# 12장 고급 응용

# 프로세스

- 수행 중인 프로그램
- 모든 프로세스는 유일한 프로세스 식별 번호(PID)를 가짐
- 유닉스에서는 ps 명령을 사용하여 프로세스 목록을 볼 수 있음

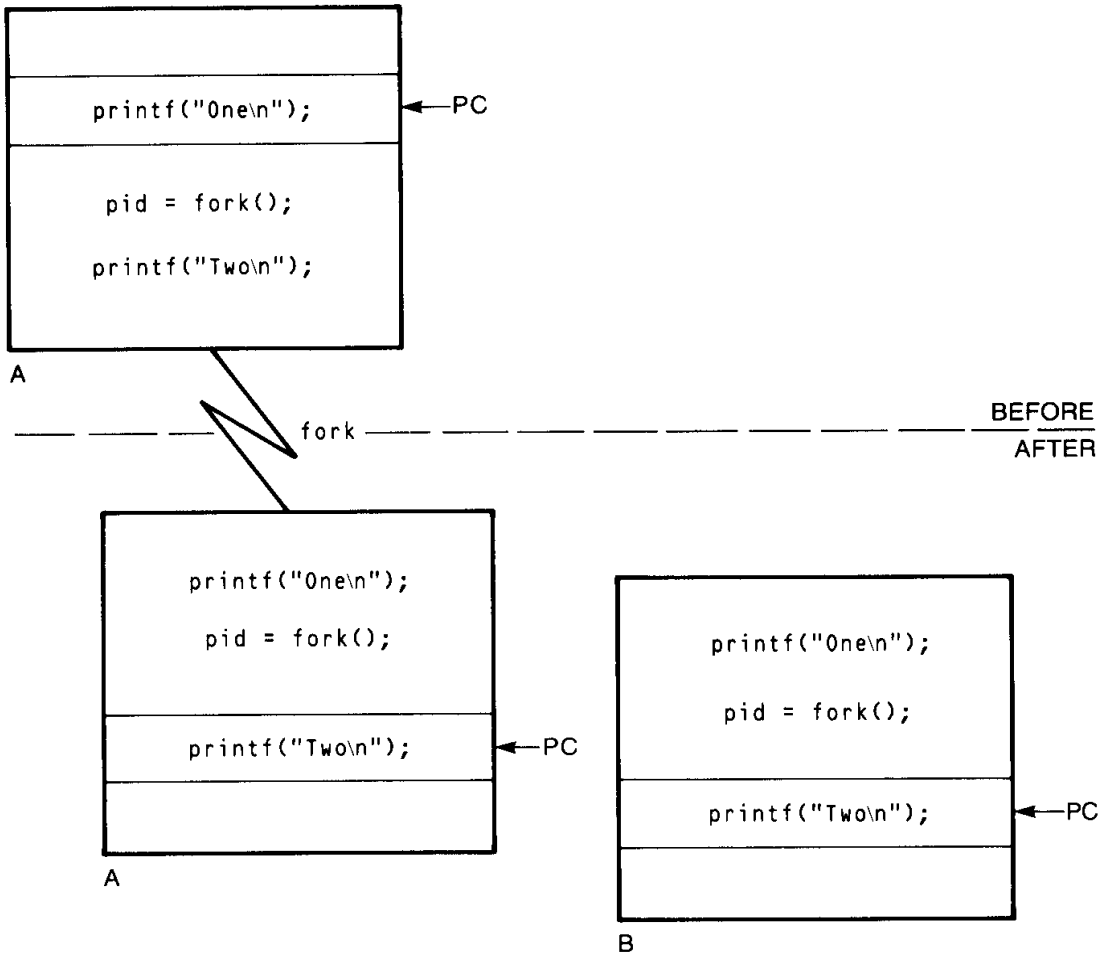
# 프로세스

```
$ ps -aux
USER       PID %CPU %MEM    SZ   RSS TT STAT   START    TIME COMMAND
blufox   17725 34.0  1.6   146   105 i2   R    15:13    0:00 ps-aux
amber    17662  1.4  7.0   636   469 j5   S    15:10    0:08 vi it.c
root       143  0.5  0.1     5     3  ?   S    Jul 31 10:17 find
.....
```

# fork()

- 새로운 프로세스를 생성하는 함수
- 부모 프로세스 / 자식 프로세스
- fork()가 호출될 때, 자식 프로세스라는 새로운 프로세스가 생성
- 새로운 프로세스는 자신의 프로세스 식별 번호를 갖는다는 것만 제외하고는 호출한 프로세스를 그대로 복사한 것임
- fork()는 자식 프로세스에게 0을 리턴하고, 부모 프로세스에게 자식 프로세스 ID를 리턴함
- 변수들은 공유하지 않지만, 사용되고 있던 파일 포인터는 공유함

# fork()


printf("One\n");
PC
pid = fork();
printf("Two\n");
A
fork
BEFORE
AFTER
printf("One\n");
pid = fork();
printf("Two\n");
PC
A
printf("One\n");
pid = fork();
printf("Two\n");
PC
B

# fork() 예제 프로그램

- 프로그램

```
#include <stdio.h>
int main(void){
   int   fork(void), value;
   value = fork();              /* new process */
   printf("In main: value = %d\n", value);
   return 0;
}
```

- 출력

```
In main: value = 17219
In main: value = 0
```

← 부모 프로세스의 출력

← 자식 프로세스의 출력

# wait()

- 자식 프로세스가 수행될 동안 프로세스를 멈추게 함

  → 자식 프로세스가 끝나면 부모 프로세스가 수행 시작

- 함수 원형

  pid_t wait(int *statloc)

  - 리턴값 : process ID 또는 -1
  - statloc : 널이 아니면 끝나는 프로세스의 상태를 저장할 곳을 나타냄 (상위 비트에 저장)

# 예제 프로그램

```
main() {
    int pid, status, exit_status;
    if ((pid = fork()) < 0 exit(1);
    if (pid == 0) { sleep(4);  exit(5); }
    if (wait(&status) < 0) exit(1);
    if ((status & 0xFF) != 0) printf("Error occurred");
    else {
       exit_status = status >> 8;
       exit_status &= 0xFF;
       printf("Exit status from %d was %d",
              pid, exit_status);
    }
}
```

- 결과

```
Exit status from 14963 was 5
```

# Exec...() 부류

```
#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, char *const 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[]);
```

# Exec...() 부류

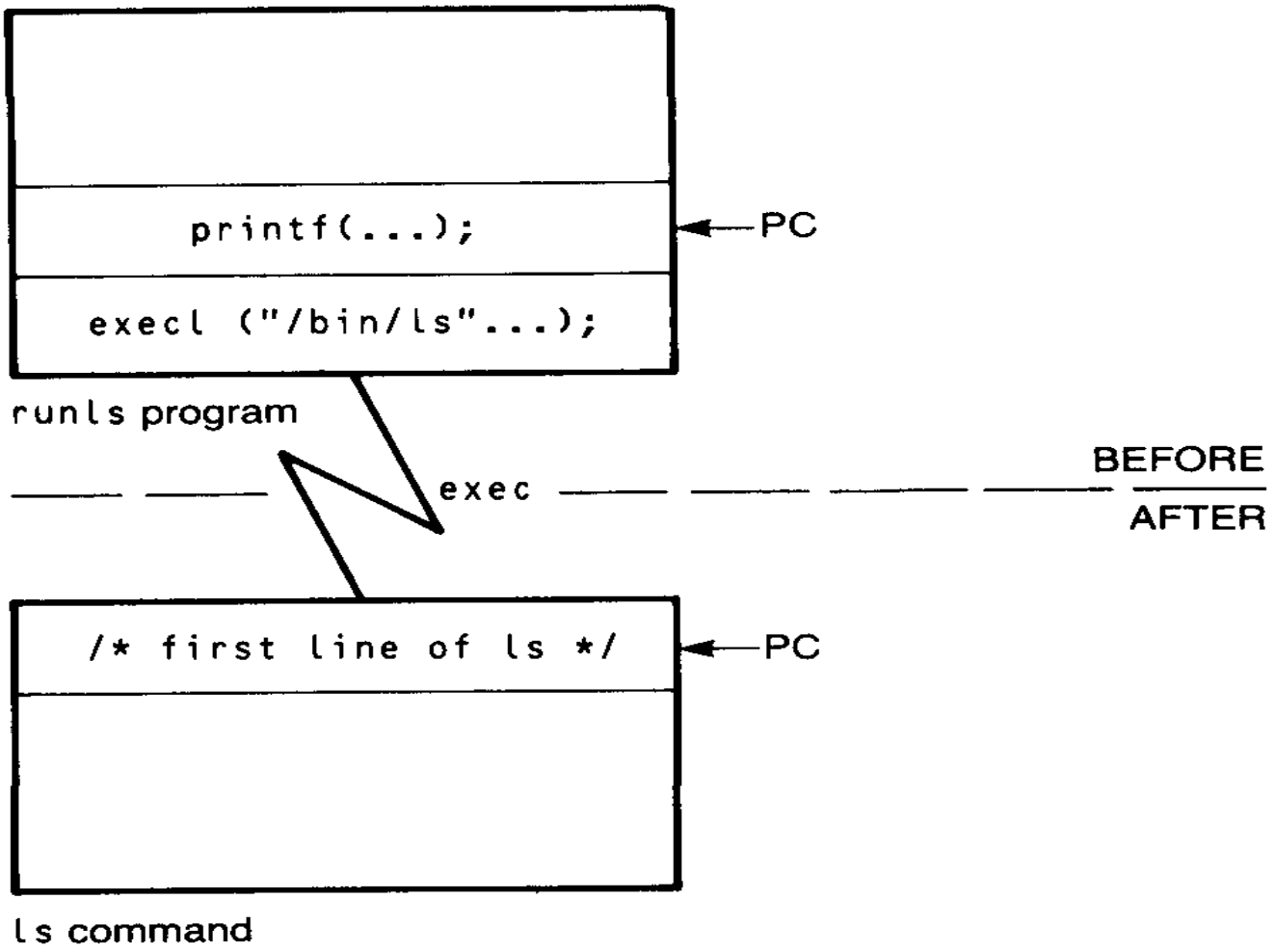
- *arg0* : 프로그램 이름, *arg1~argn* : 인자 목록
- 리턴 값 : 없음 또는 -1 (오류 발생시)
- 이것을 호출한 프로세스는 새로운 프로그램으로 대치됨
- 새로운 프로그램은 main() 함수부터 수행됨
- 프로세스 ID는 변하지 않음

# 예제 프로그램

```
main() {
    char *av[3];
    av[0] = "ls“;  av[1] = "-l"; av[2] = (char *)0;
    printf("executing ls");
    execv("/bin/ls", av);
    printf("execl error");
}
```

```
< amin:os 17 > ls -l
-rwxrwxr-x   1 kmh   sslab   5260 10월  2일  15:31 a.out*
-rw-rw-r--   1 kmh   sslab    117 10월  2일  15:31 exec.c
< amin:os 18 > a.out
executing ls
-rwxrwxr-x   1 kmh   sslab   5260 10월  2일  15:31 a.out*
-rw-rw-r--   1 kmh   sslab    117 10월  2일  15:31 exec.c
```

# Exec...() 호출

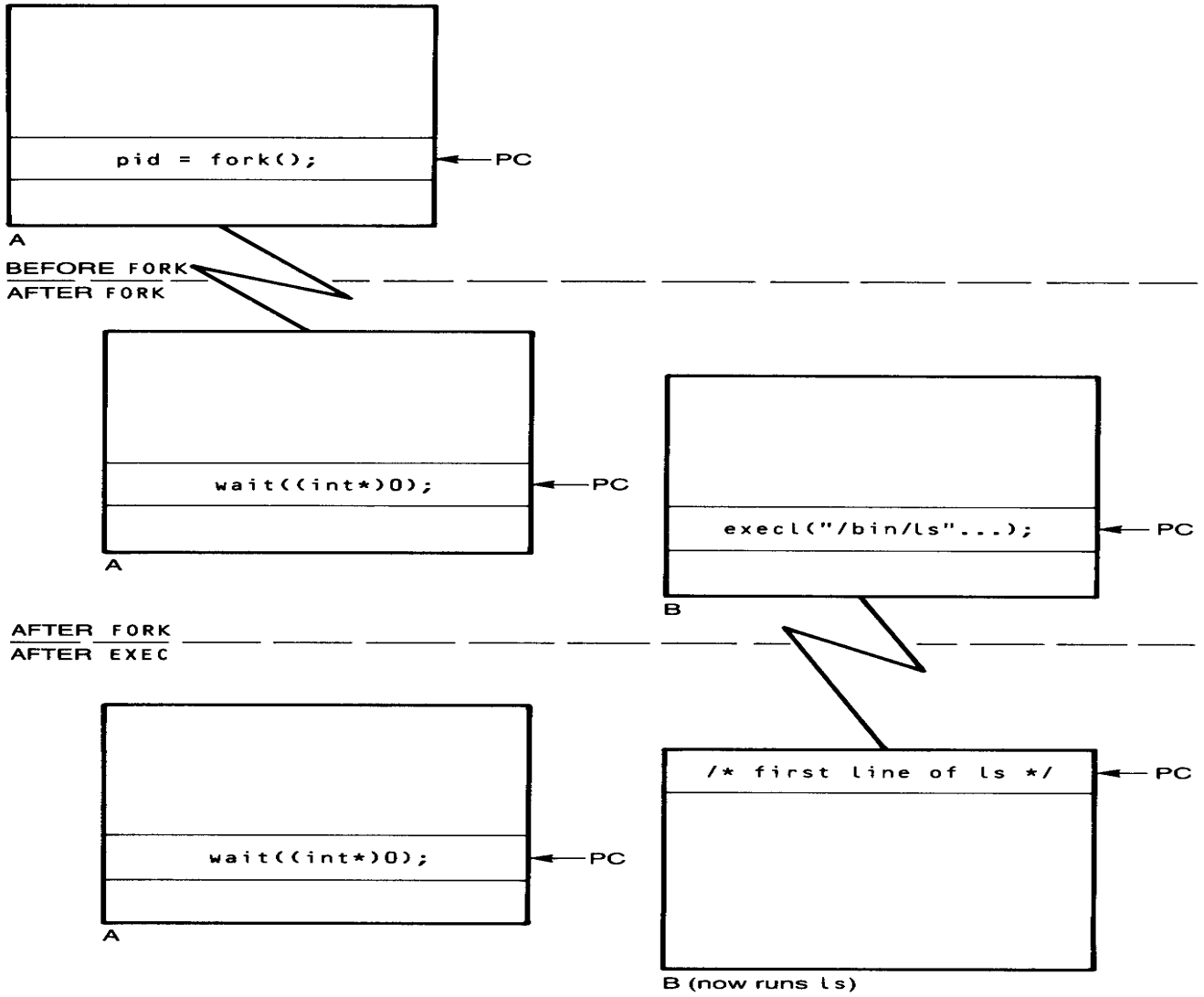

printf(...);
execl ("/bin/ls"...);
PC
runls program
exec
BEFORE
AFTER
/* first line of ls */
PC
ls command

# fork()와 Exec...()

- 보통 이 두 함수는 같이 수행되어 새로운 프로그램을 자식 프로세스로 만듦

```
main() {
    int pid;
    pid = fork();
    if (pid > 0) {
        wait((int *)0);
        printf("ls completed");
        exit(0); }
    if (pid == 0) {
        execl("/bin/ls", "ls", "-l", (char *)0);
        printf("execl error"); }
    printf("fork error");
    exit(1);
}
```

# Fork and Exec Call



# 파이프

- FIFO 이고 데이터를 동기화해서 보냄

```
$ cat temp.c | lp
$ cat [a-zA-Z]* | grep process
```

- 이름 있는 파이프와 이름 없는 파이프가 있음
- 이름 있는 파이프
  - 초기화 : open() 시스템 호출
  - 접근 권한 : 파일 허가와 유사하게 결정됨
- 이름 없는 파이프
  - 초기화 : pipe() 시스템 호출
  - 접근 권한 : 관계가 있는 프로세스로 국한됨(자식/부모 프로세스)
- read, write, close 시스템 호출을 사용할 수 있음

# 이름 없는 파이프 - pipe()

- 함수 원형

```
#include <unistd.h>

int pipe(int filedes[2]);
```

- 프로세스간 채널을 생성함
- 리턴 값 : 0, -1
- *filedes* 인자를 통해 두 개의 파일 지시자가 리턴됨
  - filedes[0] : 읽기를 위해 열림
  - filedes[1] : 쓰기를 위해 열림

# 예제 프로그램

```
#include <stdio.h>
#define MAX 16
char *msg1 = "hello, world #1";
char *msg2 = "hello, world #2";
char *msg3 = "hello, world #3";
main() {
    char inbuf[MAX];
    int fd[2], j;
    if (pipe(fd) < 0) exit(1);
    write(fd[1], msg1, MAX);
    write(fd[1], msg2, MAX);
    write(fd[1], msg3, MAX);
    for (j = 0; j < 3; j++)
    { read(fd[0], inbuf, MAX); printf("%s\n", inbuf); }
    exit(0);
}
```

# 예제 프로그램 결과

```
enterprise:[~/os ]56 a.out
hello, world #1
hello, world #2
hello, world #3
enterprise:[~/os ]57
```

# 예제 프로그램 2

```
#include <stdio.h>
#define MAX 16
char *msg1 ="hello, world #1";
char *msg2 ="hello, world #2";
char *msg3 ="hello, world #3";
main() {
    char inbuf[MAX];  int fd[2], j, pid;
    if (pipe(fd) < 0) exit(1);
    if ((pid = fork()) < 0) exit(2);
    if (pid > 0){
       write(fd[1], msg1, MAX); write(fd[1], msg2, MAX);
       write(fd[1], msg3, MAX); wait((int *)0); }
    if (pid == 0) {
       for (j = 0; j < 3; j++) { read(fd[0], inbuf, MAX);
          printf("%s\n", inbuf);  }
    }
}
```

# 예제 프로그램 2 결과

```
enterprise:[~/os ]56 a.out
hello, world #1
hello, world #2
hello, world #3
enterprise:[~/os ]57
```

# 신호

- 예외적인 조건 또는 비정상적인 사건에 의해 생성
  - 인터럽트 (ctrl-c) : 커널이 SIGINT이라는 신호를 프로세스에게 보냄
  - 잘못된 연산 : 커널이 SIGKILL이라는 신호를 프로세스에게 보냄

```
$ test
Illegal instruction - core dumped
```

  - kill 명령 : 커널이 신호를 프로세스에게 보냄

```
$ test&
1024
$ kill -9 1024
1024 terminated
```

# signal() 함수

- 프로세스가 신호를 받으면 디폴트로 그 프로세스는 종료함
- signal() 함수를 사용하여 신호를 받았을 때의 동작을 지정할 수 있음

# signal() 함수

- 함수 원형

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

  - sig 신호가 발생하면 func() 함수를 수행하게 함
  - 리턴 값 : 함수 포인터

- 자주 사용되는 매크로

```
#define   SIG_DFL  ((void (*)(int)) 0)
#define   SIG_IGN  ((void (*)(int)) 1)
```

# 예제 프로그램

```
/* Using a signal handler to catch a control-c. */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#define   MAXSTRING   100
void  cntrl_c_handler(int sig);
int   fib(int n);
int main(void){
   int   i;
   signal(SIGINT, cntrl_c_handler);
   for (i = 0; i < 46; ++i)
      printf("fib(%2d) = %d\n", i, fib(i));
      return 0;
}
```

# 예제 프로그램

```
void cntrl_c_handler(int sig){
   char   answer[MAXSTRING];
   printf("\n\n%s%d\n\n%s",
      "Interrupt received! Signal = ", sig,
      "Do you wish to continue or quit? ");
   scanf("%s", answer);
   if (*answer == 'c')
      signal(SIGINT, cntrl_c_handler);
   else
      exit(1);
}
```

# 상태 리턴

- main() 함수의 리턴 값은 운영체제가 사용함
- main() 함수가 리턴한 값은 쉘에서 status라는 쉘 변수를 보면 알 수 있음
- 예

```
main() {
    . . .
    exit(5) ;    // return 5;
}
```

```
$ a.out
$ echo $status
5
$
```