프로세스 생성[1]

프로그램 실행 : system(3)

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

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

[예제 6-1] 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
```

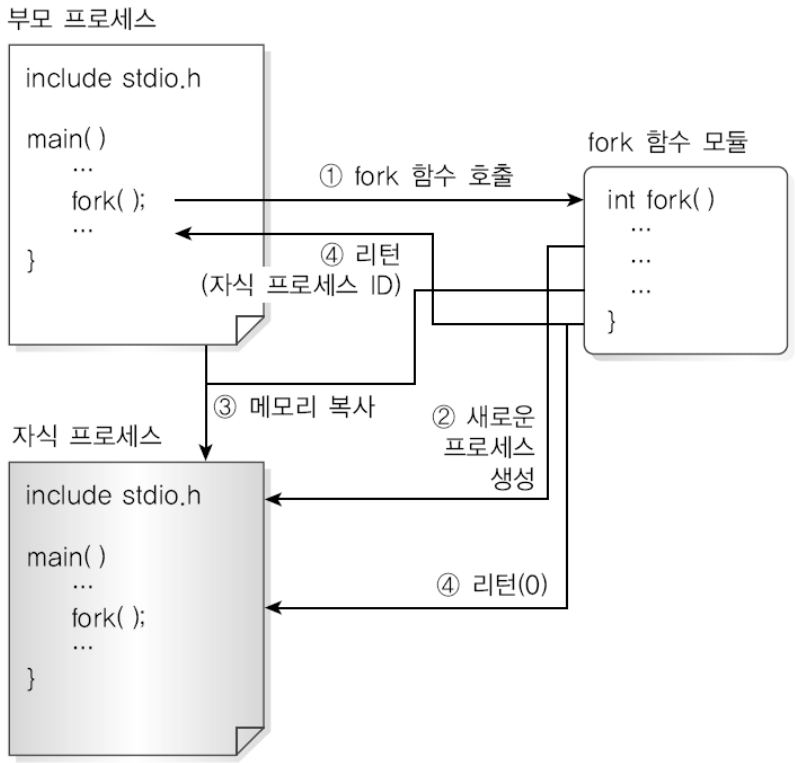
프로세스 생성[2]

프로세스 생성: fork(2)

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

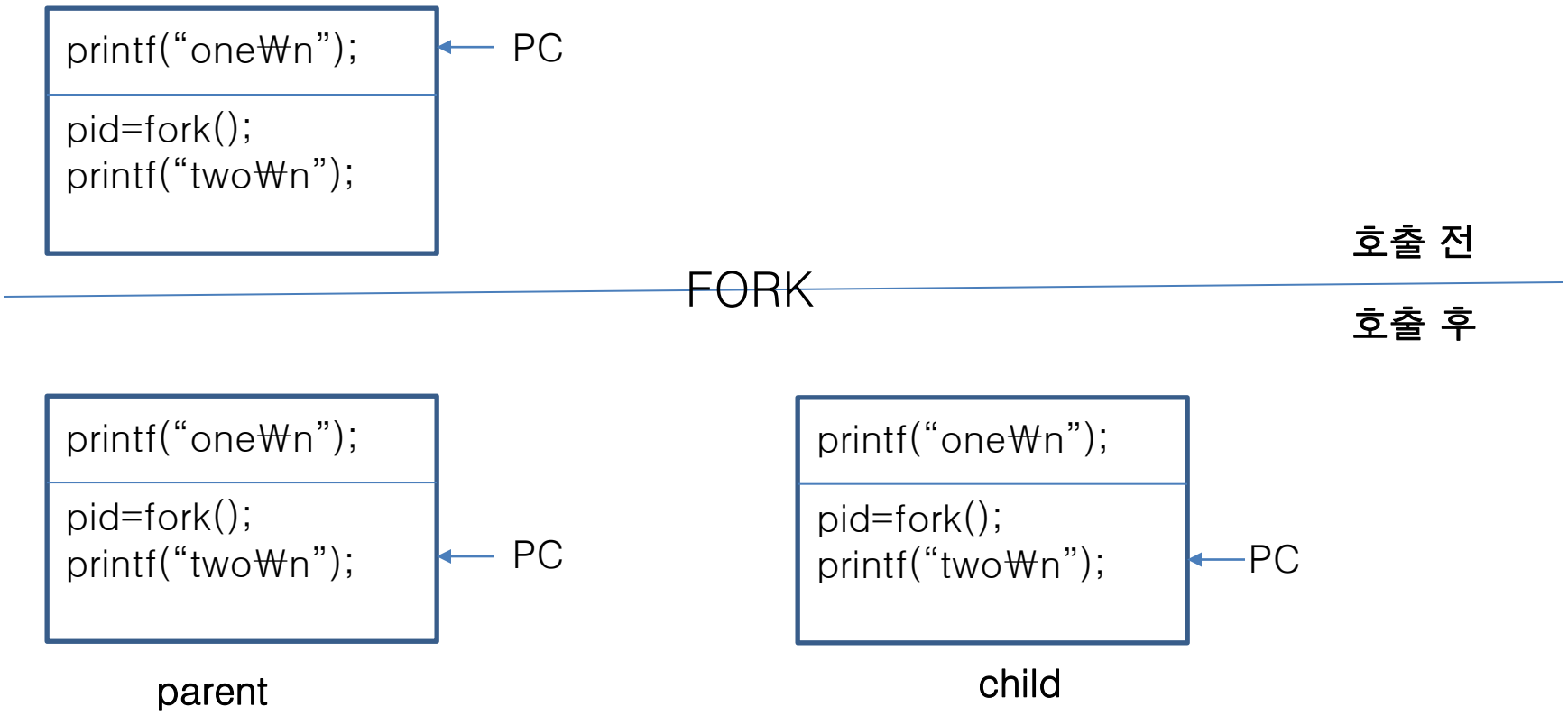
- 새로운 프로세스를 생성 : 자식 프로세스
- fork 함수를 호출한 프로세스 : 부모 프로세스
- 자식 프로세스는 부모 프로세스의 메모리를 복사
  - RUID, EUID, RGID, EGID, 환경변수
  - 열린 파일기술자, 시그널 처리, setuid, setgid
  - 현재 작업 디렉토리, umask, 사용가능자원 제한

- 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.



[그림 6-1] fork 함수를 이용한 새로운 프로세스 생성

프로세스 생성[3]


printf("one₩n");
pid=fork();
printf("two₩n");
PC
호출 전
FORK
호출 후
printf("one₩n");
pid=fork();
printf("two₩n");
PC
parent
printf("one₩n");
pid=fork();
printf("two₩n");
PC
child

[예제 6-2] fork 함수 사용하기 (test1.c)

```
...
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          default : /* parent process */
14          printf("Parent process - My PID:%d, My Parent's PID:%d, "
15              "My Child's PID:%d\n", (int)getpid(), (int)getppid(),
16              (int)pid);
17              break;
18      }
19
21      printf("End of fork\n");
22      return 0;
23  }
24
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.

참고자료

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.

[예제 6-3] exit, atexit 함수 사용하기

ex6_3.c

```
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 실행할 수 있다.
- exec 함수가 실행되면 프로세스의 메모리 이미지는 실행파일로 바뀐다.

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 execle(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.

exec 함수군 활용

```
printf(…);
execl(“/bin/ls”,…);
printf(“ERROR₩n”);
```

```
/* ls code */
```

[예제 6-4] execlp 함수 사용하기(test2.c)

```
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 (execlp("ls", "ls", "-a", (char *)NULL) == -1) {
09          perror("execlp");
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
```

[예제 6-5] execv 함수 사용하기(test3.c)

```
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
```

[예제 6-6] execve 함수 사용하기(test4.c)

```
...
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
```

[예제 6-6] (2) arg.c 파일 (arg.out)

```
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

Compiler directive 사용

```
#ifdef TIMES
        start = dclock();
#endif
                    시간측정부분
#ifdef TIMES
        end = dclock() – start;
        printf(end);
#endif
            gcc -DTIMES –o sample sample.c
```

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

[예제 6-7] 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
```