Linux System Programming #5-1
Lecture Notes

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

exit(3), _Exit(2), _exit(2)

```
#include <stdlib.h>
void exit(int status);
void _Exit(int status);

#include <unistd.h>
void _exit(int status);
```

- exit()
  - 프로그램을 정상 종료시킬 때 사용하는 라이브러리 함수
  - 종료 처리부들이나 시그널 핸들러들을 실행하고 최종적으로 _exit()를 호출
  - 표준 입출력 라이브러리 정리
  - vfork() – exit() 사용 시 유의
- _Exit(), _exit()
  - 프로그램 종료 시스템호출 함수
  - 종료 처리부들이나 시그널 처리부들을 실행하지 않고 프로세스를 종료
- int status
  - 종료 상태(exit status) -> 부모에게 전달
  - 하위 8비트만 사용(0~255)

종료 상태가 정의되지 않는 경우

-종료 함수들이 종료 상태 없이 호출될 경우

- main()가 리턴 값이 없는 return 문에 의해 리턴되었을 경우

- main()가 정수 값을 리턴되도록 선언되지 않았을 경우

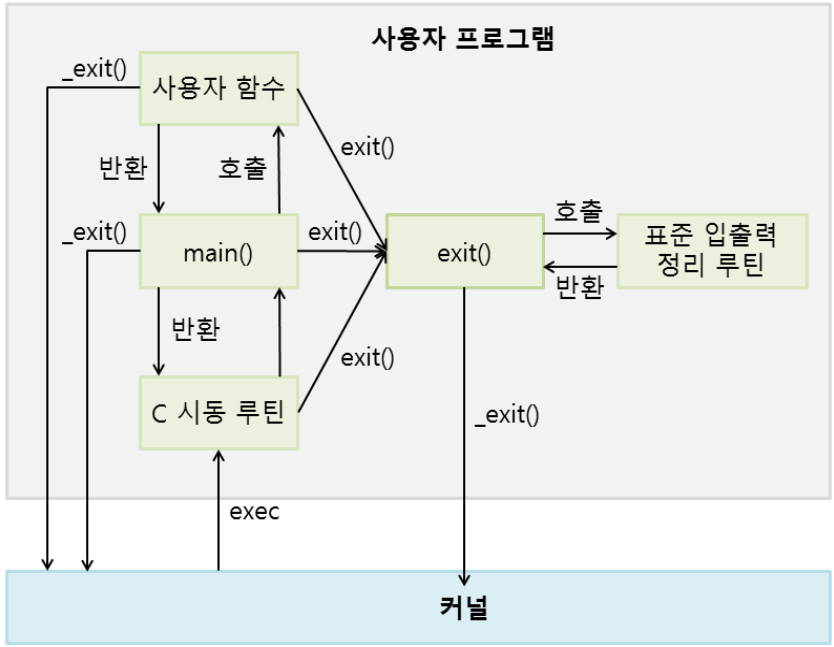
정상 종료 vs. 비정상 종료

구성 요소	내용
main()의 return 문에 의해 종료	exit() 호출과 동일
exit()를 호출하여 종료	결과적으로 _exit()가 호출되지만 표준 입출력 채널 정리 등의 추가 루틴 실행
_exit() 또는 _Exit()를 호출하여 종료	종료 상태 값을 명시적으로 지정
프로세스 마지막 스레드의 시동 루틴이 return에 의해 종료	
프로세스 마지막 스레드에서 pthread_exit()를 호출하여 종료	

비정상 종료

- abort() 호출에 의하여 종료 (SIGABRT 시그널 발생)
- 프로세스가 특정 시그널을 받아 종료 (0으로 나눈 경우, 잘못된 메모리 참조 등)
- 마지막 스레드가 취소 요청을 받아들여서 종료
- 커널이 종료 상태 값을 생성

프로세스의 종료 과정



- 해당 프로세스가 오픈한 파일 디스크립터를 모두 close
- 프로세스에 할당되었던 메모리는 모두 해제
- exit()는 오픈된 파일 스트림을 닫기 위해 fclose() 호출 -> 버퍼에 남은 데이터를 비우기 위해 표준 입출력을 정리하는 루틴 실행 -> _exit() 호출
- C 컴파일러 : main()에서 리턴하는 경우 exit() 자동 호출

_exit(), exit() 예제

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

int main(void)
{
   printf("Good afternoon?");
   _exit(0);
}

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

int main(void)
{
   printf("Good afternoon");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu__exit
root@localhost:/home/oslab# ./ssu_exit
Good afternoonroot@localhost:/home/oslab#
```

atexit(3), on_exit(3)

```
#include <stdlib.h>
int atexit(void (*__func__) (void));
int on_exit(void (*function)(int, void *), void *arg);
```
리턴 값 : 성공 시 0, 실패 시 0 이외의 값

- exit() 실행 시 자동으로 호출될 함수를 등록할 때 사용하는 라이브러리 함수
- atexit()
  - *__func__ 인자는 exit()에 의해 호출될 함수를 가리키는 함수 포인터
  - 등록될 함수는 어떠한 인자도 받지 않으며 어떠한 값도 리턴하지 않는 함수
  - 등록된 함수들을 등록의 역순으로 호출
  - 하나의 함수를 여러 번 등록할 수 있으며, 등록한 횟수만큼 여러 번 호출
  - exec류 함수들이 호출되면 등록된 함수 목록이 초기화됨

- on_exit()
  - atexit()와 동일하게 동작하지만, 등록 대상 함수 서식이 다름
  - status 인자는 exit()로 전달되거나 main()에서 리턴될 값
  - arg 인자는 on_exit()로 전달될 두 번째 인자이며, 등록된 함수가 최종적으로 호출될 시점에서 arg 인자가 가리키는 메모리가 유효해야 함.

atexit() 예제 1

<ssu_atexit_2.c>

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

void ssu_out(void);
int main(void)
{
   if (atexit(ssu_out)) {
      fprintf(stderr, "atexit error\n");
      exit(1);
   }

   exit(0);
}

void ssu_out(void) {
   printf("atexit succeeded!\n");
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_atexit_2
atexit succeeded!
```

atexit() 예제 2

<ssu_atexit_1.c>

```
#include <stdio.h>
#include <stdlib.h>
static void ssu_exit1(void);
static void ssu_exit2(void);
int main(void)
{
   if (atexit(ssu_exit2) != 0) {
      fprintf(stderr, "atexit error for ssu_exit2");
      exit(1);
   }
   if (atexit(ssu_exit1) != 0) {
      fprintf(stderr, "atexit error for ssu_exit1");
      exit(1);
   }
   if (atexit(ssu_exit1) != 0) {
      fprintf(stderr, "atexit error for ssu_exit1");
      exit(1);
   }
   printf("done\n");
   exit(0);
}static void ssu_exit1(void) {
   printf("ssu_exit1 handler\n");
}
static void ssu_exit2(void) {
   printf("ssu_exit2 handler\n");
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_atexit_1
done
ssu_exit1 handler
ssu_exit1 handler
ssu_exit2 handler
```

Command-Line 인자 예제

<ssu_command-line.c>

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

int main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++)
        printf("argv[%d] : %s\n", i, argv[i]);
    exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_command-line argv1 hello test program
argv[0] : ./ssu_command-line
argv[1] : argv1
argv[2] : hello
argv[3] : test
argv[4] : program
```

malloc(3), calloc(3), realloc(3), free(3)

```
#include <stdlib.h>
void *malloc(size_t size);
void *calloc(size_t nobj, size_t size);
void *realloc(void *ptr, size_t newsize);
리턴 값 : 성공 시 할당된 메모리를 가리키는 포인터, 에러 시 NULL
void free(void *ptr);
```

- malloc() : 할당된 메모리의 초기 값은 정의되지 않음
- calloc() : 할당된 메모리의 초기 값을 모두 0으로 정의
- free() : 메모리 할당 함수들에 의해 할당받은 메모리를 해제 -> 해제된 메모리 공간은 자유 리스트로 포함
- realloc() : 이미 할당받은 메모리 공간 크기를 늘리거나 줄이는 함수

getenv(3), putenv(3), setenv(3), unsetenv(3)

```
#include <stdlib.h>
char *getenv(const char *name);
리턴 값 : name으로 주어진 이름에 해당하는 환경 변수 값을 가리키는 포인터,그런 이름이 없는 경우에는 NULL
int putenv(char *str);
리턴 값 : 성공 시 0, 에러 시 0이 아닌 값을 리턴하고 errno가 설정됨
int setenv(const char *name, const char *value, int rewrite);
int unsetenv(const char *name);
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- 특정 환경 변수를 변경/제어, 새로운 환경 변수 생성하기 위해 사용하는 라이브러리 함수
  - 환경 변수 : "이름=값" 으로 지정
  - 환경 변수 접근 방법 (1) environ 전역 변수 이용 (2) main()의 3번째 인자를 이용
- getenv()
  - 환경 변수 리스트 중에서 "이름=값"형식의 문자열을 찾아 값에 대한 포인터를 리턴. 만약 찾는데 실패하면 NULL을 리턴
- putenv()
  - 기존 환경 변수의 값을 변경하거나 새로운 환경 변수를 등록
  - str은 "이름=값"의 형식으로 처리
  - 만약 이름에 해당하는 환경 변수가 이미 존재하면 이전에 설정된 값은 새로운 값으로 변경
  - 새로운 환경 변수는 환경 변수 리스트의 맨 끝에 추가
- unsetenv()
  - 환경 변수 이름에 대한 정의를 초기화할 때 사용
  - 만약 환경 변수 이름을 찾을 수 없는 경우 에러 리턴

·putenv() 예제 1

```
<ssu_putenv_1.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void ssu_addone(void);

extern char **environ;
char glob_var[] = "HOBBY=swimming";

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

   putenv(glob_var);
   ssu_addone();
   printf("My hobby is %s\n", getenv("HOBBY"));
   printf("My lover is %s\n", getenv("LOVER"));
   strcpy(glob_var + 6, "fishing");
   for (i = 0; environ[i] != NULL; i++)
      printf("environ[%d] : %s\n", i, environ[i]);

   exit(0);
}
```

```
void ssu_addone(void) {
   char auto_var[10];
   strcpy(auto_var, "LOVER=js");
   putenv(auto_var);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_putenv_1
environ[0] : XDG_VTNR=7
environ[1] : XDG_SESSION_ID=c2
environ[2] : CLUTTER_IM_MODULE=xim
(중략)
environ[57] : XAUTHORITY=/home/oslab/.Xauthority
environ[58] : _=./a.out
environ[59] : OLDPWD=/home/oslab
My hobby is swimming
My lover is (null)
environ[0] : XDG_VTNR=7
environ[1] : XDG_SESSION_ID=c2
environ[2] : CLUTTER_IM_MODULE=xim
(중략)
environ[59] : OLDPWD=/home/oslab
environ[60] : HOBBY=fishing
environ[61] : y
```

지역 변수를 putenv()의 인자로 사용한 후 종료하면 예기치 않은 결과가 발생 가능

·putenv() 예제 2

<ssu_putenv_2.c>

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

void ssu_printenv(char *label, char ***envpp);

extern char **environ;
int main(int argc, char *argv[], char *envp[])
{
   ssu_printenv("Initially", &envp);
   putenv("TZ=PST8PDT");
   ssu_printenv("After changing TZ", &envp);
   putenv("WARNING=Don't use envp after putenv()");
   ssu_printenv("After setting a new variable", &envp);
   printf("value of WARNING is %s\n", getenv("WARNING"));
   exit(0);
}

void ssu_printenv(char *label, char ***envpp) {
   char **ptr;

   printf("---- %s ---\n", label);
   printf("envp is at %8o and contains %8o\n", envpp, *envpp);
   printf("environ is at %8o and contains %8o\n", &environ, environ);
   printf("My environment variable are:\n");
   for (ptr = environ; *ptr; ptr++)
      printf("(%8o) = %8o -> %s\n", ptr, *ptr, *ptr);
   printf("(%8o) = %8o\n", ptr, *ptr);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_putenv_2
---- Initially ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 27754643474
My environment variable are:
(27754643474) = 27754644205 -> SHELL=/bin/bash
(27754643500) = 27754644225 -> TERM=xterm-256color
(27754643504) = 27754644251 -> USER=root
(중략)
(27754643624) = 27754647723 -> _=./ssu_putenv_2
(27754643630) = 27754647742 -> OLDPWD=/home
(27754643634) =       0
---- After changing TZ ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 1004432020
My environment variable are:
(1004432020) = 27754644205 -> SHELL=/bin/bash
(1004432024) = 27754644225 -> TERM=xterm-256color
(1004432030) = 27754644251 -> USER=root
(중략)
(1004432154) = 27754647742 -> OLDPWD=/home
(1004432160) = 1001103372 -> TZ=PST8PDT
(1004432164) =       0
---- After setting a new variable ---
envp is at 27754643250 and contains 27754643474
environ is at 1001120054 and contains 1004432020
My environment variable are:
(1004432020) = 27754644205 -> SHELL=/bin/bash
(1004432024) = 27754644225 -> TERM=xterm-256color
(1004432030) = 27754644251 -> USER=root
(중략)
(1004432160) = 1001103372 -> TZ=PST8PDT
(1004432164) = 1001103430 -> WARNING=Don't use envp after putenv()
(1004432170) =       0
value of WARNING is Don't use envp after putenv()
```

setjmp(3), longjmp(3)

```
#include <setjmp.h>
int setjmp(jmp_buf env);
리턴 값 : 직접 호출된 경우에는 0, longjmp를 통해서 호출된 경우에는 0이 아닌 값
void longjmp(jmp_buf env, int val);
```

- 함수 경계를 넘나드는 분기를 수행하기 위해 사용하는 라이브러리 함수 => 다른 함수에 설정된 레이블로 점프
- 많은 함수 호출로 인해 깊이 중첩되어 있는 경우 해당 함수로부터 빠져나와 다른 상위 레벨의 함수로 이동할 수 있는 기능 제공
  - 중첩되어 호출된 함수에서의 인터럽트나 에러를 처리하는데 유용하게 사용
- setjmp()
  - setjmp()가 위치한 함수의 스택 환경과 레지스터 환경을 env 변수에 저장하고 이후에 호출될 longjmp()에 의해 사용될 수 있게 함
  - setjmp() 직접 호출되었을 경우는 0을, longjmp()에 의해 호출되었을 경우는 0 이외의 값을 리턴.
- longjmp()
  - 현재의 스택 프레임을 setjmp()에 의해 저장된 스택의 환경과 레지스터 내용으로 복구하여 setjmp()에 의해 지정된 위치로 제어를 넘김
  - val 인자 값은 setjmp()의 리턴 값이 됨
    - (1) val 인자 값을 조정하면 한 개의 setjmp()에 대해 여러 개의 longjmp()를 가능
    - (2) setjmp()의 리턴 값으로부터 어느 longjmp()에서 호출이 일어났는지를 판단 가능하게 함

·setjmp() 예제

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

void ssu_nested_func(int loc_var, int loc_volatile, int loc_register);

static jmp_buf glob_buffer;

int main(void)
{
   register int loc_register;
   volatile int loc_volatile;
   int loc_var;

   loc_var = 10;
   loc_volatile = 11;
   loc_register = 12;

   if (setjmp(glob_buffer) != 0) {
      printf("after longjmp, loc_var = %d, loc_volatile = %d,
           loc_register = %d\n", loc_var, loc_volatile, loc_register);
      exit(0);
   }

   loc_var = 80;
   loc_volatile = 81;
   loc_register = 83;
   ssu_nested_func(loc_var, loc_volatile, loc_register);
   exit(0);
}

void ssu_nested_func(int loc_var, int loc_volatile, int loc_register) {
   printf("before longjmp, loc_var = %d, loc_volatile = %d,
        loc_register = %d\n", loc_var, loc_volatile, loc_register);
   longjmp(glob_buffer, 1);
}
```

실행 결과

```
root@localhost:/home/oslab# gcc -o ssu_setjmp ssu_setjmp.c
root@localhost:/home/oslab# ./ssu_setjmp
before longjmp, loc_var = 80, loc_volatile = 81, loc_register = 83
after longjmp, loc_var = 80, loc_volatile = 81, loc_register = 83
root@localhost:/home/oslab# gcc -o ssu_setjmp ssu_setjmp.c -O
root@localhost:/home/oslab# ./ssu_setjmp
before longjmp, loc_var = 80, loc_volatile = 81, loc_register = 83
after longjmp, loc_var = 10, loc_volatile = 81, loc_register = 12
```

getrlimit(2), setrlimit(2)

```
#include <sys/resource.h>
#include <sys/time.h>
int getrlimit(int resource, struct rlimit *rlptr);
int setrlimit(int resource, const struct rlimit *rlptr);
```
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨

- **프로세스마다 적용된 자원의 한계 값을 조회하거나 변경할 때 사용하는 시스템호출 함수**
- **getrlimit()**
  - 자원 제약을 보여주는 함수
- **setrlimit()**
  - 자원의 한계를 정하는 함수

자원의 종류 <sys/resource.h>

구성 요소	내용
RLIMIT_AS	한 프로세스의 총 자유 메모리 용량의 최댓값(바이트 단위)
RLIMIT_CORE	한 코어 파일의 최대 크기(바이트 단위)
RLIMIT_CPU	프로세스가 소비할 수 있는 최대 CPU 시간(초 단위)
RLIMIT_DATA	자료 구역 전체의 최대 크기(바이트 단위)
RLIMIT_FSIZE	파일 생성 시 파일의 최대 크기(바이트 단위)
RLIMIT_LOCKS	프로세스가 가질 수 있는 파일 자물쇠들의 최대 개수
RLIMIT_MEMLOCK	한 프로세스가 mlock으로 잠글 수 있는 메모리의 최대 용량(바이트 단위)
RLIMIT_NOFILE	한 프로세스가 열어둘 수 있는 최대 파일 개수
RLIMIT_NPROC	실제 사용자 ID당 최대 자식 프로세스 개수
RLIMIT_RSS	메모리 상주 세트 크기(RSS:resident set size)의 최댓값(바이트 단위)
RLIMIT_SBSIZE	한 사용자가 한 시점에서 사용할 수 있는 소켓 버퍼들의 최대 크기(바이트 단위)
RLIMIT_STACK	스택 최대 크기(바이트 단위)
RLIMIT_VMEM	RLIMIT_AS와 동일

```
struct rlimit {
rlim_t rlim_cur;        // 약한 한계
rlim_t rlim_max;        // 강한 한계
}
```

getrlimit() 예제 p.229

getpid(2), getppid(2), getuid(2), geteuid(2), getgid(2), getegid(2)

```
#include <unistd.h>
#include <sys/types.h>
pid_t getpid(void);
리턴 값: 호출한 프로세스의 프로세스 ID
pid_t getppid(void);
리턴 값: 호출한 프로세스의 부모 프로세스 ID
uid_t getuid(void);
리턴 값: 호출한 프로세스의 실제 사용자 ID
uid_t geteuid(void);
리턴 값: 호출한 프로세스의 유효 사용자 ID
gid_t getgid(void);
리턴 값: 호출한 프로세스의 실제 그룹 ID
gid_t getegid(void);
리턴 값: 호출한 프로세스의 유효 그룹 ID
```

- pid(프로세스 ID)들을 리턴하는 시스템호출 함수
- Pid
  - 프로세스마다 고유한 pid
  - 음이 아닌 정수
  - 프로세스를 구분하는 유일한 속성 => 프로세스의 식별자
- 프로세스 종료 => 해당 pid는 다른 프로세스의 pid로 사용

getpid() 예제

<ssu_getpid.c>

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

int main(void)
{
   printf("Process ID       = %d\n", getpid());
   printf("Parent process ID   = %d\n", getppid());
   printf("Real user ID       = %d\n", getuid());
   printf("Effective user ID   = %d\n", geteuid());
   printf("Real group ID      = %d\n", getgid());
   printf("Effective group ID  = %d\n", getegid());
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_getpid
Process ID       = 30288
Parent process ID   = 11804
Real user ID       = 0
Effective user ID   = 0
Real group ID      = 0
Effective group ID  = 0
```

fork(2)

```
#include <unistd.h>
pid_t fork(void);
리턴 값: 자식 프로세스의 경우 0, 부모 프로세스의 경우 자식의 pid,에러 시 –1을 리턴하고 errno 설정
```

- 기존 프로세스가 새 프로세스를 생성할 때 사용하는 시스템호출 함수
- 부모프로세스 vs 자식 프로세스
- 한 번 호출되나 두 개의 리턴 값을 리턴하는 함수
  - 자식에게 0 리턴
  - 부모에게 자식 프로세스의 ID
- 프로세스 생성이 실패한 경우 –1을 리턴, 에러의 내용을 errno 변수에 저장

자식에게 상속되는 프로세스 속성 vs 상속되지 않는 속성

SUID 플래그와 SGID 플래그
현재 작업 디렉토리
루트 디렉토리
파일 생성 마스크
시그널 마스크와 시그널 처리 설정들
열린 파일 디스크립터들에 대한 exec 시 닫기(close-on-exec) 플래그
환경변수
부착된 공유 메모리 영역들
메모리 매핑들
자원 한계들

fork의 리턴 값
해당 프로세스 ID들
부모 프로세스 ID
자식의 tms_utime, tms_stime, tms_cutime, tms_cstime 값들은 모두 0으로 설정
부모가 잠근 파일 lock 들은 자식에게 상속되지 않음
아직 발동되지 않은 경보(alarm)들은 자식에서 모두 해제됨
자식의 유보 중인 시그널 집합은 비워짐

fork() 예제 1

<ssu_fork_1.c>

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

char glob_str[] = "write to standard output\n";
int glob_val = 10;

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

   loc_val = 100;
   if (write(STDOUT_FILENO,
         glob_str, sizeof(glob_str)-1) != sizeof(glob_str) - 1) {
      fprintf(stderr, "write error\n");
      exit(1);
   }

   printf("before fork\n");

   if ((pid = fork()) < 0) {
      fprintf(stderr, "fork error\n");
      exit(1);
   }
   else if (pid == 0) {
      glob_val++;
      loc_val++;
   }
   else
      sleep(3);
   printf("pid = %d, glob_val = %d, loc_val = %d\n",
         getpid(), glob_val, loc_val);
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_fork_1
write to standard output
before fork
pid = 9950, glob_val = 11, loc_val = 101
pid = 9949, glob_val = 10, loc_val = 100
root@localhost:/home/oslab# ./ssu_fork_1 > temp
root@localhost:/home/oslab# cat temp
write to standard output
before fork
pid = 9952, glob_val = 11, loc_val = 101
before fork
pid = 9951, glob_val = 10, loc_val = 100
```

fork() 예제 2

<ssu_fork_2.c>

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

int main(int argc, char *argv[])
{
   pid_t pid;
   char character, first, last;
   long i;

   if ((pid = fork()) > 0) {
      first = 'A';
      last = 'Z';
   }
   else if (pid == 0) {
      first = 'a';
      last = 'z';
   }
   else {
      fprintf(stderr, "%s\n", argv[0]);
      exit(1);
   }
```

```
 for (character = first; character <= last; character++) {
      for (i = 0; i <= 100000; i++)
         ;

      write(1, &character, 1);
   }

   printf("\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_fork_2;./ssu_fork_2;./ssu_fork_2;./ssu_for
k_2
AabBcCdDeEfFghGiHjIkJlKmnLoMpNqOrPstQuRvSwxTyUz
VWXYZ
AabBcCdDeEfgFhGiHjIkJlmKnLoMpNqrOsPtQuvwxRySz
TUVWXYZ
aAbcBdCefDgEhFiGjHklImJnKopLqMrNsOtuPvQwRxSyTz
UVWXYZ
AabBcCdDefEgFhGiHjIklJmKnLoMpqNrOsPtQuvRwSxyTUVWXYZ
z
```

·fork() 예제 3 예제

<ssu_race.c>

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

static void ssu_charatatime(char *str);

int main(void)
{
   pid_t pid;

   if ((pid = fork()) < 0) {
      fprintf(stderr, "fork error\n");
      exit(1);
   }
   else if (pid == 0)
      ssu_charatatime("output from child\n");
   else
      ssu_charatatime("output from parent\n");

   exit(0);
}
```

```
static void ssu_charatatime(char *str) {
   char *ptr;
   int print_char;

   setbuf(stdout, NULL);

   for (ptr = str; (print_char = *ptr++) != 0; ) {
      putc(print_char, stdout);
      usleep(10);
   }
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_race
ooutuptputut  frforomm  pacrhielndt
```

vfork(2)

```
#include <unistd.h>
#include <sys/types.h>
pit_d vfork(void);
```
리턴 값: 자식 프로세스의 경우 0, 부모 프로세스의 경우 자식의 프로세스 ID에러 시 –1을 리턴하고 errno가 설정됨

- 생성될 새 프로세스가 exec 계열 함수로 새 프로그램을 실행하는 경우에 사용하는 시스템호출 함수
- 자식이 먼저 실행됨이 보장
  - 부모의 프로세스 공간을 자식에게 복사하지 않음
  - vfork() 호출 이후 즉시 exec 계열 함수를 호출할 것으로 생각하고 부모의 주소 공간을 참조하는 일이 없음
  - 부모의 주소 공간이 복사되지 않으므로 약간의 성능 향상
  - 자식 프로세스는 exec 계열 함수나 exit() 호출을 할 때까지 부모 프로세스의 메모리 영역에서 실행되며, 부모 프로세스는 실행을 멈춤
  - vfork()를 호출하면 자식 프로세스가 부모 프로세스의 메모리 공간에서 실행되고 있는 동안은 부모 프로세스가 실행되지 않음
  - 부모 프로세스는 vfork()를 호출하고 멈춰 있다가, 자식 프로세스가 exec 계열 함수 또는 exit()를 호출한 후에야 재개
  - 자식 프로세스가 좀비가 되는 것을 방지하고 자식 프로세스가 항상 먼저 실행되는 것을 보장

vfork() 예제

<ssu_vfork.c>

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

void ssu_local_func(void);

int main(void)
{
   printf("Before vfork\n");
   ssu_local_func();
   printf("After ssu_local_func, my PID is %d\n", getpid());
   exit(0);
}

void ssu_local_func(void) {
   pid_t pid;

   if ((pid = vfork()) == 0)
      printf("I'm child. My PID is %d\n", getpid());
   else if (pid > 0)
      sleep(3);
   else
      fprintf(stderr, "vfork error\n");
}
```

실행 결과 [Ubuntu]

```
root@localhost:/home/oslab# ./ssu_vfork
Before vfork
I'm child. My PID is 10193
After ssu_local_func, my PID is 10193
After ssu_local_func, my PID is 0
```

실행 결과 [Fedora]

```
[root@localhost oslab]# ./ssu_vfork
Before vfork
I'm child. My PID is 11488
After ssu_local_func, my PID is 11488
```

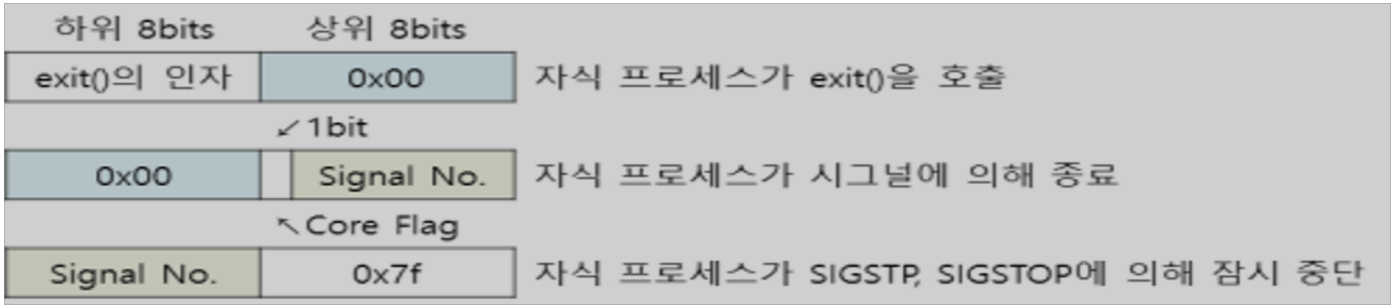
실행 결과 [Mac]

```
ssu-ui-MacBook-Air:oslab root# ./ssu_vfork
Before vfork
I'm child. My PID is 3143
After ssu_local_func, my PID is 3143
Segmentation fault: 11
```

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
```