

PROCESS 2

(EXEC)

Jo, Heeseung

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

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 }
```

ex6_3.out

Cleanup 2 is called.

Cleanup 1 is called.

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

exec 함수군 활용

exec 함수군

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

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 execlp(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 주의

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 }
```

ex6_4.out

--> Before exec function

.	ex6_1.c	ex6_3.c	ex6_4.out
..	ex6_2.c	ex6_4.c	han.txt

인자의 끝을 표시하는
NULL 포인터

첫 인자는 관례적으로
실행파일명 지정

메모리 이미지가 'ls'
명령으로 바뀌어 13행
은 실행안됨

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 }
```

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	

첫 인자는 관례적으로 실행파일명 지정

인자의 끝을 표시하는 NULL 포인터

경로로 명령 지정

역시 실행 안됨

exec 함수군과 fork 함수

fork로 생성한 자식 프로세스에서 exec 함수군을 호출

- exec 함수군
 - 자식 프로세스의 메모리 이미지로 대체
 - 부모 프로세스 이미지는 사라짐
- 부모 프로세스와 다른 프로그램 실행 가능
- 부모 프로세스와 자식 프로세스가 각기 다른 작업을 수행
 - 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 }
```

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

자식프로세스에서 execlp 함수 실행

부모프로세스는 이 부분 실행

프로세스 동기화 함수[1]

프로세스 동기화: wait(3)

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *stat_loc);
```

- stat_loc : 상태정보를 저장할 주소
- wait 함수는 자식 프로세스가 종료할 때까지 부모 프로세스를 대기시킴
- 부모 프로세스가 wait 함수를 호출하기 전에 자식 프로세스가 종료하면 wait 함수는 즉시 리턴
- wait 함수의 리턴값은 자식 프로세스의 PID
- 리턴값이 -1이면 살아있는 자식 프로세스가 하나도 없다는 의미

wait 함수 사용하기

```
int main(void) {
    int status;
    pid_t pid;
    switch (pid = fork()) {
        case -1 : /* fork failed */
            perror("fork");
            exit(1);
            break;
        case 0 : /* child process */
            printf("--> Child Process\n");
            sleep(2);
            exit(2);
            break;
        default : /* parent process */
            wait(&status);

            printf("--> Parent process\n");
            printf("Status: %d, %x\n", status, status);
            printf("Child process Exit Status:%d\n", status >> 8);
            break;
    }
    return 0;
}
```

ex6_8.out

--> Child Process

--> Parent process

Status: 512, 200

Child process Exit Status:2

자식 프로세스의 종료를 기다림

오른쪽으로 8비트 이동해야 종료 상태 값을 알 수 있음

Exercise

Ex.

- 지정된 구구단 한 단을 출력하는 `guguone` 프로그램 작성
 - `guguone`은 command line argument로 출력 단을 받음
 - `./guguone 3`
- `guguone`을 child process로 이용하여 구구단을 출력하는 `guguexec` 프로그램 작성
 - Child process는 `guguone` 프로그램을 실행 (`exec` 함수 사용)
 - Child process가 출력을 완료할 때까지 parent process는 대기

```
./guguexec 2
```

```
2 x 0 = 0
```

```
2 x 1 = 2
```

```
...
```

Exercise

Ex.

- prime number를 출력하는 psingle 프로그램을 변경하여 argument로 숫자 범위 두 개를 받도록 하는 prange 프로그램을 작성
./prange 100 400
- prange를 이용하여 5개의 프로세스를 이용하고 100,000까지의 모든 prime number를 출력할 수 있는 prangemulti 프로그램을 작성
 - 5개의 프로세스는 prange로 구간을 나누어서 병렬처리
 - Child process가 출력을 완료할 때까지 parent process는 대기
 - time 명령을 이용하여 몇 초가 걸리는지 확인할 것