Linux System Programming #5-2
Lecture Notes

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

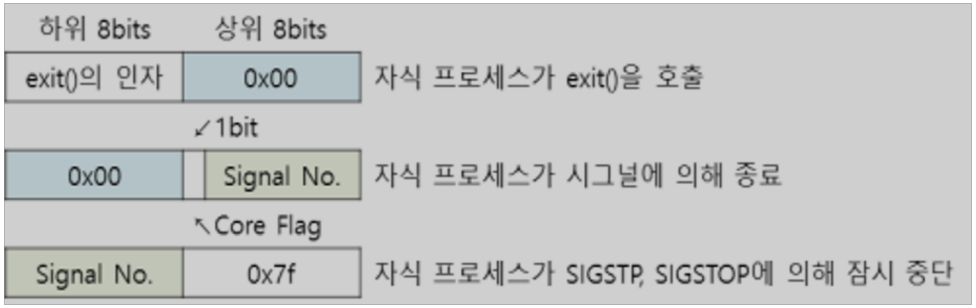
Jiman Hong

jiman@acm.org

wait(2), waitpid(2)

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *statloc);
pid_t waitpid(pid_t pid, int *statloc, int options);
```
리턴 값: 성공 시 프로세스 ID, 에러 시 –1(waitpid()의 경우, WNOHANG 옵션으로 실행되었고자식 프로세스가 종료되지 않았을 때 0을 리턴)

- **프로세스의 종료 상태를 회수하는 시스템호출 함수**
- **자식 프로세스의 종료가 언제 발생할지 부모 프로세스는 알 수 없으므로 커널은 시그널을 통해 부모 프로세스에게 알림**
  - 자식 프로세스가 종료했을 때, 커널은 부모 프로세스에게 SIGCHLD 시그널을 보냄
  - 부모 프로세스는 이 시그널을 무시할 수도 있고, 핸들러 함수(handler function)를 등록하여 핸들러에게 규정된 동작을 하도록 할 수 있음.
- **부모가 자식의 종료를 기다렸다가 종료 상태를 statloc 포인터가 가리키는 곳에 저장하고 종료된 자식 프로세스의 pid값을 리턴**
- **statloc 인자의 종료 상태 값**



두 함수로부터 얻은 종료 상태를 조사하는데 쓰이는 매크로

매크로	내용
WIFEXITED(status)	자식 프로세스가 정상적으로 종료되었으면 참
WIFEXITEDSTATUS(status)	exit()의 인자에서 하위 8비트 값을 리턴
WIFSIGNALED(status)	자식 프로세스가 시그널을 받았으나 그것을 처리하지 않아 비정상적으로 종료되었으면 참
WIFTERMSIG(status)	시그널 번호를 리턴
WCOREDUMP(status)	코어 파일이 생성된 경우에 참 값을 리턴
WIFSTOPPED(status)	자식 프로세스가 현재 중지 상태이면 참
WSTOPSIG(status)	실행을 일시 중단시킨 시그널 번호를 리턴
WIFCONTINUED(status)	자식 프로세스가 작업 제어 중지 이후 실행이 재개되었으면 참

waitpid()의 pid 인자에 따른 용도

pid	용도
pid == -1	임의의 자식 프로세스를 기다리며 이 경우 waitpid() 함수는 wait() 함수와 동일함
pid > 0	프로세스 ID가 pid인 한 자식 프로세스를 기다림
pid == 0	프로세스 그룹 ID가 호출한 프로세스의 것과 동일한 임의의 자식 프로세스를 기다림
pid < -1	프로세스 그룹 ID가 pid의 절대 값과 같은 임의의 자식 프로세스를 기다림

waitpid()의 options 인자에 사용할 수 있는 상수들

options	용도
WCONTINUED	pid로 지정된 자식들 중 중지되었다가 실행을 재개한 이후 상태가 아직 보고되지 않은 임의의 자식도 리턴함
WNOHANG	pid로 지정된 자식 프로세스의 종료 상태를 즉시 회수할 수 없는 상황이라고 하여도 waitpid() 호출이 차단되지 않으며 이 경우 리턴 값은 0임
WUNTRACED	pid로 지정된 자식들 중 중지되었으나 그 상태가 아직 보고되지 않은 임의의 자식도 리턴됨

wait() **예제** 1. p.245

wait() 예제 2

<ssu_wait_2.c>

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

#define EXIT_CODE 1

int main(void)
{
   pid_t pid;
   int ret_val, status;

   if ((pid = fork()) == 0) {
      printf("child: pid = %d ppid = %d exit_code = %d\n",
           getpid(), getppid(), EXIT_CODE);
      exit(EXIT_CODE);
   }
      printf("parent: waiting for child = %d\n", pid);
   ret_val = wait(&status);
   printf("parent: return value = %d, ", ret_val);
   printf(" child's status = %x", status);
   printf(" and shifted = %x\n", (status >> 8));
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_wait_2
parent: waiting for child = 11090
child: pid = 11090 ppid = 11089 exit_code = 1
parent: return value = 11090,  child's status = 100 and shifted = 1
```

wait() 예제 3

<ssu_wait_3.c>

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

int main(void)
{
   if (fork() == 0)
      execl("/bin/echo", "echo", "this is", "message one", (char *)0);

   if (fork() == 0)
      execl("/bin/echo", "echo", "this is", "message Two", (char *)0);

   printf("parent: waiting for children\n");

   while (wait((int *)0) != -1);

   printf("parent: all children terminated\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_wait_3
parent: waiting for children
this is message one
this is message Two
parent: all children terminated
```

wait() 예제 4

<ssu_wait_4.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(void)
{
pid_t child1, child2;
int pid, status;
if ((child1 = fork()) == 0)
execlp("date", "date", (char *)0);
if ((child2 = fork()) == 0)
execlp("who", "who", (char *)0);
printf("parent: waiting for children\n");
while ((pid = wait(&status)) != -1) {
if (child1 == pid)
printf("parent: first child: %d\n", (status >> 8));
else if (child2 == pid)
printf("parent: second child: %d\n", (status >> 8));
}
printf("parent: all children terminated\n");
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_wait_4
parent: waiting for children
oslab tty7 2017-01-17 10:44 (:0)
parent: second child: 0
Tue Jan 17 11:38:10 KST 2017
parent: first child: 0
parent: all children terminated
```

waitid(2)

```
#include <sys/types.h>
#include <sys/wait.h>
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```
리턴 값: 성공 시, 혹은 WNOHANG 옵션이 있는 상태에서자식 프로세스가 종료되지 않았을 경우 0, 에러 시 -1

- 프로세스의 종료 상태를 회수하는 시스템호출
- waitpid()와 같은 점
  - 대기할 자식 프로세스를 구체적으로 지정
  - 두 개의 개별적인 인자들로 프로세스를 지정
- waitpid()와 다른 점
  - pid 대신 idtype이라는 개별적인 인자를 지정하여 대기
  - id로 주어진 값의 의미는 idtype에 따라 달라짐
  - waitid()는 초기 Unix나 macOS에서는 구현되지 않은 함수이다
  - infop 인자
    - siginfo 구조체를 가리키는 포인터로 자식 프로세스의 상태를 변화시킨 시그널에 대한 상세한 정보가 저장

idtype 인자 및 options 인자에 사용할 수 있는 상수들

Idtype 인자 상수	용도
P_PID	특정한 프로세스를 기다림. id는 기다릴 자식 프로세스의 ID임
P_PGID	특정 프로세스 그룹에 속한 임의의 자식 프로세스를 기다림. id는 기다릴 자식 프로세스들이 속한 프로세스 그룹의 ID임
P_ALL	임의의 자식 프로세스를 기다림. id는 무시함

options	용도
WCONTINUED	중지되었다가 다시 재개되었으나 그 상태가 아직 보고되지 않은 프로세스를 기다림
WEXITED	종료된 프로세스들을 기다림
WNOHANG	상태를 알 수 있는 자식들이 없어도 호출이 차단되지 않고 즉시 리턴됨
WNOWAIT	자식의 종료 상태를 파괴하지 않음. 이후 wait나 waitid, waitpid 호출로 다시 조회할 수 있음
WSTOPPED	중지되었으나 그 상태가 아직 보고되지 않은 프로세스를 기다림

wait3(2), wait4(2)

```
#include <sys/resource.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait3(int *statloc, int options, struct rusage *rusage);
pid_t wait4(pid_t pid, int *statloc, int options, struct rusage *rusage);
```

리턴 값: 성공 시 프로세스 ID, 에러 시 –1,WNOHANG 옵션으로 실행되었고 자식 프로세스가 종료되지 않았을 때 0을 리턴

- 종료된 프로세스와 그것의 모든 자식 프로세스의 자원 사용량에 대한 정보를 커널로부터 얻을 수 있는 시스템호출
- wait3()나 wait4()
  - 종료된 프로세스와 그 자식 프로세스의 자원 사용에 관한 추가적인 정보 리턴
  - 프로세스의 자원 사용 정보는 rusage라는 구조체를 통하여 전달
    - 종료된 프로세스의 사용자 CPU 시간, 시스템 CPU 시간, 페이지 폴트 횟수, 수신된 시그널 개수 등의 정보 제공

rusage 구조체

```
struct rusage {
struct timeval ru_utime;        //사용된 유저 시간
struct timeval ru_stime;        //사용된 시스템 시간
long ru_maxrss;                 //최대 상주 세트의 사이즈
long ru_ixrss;                  //공유 텍스트 메모리 총사이즈
long ru_idrss;                  //비공유 데이터 총사이즈
long ru_isrss;                  //비공유 스택 총사이즈
long ru_minflt;                 //페이지 재생수
long ru_majflt;                 //페이지 폴트
long ru_nswap;                  //스왑
long ru_inblock;                //블록 입력 조작
long ru_oublock;                //블록 출력 조작
long ru_msgsnd;                 //송신이 끝난 메세지
long ru_msgrcv;                 //수신이 끝난 메세지
long ru_nsignals;               //수신이 끝난 시그널
long ru_nvcsw;                  //자발적인 콘텍스트 스위칭
long ru_nivcsw;                 //비자발적인 콘텍스트 스위칭
};
```

execl(3), execv(3), execle(3), execve(2), execlp(3), execvp(3)

```
#include <unistd.h>
int execl(const char *pathname, const char *arg0, ... /* (char *)0 */ );
int execv(const char *pathname, char *const argv []);
int execle(const char *pathname, const char *arg0, ... /* (char *)0, char *const envp[] */);
int execve(const char *pathname, char *const argv[], char *const envp[]);
int execlp(const char *filename, const char *arg0, ... /* (char *)0 */ );
int execvp(const char *filename, char *const argv[]);
```

리턴 값: 성공 시 리턴하지 않음, 에러 시 –1을 리턴하고 errno가 설정됨

- exec 계열 함수는 현재 수행되고 있는 프로세스를 대신하여 새로운 프로세스를 수행시키는 함수
  - exec 계열 함수들 중 하나를 호출하면 그 프로세스가 새 프로그램으로 완전히 대체되어 새 프로그램의 main()에서 수행이 다시 시작
  - 새로운 프로세스는 경로 또는 파일에 의해 지정된 한 이미지 파일로부터 생성됨
    - 현재 프로세스의 이미지(텍스트, 자료, 힙, 스택 구역들)는 디스크에서 가져온 새 프로그램의 이미지로

exec에 붙는 문자와 함수 인자와의 관계

options	용도
l	리스트(List) 형태의 명령 라인 인자(arg0, arg1, ..., argn), 마지막 인자 다음에는 NULL 포인터 하나를 지정해야 함
v	벡터(Vector) 형태의 명령 라인 인자(argv[])
e	환경 변수 인자(envp[])
p	경로 정보가 없는 실행 파일 이름(filename) filename에 슬래시(/)가 포함되어 있으면 함수는 filename을 경로 이름으로 취급하고 슬래시가 없으면 함수는 PATH 환경 변수에 나열된 디렉토리들에서 해당 실행파일을 찾음 인자가 실행 가능한 형태의 파일이 아니면 함수는 그 파일을 쉘 스크립트로 간주하고 filename을 인자로 해서 /bin/sh를 실행함 p가 없으면 패스 정보를 포함한 파일 이름(pathname)

exec 계열 함수를 호출한 프로세스에서 새 프로그램으로 상속되는 특징

- 프로세스 ID와 부모 프로세스 ID
- 실제 사용자 ID와 실제 그룹 ID
- 추가 그룹 ID
- 프로세스 그룹 ID
- 세션 ID
- 제어 터미널
- 경보(alarm) 발동까지 남은 시간
- 현재 작업 디렉토리
- 루트 디렉토리
- 파일 생성 마스크
- 파일 자물쇠들
- 프로세스 시그널 마스크
- 아직 처리되지 않은 시그널들
- 자원 한계들
- tms_utime, tms_stime, tms_cutime, tms_cstime 값들 (CPU 사용 시간)

execv() 예제 1

```
<ssu_execv_1.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/resource.h>
#include <sys/wait.h>
double ssu_maketime(struct timeval *time);
void term_stat(int stat);
void ssu_print_child_info(int stat, struct rusage *rusage);
int main(void)
{
   struct rusage rusage;
   pid_t pid;
   int status;

   if ((pid = fork()) == 0) {
      char *args[] = {"find", "/", "-maxdepth", "4", "-name", "stdio.h", NULL};
      if (execv("/usr/bin/find", args) < 0) {
         fprintf(stderr, "execv error\n");
         exit(1);
      }
   }
   if (wait3(&status, 0, &rusage) == pid)
      ssu_print_child_info(status, &rusage);
   else {
      fprintf(stderr, "wait3 error\n");
      exit(1);
   }
   exit(0);
}
double ssu_maketime(struct timeval *time) {
   return ((double)time -> tv_sec + (double)time -> tv_usec/1000000.0);
}
```

```
void term_stat(int stat) {
   if (WIFEXITED(stat))
      printf("normally terminated. exit status = %d\n", WEXITSTATUS(stat));
   else if (WIFSIGNALED(stat))
      printf("abnormal termination by signal %d. %s\n", WTERMSIG(stat),
#ifdef WCOREDUMP
         WCOREDUMP(stat)?"core dumped":"no core"
#else
         NULL
#endif
         );
   else if (WIFSTOPPED(stat))
      printf("stopped by signal %d\n", WSTOPSIG(stat));
}

void ssu_print_child_info(int stat, struct rusage *rusage) {
   printf("Termination info follows\n");
   term_stat(stat);
   printf("user CPU time : %.2f(sec)\n", ssu_maketime(&rusage->ru_utime));
   printf("system CPU time : %.2f(sec)\n", ssu_maketime(&rusage->ru_stime));
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_execv_1
/usr/include/stdio.h
find: `/run/user/1000/gvfs': 허가 거부
Termination info follows
normally terminated. exit status = 1
user CPU time : 0.06(sec)
system CPU time : 0.07(sec)
```

·execv() 예제 2

<ssu_execv_2.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(void)
{
char *argv[] = {
"ssu_execl_test_1", "param1", "param2", (char *)0
};
printf("this is the original program\n");
execv("./ssu_execl_test_1", argv);
printf("%s\n", "This line should never get printed\n");
exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_execv_2
this is the original program
argv[0]: ssu_execl_test_1
argv[1]: param1
argv[2]: param2
SHELL=/bin/bash
TERM=xterm-256color
USER=root
(중략)
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/oslab/.Xauthority
_=./ssu_execv_2
OLDPWD=/home
```

execl() 예제 1

```
<ssu_execl_test_1.c>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   extern char **environ;
   char **str;
   int i;

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

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

   exit(0);
}
```

```
<ssu_execl_1.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
   printf("this is the original program\n");
   execl("./ssu_execl_test_1", "ssu_execl_test_1",
         "param1", "param2", "param3", (char *)0);
   printf("%s\n", "this line should never get printed\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_execl_1
this is the original program
argv[0]: ssu_execl_test_1
argv[1]: param1
argv[2]: param2
argv[3]: param3
SHELL=/bin/bash
TERM=xterm-256color
USER=root

(중략)

LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/home/oslab/.Xauthority
_=./ssu_execl_1
OLDPWD=/home
```

execl() 예제 2

<ssu_execl_2.c>

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

int main(void)
{
   if (fork() == 0) {
      execl("/bin/echo", "echo", "this is", "message one", (char *)0);
      fprintf(stderr, "exec error\n");
      exit(1);
   }
   if (fork() == 0) {
      execl("/bin/echo", "echo", "this is", "message two", (char *)0);
      fprintf(stderr, "exec error\n");
      exit(1);
   }
   if (fork() == 0) {
      execl("/bin/echo", "echo", "this is", "message three", (char *)0);
      fprintf(stderr, "exec error\n");
      exit(1);
   }
   printf("Parent program ending\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_execl_2
Parent program ending
root@localhost:/home/oslab# this is message two
this is message three
this is message one
```

execve() 예제

<ssu_execve.c>

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

int main(void)
{
   char *argv[] = {
      "ssu_execl_test_1", "param1", "param2", (char *)0
   };
   char *env[] = {
      "NAME = value",
      "nextname=nextvalue",
      "HOME=/home/oslab",
      (char *)0
   };

   printf("this is the original program\n");
   execve("./ssu_execl_test_1", argv, env);
   printf("%s\n", "This line should never get printed\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_execve
this is the original program
argv[0]: ssu_execl_test_1
argv[1]: param1
argv[2]: param2
NAME = value
nextname=nextvalue
HOME=/home/oslab
```

setuid(2), setgid(2)

```
#include <unistd.h>
#include <sys/types.h>
int setuid(uid_t uid);
int setgid(gid_t gid);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- setuid(), setgid() 는 사용자 ID와 그룹 ID를 변경하는 시스템 호출
  - (1) 프로세스가 루트 권한들을 가지고 있으면, setuid()는 실제 사용자 ID(real user ID), 유효 사용자 ID(effective user ID), 저장된 사용자 ID(saved user ID)를 uid로 설정
  - (2) 프로세스가 루트 권한들을 가지고 있지 않으나 uid가 실제 사용자 ID나 저장된 사용자 ID와 같으면 setuid()는 유효 사용자 ID만 uid로 설정.
    - 단 실제 사용자 ID와 저장된 사용자 ID는 변경되지 않음
  - (3)그 외의 경우, 함수는 errno를 EPERM으로 설정하고 –1을 리턴
- setgid()는 사용자 ID가 아닌 그룹 ID라는 점만 제외하고 setuid()와 동일한 기능을 수행
- 커널이 관리하는 실행 파일과 관련 있는 사용자 ID
  - 실제(real), 유효(effective), 저장된(saved) 사용자 ID, 파일 소유자 ID

setuid(2), setgid(2)

- 실제 사용자 ID
  - 프로세스를 실행한 원래 사용자의 ID
  - 원래 사용자 ID는 프로세스 부모의 실제 사용자 ID로 설정되며, exec() 호출 도중에 변경되지 않는다. 일반적으로 login 프로세스는 login 쉘의 실제 사용자 ID를 해당 사용자의 실제 사용자 ID로 설정된다. 루트 권한을 갖는 프로세스만 실제 사용자 ID를 다른 값으로 변경할 수 있다.
- 유효 사용자 ID
  - 프로세스가 영향을 미치는 사용자 ID이며 접근 권한의 기준으로 사용
  - 프로세스 생성 초기에는 부모 프로세스의 유효 사용자 ID를 상속 받기 때문에 유효 사용자 ID는 실제 사용자 ID와 같으며, exec()를 호출할 때도 일반적으로 유효 사용자 ID는 변경되지 않음
  - 따라서 exec() 호출 도중에 실제 사용자 ID와 유효 사용자 ID가 무엇인지 확인할 수 있음
  - setuid() -> 프로세스는 유효 사용자 ID를 변경
    - 유효사용자 ID는 프로그램 파일을 소유한 사용자 ID로 변경
    - 예 "/usr/bin/passwd" 실행 파일은 setuid 파일이며, 소유자는 루트이기 때문에 일반 사용자 쉘에서 이 파일을 exec하기 위해 프로세스를 실행시킬 때 이 프로세스는 실행하는 사용자와 무관하게 유효 사용자 ID를 루트로 지정
- 저장된 사용자 ID
  - 프로세스의 본래 유효 사용자 ID
  - 프로세스가 fork()를 실행 할 때 자식 프로세스는 부모 프로세스의 저장된 사용자 ID를 상속받음
  - exec()를 호출할 때 커널은 저장된 사용자 ID를 유효 사용자 ID로 설정 => 저장된 사용자 ID는 exec() 호출 시에 유효 사용자 ID로부터 복사
  - exec()에 의해 실행되는 프로그램에 저장된 사용자 ID 비트가 설정되어 있으면, 이 프로그램의 이미지 파일의 사용자 ID가 유효 사용자 ID에 복사되는데, 이때 이전의 유효 사용자 ID는 저장된 사용자 ID에 복사
  - 루트 권한을 갖는 프로세스만이 실제 사용자 id와 동일한 값으로 변경 가능

setuid() **예제** p. 265

setreuid(2), setregid(2)

```
#include <unistd.h>
#include <sys/types.h>
int setreuid(uid_t ruid, uid_t euid);
int setregid(gid_t rgid, gid_t egid);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- 프로세스의 실제 ID와 유효 ID를 맞바꾸는 시스템호출
  - 권한이 없는 사용자라도 언제나 실제 사용자 ID와 유효 사용자 ID를 서로 교환 가능

seteuid(2), setegid(2)

```
#include <unistd.h>
#include <sys/types.h>
int seteuid(uid_t uid);
int setegid(gid_t gid);
```
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- 프로세스의 유효 사용자 ID와 유효 그룹 사용자 ID를 변경하는 함수
- 특권이 없는 사용자는 자신의 유효 사용자 ID를 실제 사용자 ID 또는 저장된 사용자 ID로 설정 가능
- 특권을 가진 사용자는 유효 사용자 ID만 uid로 설정 가능

setuid() 예제

<ssu_seteuid.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(void)
{
   int fd, state;

   state = seteuid(1000);

   if (state < 0) {
      fprintf(stderr, "seteuid error\n");
      exit(1);
   }

   if ((fd = open("ssu_test.txt", O_CREAT | O_RDWR, S_IRWXU)) < 0) {
      fprintf(stderr, "open error\n");
      exit(1);
   }

   close(fd);
   exit(0);
}
```

실행 결과

```
oslab@localhost:~$ cat /etc/passwd | grep "oslab"
oslab:x:1000:1000:,,,:/home/oslab:/bin/bash
oslab@localhost:~$ sudo ./ssu_seteuid
oslab@localhost:~$ ls -l ssu_test.txt
-rwx------ 1 oslab root 0 Jan 10 18:26 ssu_test.txt
```

system(3)

```
#include <stdlib.h>
int system(const char *cmdstring);
리턴 값: (아래 본문 참고)
```

- 프로그램 안에서 하나의 명령 문자열을 실행하는 라이브러리 함수
- 쉘에서 명령을 실행하듯이 쉽게 프로그램을 실행시킬 수
- system("clear")
- cmdstring 인자가 NULL 포인터인 경우, 시스템이 명령 처리기를 제공한다면 system() 는 0이 아닌 값을 리턴
  - 운영체제가 system()를 지원하는지의 여부를 판정할 때 유용
- system()
  - fork() -> exec() -> waitpid()
  - system()의 세 종류의 리턴 값
    - fork() 호출이 실패/ waitpid가 EINTR 이외의 에러 코드를 리턴=> errno를 해당 값으로 설정하고 –1을 리턴
    - exec 계열 함수가 실패 => 쉘이 마치 exit(127)을 호출할 때와 같은 값이 리턴
    - 그 외의 경우, 즉 fork() 호출과 exec 계열 함수 호출, waitpid() 호출 모두 성공=> 쉘의 종료 상태

·system() 예제

<ssu_system.c>

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

int main(void)
{
   printf("before system()\n");
   system("pwd");
   printf("after system()\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_system
before system()
/home/oslab
after system()
```

getlogin(3)

```
#include <unistd.h>
char *getlogin(void);
```
리턴 값: 성공 시 로그인 이름을 담은 문자열을 가리키는 포인터,에러 시 NULL을 리턴하고 errno가 설정됨

- 프로그램을 실행하고 있는 사용자의 로그인 이름을 리턴하는 라이브러리 함수
- 하나의 사용자 ID에 여러 개의 로그인 이름이 존재하면, 현재 로그인 된 이름을 리턴
- 사용자가 로그인해 있는 터미널에 부착되어 있지 않은 프로세스(ex. 디몬)에서 이 함수를 호출하면 호출이 실패할 수 있음
- 참고로 getlogin()은 다른 몇몇의 프로그램에서 tmp 파일에 접근하여 파일의 내용을 변경할 수 있기 때문에 getpwuid()를 사용 권함

getpgrp(2), getpgid(2), setpgid(2)

```
#include <unistd.h>
pid_t getpgrp(void);
pid_t getpgid(pid_t pid);
리턴 값: 성공 시 프로세스 그룹 ID, 에러 시 –1을 리턴하고 errno가 설정됨
int setpgid(pid_t pid, pid_t pgid);
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- getpgrp() =>
  - 현재 프로세스의 프로세스 그룹 ID 값을 리턴하는 시스템호출 함수
  - 각 프로세스는 고유한 프로세스 ID를 가지면서 동시에 특정 그룹에 속함 => 프로세스 그룹은 동일한 터미널로부터 시그널들을 받을 수 있는 프로세스들의 집합 => 같은 작업에 연관된 프로세스들이 같은 그룹을 형성
- getpgid()
  - 프로세스 ID를 인자로 받아 그 프로세스가 속해 있는 프로세스의 그룹 ID를 리턴하는 시스템호출 함수
  - pid 인자의 값이 0이면 호출한 프로세스의 그룹 ID를 리턴
  - 프로세스 그룹에는 하나의 프로세스 그룹 리더 존재
    - 프로세스 그룹의 리더 => 프로세스 그룹의 프로세스 그룹 ID와 동일한 프로세스 ID
    - 프로세스 그룹의 리더는 프로세스 그룹을 만들 수 있으며, 해당 그룹 내에서 프로세스들을 생성 가능
    - 프로세스 그룹이 존재하는 상황에서 리더는 자신을 종료할 수 있으며, 리더가 종료되어도 프로세스 그룹에 프로세스가 하나라도 남아 있다면 프로세스 그룹은 사라지지 않음 => 프로세스 그룹 수명

getpgrp(2), getpgid(2), setpgid(2)

int setpgid(pid_t pid, pid_t pgid);
리턴 값: 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- setpgid() : setpgrp() => setpgid(0, 0) in linux
  - 프로세스가 자신을 기존의 프로세스 그룹에 속하게 하거나 새 프로세스 그룹을 생성 시 시스템호출 함수
  - 프로세스ID가 pid인 프로세스의 프로세스 그룹 ID를 pgid로 설정
  - 두 인자가 같으면 pid로 지정된 프로세스가 그룹의 리더가 됨
  - pid가 0이면 호출자의 프로세스 ID가 사용됨
  - pgid가 0이면 pid로 지정된 프로세스 ID가 프로세스 그룹 ID로 사용
  - 프로세스는 자신이나 자신의 자식들의 프로세스 그룹 ID만 설정 가능
  - 자식 프로세스가 exec 계열 함수를 호출 했다면 그 자식 프로세스의 프로세스 그룹 ID는 변경이 불가능
  - 대부분의 작업 제어 쉘들은 fork() 호출로 자식을 생성 후 부모에서 setpgid()로 자식의 프로세스 그룹 ID를 설정
  - 자식들 역시 setpgid()를 호출해서 자신의 프로세스 그룹 ID를 설정

getpgrp(2), getpgid(2), setpgid(2)

PID 검색	pid_t getpid(void);
부모 PID 검색	pid_t getppid(void);
프로세스 그룹 ID 검색	pid_t getpgrp(void); / pid_t getpgid(pid_t pid);
프로세스 그룹 ID 변경	int setpgid(pid_t pid, pid_t pgid);
세션 리더 ID 검색	pid_t getsid(pid_t pid);
세션 생성	pid_t setsid(void);

getpgrp() 예제

<ssu_getpgrp.c>

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

int main(void)
{
    pid_t pgid;
    pid_t pid;

    pid = getpid();
    pgid = getpgrp();
    printf("pid: %d, pgid: %d\n", pid, pgid);
    exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_getpgrp
pid: 29894, pgid: 29894
```

getpgid() 예제

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

int main(int argc, char *argv[])
{
   pid_t pgid;
   pid_t pid;

   if (argc < 2) {
      fprintf(stderr, "usage: %s <file>\n",argv[0]);
      exit(1);
   }

   pid = getpid();
   pgid = getpgid(atoi(argv[1]));
   printf("pid: %d, pgid: %d\n", pid, pgid);
   exit(0);
}
```

실행 결과 [Ubuntu, Fedora]

```
root@localhost:/home/oslab# pstree –p
          ...
          ├─upowerd(8630)─┬─{gdbus}(8636)
          |               └─{gmain}(8635)
root@localhost:/home/oslab# ./ssu_getpgid 8636
pid: 29947, pgid: 8630
```

실행 결과 [Mac]

```
ssu-ui-MacBook-Air:oslab root# ps
  PID TTY           TIME CMD
 2714 ttys000    0:00.04 login -pfl user1 /bin/bash -c exec -la bash
/bin/bash
 3045 ttys000    0:00.04 su
 3046 ttys000    0:00.27 sh
 3444 ttys000    0:00.03 su user1
 2728 ttys002    0:00.03 login -pf user1
 3998 ttys002    0:00.03 su
 3999 ttys002    0:00.01 sh
 4000 ttys002    0:00.01 ps
 3613 ttys003    0:00.02 -sh
ssu-ui-MacBook-Air:oslab root# ./ssu_getpgid 3444
pid: 4019, pgid: 3444
```

setsid(2), getsid(2)

```
#include <unistd.h>
pid_t setsid(void);
리턴 값 : 성공 시 프로세스 그룹 ID, 에러 시 –1을 리턴하고 errno가 설정됨
pid_t getsid(pid_t pid);
        리턴 값 : 성공 시 pid의 세션 리더의 ID, 에러 시 –1을 리턴하고 errno가 설정됨
```

- 한 프로세스가 새로운 세션을 생성할 때에 사용하는 시스템호출
  - 세션(session)은 하나 이상의 프로세스 그룹들의 집합
  - 호출한 프로세스가 프로세스 그룹 리더가 아니면 setsid()는 새 세션을 다음과 같이 생성
    - 호출한 프로세스는 새 세션의 세션 리더가 되고, 새 프로세스 그룹의 프로세스 그룹 리더가 됨
    - 새 프로세스 그룹 ID는 호출한 프로세스의 프로세스 ID
    - 또한 호출한 프로세스는 새로 생성된 프로세스 그룹과 세션의 유일한 프로세스가 됨
    - 호출한 프로세스는 제어 터미널을 갖지 않고 프로세스가 setsid()를 호출하기 전에 제어 터미널을 가지고 있었다면 그 터미널과의 연관 관계가 사라짐
  - 호출한 프로세스가 이미 프로세스 그룹 리더이면 setsid()는 에러 리턴
    - 일반적으로 세션을 생성할 때, fork()를 호출하여 자식 프로세스를 생성한 후, 부모 프로세스는 종료하게 하고 자식 프로세스가 setsid()를 호출
- getsid() : 프로세스 세션 리더의 프로세스 그룹 ID를 얻을 때 사용하는 시스템호출
  - pid가 0 : 호출한 프로세스의 세션 리더의 프로세스 그룹 ID를 리턴

setsid() 예제

<ssu_setsid.c>

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

int main(void)
{
    pid_t pid;

    if ((pid = fork()) < 0) {
        fprintf(stderr, "fork error\n");
        exit(1);
    }
    else if(pid != 0)
        exit(1);

    printf("before pid = %d, sid = %d\n", getpid(), getsid(getpid()));
    setsid();
    printf("after pid = %d, sid = %d\n", getpid(), getsid(getpid()));
    exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_setsid
before pid = 29983, sid = 28572
after pid = 29983, sid = 29983
```

tcgetpgrp(3), tcsetpgrp(3), tcgetsid(3)

```
#include <unistd.h>
pid_t tcgetpgrp(int filedes);
리턴 값 : 성공 시 전경 프로세스 그룹의 프로세스 그룹 ID, 에러 시 –1을 리턴하고 errno가 설정됨
int tcsetpgrp(int filedes, pid_t pgrpid);
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- tcgetpgrp()
  - 전경 프로세스 그룹(foreground process group)을 조회하거나 설정할 때 사용하는 라이브러리 함수
- Tcsetpgrp()
  - filedes로 열린 터미널에 연관된 전경 프로세스 그룹의 프로세스 그룹 ID를 리턴
  - 제어 터미널에 배정되어 있는 프로세스는 tcsetpgrp()를 호출해서 프로세스 그룹 ID가 pgrpid인 프로세스 그룹을 전경 프로세스 그룹으로 만들 수있음
  - pgrpid 인자의 값은 호출자와 같은 세션에 있는 프로세스 그룹의 프로세스 그룹 ID이어야 하며 filedes 인자는 그 세션의 제어 터미널이어야 함
- 대부분의 응용프로그램은 이 두 함수들을 직접 호출하지 않으며 주로 작업 제어 쉘들이 호출

tcgetsid(int filedes)

```
#include <termios.h>
pid_t tcgetsid(int filedes);
```
리턴 값 : 성공 시 세션 리더의 프로세스 그룹 ID, 에러 시 –1을 리턴하고 errno가 설정됨

- 주어진 파일 디스크립터에 해당하는 제어 터미널과 연관된 세션 리더의 프로세스 그룹 ID를 리턴하는 라이브러리 함수
- 제어 터미널을 관리해야 하는 응용프로그램이라면, tcgetsid()를 호출하여 제어 터미널의 세션 ID(세션 리더의 프로세스 그룹 ID)를 얻을 수 있음

tcgetsid() 예제

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

int main(void)
{
   pid_t sid_stderr;
   pid_t sid_stdin;
   pid_t sid_stdout;

   sid_stdin = tcgetsid(STDIN_FILENO);

   if (sid_stdin == -1) {
      fprintf(stderr, "tcgetsid error\n");
      exit(1);
   }
   else
      printf("Session Leader for stdin: %d\n", sid_stdin);

   sid_stdout = tcgetsid(STDOUT_FILENO);
```

```
 if (sid_stdout == -1) {
      fprintf(stderr, "tcgetsid error\n");
      exit(1);
   }
   else
      printf("Session Leader for stdout: %d\n", sid_stdout);

   sid_stderr = tcgetsid(STDERR_FILENO);

   if (sid_stderr == -1) {
      fprintf(stderr, "tcgetsid error\n");
      exit(1);
   }
   else
      printf("Session Leader for stderr: %d\n", sid_stderr);

   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_tcgetsid
Session Leader for stdin: 28572
Session Leader for stdout: 28572
Session Leader for stderr: 28572
```