Process

한양대학교 소프트웨어학부

Dept. of Division of Computer Science

Hanyang University

CSE4009
System Programming

Process

- 프로세스는 수행중인 프로그램이다
- 각 프로세스는 code, data, stack 메모리를 사용한다
- 프로세스(child)는 다른 프로세스(parent)가 생성한다
- 두 프로세스는 프로세스간 통신 (IPC: Inter-Process Communication) 기능을 사용하여 통신한다
  - UNIX IPCs: signal, pipe, fifo, socket, …
- 프로세스 관련 시스템 호출
  - fork 자신의 프로세스를 복제하여 child 프로세스를 생성한다
  - exec 자신의 프로세스에 다른 프로그램을 이식한다
  - wait child 프로세스가 종료할 때까지 기다린다
  - exit 자신의 프로세스를 종료한다

Process (Cont'd)

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

pid_t getpid(void);//현재 프로세스의 프로세스ID를 되돌려준다
pid_t getppid(void);          //부모 프로세스의 PID를 되돌려준다
gid_t getgid(void);//현재 프로세스의 실제 프로세스그룹ID를 되돌림
gid_t getegid(void);          //현재 프로세스의 유효 프로세스그룹ID를 되돌림
uid_t getuid(void);//현재 프로세스의 실제 프로세스사용자ID를 되돌림
uid_t geteuid(void);          //현재 프로세스의 유효 프로세스사용자ID를 되돌림
pid_t getsid(pid_tpid);       //pid를 갖는 프로세스의 세션ID를 되돌려 준다
int setgid(gid_tgid);         //실제그룹ID를 변경한다
int setegid(gid_tgid);        //유효그룹ID를 변경한다
int setuid(uid_tuid);         //실제사용자ID를 변경한다
int seteuid(uid_tuid);        //유효사용자ID를 변경한다
pid_t setsid(void); //세션을 만들고 프로세스그룹아이디(GID)를 설정
                              //만약 호출프로세스가 프로세스 리더이면 함수는
                              //실패하고-1 반환
```

System Call – fork & vfork

사용법

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

pid_t fork(void); pid_t vfork(void);
```

자식 process를 생성

Return value
[parent process : child process #]
[child process : 0]

fork: 부모와 자식이 다른 메모리 공간 사용
vfork: 부모와 자식이 같은 메모리 공간 공유
실패경우 1 - 시스템에서 허용하는 프로세스 개수를 초과한 경우
2 – 개별 사용자가 동시에 수행할 수 있는 프로세스 초과

Example – fork #1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char*  argv[])
{
        pid_t pid = fork();
        if (pid > 0) { // getpid() - return current process pid
                printf("Parent Process %d : %d\n", getpid(), pid);
        } else if (pid == 0) {
                printf("Child Process %d\n", getpid());
        } else if (pid < 0) {
                perror("fork error");
                exit(0);
        }
}
```

```
[TA3@localhost lab8]$ ./fork
Parent Process 8028 : 8029
Child Process 8029
[TA3@localhost lab8]$
```

Example – fork & vfork

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int glob = 6;

int main(int argc, char* argv[])
{
        int var = 88;
        pid_t pid;


        printf("before fork:\t");
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
        if ((pid = fork()) < 0) {
                perror("fork error");
                exit(1);
        } else if (pid == 0) { /* child process */
                glob++; /* modify variables */
                var++;
                exit(1);
        } else
                sleep(2); /* parent process */
        printf("after fork:\t");
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
        return 0;
```

```
before fork:    pid = 17076, glob = 6, var = 88
after fork:     pid = 17076, glob = 6, var = 88
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int glob = 6;

int main(int argc, char* argv[])
{
        int var = 88;
        pid_t pid;


        printf("before vfork:\t");
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
        if ((pid = vfork()) < 0) {
                perror("fork error");
                exit(1);
        } else if (pid == 0) { /* child process */
                glob++; /* modify variables */
                var++;
                exit(1);
        } else
                sleep(2); /* parent process */
        printf("after vfork:\t");
        printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
        return 0;
```

```
before vfork:   pid = 17080, glob = 6, var = 88
after vfork:    pid = 17080, glob = 7, var = 89
```

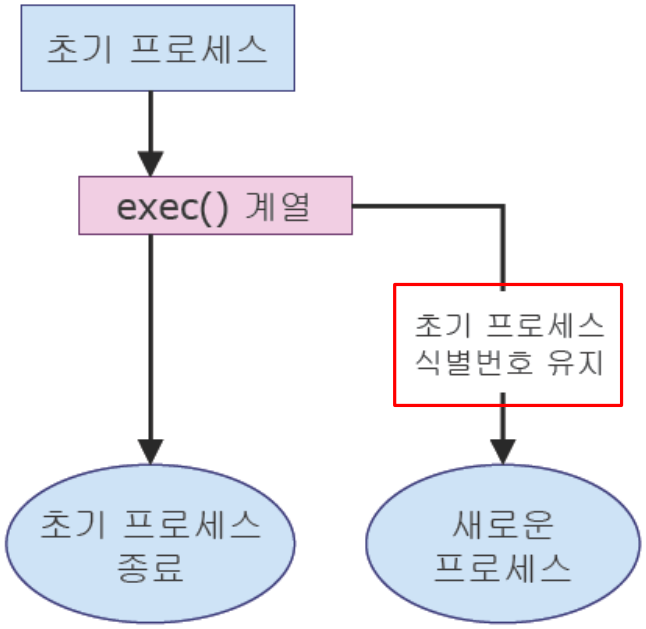
System Call – exec

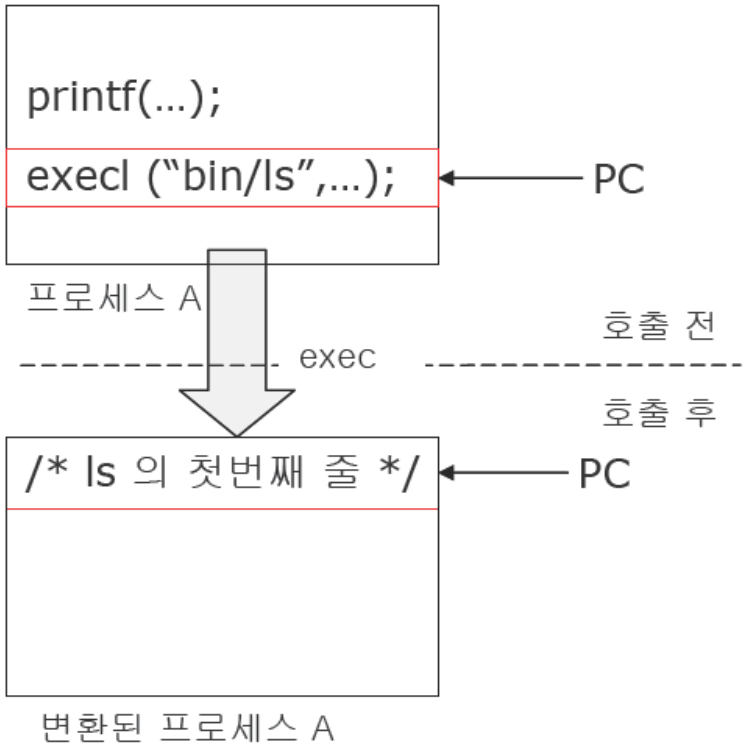
사용법

```
#include <unistd.h>
int execl(const char *path, const char *arg0, ..., const char *argn, (char *)0);
int execlp(const char *file, const char *arg0, ..., const char *argn, (char *)0);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

현재 프로세스 이미지를 새로운 프로세스 이미지로 바꾼다
exec시 새 프로그램의 수행이 시작되어 새 data와 새 stack을 형성한다
exec 이전에 open된 파일 descriptor는 exec후 그대로 남아있다
exec시 이미 open된 파일 descriptor를 close할 수 있다

System Call – exec (cont'd)





System Call – exec (cont'd)

```
#include <unistd.h>

main ()
{
  printf ( "executing ls\n" ) ;

  execl ("/bin/ls", "ls", " -l", (char *) 0) ;

  perror ("execl failed to run ls");
  exit (1);
```

bin이 PATH에 포함된다고 가정

```
execlp ("ls", "ls", " -l", (char *) 0) ;

char * const av[]={"ls", " -l", (char *) 0}
execv ("/bin/ls", av) ;

char * const av[]={"ls", " -l", (char *) 0}
execvp ("ls", av) ;
```

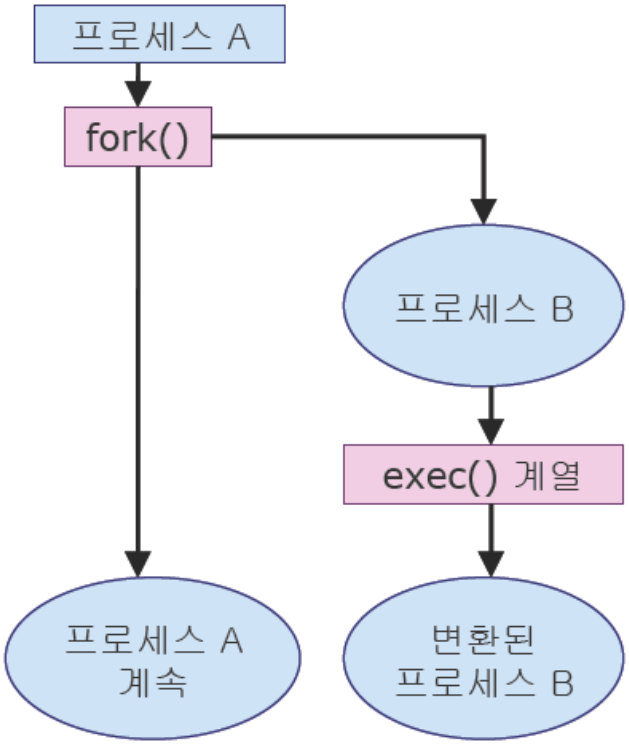
Example – exec #1

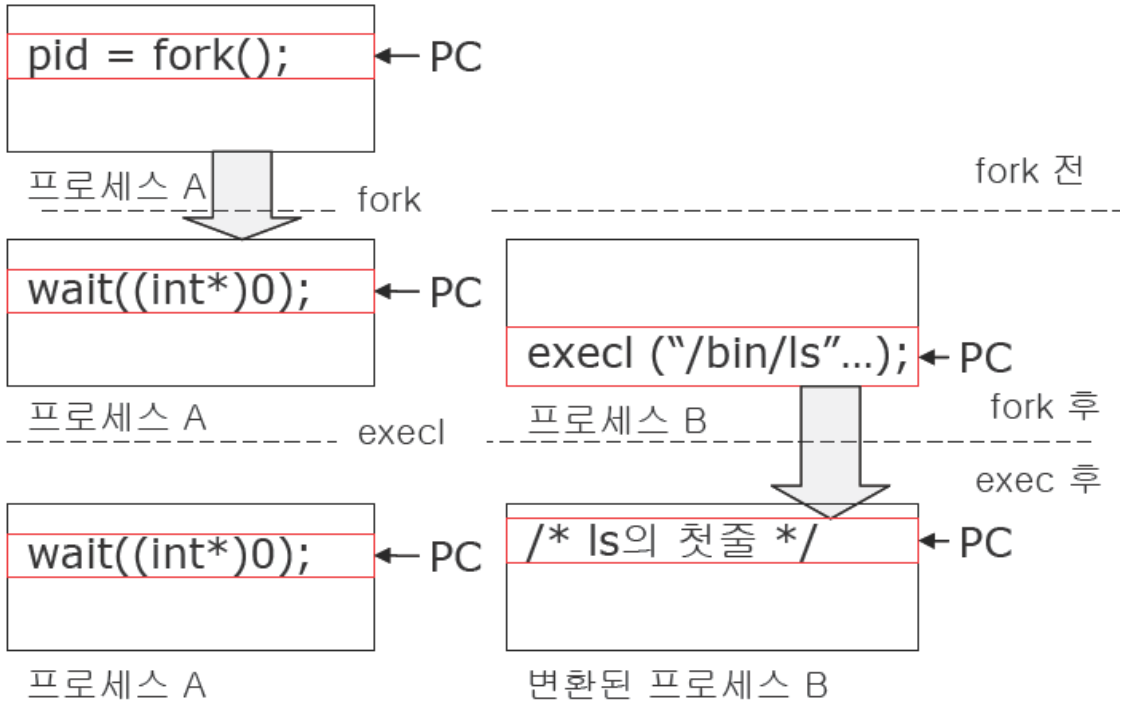
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main(void)
{
        printf("Original Process: %d\n", getpid());
        sleep(1);
        execl("/bin/sh", "sh", NULL);
        exit(0);
}
```

```
[TA3@localhost lab8]$ ./exec
Original Process: 17270
sh-4.1$ echo $$   // pid 확인
17270
sh-4.1$ exit
exit
[TA3@localhost lab8]$
```

fork & exec


프로세스 A
fork()
프로세스 B
exec() 계열
프로세스 A
계속
변환된
프로세스 B


pid = fork();
PC
프로세스 A
fork
fork 전
wait((int*)0);
PC
execl ("/bin/ls"...);
PC
프로세스 A
execl
프로세스 B
fork 후
exec 후
wait((int*)0);
PC
/* ls의 첫줄 */
PC
프로세스 A
변환된 프로세스 B

System Call – exit

사용법

```
#include <stdlib.h>

void exit(int status);
```

프로그램을 정상 종료 시킨다
status:
성공시: 0
실패시: non-zero

No return value

System Call – wait

사용법

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

pid_t wait(int *status);
```

자식 프로세스가 종료할 때까지 해당영역에서 부모 프로세스가 sleep 모드로 기다린다.
부모 프로세스가 자식 프로세스보다 먼저 종료되어서
자식 프로세스가 고아 프로세스가 되는 것을 방지하기 위한 목적
status : 자식프로세스의 종료값을 받아온다
Return value
[성공시 자식pid]
[실패시 -1]

System Call – waitpid

사용법

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pit_t pid, int *status, int option)
```

주어진 pid번호의 자식프로세스 종료를 기다린다
pid값
- -1 : 임의의 자식 프로세스를 기다린다 wait와 동일
- 0 : 프로세스 그룹ID가 호출 프로세스의 ID와 같은 자식 프로세스를 기다린다
- pid> 0 : 프로세스 ID가 pid의 값과 같은 자식 프로세스를 기다린다

options : 0 또는 다음값들과의 OR
- WNOHANG : 자식이 종료되지 않았더라도, 부모 프로세스는 블록 되지 않고 다른 작업을 수행한다
- WCONTINUED : 프로세스가 재개된 경우에도 보고한다
- WUNTRACED : 프로세스 수행이 중지된 경우에도 보고한다

Return value
- [성공시:0]
- [실패시:1]

Example – waitpid (cont'd)

status : 프로세스의 상태를 가져오기 위해서 사용
–status 가 NULL 이 아닐 경우 status가 가리키는 위치에 프로세스의 상태정보를 저장한다
–다음의 매크로들을 통해서 상태정보를 가져올 수 있다

매크로	설 명
WIFEXITED(status)	자식 프로세스가 정상적으로 종료된 경우 true 반환한다.
WEXITSTATUS(status)	WIFEXITED가 true인 경우, 종료된 자식 반환 코드의 최하위 8비트의 값을 반환한다.
WIFSIGNALED(status)	자식 프로세스가 시그널을 받아서 비정상 적으로 종료된 경우 true를 반환한다.
WTERMSIG(status)	WIFSIGNALED가 true를 반환한 경우, 자식 프로세스를 종료시킨 시그널의 번호를 반환한다.
WIFSTOPPED(status)	자식 프로세스의 수행이 중지된 경우 true를 반환한다. (WUNTRACED옵션이 설정된 경우)
WSTOPSIG(status)	WIFSTOPPED가 true를 반환한 경우, 자식 프로세스를 중지시킨 시그널의 번호를 반환한다.
WIFCONTINUED(status)	프로세스의 수행이 재개된 경우 true를 반환한다. (WCONTINUED 옵션이 설정된 경우)
WCOREDUMP(status)	core dump라고 불리는 메모리 덤프 파일이 생성된 경우에만 true를 반환한다. (프로세스가 죽은 원인을 분석하는데 유용)

Example – exec #2-1 (echoall)

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[], char *envp[])
{
        int i;
        char **ptr;
        extern char **environ; // 시스템 내의 환경변수를 가져올때

        for (i = 0; i < argc; i++)
                printf("argv[%d]: %s\n", i, argv[i]);

        for (ptr = environ; *ptr !=0; ptr++)
                printf("%s\n", *ptr);

        exit(0);
}
```

// 위 파일로 echoall 실행파일을 만든 후 다음 페이지의 실습 진행

Example – exec #2-2 (exectest)

execlp 사용하기 위해 PATH 추가

```
[TA3@localhost lab8]$ PATH=$PATH:/home/CSE4009/TA3/lab8
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void)
{
        pid_t pid;
        if ((pid = fork()) < 0) {
                perror("fork error\n");
                exit(1);
        } else if (pid == 0) {
                if (execl("/home/CSE4009/TA3/lab8/echoall", "echoall", "myarg1", "MYAGR2", (char*)0) < 0) {
                        fprintf(stderr, "execl error\n");
                        exit(1);
                }
        }

        if (waitpid(pid, NULL, 0) < 0) {
                perror("wait error\n");
                exit(1);
        } else if (pid > 0) {
                printf("\n\n-----------------------------------------\n\n");
                if (execlp("echoall", "echoall", "only 1 arg", (char*)0) < 0) {
                        fprintf(stderr, "execlp error\n");
                        exit(1);
                }
        }
        exit(0);
}
```

Environment

사용법

```
#include <stdlib.h>
char *getenv(const char *name);
int putenv(char *string);
int setenv(const char *name, const char *value, int overwrite);
```

getenv()

name 이름을 가지는 환경변수에 저장된 값을 읽어온다
만약 일치하는 name 을 가지는 환경변수가 있다면"값"을 되돌려
주고 없다면NULL 을 반환한다

putenv()

환경변수를 추가하거나, 기존의 환경변수의 값을 변경할 수 있다
성공할 경우0을 실패했을 경우에는 -1을 반환하며,
적당한 errno값을 설정한다

setenv()

name 이 존재하지 않을 경우 값 value 를 가지는 환경변수 name 을 추가시킨다
만일 name 이 환경변수에 존재할 경우, overwrite 가0이 아니라면
값은value로 변경된다 만약overwrite 가0 이라면 값은 바뀌지 않게 된다
성공할 경우0을 반환하며, 환경변수를 위한 공간이 충분하지 않다면-1 을 반환

Example #1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
        int pid;
        int status;
        int cpid;
        pid = fork();
        if (pid == 0) {
                sleep(5);
                printf("I will be back %d\n", getpid());
                exit(1);
        } else if (pid > 0) {
                printf("I'm parent %d\n", getpid());
                printf("Press any key and wait\n");
                getchar();

                // 자식 프로세스를 wait한다.
                // 자식 프로세스의 PID, 종료상태를 얻어온다.
                cpid = wait(&status);
                printf("Success: Waiting for child process");

                printf("PID: %d\n", cpid);
                printf("ExitValue: %d\n", WEXITSTATUS(status));
                printf("ExitStat: %d\n", WIFEXITED(status));
        } else {
                perror("fork error: ");
        }
}
```

```
[TA3@localhost lab8]$ ./ex1
I'm parent 25739
Press any key and wait
I will be back 25740

Success: Waiting for child processPID: 25740
ExitValue: 1
ExitStat: 1
[TA3@localhost lab8]$
```

Example #2

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>

int main()
{
        int pid;
        int status;
        pid = fork();
        if (pid < 0) {
                perror("fork error: ");
                exit(1);
        }
        if (pid == 0) {
                printf("I'm Child\n");
                sleep(5);
                exit(123);
        } else {
                printf("Parent: waiting for child(%d)\n", pid);
                waitpid(pid, &status, 0);
                if (WIFEXITED(status)) {
                        printf("Exit OK.\n");
                        printf("Return Value: %d\n", WEXITSTATUS(status));
                } else if (WIFSIGNALED(status)) {
                        printf("Signal OK.\n");
                        printf("Singnal Number: %d\n", WTERMSIG(status));
                }
        }
        return 0;
}
```

```
[TA3@localhost lab8]$ ./ex2
Parent: waiting for child(25775)
I'm Child
Exit OK.
Return Value: 123
[TA3@localhost lab8]$
```

Example #3

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main()
{
        int pid;
        pid = fork();
        if (pid == 0) {
                sleep(2);
                printf("My pid(Child): %d\n", getpid());
                printf("My Parent pid: %d\n", getppid());
        } else if (pid > 0) {
                printf("MY pid(Parent): %d\n", getpid());
                sleep(5);
        } else {
                perror("forkerror: ");
                exit(0);
        }
        return 1;
}
```

```
[TA3@localhost lab8]$ ./ex3
MY pid(Parent): 25816
My Child pid: 25817
My pid(Child): 25817
My Parent pid: 25816
[TA3@localhost lab8]$
```

Example #4 - 1

```c
#include <stdio.h>
#include <stdlib.h>
extern char **environ;

int main(int argc, char* argv[], char* envp[])
{
        int i;
        for (i = 0; argv[i] != NULL; i++)
                printf("argv[%d] = %s\n", i, argv[i]);
        printf("\n");

        for (i = 0; envp[i] != NULL; i++)
                printf("envp[%d] = %s\n", i, envp[i]);
        printf("\n");

        for (i = 0; environ[i] != NULL; i++)
                printf("environ[%d] = %s\n", i, environ[i]);
        printf("\n");

        printf("HOME = %s\n", getenv("HOME"));
        putenv("HOME=/home/CES4009/TA3/lab8");
        printf("HOME = %s\n", getenv("HOME"));
        return 0;
}
```

Example #4 - 2

```
[TA3@localhost lab8]$ ./ex4 arg1 arg2 arg3
argv[0] = ./ex4
argv[1] = arg1
argv[2] = arg2
argv[3] = arg3

envp[0] = HOSTNAME=localhost.localdomain
envp[1] = TERM=xterm-256color
envp[2] = SHELL=/bin/bash
envp[3] = HISTSIZE=1000
envp[4] = SSH_CLIENT=166.104.245.205 54623 22
envp[5] = QTDIR=/usr/lib64/qt-3.3
envp[6] = OLDPWD=/home/CSE4009/TA3
envp[7] = QTINC=/usr/lib64/qt-3.3/include
envp[8] = SSH_TTY=/dev/pts/0
envp[9] = LC_ALL=ko_KR.UTF-8
envp[10] = USER=TA3
```

```
environ[19] = SHLVL=1
environ[20] = HOME=/home/CSE4009/TA3
environ[21] = LOGNAME=TA3
environ[22] = QTLIB=/usr/lib64/qt-3.3/lib
environ[23] = CVS_RSH=ssh
environ[24] = CLASSPATH=.
environ[25] = SSH_CONNECTION=166.104.245.205 54623 166.104.229.196 22
environ[26] = LESSOPEN=||/usr/bin/lesspipe.sh %s
environ[27] = G_BROKEN_FILENAMES=1
environ[28] = _=./ex4

HOME = /home/CSE4009/TA3
HOME = /home/CES4009/TA3/lab8
[TA3@localhost lab8]$ cd ~
[TA3@localhost ~]$
```

Example #5 - 1 (openfexec)

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int fd = 3;
        long pos;

        pos = lseek(fd, 0l, SEEK_CUR);
        printf("\tpos in openfexec(): is %ld\n", pos);

        pos = lseek(fd, 50l, SEEK_CUR);
        printf("\tNew pos after lseek() in openfexec() is %ld\n\n", pos);
        exit(pos < 0 ? !0 : 0); // return non-zero if not OK
}
```

// openfexec 실행파일을 만든 후 openf로 test한다.

Example #5 - 2 (openf)

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main()
{
        int fdes, pid, excode;
        long pos;
        fdes = open("openf.c", O_RDONLY); // open source file
        printf("fdes = %d\n", fdes);

        pos = lseek(fdes, 20l, SEEK_SET);
        printf("Current position before fork() is %ld\n", pos);
        if (!fork()) { // child process
                pos = lseek(fdes, 40l, SEEK_SET);
                printf("Current position in child after fork() is %ld\n", pos);
                exit(1);
        } else { // parent process
                wait((int*)0); // wait for 1st child to terminate
                pos = lseek(fdes, 0l, SEEK_CUR);
                printf("Current position in parent after fork() is %ld\n\n", pos);
        }
```

Example #5 - 3 (openf)

```
25
26          if (!fork()) {  // fork again -- child process
27                  execl("./openfexec", 0); // overlay with another program
28                  printf("It is an error to print this line out\n");
29          }
30
31          // parent process -- no need for else
32          wait(&excode);
33          pos = lseek(fdes, 0l, SEEK_CUR);
34          printf("Current pos in parent after exec() is %ld\n", pos);
35          printf("Exitcode of a child = %d\n\n", WEXITSTATUS(excode));
36
37          if (pid = fork()) { // fork re-again -- parent process
38                  waitpid(pid, &excode, 0);
39                  printf("Exitcode of a specific child = %d\n", WEXITSTATUS(excode));
40                  exit(0);
41          }
42          execl("./openfexec", 0);
43          printf("It is an error to print this line out\n");
44  }
```

Example #5 - 4

```
[TA3@localhost lab8]$ ./openf
fdes = 3
Current position before fork() is 20
Current position in child after fork() is 40
Current position in parent after fork() is 40

        pos in openfexec(): is 40
        New pos after lseek() in openfexec() is 90

Current pos in parent after exec() is 90
Exitcode of a child = 0

        pos in openfexec(): is 90
        New pos after lseek() in openfexec() is 140

Exitcode of a specific child = 0
[TA3@localhost lab8]$
```

Q & A

- Thank you :)