UNIT

10

프로세스

로봇SW 교육원

최상훈(shchoi82@gmail.com)

학습 목표

- 리눅스 프로세스 환경의 이해

main 함수

- **프로세스의 시작**
  - exec **함수(시스템 콜) 호출**
  - **커널에 의해 실행됨**

- **시동 루틴**(start-up routine)
  - main**함수 호출되기전에 호출**
  - **시동 루틴의 주소가 프로그램의 시작주소**
    - **링커에 의해 프로그램 실행파일에 설정됨**

- **시동 루틴의 역할**
  - **실행에 필요한 제반사항을 준비**
    - **커널로 부터 명령줄 인수와 환경 변수를 전달 받음**
  - main**함수가 반환되면** exit**를 호출**

- C **프로그램의 시작**
  - main**함수의 원형**(prototype)

```
int main(int argc, char *argv[])
```

프로세스 종료

- **프로세스가 종료되는 상황**(8**가지**)
  - **정상적인 종료**(5)
    - main **의 반환**(return)
    - exit **호출**
    - _exit **또는** _Exit **호출**
    - **마지막 스레드를 시작한 스레드 시동 루틴의 반환**(return)
    - **마지막 스레드의** pthread_exit **호출**
  - **비정상적인 종료**(3)
    - abort **호출**
    - signal **수신**
    - **마지막 스레드의 실행 취소**
- main **함수의 반환**(return)
  - **시동 루틴으로 돌아감**
  - **시동 루틴에서** exit**를 호출**
  - main **함수의 반환 값은** exit**함수의 인자**

```
exit(main(argc, argv));  // exit(0) == main의 return(0)
```

프로세스 종료 : 종료 함수들


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

- exit **는 표준** I/O **라이브러리의 마무리 작업을 수행**
  - **버퍼에 남겨진 출력 자료 모두 방출**
  - **현재 열린 스트림을 모두 닫음**(fclose)
  - **모든 작업 완료 후** _Exit**와** _exit**를 호출을 통해 커널로 반환**
- _Exit **와** _exit**는 커널로 즉시 반환됨**
- **셸에서 종료 상태** (exit status) **확인**
  - **마지막 프로세스의 종료 상태 확인**
  - bash shell

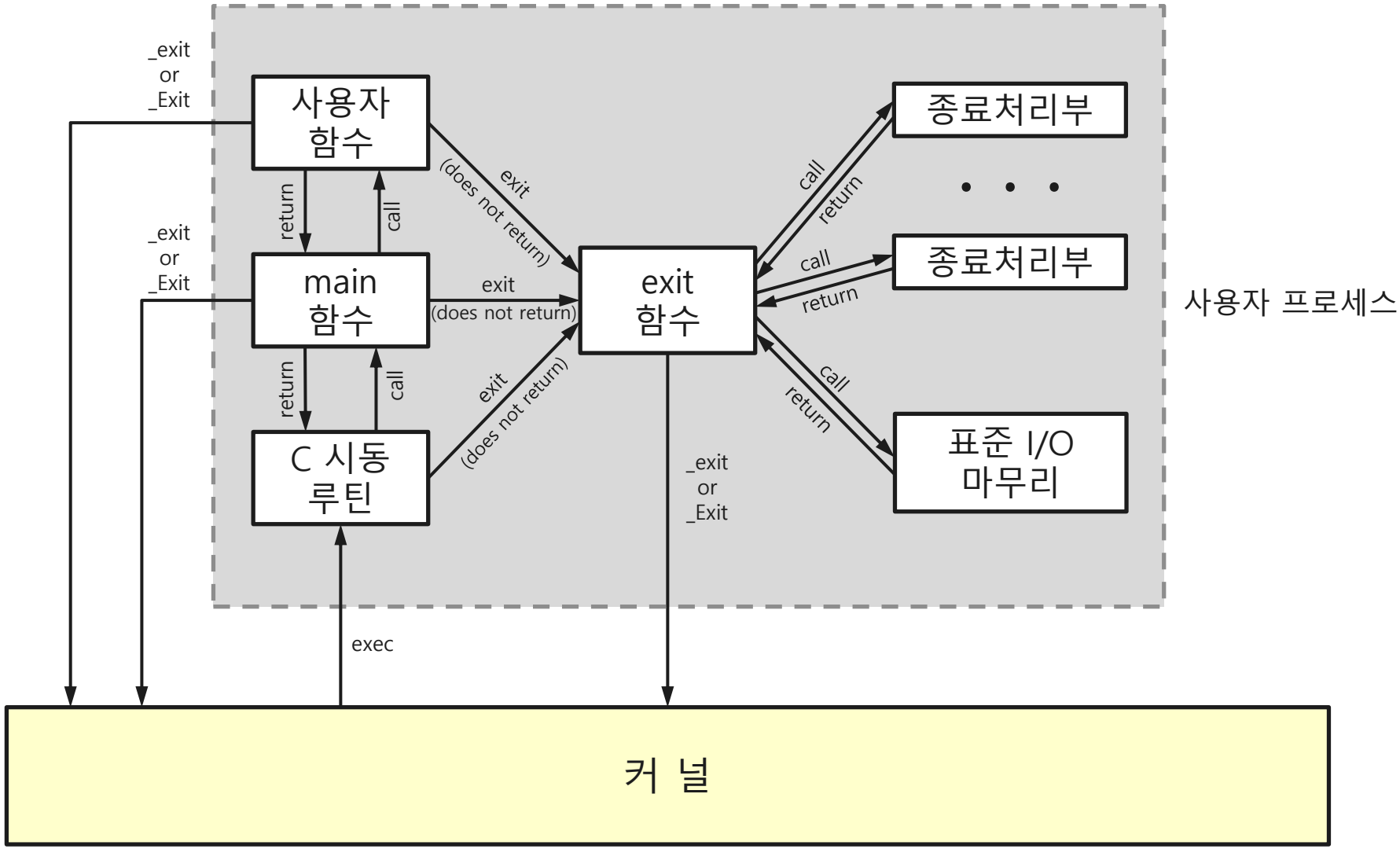
```
$ echo $?
```

프로세스 종료 처리부(exit handler)


```
#include <stdlib.h>
int atexit(void (*func)(void));
                                Returns : 0 if OK, nonzero on error
```

- **종료 처리부** (exit handler)
  - exit **호출시 자동으로 호출되는 함수**
  - **최소** 32**개 등록 가능**(ISO C**표준**)
- atexit **를 통해 등록**
  - **종료 처리부로 등록할 함수의 주소(함수 포인터)**
  - **등록한 순서의 역순으로 호출됨**
- exit **는 등록된 종료 처리부 함수들을 모두 호출한 후 열린 스트림들을 모두 닫음**(fclose)
- exec **함수가 호출되면 현재 등록된 종료 처리부 함수들은 해제됨**(POSIX.1)

프로세스의 시작과 종료


_exit
or
_Exit
사용자
함수
return
call
exit
(does not return)
종료처리부
call
return
• • •
종료처리부
_exit
or
_Exit
main
함수
exit
(does not return)
exit
함수
call
return
return
call
exit
(does not return)
call
return
C 시동
루틴
표준 I/O
마무리
_exit
or
_Exit
exec
커 널
사용자 프로세스

실습 1: atexit


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

static void my_exit1(void);
static void my_exit2(void);

int main(void)
{
    if(atexit(my_exit2) != 0)
        fprintf(stderr, "can't register my_exit2");

    if(atexit(my_exit1) != 0)
        fprintf(stderr, "can't register my_exit1");

    if(atexit(my_exit1) != 0)
        fprintf(stderr, "can't register my_exit1");

    printf("main is done\n");
    return(0);
}

static void my_exit1(void)
{
    printf("first exit handler\n");
}

static void my_exit2(void)
{
    printf("second exit handler\n");
}
```

```
$ ./atexit
main is done
first exit handler
first exit handler
second exit handler
$
```

명령줄 인수

- Command-Line Arguments
- exec**를 통해서 새 프로그램에게 명령줄 인수들을 전달**

  ※ shell**을 통한 명령어 입력**, **내부적으로** exec**를 호출**
- **main함수의 인자**

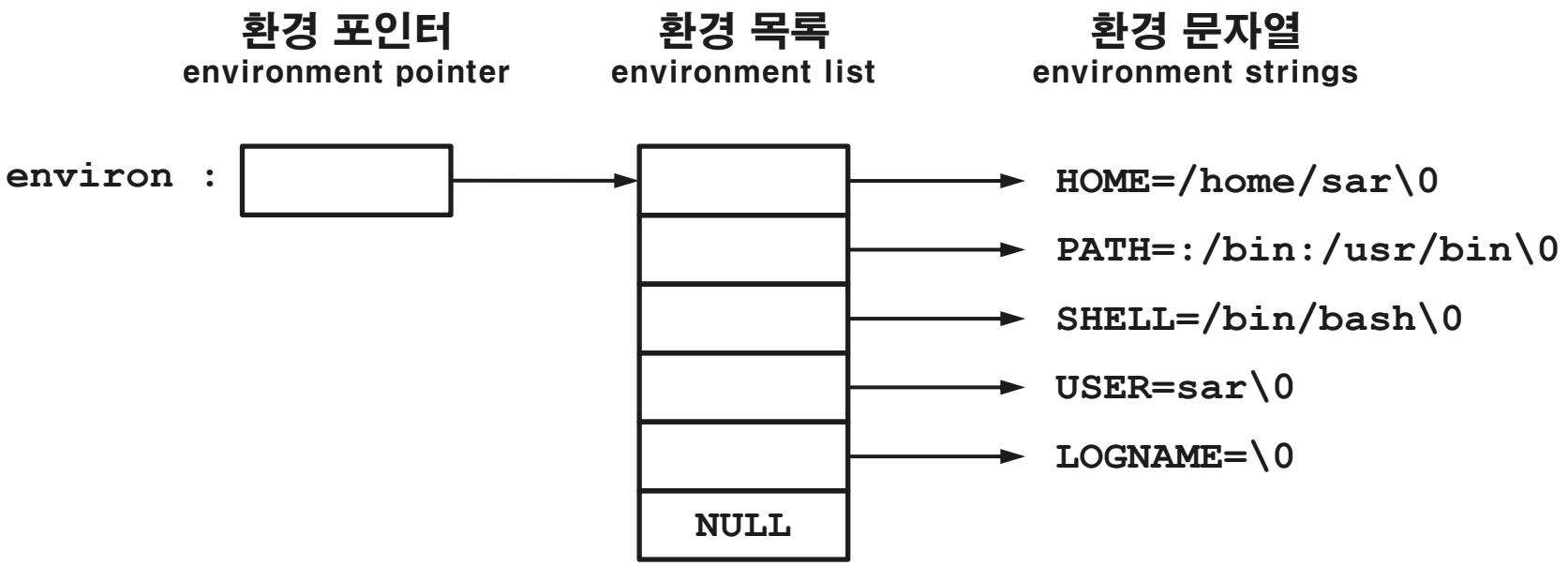
  `int argc, char *argv[]`

  `argv[argc] == NULL` (ISO C, POSIX.1)

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

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

    for(i = 0 ; i < argc ; i++)
//  for(i = 0 ; argv[i] != NULL ; i++)
        printf("argv[%d]: %s\n", i, argv[i]);
    exit(0);
}
```

```
$ ./echoarg arg1 pi raspberry
argv[0]: ./echoarg
argv[1]: arg1
argv[2]: pi
argv[3]: raspberry
$
```

환경 목록


환경 포인터
environment pointer
환경 목록
environment list
환경 문자열
environment strings
environ :
HOME=/home/sar\0
PATH=:/bin:/usr/bin\0
SHELL=/bin/bash\0
USER=sar\0
LOGNAME=\0
NULL

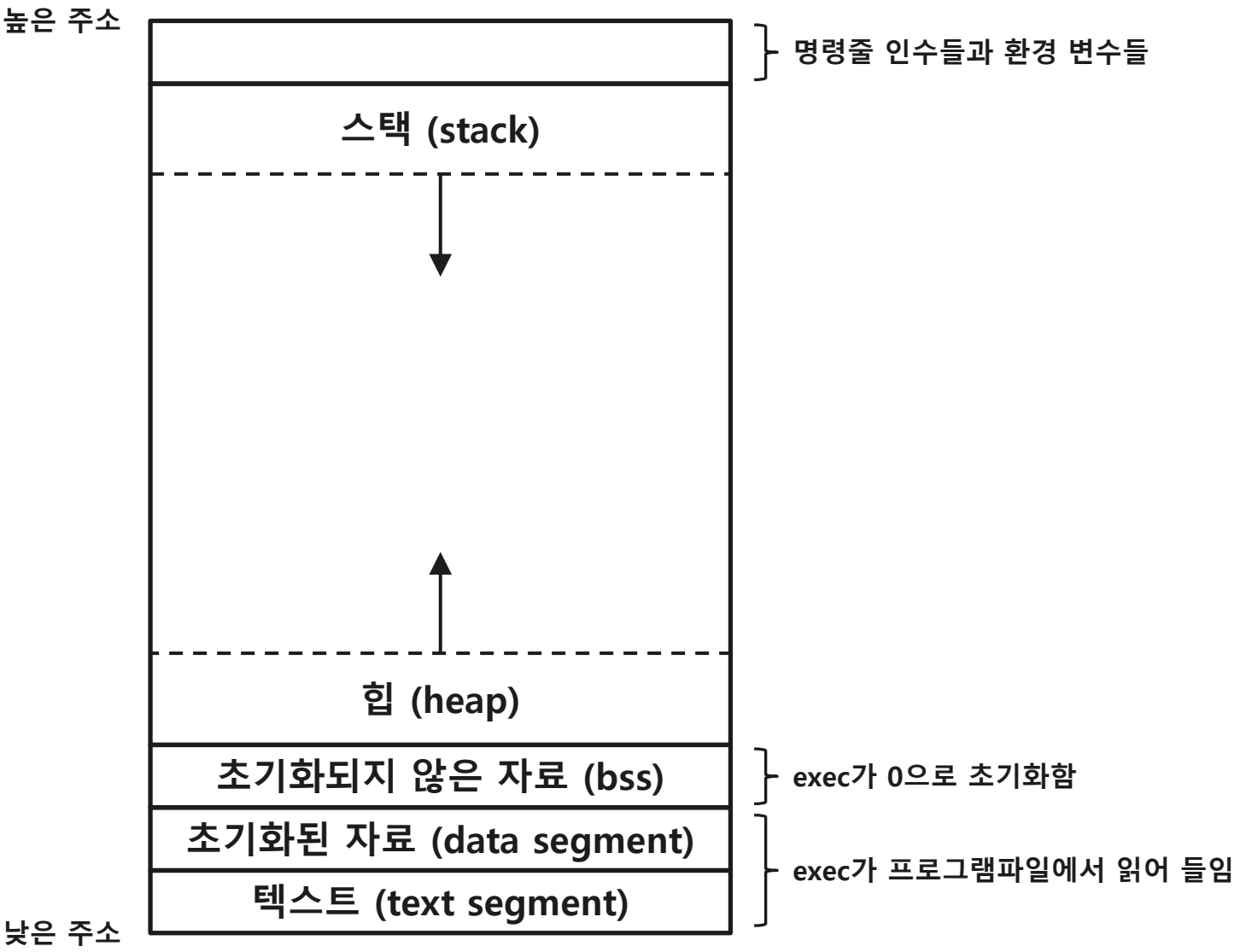
환경 목록

- **문자열 포인터들의 배열**
- **전역 변수** environ
  - extern char **environ;
- **환경 포인터, 환경 목록, 환경 문자열**
- **환경 문자열**
  - **형태 : 이름=값**
  - **이름은 보통 영문 대문자로 표현(관례)**
- **특정 환경 변수에 접근할때** getenv, putenv**을 일반적으로 사용함**
- **전체 목록을 접근할 때** environ **환경 포인터 사용**

실습 2: environ


```
#include <stdio.h>

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

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

C 프로그램의 메모리 구성



C 프로그램의 메모리 구성

- **텍스트 구역**(text segment)
  - CPU**가 실행가능한 기계어 명령들이 있는 구역**
  - **프로세스간 공유 가능**
  - **보통 읽기전용으로 지정됨**

- **초기화된 자료 구역**(initialized data segment)
  - **자료구역**(data segment) **이라고 부르기도 함**
  - **프로그램 안에서 '명시적으로 초기화된' 전역변수들이 있는 구역**

```
ex) int maxcount = 99;
```

- **초기화되지 않은 자료 구역**(uninitialized data segment)
  - 'bss' **구역 이라고 부르기도 함(고대의 어셈블러 연산자에서 따온 표현)**
    - bss : block started by symbol
  - **프로그램이 시작되기 전 커널에 의해** 0 **또는 널 포인터로 초기화됨**
  - **프로그램 안에서 '초기화되지 않은' 전역변수들이 있는 구역**

```
ex) long sum[1000];
```

C 프로그램의 메모리 구성

- **스택**(stack)
  - **함수의 자동변수와 함수 호출에 대한 정보가 저장되는 구역**
  - **함수의 자동**(auto) **변수 저장**
  - **함수 호출에 대한 정보 저장**
    - **함수의 반환주소**
    - **호출자의 환경에 대한 정보**(CPU**의 레지스터 등**)
  - **함수 호출시 새로운 스택 프레임이 생성됨**
    - **재귀호출을 가능하게 함**
      - **재귀 함수가 자신을 호출할 때마다 새로운 스택 프레임이 생성하기 때문에 한 호출의 변수들과 다른 호출의 변수들이 엉키지 않음**
  - Activation Record
- **힙**(heap)
  - **동적 메모리 할당이 주로 일어나는 곳**
  - **초기화되지 않은 자료구역과 스택 사이에 위치함**
- size(1)**명령어**

실습 3: size 명령어


```
$ size /usr/bin/passwd
   text    data     bss     dec     hex filename
  32320    3292    1248   36860    8ffc /usr/bin/passwd
$ size ./environ
   text    data     bss     dec     hex filename
   1008     292       8    1308     51c ./environ
$ size /usr/bin/cc
   text    data     bss     dec     hex filename
 273780    1956    5484  281220   44a84 /usr/bin/cc
$ size /bin/sh
   text    data     bss     dec     hex filename
  83009     916   10004   93929   16ee9 /bin/sh
$
```

공유 라이브러리

- **공유 라이브러리**(shared libraries)
  - **공통 루틴들의 복사본 하나를 메모리에 두고 실행파일에는 그 루틴으로 연결(참조)하는 데 필요한 정보만 저장**

- **장점**
  - **실행 파일의 크기를 줄일 수 있음**
  - **라이브러리가 갱신 되었을 때 그 라이브러리의 함수를 사용하는 모든 프로그램을 다시 링크하지 않고도 라이브러리를 새 버전으로 대체할 수 있음**

- **단점**
  - **프로그램 처음 실행되거나 또는 처음 공유라이브러리 함수가 호출될 때 오버헤드가 있을 수 있음**

실습 4: 공유 라이브러리


```
#include<stdio.h>

int main(int argc, char *argv[])
{
    printf("hello world\n");
    return 0;
}
```

```
$ gcc -Wall -W hello.c -o hello1
hello.c: In function ‘main’:
hello.c:3:14: warning: unused parameter ‘argc’ [-Wunused-parameter]
hello.c:3:26: warning: unused parameter ‘argv’ [-Wunused-parameter]
$ gcc -Wall -W -static hello.c -o hello2
hello.c: In function ‘main’:
hello.c:3:14: warning: unused parameter ‘argc’ [-Wunused-parameter]
hello.c:3:26: warning: unused parameter ‘argv’ [-Wunused-parameter]
$ size ./hello1 ./hello2
   text    data     bss     dec     hex filename
    840     292       4    1136     470 ./hello1
 489400    2000    6392  497792   79880 ./hello2
$
```

메모리 할당


```
#include <stdlib.h>
void *malloc(size_t size);
void *calloc(size_t nobj, size_t size);
void *realloc(void *ptr, size_t newsize);
     All three return : non-null pointer if OK, NULL on error

void free(void *ptr);
```

- malloc
  - **지정된 개수의 바이트를 할당, 메모리 초기화 하지 않음**

- calloc
  - **지정된 개수의 바이트를 할당, 메모리를 모두 0으로 초기화**

메모리 할당

- realloc
  - **이미 할당된 메모리 영역의 크기를 늘리거나 줄임**
  - **충분한 공간이 있지 않으면 다른 곳으 이동**
  - **새로운 영역으로 이동할 경우 기존의 포인터는 더 이상 유효하지 않음**
  - **확장된 영역은 초기화 되지 않음**
  - **마지막 인수는 새 영역의 전체크기**
  - realloc(NULL, newsize) = malloc(newsize)

- free
  - **할당된 영역을 해제함**
  - **메모리 누수**(memory leakage)

- sbrk **시스템 콜 호출**
  - **프로세스의** heap **영역을 확장함**

실습 5: 메모리 할당


```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#define BUFSIZE 30

int main(int argc, char *argv[])
{
    char *ptr[10];
    char fmt[10];
    int i = 0;

    for(i = 0 ; i < 10 ; i++){
        if((ptr[i] = (char *)calloc(BUFSIZE ,sizeof(char))) == NULL){
            fprintf(stderr, "calloc error\n"), exit(0);
        }
    }

    for(i = 0 ; i < 10 ; i++){
        sprintf(fmt, "string-%02d", i);
        strcpy(ptr[i], fmt);
    }

    for(i = 0 ; i < 10 ; i++)
        printf("ptr[%d] : %s\n", i, ptr[i]);

    return 1;
}
```

```
$ ./stringptarray
ptr[0] : string-00
ptr[1] : string-01
ptr[2] : string-02
ptr[3] : string-03
ptr[4] : string-04
ptr[5] : string-05
ptr[6] : string-06
ptr[7] : string-07
ptr[8] : string-08
ptr[9] : string-09
```

실습 6: 메모리 할당


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

int main(int argc, char *argv[])
{
    char *ptr = NULL;
    char *tmp = NULL;

    if((ptr = malloc(10)) == NULL)
        fprintf(stderr, "malloc error\n"),exit(1);
    if((tmp = malloc(10)) == NULL)
        fprintf(stderr, "malloc error\n"),exit(1);

    printf("ptr address : 0x%08x\n", ptr);
    printf("tmp address : 0x%08x\n", tmp);

    strcpy(ptr, "123456789" );
    printf("ptr : %s\n", ptr);

    ptr = realloc(ptr, 100);
    printf("after realloc()\nptr address : 0x%08x\n", ptr);

    strcat(ptr, "abcdefghijk" );
    printf("ptr : %s\n", ptr);

    return 1;
}
```

```
$ ./realloc
ptr address : 0x00875010
tmp address : 0x00875030
ptr : 123456789
after realloc()
ptr address : 0x00875050
ptr : 123456789abcdefghijk
$
```

환경 변수


```
#include<stdlib.h>
char *getenv(const char *name);
            Returns: pointer to value associated with name, NULL if not found

#include<stdlib.h>
int putenv(char *str);
                                        All return: 0 if OK, nonzero on error

int setenv(const char *name, const char *value, int rewrite);
int unsetenv(const char *name);
                                            All return: 0 if OK, -1 on error
```

실습 7: 환경 변수


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

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

    for(i = 0 ; *(environ+i) != NULL ; i++);
    printf("env size : %d\n", i);

    putenv("ENVTEST=abcdefg");

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

    printf("\n\nenv size : %d\n", i);
    printf("ENVTEST=%s\n", getenv("ENVTEST"));

    return 1;
}
```

setjmp함수와 longjmp함수


```
#include<setjmp.h>
int setjmp(jmp_buf env);
    Returns: 0 if called directly, nonzero if returning from a call to longjmp
void longjmp(jmp_buf env, int val);
```

- setjmp**와** longjmp **이전 스택 프레임의 함수로 분기 가능**
- setjmp
  - **현재 지점(상태) 저장**
  - longjmp**에 의해**
- longjmp
  - **저장된 지점(상태)으로 돌아감**
- **매개변수**
  - jmp_buf env : setjmp**가 호출되었을 때의 지점(상태)로 되돌리는 데 필요한 모든 정보를 담은 일종의 배열**
  - int val : **어떤** longjmp**를 실행시켰는지 식별**

※ goto**문은 함수내에서만 분기 가능**

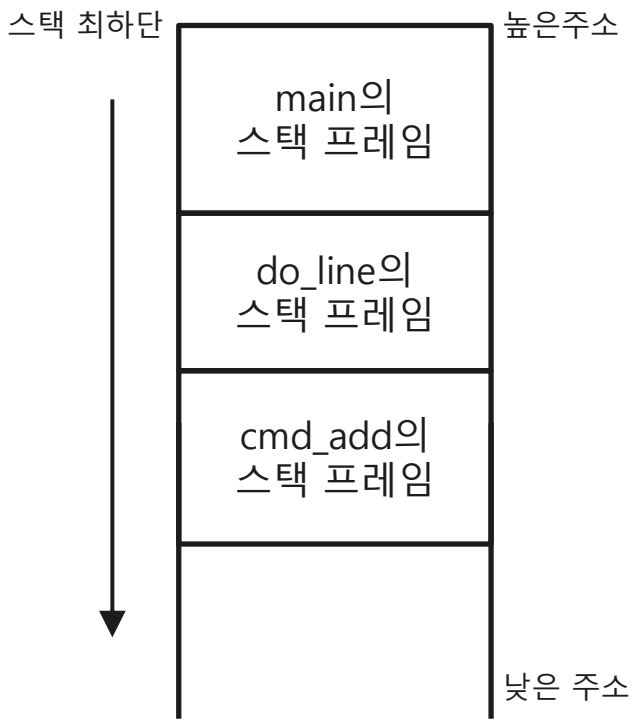
setjmp함수와 longjmp함수


```
#include<stdlib.h>

#define TOK_ADD       5
#define TOK_SUB       6
#define MAXLINE 80

void do_line(char *);
void cmd_add(void);
void cmd_sub(void);
int get_token(void);

int main(int argc, char *argv[])
{
    char line[MAXLINE];

    while(fgets(line, MAXLINE, stdin) != NULL)
        do_line(line);

    exit(0);
}

char *tok_ptr;
```

```
void do_line(char *ptr)
{
    int cmd;
    tok_ptr = ptr;
    while((cmd = get_token()) > 0){
        switch(cmd){
            case TOK_ADD:
                cmd_add();
                break;
            case TOK_SUB:
                cmd_sub();
                break;
        }
    }
}

void cmd_add(void)
{
    int token;

    token = get_token();
    printf("cmd_add\n");
}
```

setjmp함수와 longjmp함수


```
void cmd_sub(void)
{
    int token;

    token = get_token();
    printf("cmd_sub\n");
}

int get_token(void)
{
    printf("get_token\n");
    return TOK_ADD;
    //return TOK_SUB;
}
```



실습 8: setjmp함수와 longjmp함수(1/2)

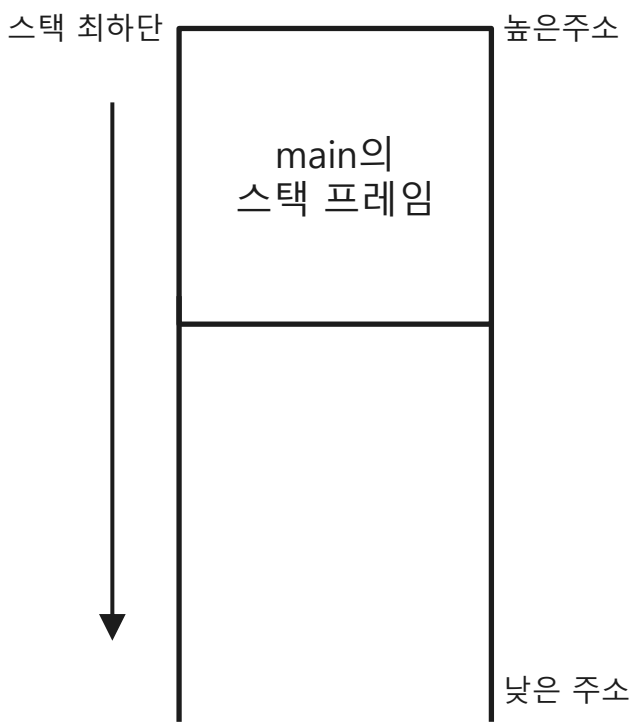

```
#include<stdio.h>
#include<setjmp.h>
#include<stdlib.h>

#define TOK_ADD 5
#define TOK_SUB 6
#define MAXLINE 80

void do_line(char *);
void cmd_add(void);
void cmd_sub(void);
int get_token(void);

jmp_buf jmpbuffer;

int main(int argc, char *argv[])
{
    char line[MAXLINE];

    if(setjmp(jmpbuffer) != 0)
        printf("error\n");
    while(fgets(line, MAXLINE, stdin) != NULL)
        do_line(line);
    exit(0);
}
```

```
char *tok_ptr;
void do_line(char *ptr)
{
    int cmd;
    tok_ptr = ptr;

    while((cmd = get_token()) > 0){
        switch(cmd){
            case TOK_ADD:
                cmd_add();
                break;
            case TOK_SUB:
                cmd_sub();
                break;
        }
    }
}

void cmd_add(void)
{
    int token = -1;
    if(token < 0)
        longjmp(jmpbuffer, 1);
    printf("cmd_add\n");
}
```

실습 8: setjmp함수와 longjmp함수(2/2)


```
void cmd_sub(void)
{
    int token = -1;

    if(token < 0)
        longjmp(jmpbuffer, 2);
    printf("cmd_sub\n");
}

int get_token(void)
{
    printf("get_token\n");
    return TOK_ADD;
    //return TOK_SUB;
}
```

```
$ ./longjmp
cmd aaa
get_token
error
$
```



longjmp가 호출된 후의 스택 프레임

실습 9: setjmp함수와 longjmp함수(1/2)


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

static void f1(int, int, int, int);
static void f2(void);

static jmp_buf jmpbuffer;

static int globval;

int main(void)
{
    int autoval;
    register int regival;
    volatile int volaval;
    static int statval;

    globval = 1;    autoval = 2;    regival = 3;    volaval = 4;    statval = 5;

    if(setjmp(jmpbuffer) != 0){
        printf("after longjmp:\n");
        printf("gloval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n"
                , globval, autoval, regival, volaval, statval);
        exit(0);
    }
    globval = 95;   autoval = 96;   regival = 97;   volaval = 98;   statval = 99;
    f1(autoval, regival, volaval, statval);
    exit(0);
}
```

실습 9: setjmp함수와 longjmp함수(2/2)


```
static void f1(int i, int j, int k, int l)
{
    printf("in f1():\n");
    printf("gloval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n"
            , globval, i, j, k, l);
    f2();
}

static void f2(void)
{
    longjmp(jmpbuffer, 1);
}
```

```
$ gcc -Wall -W longjmp.c -o longjmp
longjmp.c: In function ‘main’:
longjmp.c:15:15: warning: variable ‘regival’ might be clobbered by ‘longjmp’ or ‘vfork’ [-
Wclobbered]
$ ./longjmp
in f1():
gloval = 95, autoval = 96, regival = 97, volaval = 98, statval = 99
after longjmp:
gloval = 95, autoval = 96, regival = 97, volaval = 98, statval = 99
$ gcc -Wall -W -O3 longjmp.c -o longjmp
$ ./longjmp
in f1():
gloval = 95, autoval = 96, regival = 97, volaval = 98, statval = 99
after longjmp:
gloval = 95, autoval = 2, regival = 3, volaval = 98, statval = 99
$
```