

5프로세스 정보

학습목표

- 유닉스 시스템에서 프로세스가 무엇인지 이해한다.
- 함수를 사용해 프로세스의 속성을 검색할 수 있다.
- 프로세스의 실행시간을 측정할 수 있다.
- 환경변수를 설정하고 사용할 수 있다.



목차

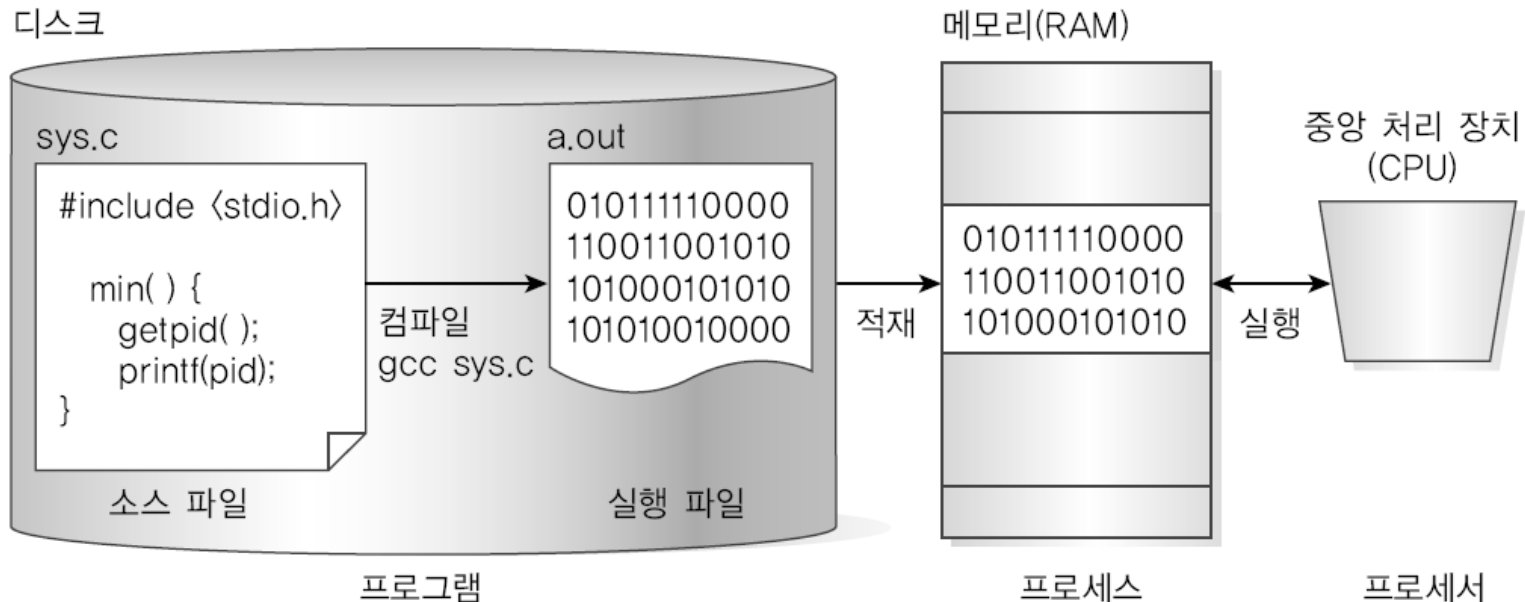
- 프로세스의 정의
- 프로세스의 구조와 상태변화
- 프로세스 목록 보기
- 프로세스 식별 함수
- 프로세스 그룹 및 세션
- 프로세스 실행시간 측정
- 환경변수의 이해
- 환경변수의 사용



프로세스의 정의

□ 프로세스

- 실행 중인 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

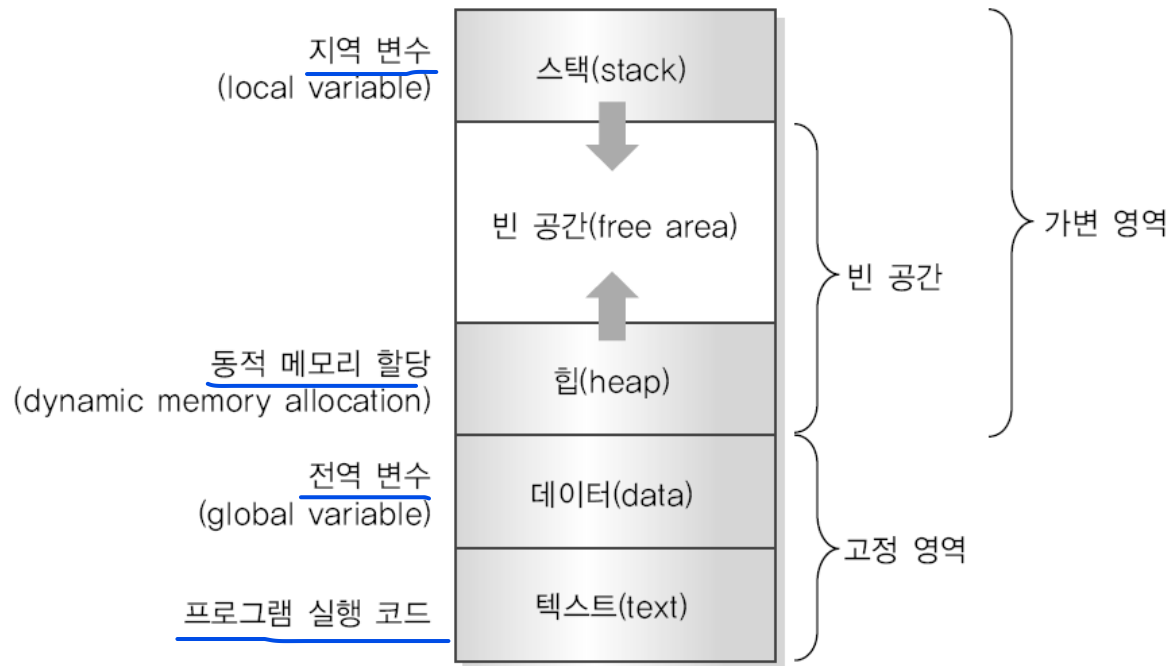


[그림 5-1] 프로그램, 프로세스, 프로세서의 관계



프로세스의 구조

□ 메모리에 적재된 프로세스의 구조



[그림 5-2] 프로세스의 기본 구조

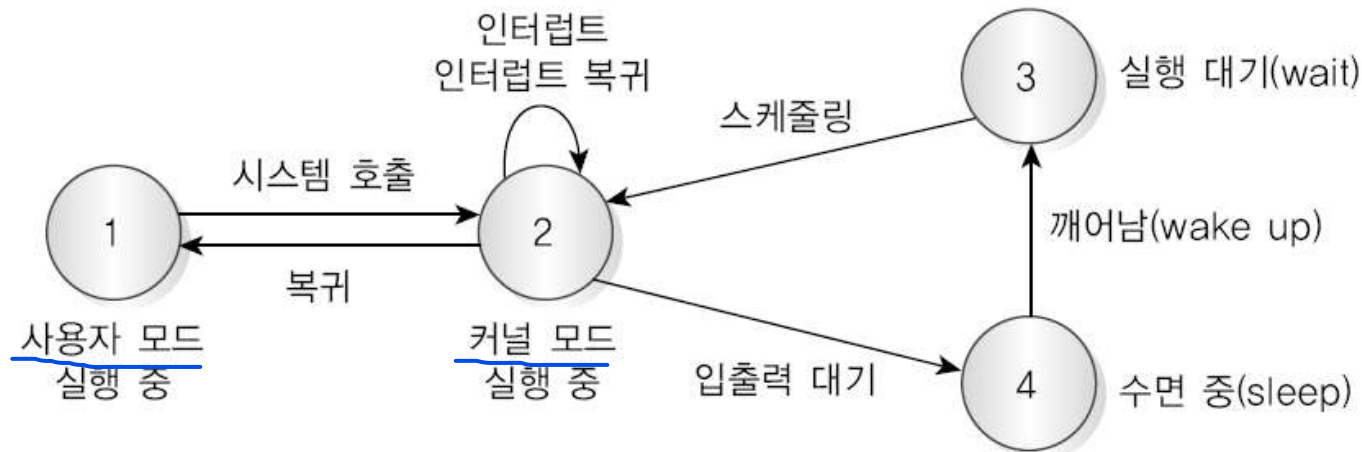
- 텍스트 영역 : 실행 코드 저장
- 데이터 영역 : 전역 변수 저장
- 힙 : 동적메모리 할당을 위한 영역
- 스택 : 지역변수를 저장하는 영역



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
  ..... 다음 페이지로 넘어가려면 스페이스 바
```

- 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

ex5_1.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 }
```

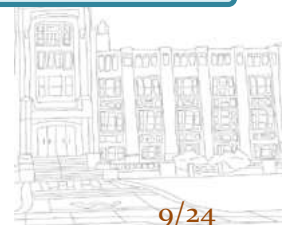
678 프로세스는
콘솔

ex5_1.out

PID : 2205

PPID : 678

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

pid를 가진 프로세스의 그룹 ID는?



```
#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);
```

```

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 }

```

```

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

```

실행방법
2287은 sleep의 PID

```

$ ps -ef | more | sleep 300 &
$ ps

```

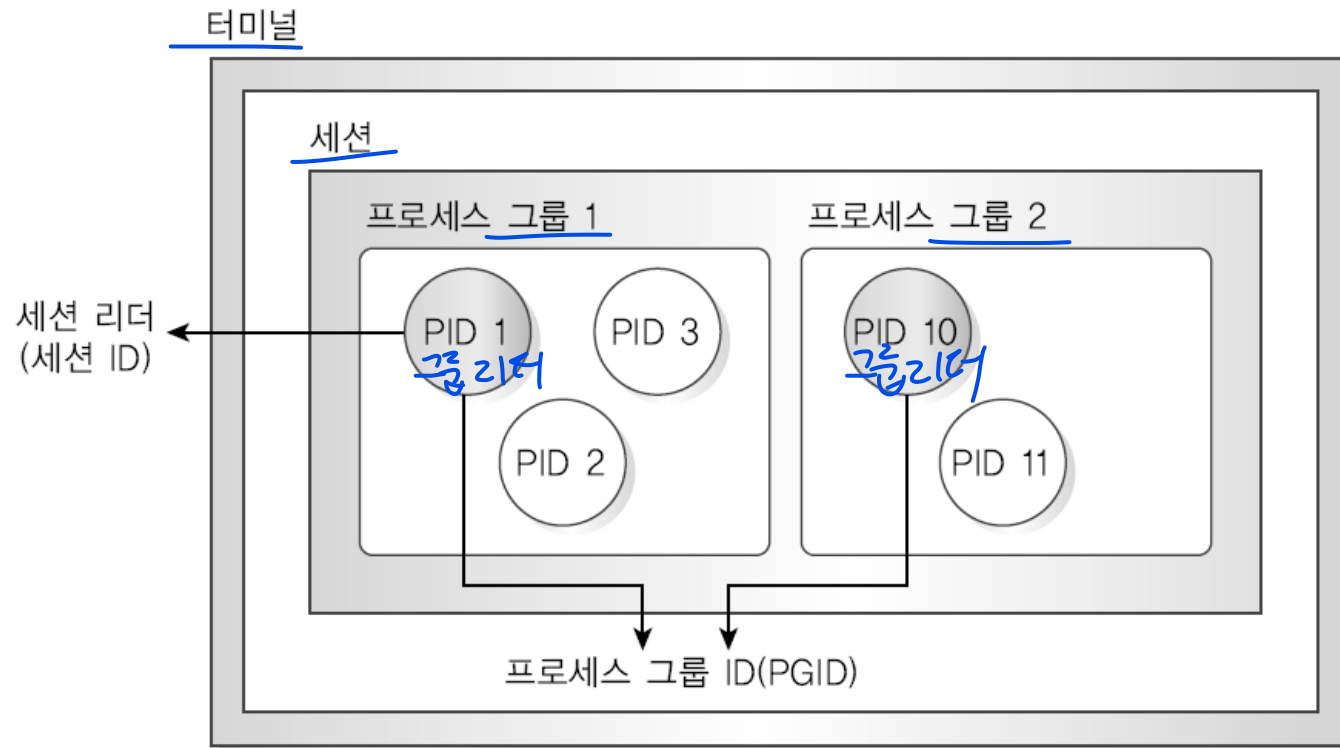
진짜 5분 기다리기 싫으면 세그널

PID	TTY	TIME	CMD
2278	pts/6	0:00	ksh
2301	pts/6	0:00	ps
2287	pts/6	0:00	sleep

세션[1]

□ 세션

- POSIX 표준에서 제안한 개념
- 사용자가 로그인해 작업하고 있는 터미널 단위로 프로세스 그룹을 묶은 것



[그림 5-4] 프로세스, 프로세스 그룹, 세션의 관계



세션 [2]

□ 세션 검색: getsid(2) *세션 번호 확인*

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

```
pid_t getsid(pid_t pid);
```

- 세션ID는 SVR4에서 정의한 개념
- 새로운 세션을 생성하면 해당 프로세스는 세션 리더가 되면 세션 리더의 PID가 세션ID
bash 쉘을 수행한 프로세스가 default로 세션리더가 됨

□ 세션 생성: setsid(2)

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

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

```
pid_t setsid(void);
```



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

ex5_3.out

PID : 2584

PGID : 2584

SID : 678

ps

PID	TTY	TIME	CMD
-----	-----	------	-----

<u>678</u>	pts/3	0:01	ksh
------------	-------	------	-----

2585	pts/3	0:00	ps
------	-------	------	----



프로세스 실행 시간 측정

□ 프로세스 실행 시간의 구성

프로세스 실행시간 = 시스템 실행시간 + 사용자 실행시간

커널 모드 사용자 모드

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

□ 프로세스 실행 시간 측정

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

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

- 사용자 실행시간과 시스템 실행시간으로 나누어 tms 구조체에 저장
- 시간 단위는 클록틱(sysconf 함수에서 _SC_CLK_TCK로 검색한 값)
- tms 구조체

클록틱이라서
100% 정확하진
않다

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

utime : 사용자 모드 실행시간
stime : 시스템 모드 실행시간
cutime : 자식프로세스의 사용자 모드 실행시간
cstime : 자식프로세스의 시스템 모드 실행시간

[예제 5-4] times 함수 사용하기

ex5_4.c

안되시면
<time.h>

```
<sys/types.h><sys/times.h><limits.h><unistd.h><stdlib.h><stdio.h>
```

```
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) { +1: 시작 시간
15         perror("times 1");
16         exit(1);
17     }
18
19     for (i = 0; i < 99999999; i++)
20         time(&t);
21
22     if ((t2 = times(&mytms)) == -1) { +2: 끝 시간
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 */

환경변수의 이해

□ 환경변수

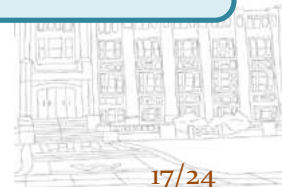
- 프로세스가 실행되는 기본 환경을 설정하는 변수
- 로그인명, 로그인 셸, 터미널에 설정된 언어, 경로명 등
- 환경변수는 “환경변수=값”의 형태로 구성되며 관례적으로 대문자로 사용
- 현재 셸의 환경 설정을 보려면 env 명령을 사용

```
# 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
...
```

env | grep SHELL

하면 SHELL만 볼 수 있지

SHELL = /usr/bin/ksh



환경변수의 사용[1]

- 전역변수 사용 : environ *env 와 똑같은 걸로*

```
#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`
```

환경변수의 사용[2]

□ main 함수 인자 사용 *env와 똑같은 결과*

```
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
```

환경변수의 사용[3]

- 환경변수 검색: getenv(3) *특정 환경변수의 값만 볼 수 있음*

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

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

[예제 5-7] getenv 함수 사용하기

ex5_7.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) 프로그램이 실행되는 동안에만 바뀌어 있음

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

```
int putenv(char *string);
```

[예제 5-8] putenv 함수 사용하기

ex5_8.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);
```



```

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  }

```

프로그램이 실행되는 동안에만 바뀜

환경변수의 덮어쓰기가 되지 않음
= 안바뀌는 것과 같음

환경변수의 덮어쓰기 설정
= 바뀜

ex5_9.out

```

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

```



Thank You !

IT CookBook, 유닉스 시스템 프로그래밍