41865 Unix Systems

실습 10-1

외부 명령어 실행

Jaehong Shim

Dept. of Computer Engineering

조선대학교
CHOSUN UNIVERSITY

cmd: 외부 명령어의 실행

- 지금껏 우리가 구현한 내부 명령어
  - cat, cd, chmod, cp, date, echo, help, hostname, id, ln, ls, mkdir, mv, pwd, exit, rm, rmdir, touch, uname, whoami
- 위의 내부 명령어를 제외한 다른 외부 명령어들을 cmd에서 실행시킴
  - gcc, make, more, vi, man, find, grep, tail,
  - ps, kill, who, cal [2011], clear, whereis
  - 또는 본인이 직접 작성한 프로그램 등을 cmd에서 실행시켜 줌
- 강의노트 8장 및 교재 8장 참조

proc_cmd() 함수의 수정

```
void proc_cmd(void)
{
   int  k;

   for (k = 0; k < num_cmd; ++k) {
        if (EQUAL(cmd, cmd_tbl[k].cmd)) {     // 명령어 찾았음
            // 명령어 인자와 옵션 체크
            if ((check_arg(cmd_tbl[k].argc) < 0) ||
                (check_opt(cmd_tbl[k].opt)  < 0))
                print_usage("사용법: ", cmd_tbl[k].cmd, cmd_tbl[k].opt,
                            cmd_tbl[k].arg);
            else
                cmd_tbl[k].func();            // 명령어 처리함수 호출
            return;
        }
   }
   // 내부 명령어 찾지 못한 경우
   printf("%s : 지원되지 않는 명령어입니다.\n", cmd);
   run_cmd();  // 새로 추가
}
```

외부 명령어 실행: run_cmd()

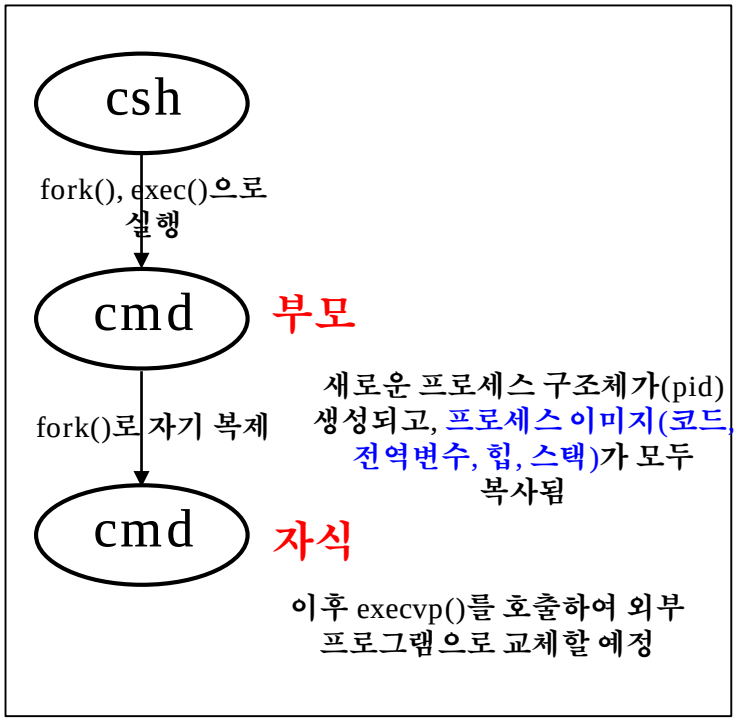
```
// fork() exec() waitpid() 함수들의 헤드파일을 찾아 include시킨다.
```

```
// proc_cmd() 함수 앞에 아래 함수를 입력한다.

// run_cmd():
// 외부 명령어를 실행해 주는 함수
// cmd 자신을 복제하여 부모 프로세스는 자식이 종료할 때까지 대기하고,
// 자식 프로세스는 외부 명령어 프로그램으로 대체하여 실행함
void
run_cmd(void)
{
   pid_t pid;

   // 뒤 페이지들의 코드들을 여기에 삽입할 것
}
```

새로운 프로세스 생성하기: fork()

부모 cmd

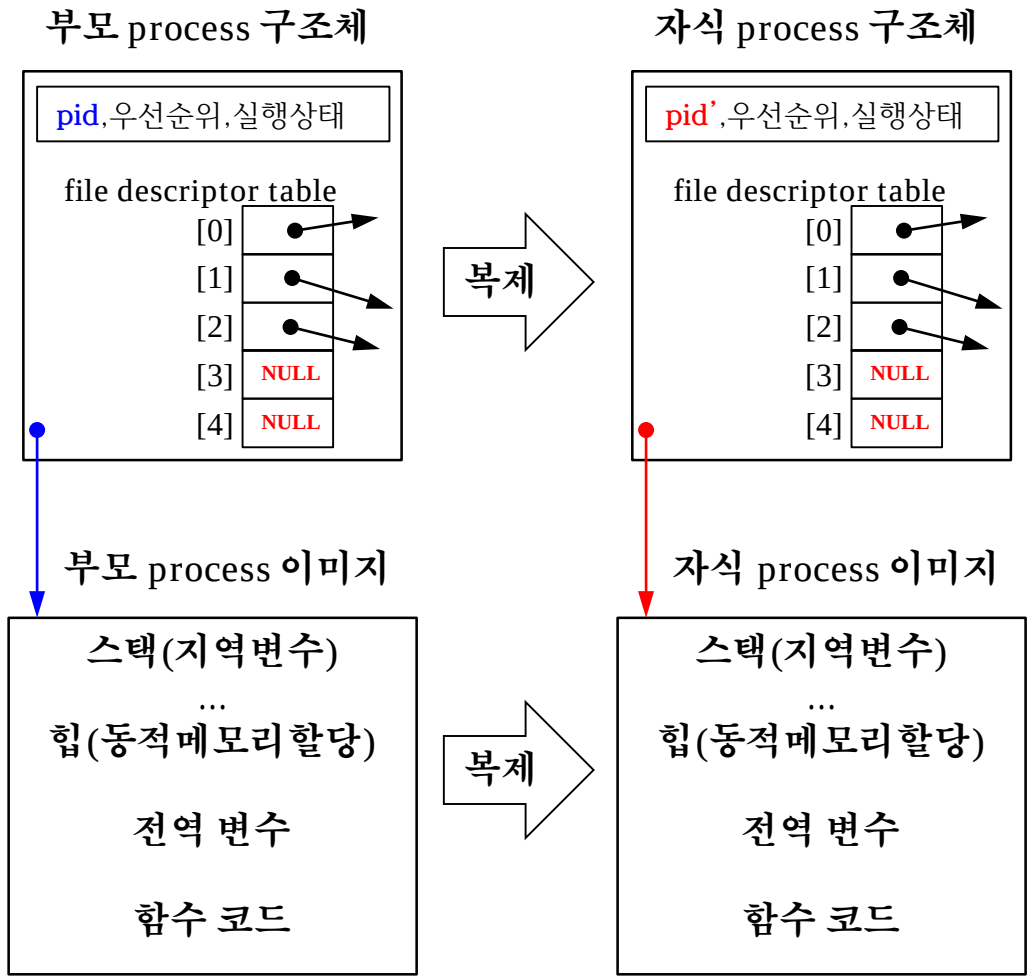
```
// fork()를 호출하여 기존에 실행 중인 cmd를
// 복제하여 새로운 프로세스를 생성한다.
// 에러가 발생했으면 에러 원인 출력하고 리턴함
// 강의노트 8장 p.4

if ( (pid = fork()) < 0 )
   PRINT_ERR_RET();

// 이 순간부터 동일한 프로그램(cmd) 2개가
// 동시에 실행되고 있으며,
// 프로그램 내에서 실행되는 위치도 바로 이
// 위치에서 계속 실행된다.

// fork()의 리턴 값(pid)이
// 0이면 자식 프로세스(새로 복제된)이고,
// 0보다 크면 부모 프로세스임(원래 돌던
   프로그램)
```



자식 cmd (코딩하지 말고 참고만 할 것)

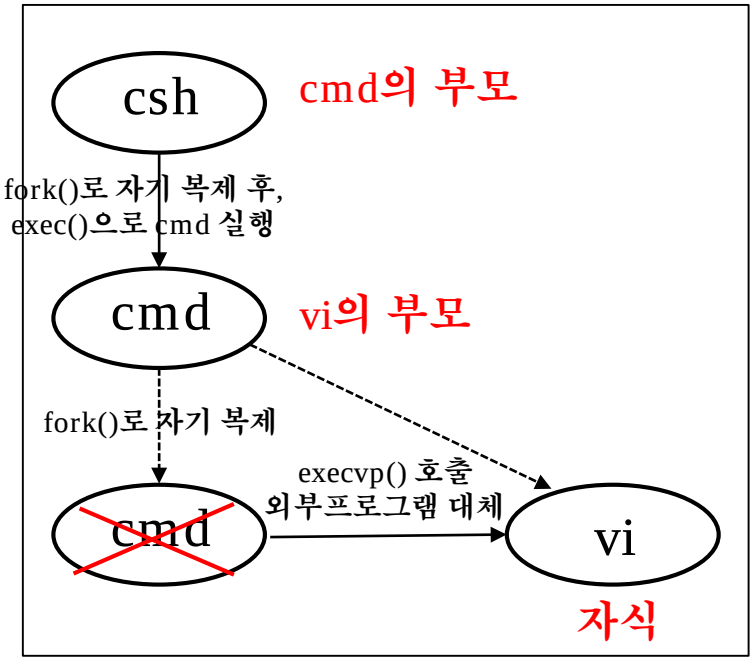
```
if ( (pid = fork()) < 0 )
   PRINT_ERR_RET();
else if (pid == 0)
     ... // 자식 프로세스
else     // pid > 0
     ... // 부모 프로세스
```

fork() 한 이후의 cmd 프로세스의 메모리 모습

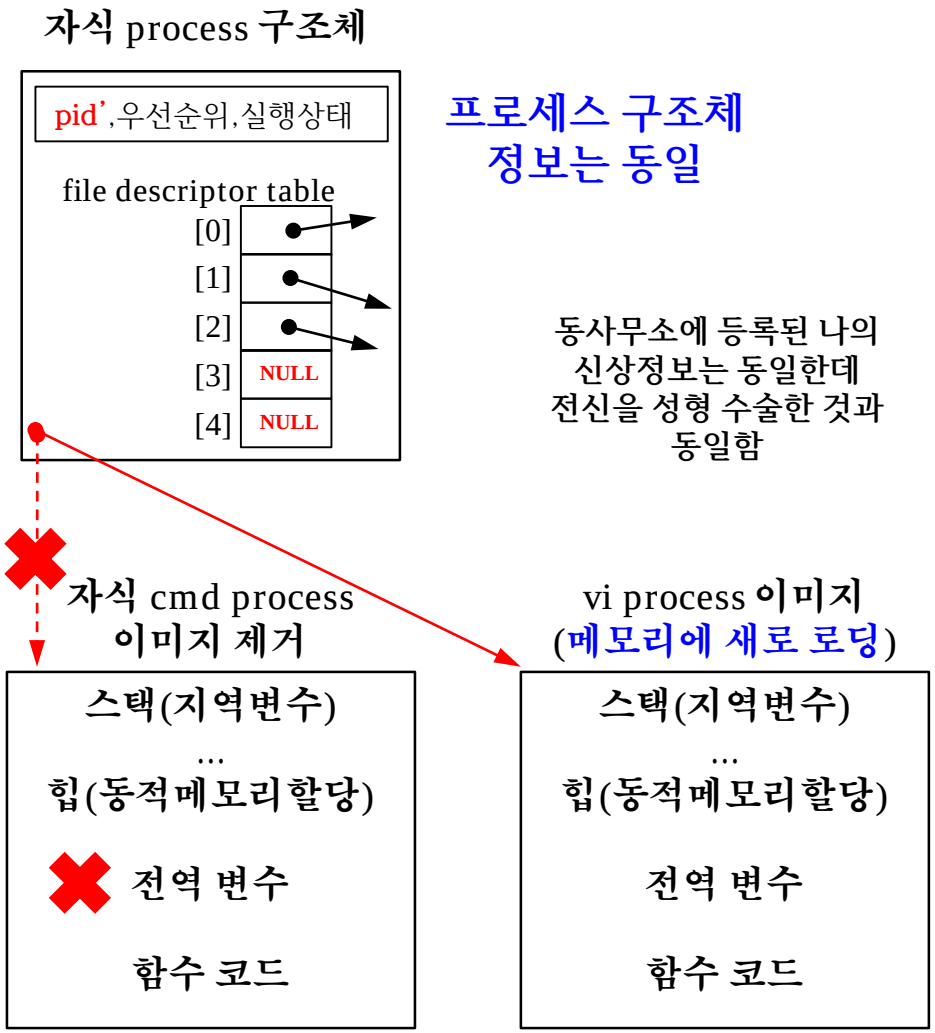


프로세스 복제 후 process id와 process 이미지 포인터 등 일부 멤버들 수정

자식 프로세스: exec() 호출하여 외부 명령어로 대체



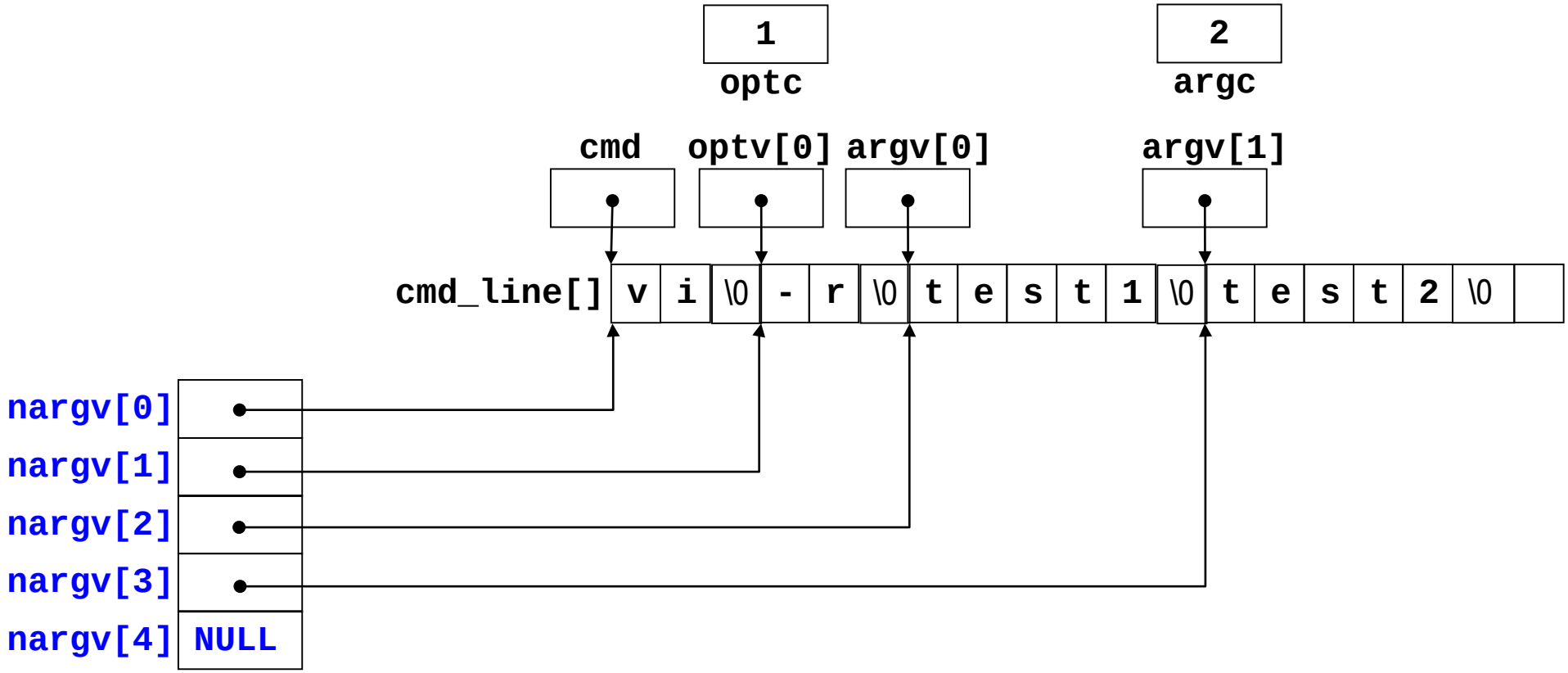
프로세스 구조체 정보(pid)는 동일한데,
프로세스 이미지(코드, 전역변수, 힙,
스택)만 외부 프로그램 vi로 교체됨



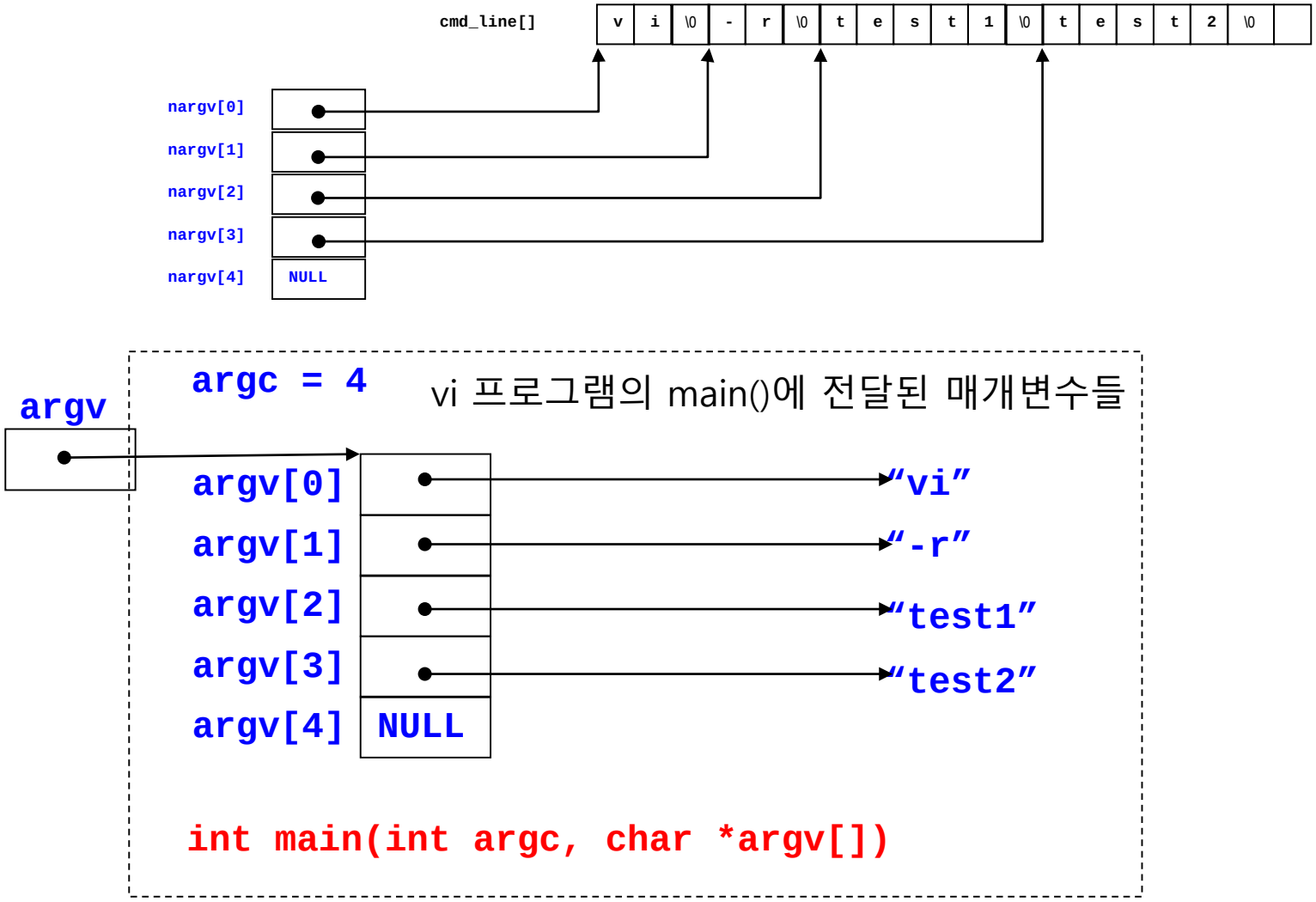
자식 프로세스: 외부 명령어로 대체 과정

```
// 자식 프로세스는 외부 프로그램으로 자기 자신을 대체 한다. 이 과정에서 에러가
// 발생했으면 에러 원인 출력하고 스스로 종료한다.
else if (pid == 0) {   // 자식 프로세스
   int   i, cnt = 0;
   char *nargv[100];
   // 기존의 명령어 cmd, 옵션 optv[], 인자 argv[]에 저장된 포인터 값을
   // 순서대로 nargv[]에 연속적으로 저장함; nargv[]의 마지막은 NULL이어야 함
   nargv[cnt++] = cmd;                  // 기존의 cmd를 nargv[]에 저장
   for (i = 0; i < optc; ++i)
        nargv[cnt++] = optv[i];         // 기존의 optv[]를 nargv[]에 저장
   for (i = 0; i < argc; ++i)
        nargv[cnt++] = argv[i];         // 기존의 argv[]를 nargv[]에 저장
   nargv[cnt++] = NULL;                 // nargv[]의 마지막은 NULL이어야 함

   if (execvp(cmd, nargv) < 0) {// 외부 프로그램으로 대체: 강의노트 8장 p.34
        // 해당 프로그램이나 접근권한이 없거나 또는 실행 프로그램이 아닌 경우 에러
        perror(cmd);   // 여기서 PRINT_ERR_RET()를 호출하면 안됨
        exit(1);       // 에러 발생 시 자식 프로세스는 종료해야 함
   }
} // 자식 프로세스는 외부 프로그램으로 대체되었기 때문에 이 이후의 코드는 실행되지 않는다.
```

자식 프로세스: nargv[ ]의 값들

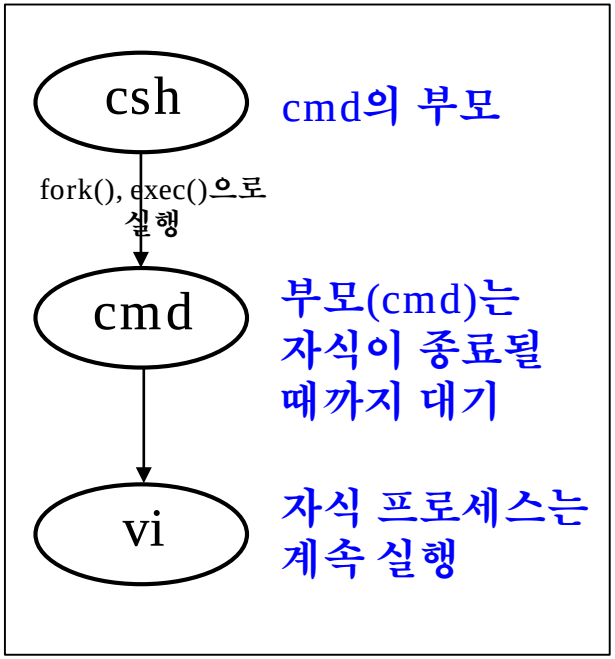


vi 프로그램의 main(int argc, char *argv[ ])에 전달



부모 프로세스: 자식이 종료할 때까지 대기

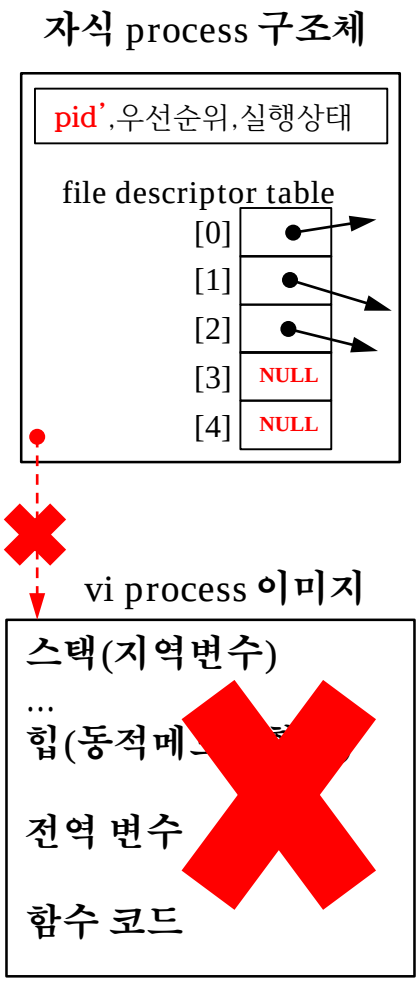
```
// fork()의 리턴 값(pid)이 양수(pid=자식 프로세스의
// ID)이면 부모 프로세스임
// 부모 프로세스(cmd 프로그램)는 자식 프로세스(외부
// 프로그램)가 종료될 때까지 더 이상 실행되지 않고
// 여기서 대기해야 함. 이 과정에서 에러가 발생했으면
// 에러 원인 출력하고 스스로 종료한다.

else { // pid > 0 경우, 즉 부모 프로세스
    // 자식(pid)이 종료될 때까지 대기
    // sleep(60); // 좀비 프로세스 생성을 위해
    if (waitpid(pid, NULL, 0) < 0)
        PRINT_ERR_RET(); // 강의노트 8장 p.21
    // 자식 프로세스가 종료되면 waitpid()는
    // 자식 프로세스의 pid 값을 리턴함
}
// 만약 여기서 부모가 자식 종료를 기다리지 않으면
// 자식은 어떻게 될까? Zombie 프로세스가 됨
// 즉, 죽어서 무덤까지 만들어 주었지만 (시스템 내에
// 종료된 프로세스 이미지 제거), 부모가 사망 통지서를
   수신(waitpid()를 호출하지 않음) 하지 않은 경우임
// (종료된 프로그램의 정보는 계속 시스템에 남아 있음)
```



프로세스 이미지:
프로그램 코드,
전역변수,
힙(동적메모리할당),
스택(지역변수)

Zombie Process

자식이 죽어서 무덤까지 만들어 주었지만 동사무소에 자식의 사망 신고를 하지 않아 자식의 인적 정보가 계속 남아 있는 것과 유사함



프로세스 구조체는 아직 시스템에 남아 있음

자식 프로세스는 메모리에서 사라졌는데 아직 그 프로세스에 대한 정보가 시스템에 남아 있음;
부모 프로세스가 빨리 **waitpid()**를 호출하여 자식 프로세스 구조체도 빨리 제거되게 해야 함

vi process 이미지는 이미 제거되었음