8장 프로세스 다루기(2)

- 서론
- 예제 프로그램
- 함수
  - wait
  - waitpid
  - getpid, getppid
  - getpgrp, getpgid, setpgrp,setpgid
  - getsid, setsid
  - getenv, putenv

한빛미디어(주)

프로세스를 동기화하고 속성과 환경 변수를 다루는 시스템 호출/표준 라이브러리 함수

함수	의미
wait	자신의 자식 프로세스가 종료할 때까지 대기 상태가 된다.
waitpid	지정한 자신의 자식 프로세스가 종료할 때까지 대기 상태가 된다.
getpid, getppid	자신(또는 부모)의 프로세스 식별 번호를 구한다.
getpgrp, setpgrp	자신의 프로세스 그룹 식별 번호를 구하거나 변경한다.
getpgid, setpgid	지정한 프로세스의 그룹 식별 번호를 구하거나 변경한다.
getsid	지정한 프로세스의 세션 식별 번호를 구한다.
setsid	현재 프로세스가 새로운 세션을 생성한다.
getenv, putenv	환경 변수의 값을 구하거나, 새로운 환경 변수를 등록/변경한다.
setenv	새로운 환경 변수를 등록하거나 변경한다.
unsetenv	등록된 환경 변수를 삭제한다.

【예제】 예제 프로그램(1/3)


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main()
05 {
06    pid_t pid;
07     int status;
08
09     pid = fork();
10
11    putenv("APPLE=RED");
```

09 자식 프로세스를 생성한다

11 환경변수를 등록한다

표준입력 스트림

【예제】 예제 프로그램(2/3)


```
12    if(pid > 0)
13    {
14        printf("[parent] PID : %d₩n", getpid());
15        printf("[parent] PPID: %d₩n", getppid());
16        printf("[parent] GID : %d₩n", getpgrp());
17        printf("[parent] SID : %d₩n", getsid(0));
18
19        waitpid(pid, &status, 0);
20
21        printf("[parent] status is %d₩n", status);
22
23        unsetenv("APPLE");
24    }
```

12 부모 프로세스가 수행하는 부분이다

14 자신의 PID, PPID, GID, SID를 출력한다

19 pid를 식별번호로 가지는 자식 프로세스의 종료를 기다린다

21 자식 프로세스가 종료하면서 전달한 종료 상태 값을 출력한다.

23 환경변수를 삭제한다

표준입력 스트림

【예제】 예제 프로그램(3/3)


```
25    else if(pid == 0)
26    {
27        printf("[child] PID : %d₩n", getpid());
28        printf("[child] PPID: %d₩n", getppid());
29        printf("[child] GID : %d₩n", getpgid(0));
30        printf("[child] SID : %d₩n", getsid(0));
31
32        sleep(1);
33
34        printf("[child] APPLE=%s₩n", getenv("APPLE"));
35
36        exit(1);
37    }
38    else
39        printf("fail to fork₩n");
40 }
```

25 자식 프로세스가 수행하는 부분이다.

34 환경변수 값을 읽어와 출력한다

표준입력 스트림

실행 결과

```
$ ex08-01
[parent] PID : 13011
[parent] PPID: 3913
[parent] GID : 13011
[parent] SID : 3913
[child] PID : 13012
[child] PPID: 13011
[child] GID : 13011
[child] SID : 3913
[child] APPLE=RED
[parent] status is 256
$
```

자신의 자식 프로세스가 종료할 때까지 대기한다.

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

pid_t wait(int *status);
```

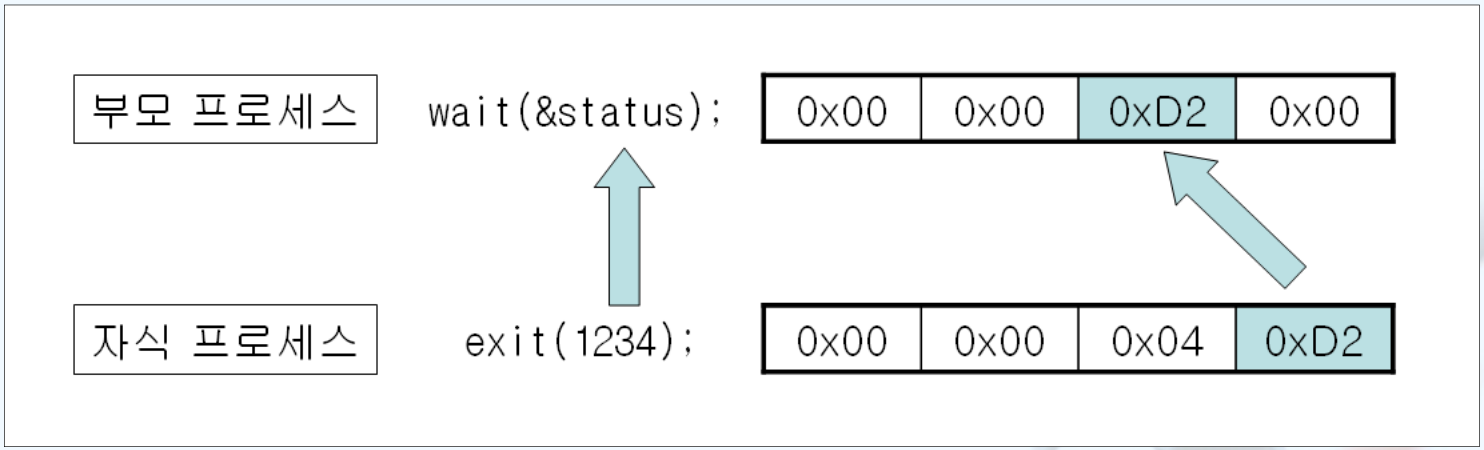
status	자식 프로세스가 exit 함수로 종료하면서 전달하는 종료 상태 값으로 0에서 255 사이의 값을 가진다.
반환값	호출이 성공했을 경우 종료한 자식 프로세스의 식별 번호가 반환되고, 실패할 경우 -1이 반환된다.

- **자식 프로세스를 가진 부모 프로세스가 wait를 호출하면**
  - **자식 프로세스가 종료할 때까지 실행이 중단된다. (대기 상태)**
  - **자식 프로세스가 종료하면 이를 처리한다.**

    wait 호출 이전에 자식이 종료했다면 대기 상태가 되지 않고 처리

status

- 자식 프로세스가 exit를 호출하면서 지정한 값을 부모 프로세스는 status 변수로 받는다.
- 자식 프로세스가 exit(n); 을 실행했을 때 부모 프로세스에게 전달되는 실제 값은 n의 하위 1 바이트 뿐이다.
- 자식 프로세스가 전달한 1바이트 값은 부모 프로세스 쪽의 status 변수의 하위 두 번째 바이트에 저장된다.



【예제 8-2】 ex08-02.c


```
01 #include <stdio.h>
02 #include <sys/types.h>
03
04 main()
05 {
06     pid_t pid;
07     int status;
08
09     pid = fork();
10
11     if(pid > 0) {                    /* 부모 프로세스 */
12         printf("parent: waiting..₩n");
13         wait(&status);
14         printf("parent: status is %d₩n", status);
15     }
16     else if(pid == 0) { /* 자식 프로세스 */
17         sleep(1);
18         printf("child: bye!₩n");
19         exit(1234);
20     }
21     else
22         printf("fail to fork₩n");
23
24     printf("bye!₩n");
25 }
```

```
$ ex08-02
parent: waiting..
child: bye!
parent: status is 53760
bye!
$
```

자식 프로세스가 1234로 exit했다. 부모 프로세스가 실제로 받는 값은 얼마인가?

좀비 프로세스 (zombie process)

- 부모 프로세스가 wait를 수행하지 않고 있는 상태에서 자식이 종료
  - ▸자식 프로세스의 종료를 부모 프로세스가 처리해주지 않으면 자식 프로세스는 좀비 프로세스가 된다.
  - ▸좀비 프로세스는 CPU, Memory 등의 자원을 사용하지 않으나, 커널의 작업 리스트에는 존재한다.

고아 프로세스 (orphan process)

- 하나 이상의 자식 프로세스가 수행되고 있는 상태에서 부모가 먼저 종료
  부모 프로세스가 수행 중인 자식 프로세스를 기다리지 않고 먼저 종료

init 프로세스

- 좀비와 고아 프로세스의 관리는 결국 시스템의 init 프로세스로 넘겨진다.
  - ▸ init 프로세스가 새로운 부모가 된다.

프로세스 식별번호로 지정한 자식 프로세스가 종료할 때까지 대기한다

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

pid_t waitpid(pid_t pid, int *status, int options);
```

pid	자식 프로세스의 식별 번호이다.
status	자식 프로세스가 exit 함수로 종료하면서 전달하는 종료 상태 값이다.
options	부모 프로세스의 대기 방법을 선택한다. 일반적으로 0이 사용된다.
반환값	호출이 성공했을 때 종료한 자식 프로세스가 있다면 자식 프로세스의 식별번호가 반환되고, WNOHANG 옵션을 사용할 때 종료한 자식 프로세스가 없으면 0을 반환한다. 호출이 실패할 경우 -1을 반환한다.

- **wait와 waitpid의 차이점**
  - **wait는 자식 프로세스 중 가장 먼저 종료되는 것을 처리해주나, waitpid는 PID로 지정한 자식 프로세스의 종료만 처리해준다.**

wait와 waitpid의 차이점 (계속)

- wait
  - 부모 프로세스가 특정 자식 프로세스를 기다리지 않는다.

    먼저 종료되는 것을 먼저 처리해준다.
  - 종료되는 자식 프로세스를 wait의 반환 값으로 알 수 있다.
  - 종료하는 자식 프로세스가 있을 때까지 부모 프로세스는 대기 상태가 된다.
- waitpid
  - 부모 프로세스가 PID로 자식 프로세스를 지정하여 기다린다.

    지정하지 않은 자식 프로세스의 종료를 처리해주지 않는다.

    자식 프로세스의 종료 순서에 상관없이 부모 프로세스가 처리 순서를 결정할 수 있다.
  - 옵션에 따라서 자식 프로세스가 종료할 때까지 대기 상태가 될 수도 있고 아닐 수도 있다.

【예제 8-3】 ex08-03.c


```
01 #include <unistd.h>
02 #include <sys/types.h>
03
04 main()
05 {
06     pid_t pid1, pid2;
07     int status;
08
09     pid1 = pid2 = -1;
10
11     pid1 = fork();
12     if(pid1 > 0)
13         pid2 = fork();
14
15     if(pid1 > 0 && pid2 > 0)
16     {
17         waitpid(pid2, &status, 0);
18         printf("parent: child2 - exit(%d)₩n", status);
19         waitpid(pid1, &status, 0);
20         printf("parent: child1 - exit(%d)₩n", status);
21     }
```

12 부모프로세스라면 자식을 한 번 더 생성

15 부모 프로세스

표준입력 스트림

【예제 8-3】 ex08-03.c


```
22      else if(pid1 == 0 && pid2 == -1)
23      {
24          sleep(1);
25          exit(1);
26      }
27      else if(pid1 > 0 && pid2 == 0)
28      {
29          sleep(2);
30          exit(2);
31      }
32      else
33          printf("fail to fork₩n");
34  }
```

22 첫번째 자식 프로세스

27 두 번째 자식 프로세스

표준입력 스트림

```
$ ex08-03
parent: child2 - exit(512)
parent: child1 - exit(256)
$
```

WNOHANG 옵션

waitpid 함수의 동작 방법을 변경한다.

- 기본 동작 (WNOHANG 사용하지 않음)

  PID로 지정한 자식 프로세스가 종료할 때까지 대기한다.

- WNOHANG 사용

  PID로 지정한 자식 프로세스가 종료하였는지 확인

  종료했으면 이를 처리하고

  종료하지 않았으면 자신의 일을 계속 수행한다.

【예제 8-4】 ex08-04.c


```
01 #include <unistd.h>
02 #include <sys/types.h>
03 #include <sys/wait.h>
04
05 main()
06 {
07     pid_t pid;
08     int status = 0;
09
10     if((pid = fork()) > 0)
11     {
12         while(!waitpid(pid, &status, WNOHANG))
13         {
14             printf("parent: %d₩n", status++);
15             sleep(1);
16         }
17         printf("parent: child - exit(%d)₩n", status);
18     }
```

10 부모프로세스

12 WNOHANG 사용

표준입력 스트림

【예제 8-4】 ex08-04.c


```
19     else if(pid == 0)
20     {
21         sleep(5);
22         printf("bye!₩n");
23         exit(0);
24     }
25     else
26         printf("fail to fork₩n");
27 }
```

19 자식 프로세스

21 5초 동안 대기한다

표준입력 스트림

```
$ ex08-04
parent: 0
parent: 1
parent: 2
parent: 3
parent: 4
bye!
parent: child - exit(0)
$
```

자신이나 부모의 프로세스 식별 번호(PID)를 구한다.

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

pid_t getpid(void);
pid_t getppid(void);
```

반환값	프로세스의 식별 번호이다.

- **getpid**
  - **자신의 프로세스 식별 번호를 구한다.**
- **getppid**
  - **부모의 프로세스 식별 번호를 구한다.**

【예제 8-5】 ex08-05.c


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main()
05 {
06    pid_t pid;
07
08     if((pid = fork()) > 0) {
09         printf("[ex08-05.c] PPID:%d, PID:%d₩n",
           getppid(), getpid());
10         sleep(1);
11     }
12     else if(pid == 0) {
13         printf("[ex08-05.c] PPID:%d, PID:%d₩n",
           getppid(), getpid());
14         execl("0806", "0806", (char *)0);
15     }
16     else
17         printf("fail to fork₩n");
18 }
```

표준입력 스트림

【예제 8-6】 ex08-06.c


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main()
05 {
06     printf("[ex08-06.c] PPID:%d, PID:%d₩n",
       getppid(), getpid());
07 }
```

표준입력 스트림

```
$ ps
  PID TTY          TIME CMD
23588 pts/3    00:00:00 bash
2587 pts/3    00:00:00 ps
$ ex08-05
[ex08-05.c] PPID:23588, PID:2588
[ex08-05.c] PPID:2588, PID:2589
[ex08-06.c] PPID:2588, PID:2589
$
```

프로세스의 그룹 식별 번호를 구하거나 변경한다.

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

int setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);
int setpgrp(void);
pid_t getpgrp(void);
```

pid	프로세스 식별 번호이다.
pgid	프로세스 그룹의 식별 번호이다.
반환값	setpgid와 setpgrp는 호출이 성공할 경우 0을 반환하고, 실패할 경우 -1을 반환한다. getpgid는 호출이 성공할 경우 프로세스의 그룹 식별 번호를 반환하고, 실패할 경우 -1을 반환한다. getpgrp은 항상 프로세스의 그룹 식별 번호를 반환한다.

프로세스 식별번호 (process id)

- 한 시점에서 프로세스를 식별하기 위한 유일한 번호
- 음이 아닌 정수

프로세스 그룹

- 여러 개의 프로세스들이 하나의 그룹에 속할 수 있음
- 시그널을 사용하여 동일 그룹의 프로세스들의 한 집합을 한꺼번에 처리할 수 있음

프로세스 그룹 식별번호 (process group id)

- 한 시점에서 프로세스 그룹을 식별하기 위한 유일한 번호
- 그룹 식별번호와 동일한 값을 프로세스 식별번호로 가지는 프로세스
  - 프로세스 그룹의 리더

setpgid

- pid로 지정한 프로세스의 그룹 식별 번호를 pgid로 변경한다.
- pid = 0 일 경우 현재 프로세스에 적용한다.
- pgid = 0 일 경우 pid로 지정한 값을 pgid로 사용한다.
- 프로세스 그룹의 변경은 같은 세션 내에서만 가능하다

setpgrp, getpgrp

- setpgrp은 setpgid(0, 0)과 같고
- getpgrp은 getpgid(0)과 같다.

【예제 8-7】 ex08-07.c


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main()
05 {
06     printf("getpgrp():%d₩n", getpgrp());
07     printf("getpgid(0):%d₩n", getpgid(0));
08     printf("getpgid(getpid()):%d₩n", getpgid(getpid()));
09}
```

표준입력 스트림

```
$ ex08-07
getpgrp():2657
getpgid(0):2657
getpgid(getpid()):2657
$
```

프로세스의 세션 식별 번호를 구하거나, 새로운 세션을 생성한다

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

pid_t getsid(pid_t pid);
pid_t setsid(void);
```

pid	프로세스의 식별 번호이다.
반환값	호출이 성공하면 프로세스의 세션 식별 번호를 반환하고, 실패하면 -1을 반환한다.

- **세션 (session)**
  - **일반적으로 시스템과 연결된 하나의 제어 단말기를 포함한 단위**
  - **식별 번호(id)가 부여되어 있다.**
  - **세션 ⊃ 그룹 ⊃ 프로세스**

getsid

- pid로 지정한 프로세스가 포함된 세션의 식별 번호를 알아온다.
- pid = 0 일 경우 현재 프로세스에 대해 알아온다.
- 세션의 리더 (프로세스)
- 자신의 PID = 자신의 PGID = 자신의 PSID 일 경우

setsid

- 호출하는 프로세스가 그룹의 리더가 아닌 경우 새로운 세션을 생성한다.
  - 일반적인 경우에 셸 프로세스가 세션과 그룹의 리더이다.
- 호출이 성공하면 프로세스는 새로운 세션과 그룹의 리더가 된다.
  - 이때 제어 터미널을 가지지 않는다.
  - 세션과 그룹 내에서 유일한 프로세스가 된다.

【예제 8-8】 ex08-08.c


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main(int argc, char *argv[])
05 {
06     pid_t pid;
07     int interval;
08
09     if(argc != 3)
10         exit(1);
11
12     pid = atoi(argv[1]);
13     interval = atoi(argv[2]);
14
15     printf("shell process...₩n");
16    printf("process id:%d, group id:%d, session id:%d₩n",
17         pid, getpgid(pid), getsid(pid));
18     printf("current process.. not daemon...₩n");
19     printf("process id:%d, group id:%d, session id:%d₩n",
20         getpid(), getpgrp(), getsid(0));
21
22     sleep(interval);
23 }
```

```
$ ps
  PID TTY          TIME CMD
2849 pts/4    00:00:00 bash
3030 pts/4    00:00:00 ps
$ ex08-08 2849 0
shell process...
process id:2849, group id:2849, session id:2849
current process.. not daemon...
process id:3031, group id:3031, session id:2849
$
```

연결 종료

- 로그인 셸
  - 제어 터미널의 연결 상태를 가진다. (제어 터미널을 포함한 세션)
  - 세션과 그룹 내에서 리더이다.
  - 셸 프롬프트 상에서 실행한 많은 프로세스가 있다.

로그인 셸의 종료

- 세션의 리더이기 때문에 연결이 끊어진다.
- 같은 세션에 있는 다른 프로세스도 종료된다.
  - 후면(background) 실행 중인 프로세스도 종료된다.

연결 종료 시 같은 세션의 프로세스 모두 종료된다. (1)

```
$ ps
  PID TTY          TIME CMD
2849 pts/4    00:00:00 bash
3030 pts/4    00:00:00 ps
$ ex08-08 2849 600 &
[1] 3158
shell process...
process id:2849, group id:2849, session id:2849
current process.. not daemon...
process id:3158, group id:3158, session id:2849
$ ex08-08 2849 600 &
[2] 3159
$ shell process...
process id:2849, group id:2849, session id:2849
current process.. not daemon...
process id:3159, group id:3159, session id:2849
```

연결 종료 시 같은 세션의 프로세스는 모두 종료된다. (2)

```
$ ps
  PID TTY          TIME CMD
2849 pts/4    00:00:00 bash
3158 pts/4    00:00:00 ex08-08
3159 pts/4    00:00:00 ex08-08
3160 pts/4    00:00:00 ps
$
```

※터미널 연결을 끊고 다시 접속한다.

```
$ ps
  PID TTY          TIME CMD
3177 pts/4    00:00:00 bash
3201 pts/4    00:00:00 ps
$
```

【예제 8-9】 ex08-09.c


```
01 #include <sys/types.h>
02 #include <unistd.h>
03
04 main()
05 {
06         pid_t pid;
07
08         if((pid = fork()) > 0)
09         {
10                 sleep(1);
11                 exit(1);
12         }
13         else if(pid == 0)
14         {
15                 printf("old session id: %d₩n", getsid(0));
16                 printf("new session id: %d₩n", setsid());
17                 sleep(600);
18         }
19 }
```

새로운 세션을 만들면서 현재 프로세스가 리더가 된다.

실행 결과 (1)

```
$ ex08-09
old session id: 3615
new session id: 3862
$ ps
  PID TTY          TIME CMD
3615 pts/5    00:00:00 bash
3867 pts/5    00:00:00 ps
$ ps -ef | grep usp
usp       3614  3610  0 22:52 ?        00:00:00 /usr/local/ssh/sbin/sshd
usp       3615  3614  0 22:52 pts/5    00:00:00 -bash
usp       3862     1  0 22:55 ?        00:00:00 ex08-09
usp       3876  3615  0 22:55 pts/5    00:00:00 ps -ef
usp       3877  3615  0 22:55 pts/5    00:00:00 grep usp
$
```

※터미널 연결을 끊고 다시 접속한다.

실행 결과 (2)

※터미널 연결을 끊고 다시 접속한다.

```
로그인: usp
usp의 비밀번호:
Last login: ..........
$ ps
  PID TTY          TIME CMD
3913 pts/2    00:00:00 bash
3949 pts/2    00:00:00 ps
$ ps -ef | grep usp
usp       3862     1  0 22:55 ?        00:00:00 ex08-09
usp       3912  3904  0 22:56 ?        00:00:00 /usr/local/ssh/sbin/sshd
usp       3913  3912  0 22:56 pts/2    00:00:00 -bash
usp       3956  3913  0 22:56 pts/2    00:00:00 ps -ef
usp       3957  3913  0 22:56 pts/2    00:00:00 grep usp
$
```

환경 변수의 값을 알아오거나 새로운 값을 등록한다.

```
#include <stdlib.h>

char *getenv(const char *name);
int putenv(char *string);
int setenv(const char *name, const char *value, int overwrite);
void unsetenv(const char *name);
```

string	name=value 형식으로 구성된 문자열이다.
name	환경 변수의 이름에 해당하는 문자열이다.
value	환경 변수의 값에 해당하는 문자열이다.
overwrite	name으로 지정한 환경 변수가 이미 존재할 경우 덮어쓰기 여부를 결정한다. 0이면 덮어쓰기를 하고 0이 아니면 덮어쓰기를 하지 않는다.
반환값	getenv는 호출이 성공할 경우 name에 해당하는 환경 변수의 값에 대한 문자열 포인터를 반환하고, 검색에 실패할 경우 NULL을 반환한다. putenv와 setenv는 호출이 성공할 경우 0을 반환하고, 실패할 경우 -1을 반환한다.

환경 변수

- 프로세스가 exec 계열의 함수를 사용하여 새로운 프로세스를 생성할 때 넘겨 주는 값
- 문자열의 형태

  “name=value”의 형식이며 NULL로 종결되어야 한다.

  여러 개의 문자열이 환경 변수로 존재할 수 있다.

  마지막 원소는 NULL 포인터여야 한다.

- putenv, setenv

  새로운 환경 변수를 등록하거나 기존의 환경 변수를 변경한다.
- getenv

  등록된 환경 변수의 값을 구한다.
- unsetenv

  등록된 환경 변수를 제거한다.

【예제 8-10,11】 ex08-10,11.c


```
$ ex08-10
BANANA
BANANA
APPLE not found
$
```

```c
01 #include <unistd.h>
02
03 main()
04 {
05     putenv("APPLE=BANANA");
06     printf("%s₩n", getenv("APPLE"));
07
08     execl("ex08-11", "ex08-11", (char *)0);
09 }
```

```c
01 #include <unistd.h>
02
03 main()
04 {
05     printf("%s₩n", getenv("APPLE"));
06     unsetenv("APPLE");
07
08     if(!getenv("APPLE"))
09         printf("APPLE not found₩n");
10 }
```

환경 변수를 전하는 또다른 방법

- 기존의 방법
  - 자신이 가지고 있는 이미 등록되어 있는 환경 변수를 전달한다.
- 또다른 방법
  - 프로세스 내에서 문자열을 새롭게 구성하여 (환경 변수 등록이 아님) 이를 새로운 프로세스를 생성하면서 전달한다.
  - exec 계열의 새로운 함수를 사용한다.

    execle, execve

마지막 값에 주의!!

```
char *envlist[] = {"APPLE=BANANA", (char *)0};
...
execle("myprogram", "myprogram", (char *)0, envlist);
execve("myprogram", arglist, envlist);
```

전달되는 환경 변수 받기

```
extern char **environ;
main()
{
    while(*environ)
        printf("%s₩n", *environ++);
}
```

또는

```
main(int argc, char *argv[], char *envlist[])
{
    while(*envlist)
        printf("%s₩n", *envlist++);
}
```

【예제 8-12,13】 ex08-12,13.c


```
01 #include <unistd.h>
02
03 main()
04 {
05         char *envlist[] = {"APPLE=0", "BANANA=1", (char *)0 };
06
07
08         execle("ex08-13", "ex08-13", (char *)0, envlist);
09 }
```

```
01 #include <unistd.h>
02
03 extern char **environ;
04
05 main()
06 {
07         while(*environ)
08                 printf("%s₩n", *environ++);
09 }
```