ❑ /etc/shadow 파일에 password 저장

```
# cat /etc/passwd
root:x:0:0:Super-User:/:/usr/bin/ksh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
.....
hbooks:x:100:1:Hanbit Books:/export/home/han:/bin/ksh
```



[그림 4-1] 사용자 계정의 예

패스워드 파일 검색[2]

- ❑ UID로 passwd 파일 읽기 : getpwuid(3)

```
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
```

- ❑ 이름으로 passwd 파일 읽기 : getpwnam(3)

```
#include <pwd.h>

struct passwd *getpwnam(const char *name);
```

✓ passwd 구조체

```
struct passwd {
    char    *pw_name;
    char    *pw_passwd;
    uid_t   pw_uid;
    gid_t   pw_gid;
    char    *pw_age;
    char    *pw_comment;
    char    *pw_gecos;
    char    *pw_dir;
    char    *pw_shell;
};
```

getpwuid 함수 사용하기

```
01 #include <unistd.h>
02 #include <pwd.h>
03
04 int main(void) {
05     struct passwd *pw;
06
07     pw = getpwuid(getuid());
08     printf("UID : %d\n", (int)pw->pw_uid);
09     printf("Login Name : %s\n", pw->pw_name);
10
11     return 0;
12 }
```

```
# ex4_7.out
UID : 0
Login Name : root
```

getpwnam 함수 사용하기

```
01 #include <pwd.h>
02
03 int main(void) {
04     struct passwd *pw;
05
06     pw = getpwnam("hbooks");
07     printf("UID : %d\n", (int)pw->pw_uid);
08     printf("Home Directory : %s\n", pw->pw_dir);
09
10     return 0;
11 }
```

```
# ex4_8.out
UID : 100
Home Directory : /export/home/han
```

패스워드 파일 검색[3]

- ❑ /etc/passwd 파일 순차적으로 읽기

```
#include <pwd.h>

struct passwd *getpwent(void);
void setpwent(void);
void endpwent(void);
struct passwd *fgetpwent(FILE *fp);
```

getpwent 함수 사용하기

```
01  #include <pwd.h>
02
03  int main(void) {
04      struct passwd *pw;
05      int n;
07      for (n = 0; n < 3; n++) {
08          pw = getpwent();
09          printf("UID: %d, LoginName: %s\n", (int)pw->pw_uid,
                pw->pw_name);
10      }
11
12      return 0;
13  }
```

```
# ex4_9.out
UID: 0, LoginName: root
UID: 1, LoginName: daemon
UID: 2, LoginName: bin
```

새도우 파일 검색[1]

□ /etc/shadow 파일의 구조

```
# cat /etc/shadow
root:lyTy6ZkWh4RYw:13892::::::
daemon:NP:6445::::::
bin:NP:6445::::::
.....
hbooks:KzV35jsiil./6:14273:3:30:7:10:14344:
```



[그림 4-2] 사용자 패스워드 정보의 예

새도우 파일 검색[2]

- ❑ /etc/shadow 파일 읽기 : getspnam(3)

```
#include <shadow.h>
```

```
struct spwd *getspnam(const char *name);
```

✓ spwd 구조체

```
struct spwd {  
    char *sp_namp;  
    char *sp_pwdp;  
    int    sp_lstchg;  
    int    sp_min;  
    int    sp_max;  
    int    sp_warn;  
    int    sp_inact;  
    int    sp_expire;  
    unsigned int sp_flag;  
};
```

getspnam 함수 사용하기

```
01  #include <shadow.h>
02
03  int main(void) {
04      struct spwd *spw;
05
06      spw = getspnam("hbooks");
07      printf("Login Name : %s\n", spw->sp_namp);
08      printf("Passwd : %s\n", spw->sp_pwdp);
09      printf("Last Change : %d\n", spw->sp_lstchg);
10
11      return 0;
12  }
```

```
# ex4_10.out
Login Name : hbooks
Passwd : KzV35jsiil./6
Last Change : 14273
```

새도우 파일 검색[3]

- /etc/shadow 파일 순차적으로 읽기

```
#include <shadow.h>

struct spwd *getspent(void);
void setspent(void);
void endspent(void);
struct spwd *fgetspent(FILE *fp);
```

getspent 함수 사용하기

```
01 #include <shadow.h>
02
03 int main(void) {
04     struct spwd *spw;
05     int n;
06
07     for (n = 0; n < 3; n++) {
08         spw = getspent();
09         printf("LoginName: %s, Passwd: %s\n", spw->sp_namp, spw->sp_pwdp);
10     }
11
12     return 0;
13 }
```

ex4_11.out

LoginName: root, Passwd: lyTy6ZkWh4RYw

LoginName: daemon, Passwd: NP

LoginName: bin, Passwd: NP

그룹 정보 검색

- ❑ 그룹 ID 검색하기 : `getgid(2)`, `getegid(2)`

```
#include <sys/types.h>
#include <unistd.h>

gid_t getgid(void);
gid_t getegid(void);
```

getid, getegid 함수 사용하기

```
01  #include <sys/types.h>
02  #include <unistd.h>
03  #include <stdio.h>
04
05  int main(void) {
06      gid_t gid, egid;
07
08      gid = getgid();
09      egid = getegid();
10
11      printf("GID=%d, EGID=%d\n", (int)gid, (int)egid);
12
13      return 0;
14  }
```

```
# ex4_12.out
GID=1, EGID=1
```

그룹 파일 검색[1]

□ /etc/group 파일의 구조

```
# cat /etc/group
root::0:
other::1:root
bin::2:root,daemon
sys::3:root,bin,adm
adm::4:root,daemon
uucp::5:root
.....
```

✓ group 구조체

```
struct group {
    char    *gr_name;
    char    *gr_passwd;
    gid_t   gr_gid;
    char    **gr_mem;
};
```

그룹 파일 검색[2]

- ❑ /etc/group 파일 검색 : getgrnam(3), getgrgid(3)

```
#include <grp.h>

struct group *getgrnam(const char *name);
struct group *getgrgid(gid_t gid);
```

- ❑ /etc/group 파일 순차적으로 읽기

```
#include <grp.h>

struct group *getgrent(void);
void setgrent(void);
void endgrent(void);
struct group *fgetgrent(FILE *fp);
```

getgrnam 함수 사용하기

```
01 #include <grp.h>
02
03 int main(void) {
04     struct group *grp;
05     int n;
06
07     grp = getgrnam("adm");
08     printf("Group Name : %s\n", grp->gr_name);
09     printf("GID : %d\n", (int)grp->gr_gid);
10
11     n = 0;
12     printf("Members : ");
13     while (grp->gr_mem[n] != NULL)
14         printf("%s ", grp->gr_mem[n++]);
15     printf("\n");
16
17     return 0;
18 }
```

```
# ex4_13.out
Group Name : adm
GID : 4
Members : root daemon
```

getgrent 함수 사용하기

```
01 #include <grp.h>
02
03 int main(void) {
04     struct group *grp;
05     int n,m;
06
07     for (n = 0; n < 3; n++) {
08         grp = getgrent();
09         printf("GroupName: %s, GID: %d ", grp->gr_name,
10                (int)grp->gr_gid);
11
12         m = 0;
13         printf("Members : ");
14         while (grp->gr_mem[m] != NULL)
15             printf("%s ", grp->gr_mem[m++]);
16         printf("\n");
17     }
18     return 0;
19 }
```

ex4_14.out

GroupName: root, GID: 0 Members :

GroupName: other, GID: 1 Members : root

GroupName: bin, GID: 2 Members : root daemon

로그인 기록 검색[1]

- ❑ who 명령: 현재 시스템에 로그인하고 있는 사용자 정보
- ❑ last 명령: 시스템의 부팅 시간 정보와 사용자 로그인 기록 정보
- ❑ utmpx 구조체

```
struct utmpx {  
    char    ut_user[32];        /* 사용자 로그인명 */  
    char    ut_id[4];          /* inittab id */  
    char    ut_line[32];       /* 로그인한 장치이름 */  
    pid_t   ut_pid;            /* 실행중인 프로세스 PID*/  
    short   ut_type;           /* 현재 읽어온 항목의 종류 */  
    struct  exit_status ut_exit; /* 프로세스 종료 상태 코드 */  
    struct  timeval ut_tv;      /* 해당정보를 변경한 시간 */  
    int     ut_session;        /* 세션 번호 */  
    int     pad[5];            /* 예약 영역 */  
    short   ut_syslen;         /* ut_host의 크기 */  
  
    char    ut_host[257];      /* 원격호스트명 */  
};
```

로그인 기록 검색[2]

❑ ut_type : 현재 읽어온 항목의 종류

- ✓ EMPTY(0): 비어 있는 항목
- ✓ RUN_LVL(1): 시스템의 런레벨의 변경. 바뀐 런레벨은 ut_id에 저장
- ✓ BOOT_TIME(2): 시스템 부팅 정보. 부팅 시간은 ut_time에 저장
- ✓ OLD_TIME(3): date 명령으로 시스템 시간의 변경 표시. 변경되기 전의 시간을 저장
- ✓ NEW_TIME(4): date 명령으로 시스템 시간의 변경 표시. 변경된 시간을 저장
- ✓ INIT_PROCESS(5): init에 의해 생성된 프로세스임을 표시. 프로세스명은 ut_name에 저장하고 프로세스 ID는 ut_pid에 저장
- ✓ LOGIN_PROCESS(6): 사용자가 로그인하기를 기다리는 getty 프로세스를 표시
- ✓ USER_PROCESS(7): 사용자 프로세스를 표시
- ✓ DEAD_PROCESS(8): 종료한 프로세스를 표시
- ✓ ACCOUNTING(9): 로그인 정보를 기록한 것임을 표시
- ✓ DOWN_TIME(10): 시스템을 다운시킨 시간. ut_type이 가질 수 있는 가장 큰 값

로그인 기록 검색[3]

- ❑ /var/adm/utmpx 파일 순차적으로 읽기

```
#include <utmpx.h>

struct utmpx *getutxent(void);
void setutxent(void);
void endutxent(void);
int utmpxname(const char *file);
```

getutxent 함수 사용하기

```
...
05 int main(void) {
06     struct utmpx *utx;
07
08     printf("LoginName  Line\n");
09     printf("-----\n");
10
11     while ((utx=getutxent()) != NULL) {
12         if (utx->ut_type != USER_PROCESS)
13             continue;
14
15         printf("%s      %s\n", utx->ut_user, utx->ut_line);
16     }
17
18     return 0;
19 }
```

```
# ex4_15.out
LoginName  Line
-----
root       console
root       pts/3
root       pts/5
root       pts/4
```


프로그램 실행 : system(3)

```
#include <stdlib.h>
int system(const char *string);
```

- ✓ 새로운 프로그램을 실행하는 가장 간단한 방법
- ✓ 비효율적이므로 남용하지 말 것
- ✓ 실행할 프로그램명을 인자로 지정

system 함수 사용하기

```
01 #include <stdlib.h>
02 #include <stdio.h>
03
04 int main(void) {
05     int a;
06     a = system("ps -ef | grep han > han.txt");
07     printf("Return Value : %d\n", a);
08
09     return 0;
10 }
```

```
# ex6_1.out
Return Value : 0
# cat han.txt
root 736 735 0 10:31:02 pts/3 0:00 grep han
root 735 734 0 10:31:02 pts/3 0:00 sh -c ps -ef | grep han> han.txt
```

시간 관리 함수[1]

❑ 유닉스 시스템에서 시간관리

- ✓ 1970년 1월 1일 0시 0분 0초(UTC)를 기준으로 현재까지 경과한 시간을 초 단위로 저장하고 이를 기준으로 시간 정보 관리

❑ 초 단위로 현재 시간 정보 얻기 : time(2)

```
#include <sys/types.h>
#include <time.h>

time_t time(time_t *tloc);
```

time 함수 사용하기

```
01 #include <sys/types.h>
02 #include <time.h>
03 #include <stdio.h>
04
05 int main(void) {
06     time_t tt;
07
08     time(&tt);
09     printf("Time(sec) : %d\n", (int)tt);
10
11     return 0;
12 }
```

```
# ex4_16.out
Time(sec) : 1233361205
```

POSIX 시간

- ❑ UTC (Coordinated Universal Time)
- ❑ Epoch 이후 시간을 초 단위로 정의
 - ✓ Epoch – 1970년 1월 1일 00:00를 의미
- ❑ 하루 86,400 초
- ❑ 정밀도
- ❑ 시간 측정 방식
 - ✓ 클록 (Clock)
 - 고정된 짧은 시간 구간마다 증가되는 수
 - ✓ 시간 (wall-clock)
 - 벽에 걸린 시계의 시간을 의미, 즉 현재 사용되는 시간의 단위
 - 년, 월, 일, 시, 분, 초
 - calendar time 이라고도 불림

변수 형 정의

- ❑ time_t
- ❑ struct tm
- ❑ struct timeval
- ❑ struct timespec
- ❑ clockid_t
 - ✓ 특정 클럭을 지정
 - ✓ CLOCK_REALTIME
- ❑ struct tms

시간 표현

- ❑ 두 시간 사이의 차이 표현에 사용
- ❑ Epoch 이후 시간을 초 단위로 표현 함수

```
#include <time.h>  
time_t time (time_t *t)
```

- ❑ time_t
 - ✓ long 형을 재 정의
- ❑ *t 는 NULL 또는 값을 저장할 공간의 포인터

time(), difftime()

```
#include <stdio.h>
#include <time.h>

void function_to_time(void);

int main(void)
{
    time_t tstart;

    tstart = time(NULL);
    function_to_time();
    printf ("function_to_time took %f seconds of elapsed time\n", difftime(time(NULL), tstart));
    return 0;
}
```

시간 표현 - 월력 시간 표현법

- 시간 구성 요소 (년, 월, 일) + 지역적 특성 (time zone, daylight-saving time, leap second)

- struct tm {

```
int tm_sec;    // 초 [0, 60]
int tm_min;    // 분 [0, 60]
int tm_hour;   // 시 [0, 23]
int tm_mday;   // 날 [1, 31]
int tm_mon;    // 달 [0, 11]
int tm_year;   // 년 [1900, ]
int tm_wday;   // 요일
int tm_yday;   // 1월1일 이후 날 [0, 365]
int tm_isdst;  // 일광절약시간 표시 플래그
```

```
}
```

시간 표현 - 월력 시간 표현법

- 구조체 tm을 통하여 년, 월, 일, 시, 분, 초 표현
 - ✓ calendar time
 - ✓ 재호출 시 오류 발생

```
char *ctime (const time_t *clock);  
struct tm *localtime (const time_t *timer);
```

- 예
 - ✓ Sun Oct 06 02:21:35 2009

형식 지정 시간 출력[1]

- 초 단위 시간을 변환해 출력하기: ctime(3)

```
#include <time.h>

char *ctime(const time_t *clock);
```

ctime 함수 사용하기

```
01 #include <time.h>
02 #include <stdio.h>
03
04 int main(void) {
05     time_t t;
06
07     time(&t);
08
09     printf("Time(sec) : %d\n", (int)t);
10     printf("Time(date) : %s\n", ctime(&t));
11
12     return 0;
13 }
```

```
# ex4_21.out
Time(sec) : 1233370759
Time(date) : Sat Jan 31 11:59:19 2009
```

형식 지정 시간 출력[2]

- ❑ tm 구조체 시간을 변환해 출력하기: asctime(3)

```
#include <time.h>

char *asctime(const struct tm *tm);
```

asctime 함수 사용하기

```
01 #include <time.h>
02 #include <stdio.h>
03
04 int main(void) {
05     struct tm *tm;
06     time_t t;
07
08     time(&t);
09     tm = localtime(&t);
10
11     printf("Time(sec) : %d\n", (int)t);
12     printf("Time(date) : %s\n", asctime(tm));
13
14     return 0;
15 }
```

```
# ex4_22.out
Time(sec) : 1233371061
Time(date) : Sat Jan 31 12:04:21 2009
```

시간 표현

```
#include <stdio.h>
#include <time.h>

void function_to_time(void);

int main(void) {
    time_t tend, tstart;

    tstart = time(NULL);
    function_to_time();
    tend = time(NULL);
    printf("The time before was %s The time after was %s", ctime(&tstart), ctime(&tend));
    return 0;
}
```

ctime()는 시간 문자열을 정적변수에 저장하기에 두 번 호출은 오류

시간 표현

❑ 스레드 안전 함수 (Thread safety)

```
#include <time.h>

char *ctime_r (const time_t *clock, char *buf);
struct tm *localtime_r (const time_t *restrict timer, struct tm *restrict result);
```

- ✓ 함수 실행에 성공하면:
 - 결과값이 저장된 매개변수 포인터 반환
 - ctime_r()는 buf에 결과 값을 저장
 - localtime_r () 은 result에 결과 값을 저장
- ✓ 실패하면 NULL 반환

localtime_r ()

- ❑ 1월1일부터 오늘까지 지난 날 수를 출력

```
struct tm tbuf;
```

```
if (localtime_r (&(time(NULL)), &tbuf) != NULL)
    printf ("1월1일 이후 오늘까지 %d 날입니다.\n", tbuf.tm_yday);
```

- ❑ 월력에 따른 시간 표현

```
struct tm curtime;
```

```
time_t curtimep;
```

```
curtimep = time(NULL);
```

```
if (localtime_r (&curtimep, &curtime) != NULL)
    printf ("현재 시간: %04d년 %02d월 %02d일: %02d시 %02d분 %02d초\n",
            curtime.tm_year + 1900, curtime.tm_mon + 1, curtime.tm_mday,
            curtime.tm_hour, curtime.tm_min, curtime.tm_sec);
```

실습 & 과제:

□ 디렉터리 생성

✓ ~HOME/class/ACS30021/chap05/date/yyyy/mm/dd/hh/

➤ y: year

➤ yyyy: 4자리로 년도 표시

➤ m: month

➤ d: day

➤ h: hour

➤ t: ten minute, 즉, 00, 10, 20, 30, 40, 50 중 하나

□ 각각의 최종 서브디렉터리에 파일 저장

✓ 파일명 yyyyymmdd_hhmm.vodo 파일

✓ 각 파일의 내용에는 년 월 일 시 분 저장

시간 표현 – struct timeval

- 시간을 계산하는 프로그램이나 이벤트 제어를 위해 사용

- ✓ select ()
- ✓ gettimeofday() – 마이크로 초단위까지

- struct timeval

```
struct timeval {  
    time_t tv_sec; // Epoch 이후부터의 초 단위 시간  
    time_t tv_usec; // 마이크로 초 (μsec)  
}
```

실행 시간 측정 - 마이크로 초 단위

```
#include <stdio.h>
#include <sys/time.h>
#define MILLION 1000000L

void function_to_time(void);

int main(void)
{
    long timedif;
    struct timeval tpend, tpstart;

    if (gettimeofday(&tpstart, NULL)) {
        fprintf(stderr, "Failed to get start time\n");
        return 1;
    }
    function_to_time(); /* timed code goes here */
    if (gettimeofday(&tpend, NULL)) {
        fprintf(stderr, "Failed to get end time\n");
        return 1;
    }
    timedif = MILLION*(tpend.tv_sec - tpstart.tv_sec) + tpend.tv_usec - tpstart.tv_usec;
    printf("The function_to_time took %ld microseconds\n", timedif);
    return 0;
}
```


실행 시간 측정 - 나노 초 단위

❑ struct timespec

```
struct timespec {  
    time_t tv_sec;    // 초 단위 (sec)  
    long   tv_nsec;   // 나노 초 단위 (nsec)  
}
```

❑ 스레드 함수의 시간 종료 매개변수에 사용되는 타이머 변수를 설정

예제 프로그램

```
#include <stdio.h>
#include <time.h>
#define BILLION 1000000000L
#define NUMDIF 20

int main(void)
{
    int l, numcalls = 1, numdone = 0;
    long sum = 0;
    long timedif[NUMDIF];
    struct timespec tlast, tthis;

    if (clock_getres(CLOCK_REALTIME, &tlast))
        perror("Failed to get clock resolution");
    else if (tlast.tv_sec != 0)
        printf("Clock resolution no better than one second\n");
    else
        printf("Clock resolution: %ld nanoseconds\n", (long)tlast.tv_nsec);

    if (clock_gettime(CLOCK_REALTIME, &tlast)) {
        perror("Failed to get first time");
        return 1;
    }
```

예제 프로그램

```
while (numdone < NUMDIF) {
    numcalls++;
    if (clock_gettime(CLOCK_REALTIME, &tthis)) {
        perror("Failed to get a later time");
        return 1;
    }
    timedif[numdone] = BILLION*(tthis.tv_sec - tlast.tv_sec) + tthis.tv_nsec - tlast.tv_nsec;
    if (timedif[numdone] != 0) {
        numdone++;
        tlast = tthis;
    }
}
printf("Found %d differences in CLOCK_REALTIME:\n", NUMDIF);
printf("%d calls to CLOCK_REALTIME were required\n", numcalls);
for (i = 0; i < NUMDIF; i++) {
    printf("%2d: %10ld nanoseconds\n", i, timedif[i]);
    sum += timedif[i];
}
printf("The average nonzero difference is %f\n", sum/(double)NUMDIF);
return 0;
}
```

프로세싱 시간 측정 – times ()

- ❑ 프로세스가 실행 상태로 있는 동안 사용한 시간
 - ✓ 프로세스의 실행 시간을 인자로 지정한 tms 구조체에 저장
 - ✓ clock_t 시작 시점부터 경과된 시간으로 클록 틱(tick)값으로 표현
 - ✓ 1초당 100틱
 - ✓ 실행 중 컨텍스트 스위칭 시간 포함

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

```
clock_t times (struct tms *buff); // 실행 시간을 저장할 tms 구조체의 주소
```

❑ struct tms

```
struct tms {  
    clock_t tms_utime; //프로세스에서 사용자가(명령어) 사용한 CPU 시간  
    clock_t tms_stime; // 프로세스에서 시스템 CPU를 사용한 시간  
    clock_t tms_cutime; // 프로세스와 자식 프로세스에서 사용자가 사용한 CPU 시간  
    clock_t tms_cstime;  
}
```

예제 프로그램

- 프로그램을 실행하는데 걸린 시간을 반환

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/times.h>

void function_to_time(void);

int main(void)
{
    double clockticks, cticks;
    clock_t tcend, tcstart;
    struct tms tmend, tmstart;

    if ((clockticks = (double) sysconf(_SC_CLK_TCK)) == -1) {
        perror("Failed to determine clock ticks per second");
        return 1;
    }
    printf("The number of ticks per second is %fWn", clockticks);

    if (clockticks == 0) {
        fprintf(stderr, "The number of ticks per second is invalidWn");
        return 1;
    }
}
```

예제 프로그램

```
if ((tcstart = times(&tmstart)) == -1) {
    perror("Failed to get start time");
    return 1;
}

function_to_time();

if ((tcend = times(&tmend)) == -1) {
    perror("Failed to get end times");
    return 1;
}
cticks = tmend.tms_utime + tmend.tms_stime - tmstart.tms_utime - tmstart.tms_stime;
printf("Total CPU time for operation is %f seconds\n", cticks/clockticks);
if ((tcend <= tcstart) || (tcend < 0) || (tcstart < 0)) {
    fprintf(stderr, "Tick time wrapped, couldn't calculate fraction\n");
    return 1;
}
printf("Fraction of CPU time used is %f\n", cticks/(tcend - tcstart));
return 0;
}
```

시간 관리 함수[3]

□ 시간대 정보 : tzset(3)

- ✓ 현재 지역의 시간대로 시간을 설정
- ✓ 이 함수를 호출하면 전역변수 4개에 정보를 설정
 - timezone : UTC와 지역 시간대와 시차를 초 단위로 저장
 - altzone : UTC와 일광절약제 등으로 보정된 지역시간대와의 시차를 초 단위로 저장
 - daylight : 일광절약제를 시행하면 0이 아니고, 아니면 0
 - tzname : 지역시간대와 보정된 시간대명을 약어로 저장

```
#include <time.h>

void tzset(void);
```

tzset 함수 사용하기

```
01 #include <time.h>
02 #include <stdio.h>
03
04 int main(void) {
05     tzset();
06
07     printf("Timezone : %d\n", (int)timezone);
08     printf("Altzone : %d\n", (int)altzone);
09     printf("Daylight : %d\n", daylight);
10     printf("TZname[0] : %s\n", tzname[0]);
11     printf("TZname[1] : %s\n", tzname[1]);
12
13     return 0;
14 }
```

```
# ex4_18.out
Timezone : -32400
Altzone : -36000
Daylight : 1
TZname[0] : KST
TZname[1] : KDT
```

UTC와 9시간(32,400초) 시차가 발생

시간의 형태 변환

- 초 단위 시간 정보 분해 : `gmtime(3)`, `localtime(3)`

```
#include <time.h>
```

```
struct tm *localtime(const time_t *clock);  
struct tm *gmtime(const time_t *clock);
```

- ✓ 초를 인자로 받아 tm구조 리턴,
- ✓ `gmtime`은 UTC기준,
- ✓ `localtime`은 지역시간대 기준

- 캘린더 단위 시간을 초 단위로 역산 : `mktime(3)`

```
#include <time.h>
```

```
time_t mktime(struct tm *timeptr);
```


gmtime, localtime 함수 사용하기

```
01 #include <time.h>
02 #include <stdio.h>
03
04 int main(void) {
05     struct tm *tm;
06     time_t t;
07
08     time(&t);
09     printf("Time(sec) : %d\n", (int)t);
10
11     tm = gmtime(&t);
12     printf("GMTIME=Y:%d ", tm->tm_year);
13     printf("M:%d ", tm->tm_mon);
14     printf("D:%d ", tm->tm_mday);
15     printf("H:%d ", tm->tm_hour);
16     printf("M:%d ", tm->tm_min);
17     printf("S:%d\n", tm->tm_sec);
18
19     tm = localtime(&t);
20     printf("LOCALTIME=Y:%d ", tm->tm_year);
21     printf("M:%d ", tm->tm_mon);
22     printf("D:%d ", tm->tm_mday);
```

gmtime, localtime 함수 사용하기

```
23     printf("H:%d ", tm->tm_hour);  
24     printf("M:%d ", tm->tm_min);  
25     printf("S:%d\n", tm->tm_sec);  
26  
27     return 0;  
28 }
```

```
# ex4_19.out  
Time(sec) : 1233369331  
GMTIME=Y:109 M:0 D:31 H:2 M:35 S:31  
LOCALTIME=Y:109 M:0 D:31 H:11 M:35 S:31
```

연도가 109?
어떻게 해석해야하나?

mktime 함수 사용하기

```
01  #include <time.h>
02  #include <stdio.h>
03
04  int main(void) {
05      struct tm tm;
06      time_t t;
07
08      time(&t);
09      printf("Current Time(sec) : %d\n", (int)t);
10
11      tm.tm_year = 109;
12      tm.tm_mon = 11;
13      tm.tm_mday = 31;
14      tm.tm_hour = 12;
15      tm.tm_min = 30;
16      tm.tm_sec = 0;
17
18      t = mktime(&tm);
19      printf("2009/12/31 12:30:00 Time(sec) : %d\n", (int)t);
20
21      return 0;
22  }
```

ex4_20.out

Current Time(sec) : 1233370219

2009/12/31 12:30:00 Time(sec) : 1262226600

형식 지정 시간 출력[3]

❑ 출력 형식 기호 사용 : strftime(3)

```
#include <time.h>
```

```
size_t strftime(char *restrict s, size_t maxsize,  
const char *restrict format, const struct tm *restrict timeptr);
```

- ✓ s : 출력할 시간 정보를 저장할 배열 주소
- ✓ maxsize : s의 크기
- ✓ format : 출력 형식
- ✓ timeptr : 출력할 시간정보를 저장한 구조체 주소

형식 지정 시간 출력[3]

지정자	기능	지정자	기능
%a	지역 시간대의 요일명 약자	%A	지역 시간대의 요일명
%b	지역 시간대의 월 이름 약자	%B	지역 시간대의 월 이름
%c	지역 시간대에 적합한 날짜와 시간 표현	%C	date 명령의 결과와 같은 형태로 날짜와 시간 표현
%d	날짜(0~31)	%D	날짜(%m/%d/%y)
%e	날짜(0~31). 한 자리 수는 앞에 공백 추가	%F	%Y-%m-%d 형태로 표현
%g	년도(00~99)	%G	년도(0000~9999)
%h	지역 시간대 월 이름 약자	%j	1년 중 일 수(001~365)
%H	24시간 기준 시간(00~23)	%I	12시간 기준 시간(01~12)
%k	24시간 기준 시간(00~23), 한 자리 수는 앞에 공백 추가	%I	12시간 기준 시간(01~12), 한 자리 수는 앞에 공백 추가
%m	월(01~12)	%M	분(00~59)
%p	지역 시간대 a.m, p.m	%r	%p와 함께 12시간 표시
%R	%H:%M 형태로 시간 표시	%T	%H:%M:%S 형태로 시간 표시
%S	초(00~60)		

형식 지정 시간 출력[3]

%n	개행	%t	탭 추가
%U	연중 주간 수 표시(00~53)	%V	ISO 8601 표준으로 연중 주간 수 표시(01~53)
%w	요일(0~6, 0을 일요일)	%W	연중 주간 수 표시(00~53), 1월 첫 월요일이 01주, 그 이전은 00주로 표시
%x	지역 시간대에 적합한 날짜 표시	%X	지역 시간대에 적합한 시간 표시
%y	년도(00~99)	%Y	네 자리 수 년도
%z	UTC와의 시차 표시	%Z	시간대명 약자

strftime 함수 사용하기

```
01 #include <time.h>
02 #include <stdio.h>
03
04 char *output[] = {
05     "%x %X",
06     "%G년, %m월 %d일 %U주 %H:%M ",
07     "%r"
08 };
09
10 int main(void) {
11     struct tm *tm;
12     int n;
13     time_t t;
14     char buf[257];
15
16     time(&t);
17     tm = localtime(&t);
18
19     for (n = 0; n < 3; n++) {
20         strftime(buf, sizeof(buf), output[n], tm);
21         printf("%s = %s\n", output[n], buf);
22     }
23
24     return 0;
25 }
```

ex4-23.out

%x %X = 01/31/09 12:43:12

%G년 %m월 %d일 %U주 %H:%M = 2009년 01월 31일 04주 12:43

%r = 12:43:12 PM

sleep ()

- 호출하는 프로세스를 지정된 시간이 경과될 때까지 또는 시그널을 받을 때까지 실행을 중지시킴

```
#include <unistd.h>
```

```
unsigned sleep (unsigned sec);
```

- rntp에서 지정한 시간이 경과되거나, 프로세스가 시그널을 받을 때까지 호출하는 프로세스의 실행을 중지

```
#include <time.h>
```

```
int nanosleep (const struct timespec *rntp, struct timespec *rmtp);
```

- ✓ nanosleep()가 시그널에 의해 unblock되고 rmtp가 NULL이 아니면 시그널 처리 후 rmtp에 저장된 남은 시간 만큼 nanosleep()가 재실행 된다.

sleep ()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int sleeptime;

    if (argc != 2) {
        fprintf(stderr, "Usage:%s nWn", argv[0]);
        return 1;
    }
    sleeptime = atoi(argv[1]);
    fprintf(stderr, "Sleep time is %dWn", sleeptime);
    for ( ; ; ) {
        sleep(sleeptime);
        printf("W007");
        fflush(stdout);
    }
}
```

예제 프로그램 – nanosleep ()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>

#define COUNT 100
#define D_BILLION 1000000000.0
#define D_MILLION 1000000.0
#define MILLION 1000000L
#define NANOSECONDS 1000

int main(void)
{
    int i;
    struct timespec slptm;
    long tdif;
    struct timeval tend, tstart;

    slptm.tv_sec = 0;
    slptm.tv_nsec = NANOSECONDS;
    if (gettimeofday(&tstart, NULL) == -1) {
        fprintf(stderr, "Failed to get start time\n");
        return 1;
    }
}
```

예제 프로그램

```
for (i = 0; i < COUNT; i++)
    if (nanosleep(&slptm, NULL) == -1) {
        perror("Failed to nanosleep");
        return 1;
    }
if (gettimeofday(&tend, NULL) == -1) {
    fprintf(stderr, "Failed to get end time\n");
    return 1;
}
tdif = MILLION*(tend.tv_sec - tstart.tv_sec) + tend.tv_usec - tstart.tv_usec;

printf("%d nanosleeps of %d nanoseconds\n", COUNT, NANOSECONDS);
printf("Should take  %11d microseconds or %f seconds\n",
        NANOSECONDS*COUNT/1000, NANOSECONDS*COUNT/D_BILLION);
printf("Actually took %11d microseconds or %f seconds\n", tdif, tdif/D_MILLION);
printf("Number of seconds per nanosleep was      %f\n", (tdif/(double)COUNT)/MILLION);
printf("Number of seconds per nanosleep should be %f\n", NANOSECONDS/D_BILLION);
return 0;
}
```

구간 타이머

□ 타이머

- ✓ 지정된 시간이 경과된 후 프로세스에게 통지
- ✓ 시간이 경과될수록 타이머 값을 감소 시켜 0이 되면 시그널 발생

□ 구간 타이머 (Interval Timer)

- ✓ 운영체제가 주기적으로 어떠한 카운터 값을 증가 시키기 위해 사용
- ✓ 주기적으로 인터럽트를 발생 할 수 있음
- ✓ 시간 공유 (Time sharing) 운영체제의 프로세스 스케줄링
 - 1 Quantum에 1 구간 타이머 작동

구간 타이머

❑ struct itimerval

```
struct itimerval {  
    struct timeval it_value; //만료까지의 시간  
    struct timeval it_interval; // 타이머 리셋 시 사용  
}
```

❑ 3개의 사용자 구간 타이머

✓ ITIMER_REAL

- 실시간으로 시간이 감소
- 시간 만료 시 SIGALRM 시그널 발생

✓ ITIMER_VIRTUAL

- 가상 시간 (프로세스가 사용한 시간)으로 시간이 감소
- 시간 만료 시 SIGVTALRM 시그널 발생

✓ ITIMER_PROF

- 실/가상 시간으로 시간 감소 모두 가능
- 시간 만료 시 SIGPROF 시그널 발생

구간 타이머 관련 함수

```
#include <sys/time.h>
```

```
/* 현재의 구간 타이머 값 획득 */
```

```
int getitimer (int which, struct itimerval *val);
```

```
/* 구간 타이머의 구동과 정지 설정 */
```

```
int setitimer (int which, const struct itimerval *restrict val, struct  
itimerval *restrict oval);
```

- ❑ which – 타이머 지시자 (ITIMER_REAL, ITIMER_VIRTUAL, ITIMER_PROF) 중 하나
- ❑ getitimer () 는 *val 변수에 타이머 시간 값 저장
- ❑ setitimer ()

setitimer ()

- ❑ *val 변수 값으로 타이머 설정
- ❑ *oval 이 NULL이 아니면 새 값이 설정되기 이전 값을 저장
 - ✓ 타이머가 실행 중이었으면 *oval 변수의 it_value 멤버는 타이머 만료까지 남은 시간 저장
- ❑ *val.it_interval 변수의 멤버가 0이 아니면 타이머 만료 후 이 값으로 재시작
- ❑ *val.it_interval이 0 이고 타이머가 구동 중이면 이 함수는 타이머를 중지

예제 프로그램

```
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>

/* ARGSUSED */
static void myhandler(int s)
{
    char aster = '*';
    int errsave;
    errsave = errno;
    write(STDERR_FILENO, &aster, 1);
    errno = errsave;
}

static int setupinterrupt(void)      /* set up myhandler for SIGPROF */
{
    struct sigaction act;
    act.sa_handler = myhandler;
    act.sa_flags = 0;
    return (sigemptyset(&act.sa_mask) || sigaction(SIGPROF, &act, NULL));
}
```


예제 프로그램

```
static int setupitimer(void)    /* set ITIMER_PROF for 2-second intervals */
{
    struct itimerval value;
    value.it_interval.tv_sec = 2;
    value.it_interval.tv_usec = 0;
    value.it_value = value.it_interval;
    return (setitimer(ITIMER_PROF, &value, NULL));
}

int main(void)
{
    if (setupinterrupt()) {
        perror("Failed to set up handler for SIGPROF");
        return 1;
    }
    if (setupitimer() == -1) {
        perror("Failed to set up the ITIMER_PROF interval timer");
        return 1;
    }
    for ( ; ; );                /* execute rest of main program here */
}
```

타이머 시간의 정밀도와 오차

- 지연 요소
 - ✓ 타이머 함수 호출 시간
 - ✓ 인터럽트, 시그널 핸들러 작동
 - ✓ 컨텍스트 스위칭

- 일반적 타이머의 정밀도 - 10msec

환경변수의 이해

□ 환경변수

- ✓ 프로세스가 실행되는 기본 환경을 설정하는 변수
 - 전역변수 사용
 - getenv () 사용
 - main 함수 인자로 받아 사용
- ✓ 로그인명, 로그인 셸, 터미널에 설정된 언어, 경로명 등
- ✓ 환경변수는 “환경변수=값”의 형태로 구성되며, 관례상 대문자 사용
- ✓ 현재 셸의 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
...
```

환경변수의 사용[1]

□ 전역변수 사용 : environ

```
#include <stdlib.h>

extern char **environ;
```

environ 전역 변수사용하기

```
01 #include <stdlib.h>
02 #include <stdio.h>
03
04 extern char **environ;
05
06 int main(void)
07 {
08     char **env;
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`
```

환경변수의 사용[2]

❑ main 함수 인자 사용

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

main 함수 인자

```
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
```

환경변수의 사용[3]

□ 환경변수 검색: getenv(3)

```
#include <stdlib.h>

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

getenv 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("SHELL = %s\n", val);
12
13     return 0;
14 }
```

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

환경변수의 사용[4]

□ 환경변수 설정: putenv(3)

```
#include <stdlib.h>

int putenv(char *string);
```

putenv 함수 사용하기

```
...
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);
17     return 0;
18 }
19 }
```

ex5_8.out

1. SHELL = /usr/bin/ksh
2. SHELL = /usr/bin/csh

설정하려는 환경변수를
“환경변수=값”형태로 지정

환경변수의 사용[5]

❑ 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, !0이면 덮어쓰고, 0이면 덮어쓰지 않음

❑ 환경변수 설정 삭제: unsetenv(3)

```
#include <stdlib.h>
```

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


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
```

프로세스 정보 - top 명령어

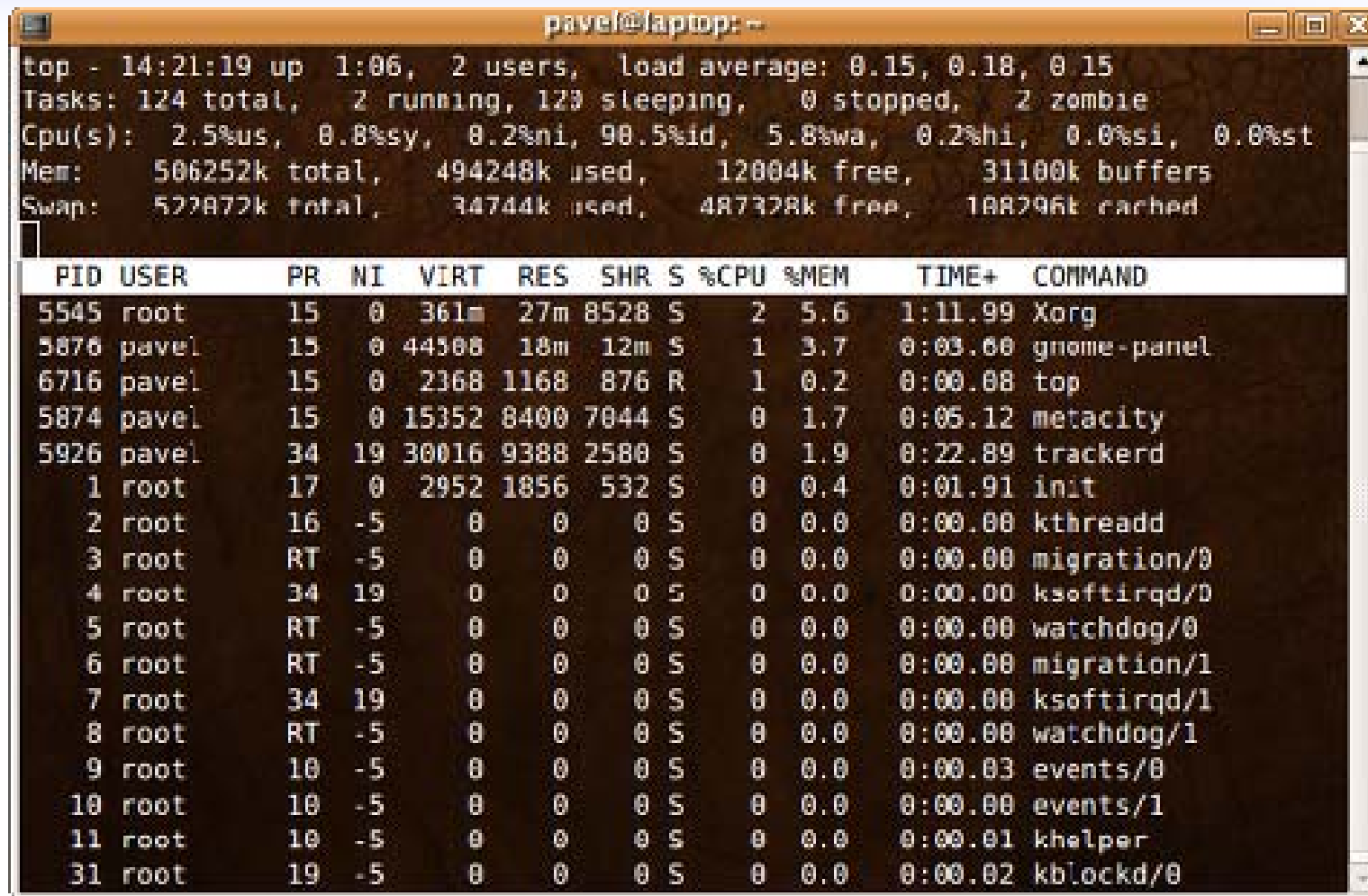
❑ top 명령 실행 시 추가할 수 있는 옵션

- * (top) -d [sec]: 설정된 초단위로 Refresh
- * (top) -c : command뒤에 인자값 표시

❑ top 명령 실행 후 사용할 수 있는 옵션

- * shift + t : 실행된 시간이 큰 순서로 정렬
- * shift + m : 메모리 사용량이 큰 순서로 정렬
- * shift + p : cpu 사용량이 큰 순서로 정렬
- * k : Process 종료
 - o k 입력 후 종료할 PID를 입력한다
 - o signal을 입력하라 표시되면 9를 넣어준다
- * c : 명령 인자 표시/비표시
- * l(소 문자엘) : uptime line(첫번째 행)을 표시/비표시
- * space bar : Refresh
- * u : 입력한 유저 소유의 Process만 표시
 - o which user: 와 같이 유저를 입력하라 표시될 때 User를 입력
 - o blank(공백) 입력 시 모두 표시
- * shift + b : 상단의 uptime 및 기타 정보값을 표시 선택
- * f : 화면에 표시될 프로세스 관련 항목 설정

프로세스 정보 - top 명령어



```
pavel@laptop: ~  
top - 14:21:19 up 1:06, 2 users, load average: 0.15, 0.18, 0.15  
Tasks: 124 total, 2 running, 123 sleeping, 0 stopped, 2 zombie  
Cpu(s): 2.5%us, 0.8%sy, 0.2%ni, 90.5%id, 5.8%wa, 0.2%hi, 0.0%si, 0.0%st  
Mem: 506252k total, 494248k used, 12004k free, 31100k buffers  
Swap: 527872k total, 34744k used, 487328k free, 108296k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5545	root	15	0	361m	27m	8528	S	2	5.6	1:11.99	Xorg
5876	pavel	15	0	44508	18m	12m	S	1	3.7	0:03.00	gnome-panel
6716	pavel	15	0	2368	1168	876	R	1	0.2	0:00.00	top
5874	pavel	15	0	15352	8400	7044	S	0	1.7	0:05.12	metacity
5926	pavel	34	19	30016	9388	2580	S	0	1.9	0:22.89	trackerd
1	root	17	0	2952	1856	532	S	0	0.4	0:01.91	init
2	root	16	-5	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	RT	-5	0	0	0	S	0	0.0	0:00.00	migration/0
4	root	34	19	0	0	0	S	0	0.0	0:00.00	ksoftirqd/0
5	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/0
6	root	RT	-5	0	0	0	S	0	0.0	0:00.00	migration/1
7	root	34	19	0	0	0	S	0	0.0	0:00.00	ksoftirqd/1
8	root	RT	-5	0	0	0	S	0	0.0	0:00.00	watchdog/1
9	root	10	-5	0	0	0	S	0	0.0	0:00.03	events/0
10	root	10	-5	0	0	0	S	0	0.0	0:00.00	events/1
11	root	10	-5	0	0	0	S	0	0.0	0:00.01	khelper
31	root	19	-5	0	0	0	S	0	0.0	0:00.02	kblockd/0

❑ 로드 에버리지(load average)란?

- 작업의 대기시간, 값이 1이라면 1분동안 평균 1개의 프로세스가 대기상태임을 나타낸다.

보통 5이면 서버가 부하를 받는다고 생각함, 10~15면 과부하

❑ Tasks: 131 total, 1 running, 130 sleeping, 0 stopped, 0 zombie
전체 프로세스 수, 현재 실행중인 프로세스, 유휴상태 프로세스, 정지상태 프로세스, 좀비 프로세스

❑ Cpu(s): 2.4%us, 0.3%sy, 0.0%ni,
97.0%id, 0.2%wa, 0.0%hi, 0.0%si, 0.0%st

사용자가 사용중인 CPU 사용률(us), 시스템이 사용하는 CPU 사용률(sy), NICE 정책에 의해 사용되는 CPU 사용률(ni), 사용되지 않은 CPU의 미사용률(id), 입출력 대기상태의 사용률(wa)

❑ Mem: 8140668k total, 7900820k used, 239848k free, 3074544k buffers

전체 물리적인 메모리, 사용중인 메모리(used), 사용되지 않는 여유 메모리(free), 버퍼된 메모리(buffers)

□ 세부 정보 필드별 항목

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

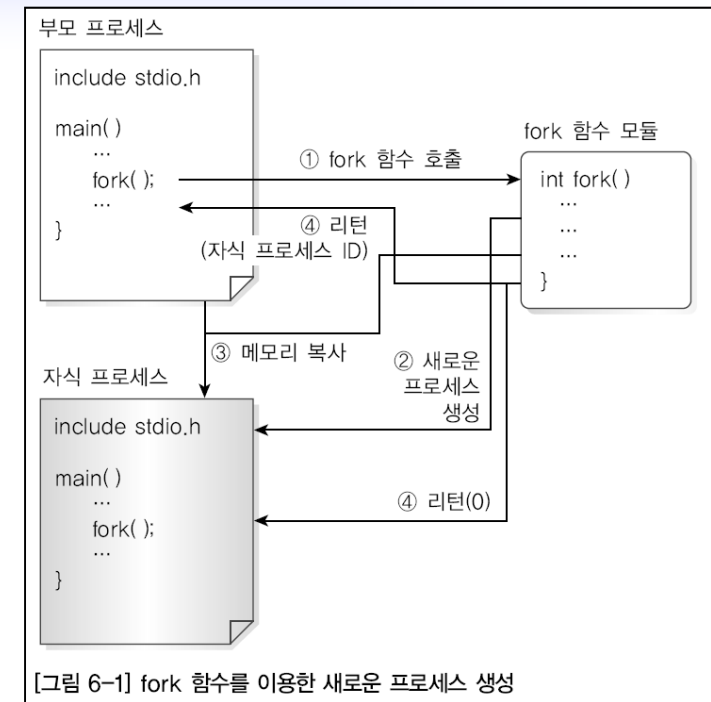
- * PID : 프로세스 ID (PID)
- * USER : 프로세스를 실행시킨 사용자 ID
- * PRI : 프로세스의 우선순위 (priority)
- * NI : NICE 값. 일의 nice value값. 마이너스를 가지는 nice value는 우선순위가 높음.
- * VIRT : 가상 메모리의 사용량(SWAP+RES)
- * RES : 현재 페이지가 상주하고 있는 크기(Resident Size)
- * SHR : 분할된 페이지, 프로세스에 의해 사용된 메모리를 나눈 메모리의 총합.
- * S : 프로세스의 상태 [S(sleeping), R(running), W(swapped out process), Z(zombies)]
- * %CPU : 프로세스가 사용하는 CPU의 사용율
- * %MEM : 프로세스가 사용하는 메모리의 사용율
- * COMMAND : 실행된 명령어

프로세스 생성

❑ fork(2)

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

- ✓ 새로운 프로세스를 생성 : 자식 프로세스
- ✓ fork 함수를 호출한 프로세스 : 부모 프로세스
- ✓ 자식 프로세스는 부모 프로세스의 메모리를 복사
 - RUID, EUID, RGID, EGID, 환경변수
 - 열린 파일기술폰자, 시그널 처리, setuid, setgid
 - 현재 작업 디렉토리, umask, 사용가능자원 제한
- ✓ 부모 프로세스와 다른 점
 - 자식 프로세스는 유일한 PID를 갖는다
 - 자식 프로세스는 유일한 PPID를 갖는다.
 - 부모 프로세스가 설정한 프로세스잠금, 파일 잠금, 기타 메모리 잠금은 상속 안함
 - 자식 프로세스의 tms구조체 값은 0으로 설정
- ✓ 부모 프로세스와 자식 프로세스는 열린 파일을 공유하므로 읽거나 쓸 때 주의해야 한다.



fork 함수 사용하기

```
...
06 int main(void) {
07     pid_t pid;
08
09     switch (pid = fork()) {
10         case -1 : /* fork failed */
11             perror("fork");
12             exit(1);
13             break;
14         case 0 : /* child process */
15             printf("Child Process - My PID:%d, My Parent's PID:%d\n",
16                 (int)getpid(), (int)getppid());
17             break;
18         default : /* parent process */
19             printf("Parent process - My PID:%d, My Parent's PID:%d, "
20                 "My Child's PID:%d\n", (int)getpid(), (int)getppid(),
21                 (int)pid);
22             break;
23     }
24     printf("End of fork\n");
25     return 0;
26 }
27 }
```

fork함수의 리턴값 0은
자식 프로세스가 실행

ex6_2.out

Child Process - My PID:796, My Parent's PID:795

End of fork

Parent process - My PID:795, My Parent's PID:695, My Child's PID:796

End of fork

프로세스 종료 함수[1]

❑ 프로그램 종료: exit(2)

```
#include <stdlib.h>
void exit(int status);
```

✓ status : 종료 상태값

❑ 프로그램 종료시 수행할 작업 예약: atexit(2)

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

✓ func : 종료시 수행할 작업을 지정한 함수명

❑ 프로그램 종료: _exit(2)

```
#include <unistd.h>
void _exit(int status);
```

✓ 일반적으로 프로그램에서 직접 사용하지 않고 exit 함수 내부적으로 호출

프로세스 종료 함수[2]

- 프로그램 종료 함수의 일반적 종료 절차
 1. 모든 파일 기술자를 닫는다.
 2. 부모 프로세스에 종료 상태를 알린다.
 3. 자식 프로세스들에 SIGHUP 시그널을 보낸다.
 4. 부모 프로세스에 SIGCHLD 시그널을 보낸다.
 5. 프로세스간 통신에 사용한 자원을 반납한다.

exit, atexit 함수 사용하기

```
01 #include <stdlib.h>
02 #include <stdio.h>
03
04 void cleanup1(void) {
05     printf("Cleanup 1 is called.\n");
06 }
07
08 void cleanup2(void) {
09     printf("Cleanup 2 is called.\n");
10 }
11
12 int main(void) {
13     atexit(cleanup1);
14     atexit(cleanup2);
15
16     exit(0);
17 }
```

종료시 수행할 함수 지정
지정한 순서의 역순으로 실행(실행결과 확인)

ex6_3.out

Cleanup 2 is called.

Cleanup 1 is called.

exec 함수군 활용

❑ exec 함수군

- ✓ exec로 시작하는 함수들로, 명령이나 실행 파일을 실행
- ✓ exec 함수가 실행되면 프로세스의 메모리 이미지는 실행파일로 바뀜

❑ exec 함수군의 형태 6가지

```
#include <unistd.h>
int execl(const char *path, const char *arg0, ..., const char *argn, (char *)0);
int execv(const char *path, char *const argv[]);
int execl(const char *path, const char *arg0, ..., const char *argn,
(char *)0, char *const envp[]);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ..., const char *argn, (char *)0);
int execvp(const char *file, char *const argv[]);
```

- ✓ path : 명령의 경로 지정
- ✓ file : 실행 파일명 지정
- ✓ arg#, argv : main 함수에 전달할 인자 지정
- ✓ envp : main 함수에 전달할 환경변수 지정
- ✓ 함수의 형태에 따라 NULL 값 지정에 주의해야 한다.

execvp 함수 사용하기

```
01 #include <unistd.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main(void) {
06     printf("--> Before exec function\n");
07
08     if (execvp("ls", "ls", "-a", (char *)NULL) == -1) {
09         perror("execvp");
10         exit(1);
11     }
12
13     printf("--> After exec function\n");
14
15     return 0;
16 }
```

인자의 끝을 표시하는 NULL 포인터

첫 인자는 관례적으로 실행파일명 지정

메모리 이미지가 'ls' 명령으로 바뀌어 13행은 실행안됨

```
# ex6_4.out
--> Before exec function
.      ex6_1.c      ex6_3.c      ex6_4.out
..     ex6_2.c      ex6_4.c      han.txt
```

execv 함수 사용하기

```
01 #include <unistd.h>
02 #include <stdlib.h>
03 #include <stdio.h>
04
05 int main(void) {
06     char *argv[3];
07
08     printf("Before exec function\n");
09
10     argv[0] = "ls";
11     argv[1] = "-a";
12     argv[2] = NULL;
13     if (execv("/usr/bin/ls", argv) == -1) {
14         perror("execv");
15         exit(1);
16     }
17
18     printf("After exec function\n");
19
20     return 0;
21 }
```

첫 인자는 관례적으로 실행파일명 지정

인자의 끝을 표시하는 NULL 포인터

경로로 명령 지정

역시 실행안 됨

ex6_5.out

--> Before exec function

. ex6_1.c ex6_3.c ex6_5.c han.txt

.. ex6_2.c ex6_4.c ex6_5.out

execve 함수 사용하기

```
...
05 int main(void) {
06     char *argv[3];
07     char *envp[2];
08
09     printf("Before exec function\n");
10
11     argv[0] = "arg.out";
12     argv[1] = "100";
13     argv[2] = NULL;
14
15     envp[0] = "MYENV=hanbit";
16     envp[1] = NULL;
17
18     if (execve("./arg.out", argv, envp) == -1) {
19         perror("execve");
20         exit(1);
21     }
22
23     printf("After exec function\n");
24
25     return 0;
26 }
```

실행파일명 지정

인자의 끝을 표시하는 NULL 포인터

환경변수 설정

ex6_6_arg.c를 컴파일하여 생성

```
# ex6_6.out
--> Before exec function
--> In ex6_6_arg.c Main
argc = 2
argv[0] = arg.out
argv[1] = 100
MYENV=hanbit
```

ex6_6_arg.c 파일

```
01  #include <stdio.h>
02
03  int main(int argc, char **argv, char **envp) {
04      int n;
05      char **env;
06
07      printf("\n--> In ex6_6_arg.c Main\n");
08      printf("argc = %d\n", argc);
09      for (n = 0; n < argc; n++)
10          printf("argv[%d] = %s\n", n, argv[n]);
11
12      env = envp;
13      while (*env) {
14          printf("%s\n", *env);
15          env++;
16      }
17
18      return 0;
19  }
```

인자 값 출력

환경변수 출력

exec 함수군과 fork 함수

- ❑ fork로 생성한 자식 프로세스에서 exec 함수군을 호출
 - ✓ 자식 프로세스의 메모리 이미지가 부모 프로세스 이미지에서 exec 함수로 호출한 새로운 명령으로 대체
 - ✓ 자식 프로세스는 부모 프로세스와 다른 프로그램 실행 가능
 - ✓ 부모 프로세스와 자식 프로세스가 각기 다른 작업을 수행해야 할 때 fork와 exec 함수를 함께 사용

fork와 exec 함수 사용하기

```
...
06 int main(void) {
07     pid_t pid;
08
09     switch (pid = fork()) {
10         case -1 : /* fork failed */
11             perror("fork");
12             exit(1);
13             break;
14         case 0 : /* child process */
15             printf("--> Child Process\n");
16             if (execlp("ls", "ls", "-a", (char *)NULL) == -1) {
17                 perror("execlp");
18                 exit(1);
19             }
20             exit(0);
21             break;
22         default : /* parent process */
23             printf("--> Parent process - My PID:%d\n", (int)getpid());
24             break;
25     }
27     return 0;
28 }
```

자식프로세스에서 execlp 함수 실행

부모프로세스는 이 부분 실행

```
# ex6_7.out
--> Child Process
ex6_1.c  ex6_3.c  ex6_5.c  ex6_6_arg.c  ex6_7.out
ex6_2.c  ex6_4.c  ex6_6.c  ex6_7.c      han.txt
--> Parent process - My PID:10535
```