

디렉토리 정보 검색[1]

□ 디렉토리 열기: opendir(3)

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *dirname);
```

- 성공하면 열린 디렉토리를 가리키는 DIR 포인터를 리턴

□ 디렉토리 닫기: closedir(3)

```
#include <sys/types.h>
#include <dirent.h>
int closedir(DIR *dirp);
```

□ 디렉토리 정보 읽기: readdir(3)


```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
```

- 디렉토리의 내용을 한 번에 하나씩 읽어옴

```
typedef struct dirent {
    ino_t      d_ino;
    off_t      d_off;
    unsigned short d_reclen;
    char       d_name[1];
} dirent_t;
```

이름의 최대 크기는
255지만, 배열의 크기가
1인 이유는 필요한 메모리
공간을 최대한 작게 하기
위해서

[예제 3-15] 디렉토리 열고 정보 읽기 (test1.c)

```
01  #include <dirent.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      DIR *dp;
07      struct dirent *dent;
08
09      if ((dp = opendir("hanbit")) == NULL) {
10          perror("opendir: hanbit");
11          exit(1);
12      }
13      
14      while (((dent = readdir(dp)))) {
15          printf("Name : %s  ", dent-<u>d_name
```

ex3_15.out

Name : . Inode : 208

Name : .. Inode : 189

[예제 3-16] 디렉토리 항목의 상세 정보 검색하기 (test2.c)

```
01 #include <sys/types.h>
02 #include <sys/stat.h>
03 #include <dirent.h>
04 #include <stdlib.h>
05 #include <stdio.h>
06
07 int main(void) {
08     DIR *dp;
09     struct dirent *dent;
10     struct stat sbuf;
11     char path[BUFSIZ];
12
13     if ((dp = opendir("hanbit")) == NULL) {
14         perror("opendir: hanbit");
15         exit(1);
16     }
17
18     while ((dent = readdir(dp))) {
19         if (dent->d name[0] == '.') continue;
20         else break;
21     }
22 }
```

'.', '..' 을 건너뛰기 위해서



[예제 3-16] 디렉토리 항목의 상세 정보 검색하기

```
23     sprintf(path, "hanbit/%s", dent->d_name);  
24     stat(path, &sbuf);  
25  
26     printf("Name : %s\n", dent->d_name);  
27     printf("Inode(dirent) : %d\n", (int)dent->d_ino);  
28     printf("Inode(stat) : %d\n", (int)sbuf.st_ino);  
29     printf("Mode : %o\n", (unsigned int)sbuf.st_mode);  
30     printf("Uid : %d\n", (int)sbuf.st_uid);  
31  
32     closedir(dp);  
33  
34     return 0;  
35 }
```

디렉토리의 항목을 읽고
다시 stat 함수로 상세 정보 검색

```
# ls -ai hanbit  
208 .    189 ..    213 han.c  
# ex3_16.out  
Name : han.c  
Inode(dirent) : 213  
Inode(stat) : 213  
Mode : 100644  
Uid : 0
```

디렉토리 정보 검색[2]

□ 디렉토리 오프셋: telldir(3), seekdir(3), rewinddir(3)

```
#include <dirent.h>
long telldir(DIR *dirp);
void seekdir(DIR *dirp, long loc);
void rewinddir(DIR *dirp);
```

- **telldir** : 디렉토리 오프셋의 현재 위치를 알려준다.
- **seekdir** : 디렉토리 오프셋을 loc에 지정한 위치로 이동시킨다.
- **rewinddir** : 디렉토리 오프셋을 디렉토리의 시작인 0으로 이동시킨다.



[예제 3-17] 디렉토리 오프셋 변화 확인하기

```
01 #include <sys/stat.h>
02 #include <dirent.h>
03 #include <stdlib.h>
04 #include <stdio.h>
05
06 int main(void) {
07     DIR *dp;
08     struct dirent *dent;
09
10     if ((dp = opendir("hanbit")) == NULL) {
11         perror("opendir");
12         exit(1);
13     }
14
15     printf("** Directory content **\n");
16     printf("Start Offset : %ld\n", tellidir(dp));
17     while ((dent = readdir(dp))) {
18         printf("Read : %s  ", dent->d_name);
19         printf("Cur Offset : %ld\n", tellidir(dp));
20     }
21
22     printf("** Directory Pointer Rewind **\n");
23     rewinddir(dp);
24     printf("Cur Offset : %ld\n", tellidir(dp));
```

[예제 3-17] 디렉토리 오프셋 변화 확인하기

```
25
26     printf("** Move Directory Pointer **\n");
27     seekdir(dp, 24);
28     printf("Cur Offset : %ld\n", telldir(dp));
29
30     dent = readdir(dp);
31     printf("Read %s ", dent->d_name);
32     printf("Next Offset : %ld\n", telldir(dp));
33
34     closedir(dp);
35     return(0);
36
37 }
```

cp a.c /tmp

```
# ex3_17.out
** Directory content
Start Offset : 0
Read : .
Cur Offset : 12

Read : ..
Cur Offset : 24

Read : han.c
Cur Offset : 512

**      Directory      Pointer
Rewind **
Cur Offset : 0
**      Move      Directory
Pointer **
Cur Offset : 24
Read han.c
Next Offset : 512
```

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); 함수를 통해 정보 추출 가능.
```

▪ 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 { 다 제시해줄 것임.  
    char sysname[_SYS_NMLN];  
    char nodename[_SYS_NMLN];  
    char release[_SYS_NMLN];  
    char version[_SYS_NMLN];  
    char machine[_SYS_NMLN];  
};
```


[예제 4-1] uname 함수 사용하기 (test3.c)

```
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 자원 정보 검색[3]

□ 파일과 디렉토리 관련 자원 검색 : fpathconf(3), pathconf(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
<u>_PC_LINK_MAX(1)</u>	디렉토리 혹은 파일 하나에 가능한 최대 링크 수를 나타낸다.
<u>_PC_NAME_MAX(4)</u>	파일명의 최대 길이를 바이트 크기로 나타낸다.
<u>_PC_PATH_MAX(5)</u>	경로명의 최대 길이를 바이트 크기로 나타낸다.



[예제 4-5] pathconf 함수 사용하기 (test4.c)

```
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);
```

[예제 4-6] getuid, geteuid 함수 사용하기

ex4_6.c

```
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     return 0;
18 }
```

유효유저아이디. 최대 단계까지 내려감.

일시적으로 파일 소유자의 권한을 얻게 됨.

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

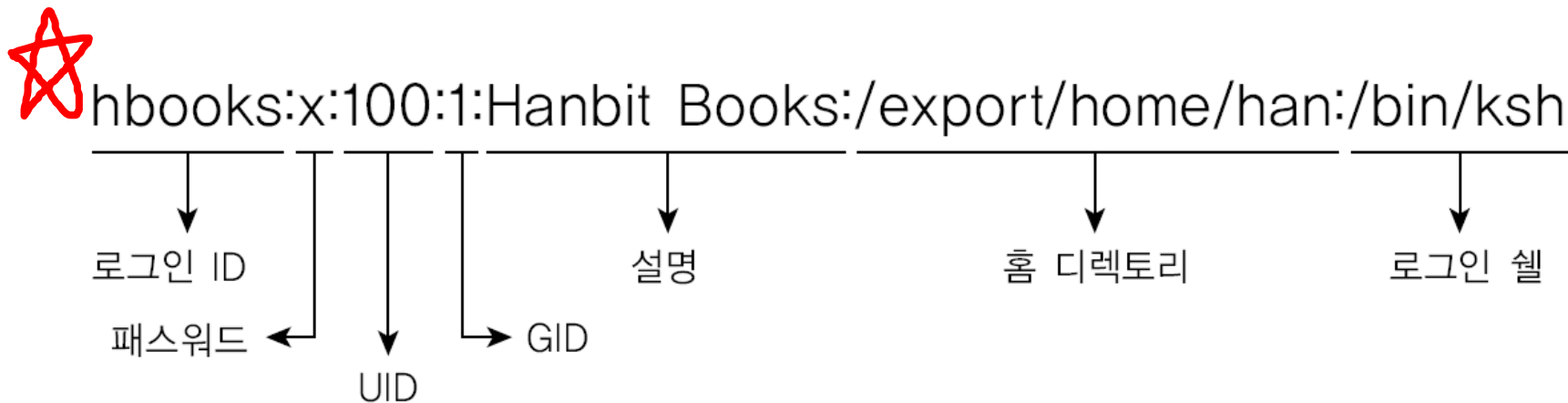
5964 1월 29일 15:11 ex4_6.out

setuid 설정 후 일반사용자가 이 파일을 실행하면?

패스워드 파일 검색[1]

□ /etc/passwd 파일의 구조

```
# 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

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);
```

[예제 4-12] `getid`, `getegid` 함수 사용하기

ex4_12.c

```
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,bm,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;
};
```



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);
```

[예제 4-16] time 함수 사용하기 (test5.c)

```
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
```



시간 관리 함수[2]

□ 마이크로 초 단위로 시간 정보얻기 : gettimeofday(3)

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

초 단위로 재면 너무 빠르기 때문에 0 이 나올 가능성이 높음.

```
int gettimeofday(struct timeval *tp, void *tzp);  
int settimeofday(struct timeval *tp, void *tzp);
```

▪ timeval 구조체

```
struct timeval {  
    time_t      tv_sec; /* 초 */  
    suseconds_t tv_usec; /* 마이크로 초 */  
};
```

[예제 4-17] gettimeofday 함수 사용하기 (test6.c)

```
..  
04 int main(void) {  
05     struct timeval tv;  
06  
07     gettimeofday(&tv, NULL);  
08     printf("Time(sec) : %d\n", (int)tv.tv_sec);  
09     printf("Time(micro-sec) : %d\n", (int)tv.tv_usec);  
10  
11     return 0;  
12 }
```

```
# ex4_17.out  
Time(sec) : 1233362365  
Time(micro-sec) : 670913
```

시간의 형태 변환

글로벌타임

□ 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, gmtime은 UTC기준, localtime은 지역시간대 기준

□ 초 단위 시간으로 역산 : mktime(3)

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

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

- tm구조체

```
struct tm {  
    int tm_sec;  
    int tm_min;  
    int tm_hour;  
    int tm_mday;  
    int tm_mon;  
    int tm_year;  
    int tm_wday;  
    int tm_yday;  
    int tm_isdst;  
};
```

tm_mon(월) : 0(1월)~11(12월)
tm_year(연도) : 년도 - 1900
tm_wday(요일) : 0(일)~6(토)
tm_isdst(일광절약제) : 1(시행)



[예제 4-19] gmtime, localtime 함수 사용하기 (test7.c)

```
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);
```

[예제 4-19] 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? 2009 년을 의미.
어떻게 해석해야하나?



형식 지정 시간 출력[1]

□ 초 단위 시간을 변환해 출력하기: ctime(3)

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

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

[예제 4-21] ctime 함수 사용하기 (test8.c)

```
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
```

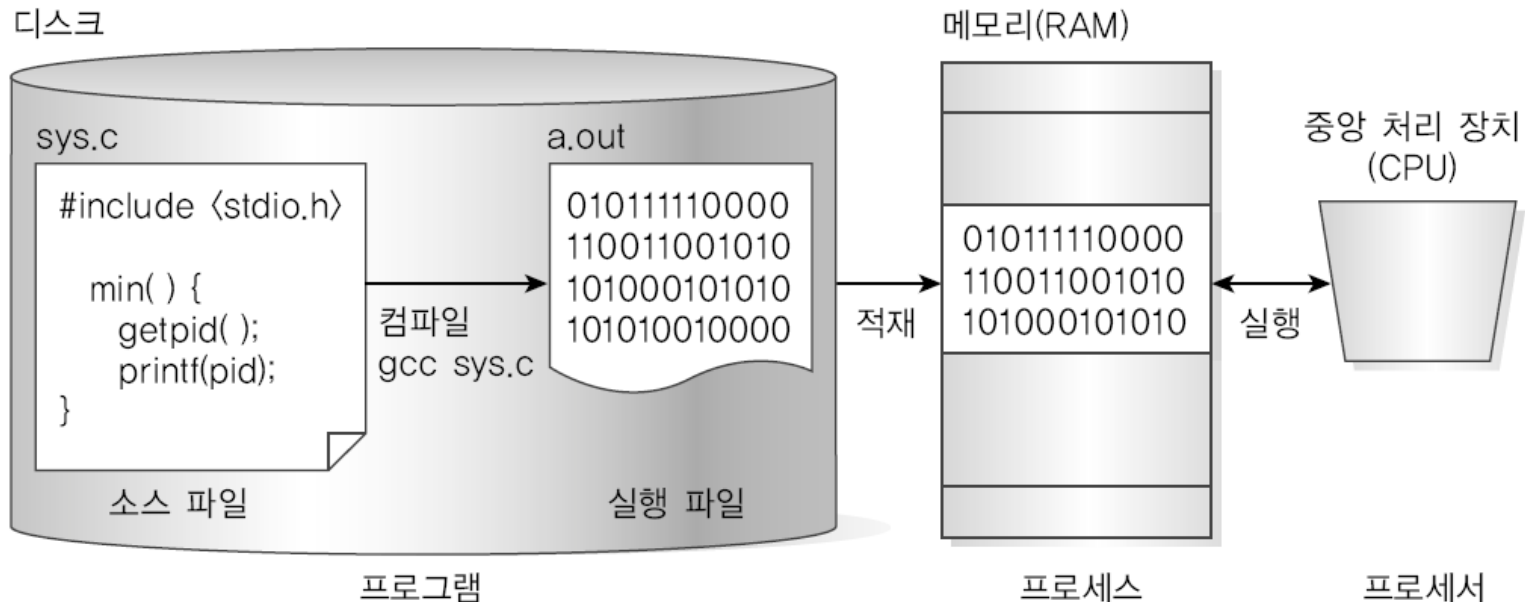
```
Time(sec) : 1233370759
```

```
Time(date) : Sat Jan 31 11:59:19 2009
```

프로세스의 정의

□ 프로세스

- 실행중인 프로그램을 의미
 - 프로세서(processor) : 중앙처리장치(예: 펜티엄, 쿼드코어 등)
 - 프로그램(program) : 사용자가 컴퓨터에 작업을 시키기 위한 명령어의 집합
- 고급언어로 작성한 프로그램은 기계어 프로그램으로 변환해야 실행이 가능

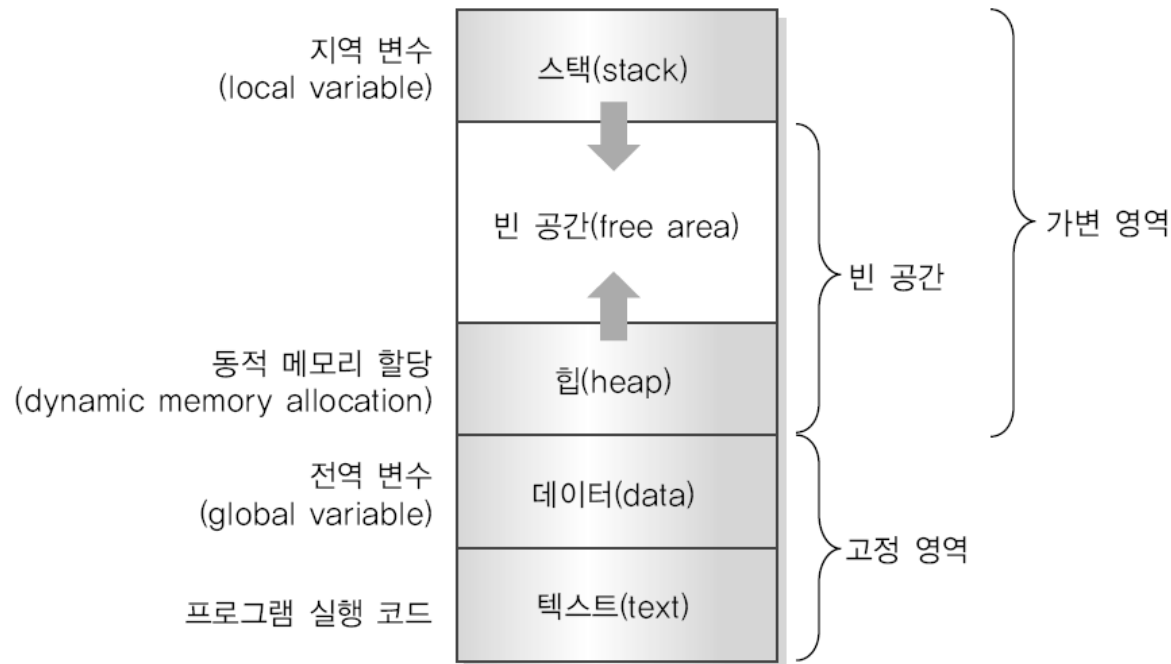


[그림 5-1] 프로그램, 프로세스, 프로세서의 관계



프로세스의 구조

□ 메모리에 적재된 프로세스의 구조



[그림 5-2] 프로세스의 기본 구조

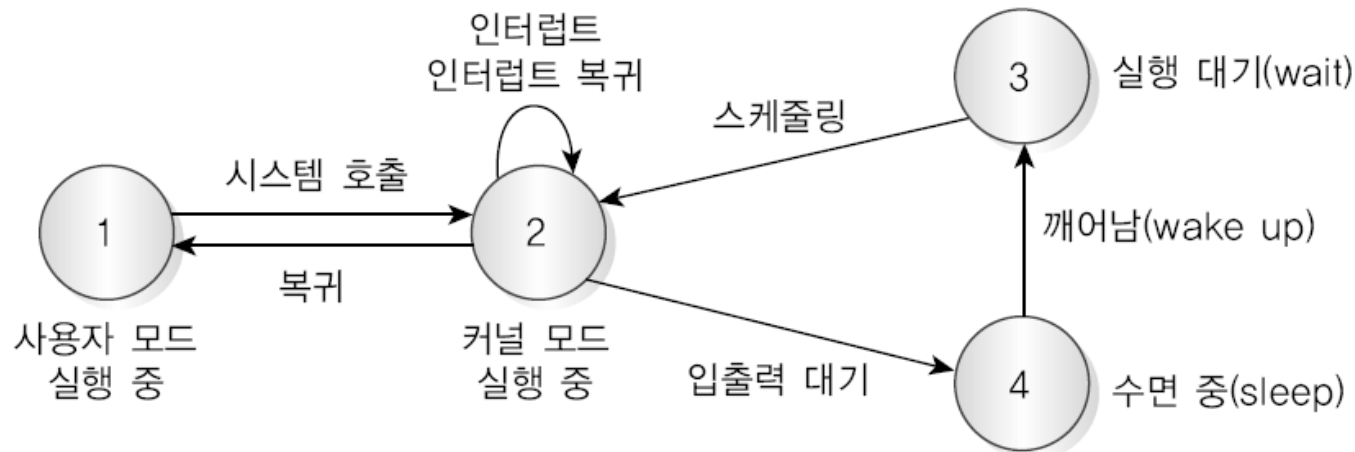
- **텍스트 영역** : 실행 코드 저장
- **데이터 영역** : 전역 변수 저장
- **힙** : 동적메모리 할당을 위한 영역
- **스택** : 지역변수를 저장하는 영역



프로세스 상태 변화

□ 프로세스의 상태는 규칙에 따라 여러 상태로 변함

- 커널의 프로세스 관리 기능이 프로세스의 스케줄링 담당



[그림 5-3] 프로세스의 상태 및 전이

1. 프로세스는 먼저 사용자 모드에서 실행
2. 사용자모드에서 시스템 호출을 하면 커널 모드로 전환
3. 수면 중이던 프로세스가 깨어나 실행 대기 상태로 전환되면 실행 준비
4. 커널 모드에서 실행 중 입출력을 기다릴 때처럼 실행을 계속할 수 없으면 수면상태로 전환



프로세스 목록 보기

□ 현재 실행중인 프로세스 목록을 보려면 ps 명령 사용

```
# ps
  PID TTY          TIME CMD
  678 pts/3        0:00 ksh
 1766 pts/3        0:00 ps
```

■ 전체 프로세스를 보려면 -ef 옵션 사용

```
# ps -ef | more
  UID    PID  PPID    C   STIME     TTY   TIME   CMD
  root      0      0    0   1월 30일 ?    175:28 sched
  root      1      0    0   1월 30일 ?      0:02  /sbin/init
  root      2      0    0   1월 30일 ?      0:00  pageout
  .....
```

□ 현재 실행중인 프로세스를 주기적으로 확인

- 솔라리스 기본 명령 : prstat, sdtprocess
- 공개소프트웨어 : top



프로세스 식별

□ PID 검색: getpid(2)

```
#include <unistd.h>
pid_t getpid(void);
```

- 이 함수를 호출한 프로세스의 PID를 리턴

□ PPID 검색 : getppid(2)

```
#include <unistd.h>
pid_t getppid(void);
```

- 부모 프로세스의 PID를 리턴

```
# ps -ef | more
  UID  PID  PPID  C   STIME  TTY  TIME  CMD
root    0     0   0   1월 30일 ?    175:28 sched
root    1     0   0   1월 30일 ?     0:02 /sbin/init
root    2     0   0   1월 30일 ?     0:00 pageout
.....
```

부모 프로세스ID

[예제 5-1] getpid, getppid 함수 사용하기(test9.c)

```
01 #include <unistd.h>
02 #include <stdio.h>
03
04 int main(void) {
05     printf("PID : %d\n", (int)getpid());
06     printf("PPID : %d\n", (int)getppid());
07
08     return 0;
09 }
```

678 프로세스는
콘솔

ex5_1.out

PID : 2205

PPID : 678

```
# ps
PID TTY          TIME CMD
678 pts/3        0:00 ksh
2206 pts/3       0:00 ps
```



프로세스 그룹

□ 프로세스 그룹

- 관련 있는 프로세스를 묶은 것으로 프로세스 그룹ID(PGID)가 부여됨
- 작업제어 기능을 제공하는 C셸이나 콘셸은 명령을 파이프로 연결하여 프로세스 그룹 생성 가능

□ 프로세스 그룹 리더

- 프로세스 그룹을 구성하는 프로세스 중 하나가 그룹 리더가 됨
- 프로세스 그룹 리더의 PID가 PGID
- 프로세스 그룹 리더는 변경 가능

□ PGID 검색 : getpgrp(2), getpgid(2)

getpgid는 pid 인자로 지정된 프로세스의 그룹id를 return
getpgrp의 인자가 0이면 getpgid함수를 호출한 프로세스가
속한 그룹의 pgid를 return

```
#include <unistd.h>

pid_t getpgrp(void);
pid_t getpgid(pid_t pid);
```

□ PGID 변경: setpgid(2)

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

int setpgid(pid_t pid, pid_t pgid);
```



[예제 5-2] getpgrp, getpgid 함수 사용하기 (test10.c)

```
01 #include <unistd.h>
02 #include <stdio.h>
03
04 int main(void) {
05     printf("PID : %d\n", (int)getpid());
06     printf("PGRP : %d\n", (int)getpgrp());
07     printf("PGID(0) : %d\n", (int)getpgid(0));
08     printf("PGID(2287) : %d\n", (int)getpgid(2287));
09
10     return 0;
11 }
```

실행방법
2287은 sleep의 PID

```
# ex5_2.out
PID : 2297
PGRP : 2297
PGID(0) : 2297
PGID(2287) : 2285
```

```
$ ps -ef | more | sleep 300 &
$ ps
PID TTY          TIME CMD
2278 pts/6        0:00 ksh
2301 pts/6        0:00 ps
2287 pts/6        0:00 sleep
```

환경변수의 이해

□ 환경변수

- 프로세스가 실행되는 기본 환경을 설정하는 변수
- 로그인명, 로그인 셸, 터미널에 설정된 언어, 경로명 등
- 환경변수는 “환경변수=값” 의 형태로 구성되며 관례적으로 대문자로 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;
```

[예제 5-5] environ 전역 변수사용하기 (test11.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`
```


환경변수의 사용[2]

□ main 함수 인자 사용

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

[예제 5-6] main 함수 인자 (test12.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
```

환경변수의 사용[3]

□ 환경변수 검색: getenv(3)

```
#include <stdlib.h>

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

[예제 5-7] getenv 함수 사용하기 (test13.c)

```
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);
```

[예제 5-8] putenv 함수 사용하기 (test14.c)

```
...
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, 0이 아니면 덮어쓰고, 0이면 덮어쓰지 않음

□ 환경변수 설정 삭제: `unsetenv(3)`

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

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



[예제 5-9] setenv 함수 사용하기(test15.c)

```
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
```