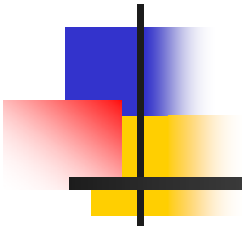
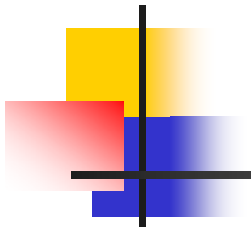
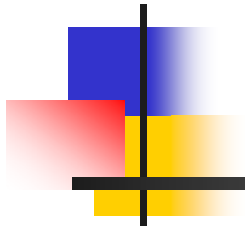


제6장 프로세스 (Process)



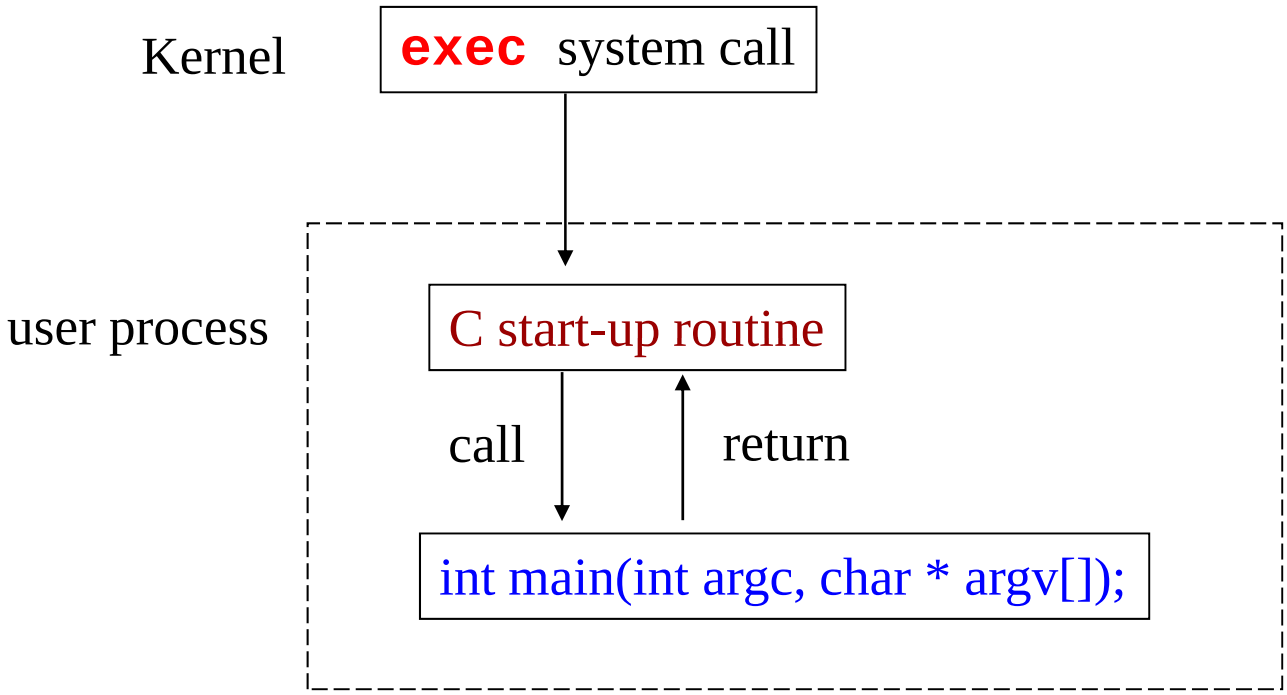
내용

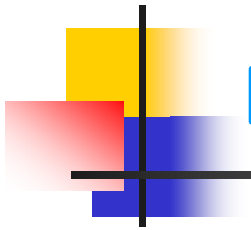
- 프로세스 시작/종료
- 명령중 인수/환경 변수
- 메모리 배치/할당
- 비지역 점프



프로세스 시작/종료

Process Start

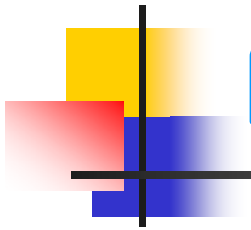




main()

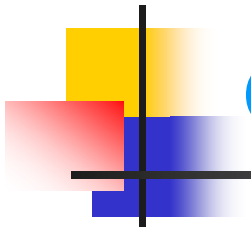
- int main(int argc, char *argv[]);
  - argc : the number of command-line arguments
  - argv[] : an array of pointers to the arguments
- C start-up routine
  - Started by the kernel (by the exec system call)
  - Take the command-line arguments and the environment from the kernel
  - Call main

    exit(main(argc, argv));



Process Termination

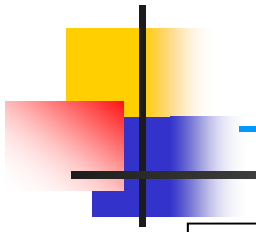
- Normal termination
  - return from main()
  - calling exit() :
  - calling _exit()
- Abnormal termination
  - calling abort()
  - terminated by a signal



exit()

```
#include <stdlib.h>

void exit(int status);
```

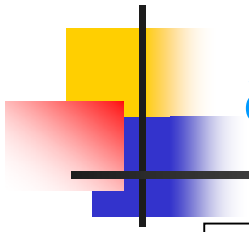
- 프로세스를 정상적으로 종료한다
- cleanup processing 을 수행한다
  - 모든 열려진 스트림을 닫고(fclose)
  - 출력 버퍼의 내용을 디스크에 쓴다(fflush)
- *status*
  - the exit status of a process
  - 이 값은 UNIX shell 에 의해서 사용됨



_exit()

```
#include <unistd.h>

void _exit(int status);
```

- 프로세스를 정상적으로 종료한다
- 커널로 즉시 리턴한다

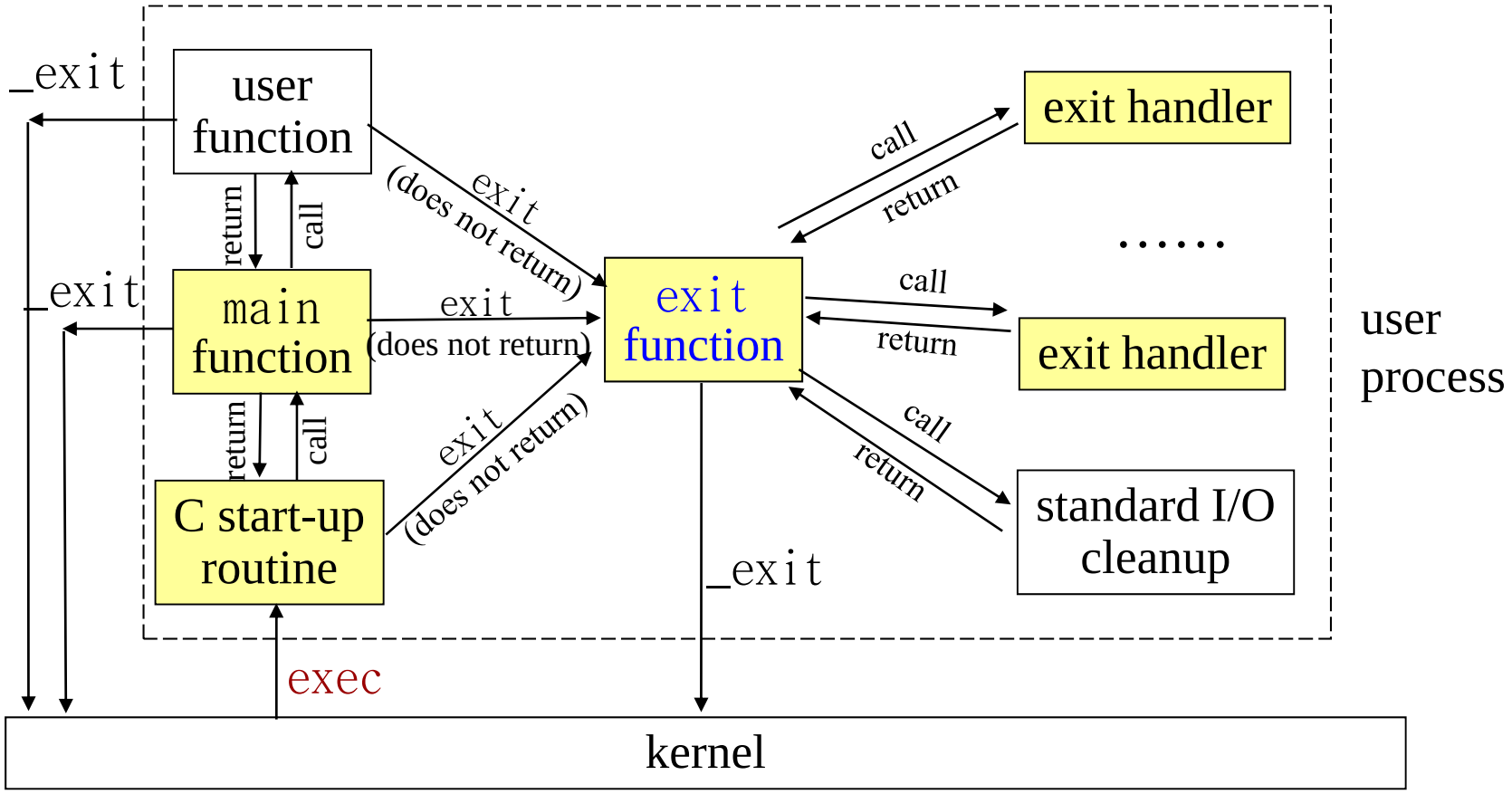


atexit()

```
#include <stdlib.h>

void atexit(void (*func)(void));
                        returns: 0 if OK, nonzero on error
```

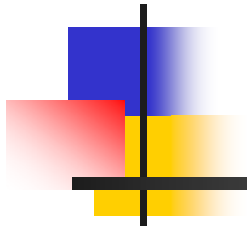
- exit handler 를 등록한다
  - 프로세스당 32개까지
- *func*
  - an exit handler
  - a function pointer
- exit() 는 exit handler 들을 등록된 역순으로 호출한다

C Program Start and Termination



Example of exit handlers

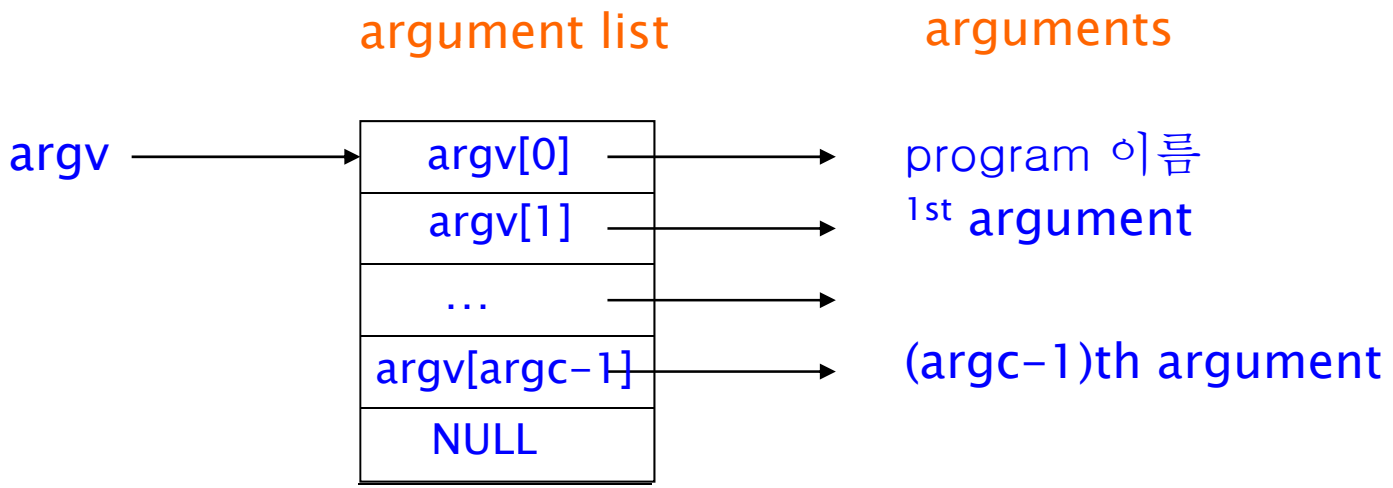
```
/* doatexit.c */
static void my_exit1(void), my_exit2(void);

int main(void) {
  if (atexit(my_exit2) != 0)          perror("can't register my_exit2");
  if (atexit(my_exit1) != 0)          perror("can't register my_exit1");
  if (atexit(my_exit1) != 0)          perror("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");
}
```



명령줄 인수/ 환경 변수

Command-Line Arguments

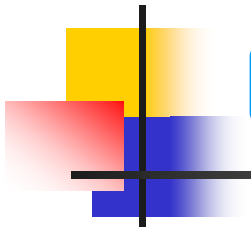
- exec() pass command-line arguments to a new program
  - argv[0], argv[1], …, argv[argc-1]
  - argv[argc] is NULL (ANSI, POSIX.1)



Echo command-line arguments

```
#include "ourhdr.h" /* echoarg.c */
int main(int argc, char *argv[]) {
  int    i;

  for (i = 0; i < argc; i++)    /* echo all command-line args */
    printf("argv[%d]: %s\n", i, argv[i]);
  exit(0);
}
```

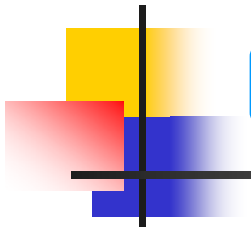
```
#include "ourhdr.h" /* echoarg1.c */
int main(int argc, char *argv[]) {
  int i = 0;
  char **p = argv;

  while (*p)
    printf("argv[%d]: %s\n", i++, *p++);
  exit(0);
}
```



Environment Variables

- 환경 변수(environment variable)
  - 부모 프로세스에서 자식 프로세스로 전달된다
- .login 또는 .cshrc 파일에서 환경 변수를 설정한다.
- 환경변수: 이름=값

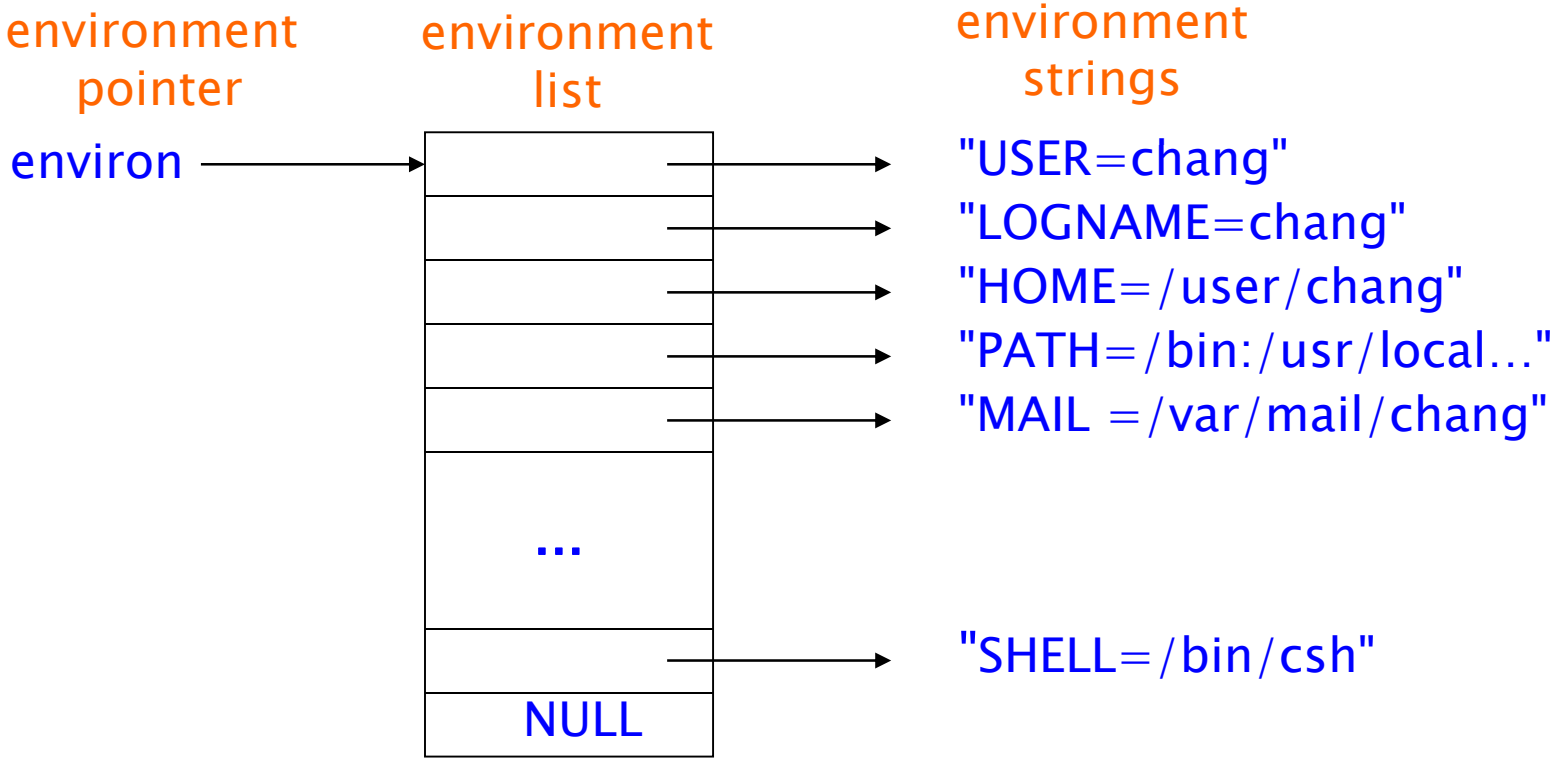
```
$ env
USER=lsj
LOGNAME=lsj
HOME=/user1/lsj
PATH=/bin:/usr/bin:/usr/local/bin:/usr/ccs/bin:/usr/ucb:/usr/open
    win/bin:/etc:.
MAIL=/var/mail/lsj
SHELL=/bin/csh
```

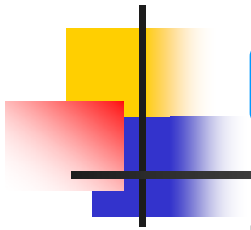


Environment List

- 전역 변수 **environ**을 이용하여 환경 변수에 접근한다.
  - extern char ** environ;
- 각 항목은 "환경 변수 이름=값" 의 형식
  - 각 문자열은 '\0'로 끝난다
  - 환경 변수 리스트의 마지막은 NULL 포인터
- argv 와 같은 구조

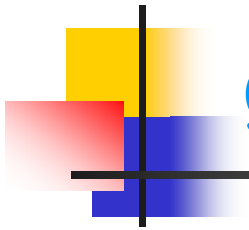
Environment List





Echoall.c

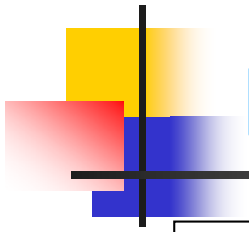
```
int main(int argc, char *argv[])
{
  int      i;
  char     **ptr;
  extern char  **environ;

  for (i = 0; i < argc; i++)     /* echo all command-line args */
    printf("argv[%d]: %s\n", i, argv[i]);

  for (ptr = environ; *ptr != 0; ptr++)   /* and all env strings */
    printf("%s\n", *ptr);

  exit(0);
}
```



getenv()

```
#include <stdlib.h>

char *getenv(const char *name);

Returns : pointer to value associated with name, NULL if not found
```

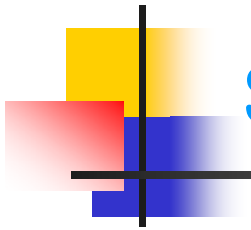
- 환경 변수 리스트에서 이름이 *name* 인 것을 찾아
- 값에 대한 포인터를 리턴한다
- 실패하면 NULL 포인터를 리턴



putenv()

```
#include <stdlib.h>

int putenv(const char *str);
                                    Returns: 0 if OK, nonzero on error
```

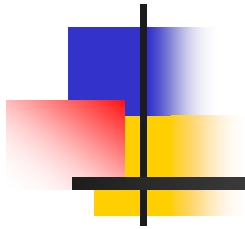
- 환경 변수를 추가한다
- *str*은 "name=value" 형식의 문자열
- 성공적으로 실행된 경우 0을 리턴
- 같은 이름의 환경 변수가 이미 있다면 새 값으로 변경된다



setenv() unsetenv()

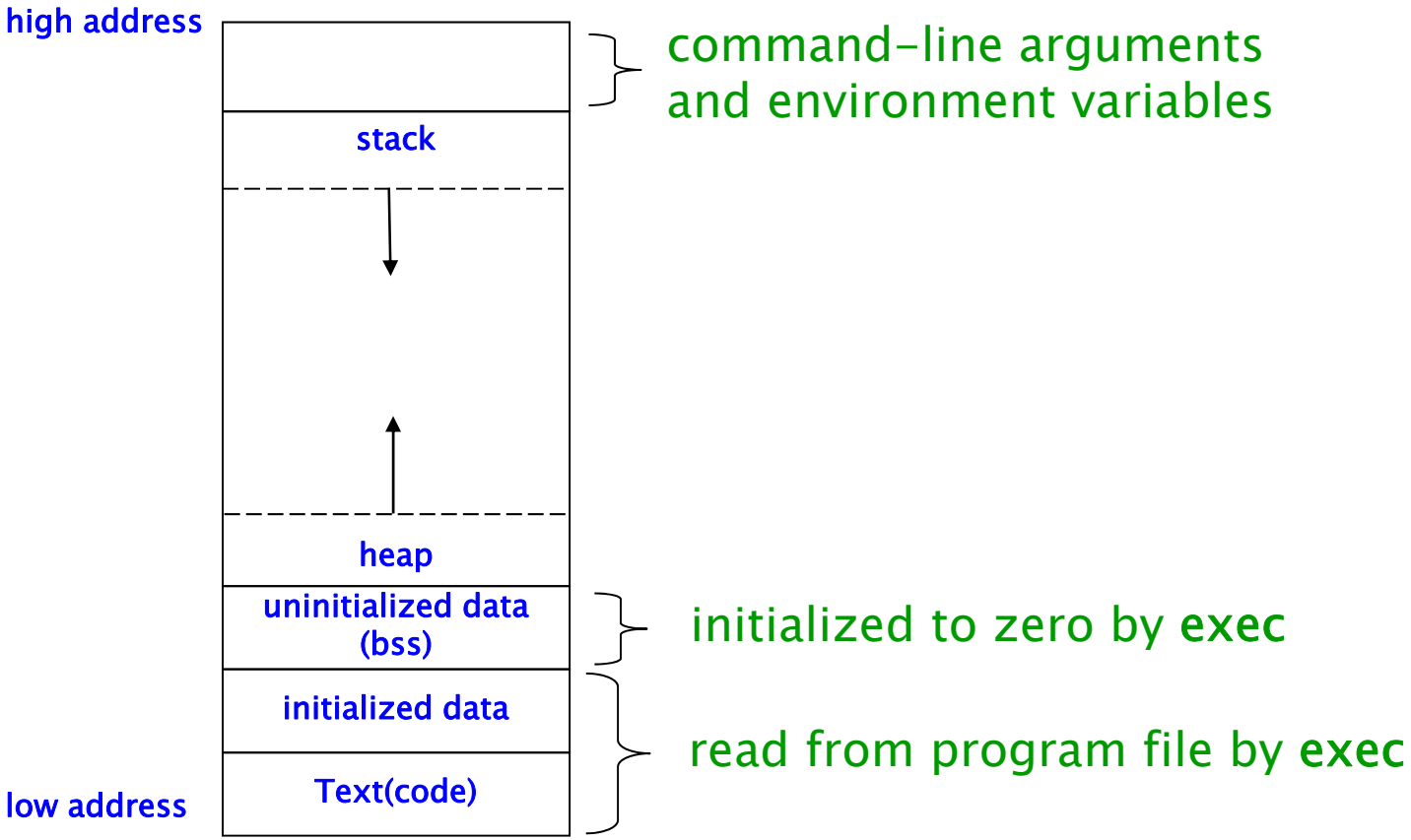
```
#include <stdlib.h>
int setenv(const char *name, const char *value, int rewrite);
                                        Returns: 0 if OK, nonzero on error
void unsetenv(const char *name);
```

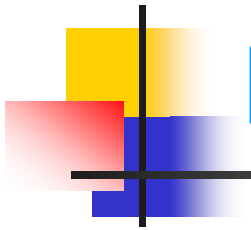
- setenv 는 환경 변수 *name* = *value* 을 등록한다
- *name* 의 환경변수가 이미 있을 경우
  - *rewrite* != 0 이면 새 값으로 변경되고
  - *rewrite* == 0 이면 값이 변경되지 않는다
- unsetenv 는 환경 변수 *name* 을 제거한다



메모리 배치/할당

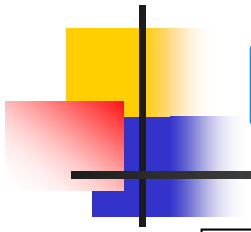
Memory Layout of a C program





Memory Layout of a C program

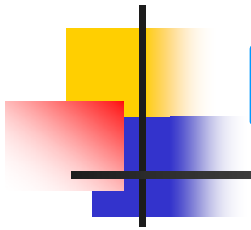
- Text(code) segment
  - Machine instructions (read-only, sharable)
- Initialized data segment
  - e.g. int maxcount = 99; (initialized)
- Uninitialized data segment
  - (bss: block started by symbol)
  - e.g. long sum[1000];
- Stack
  - automatic local variables, temporary variables, return address, caller's environment (registers)
- Heap
  - dynamic memory allocation



Memory Allocation

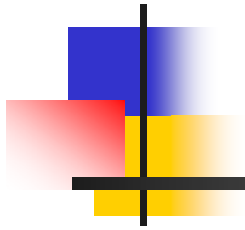
```
#include <stdlib.h>
void *malloc(size_t size);
void *calloc(size_t nobj, size_t size);
void *realloc(void *ptr, size_t newsize);
                        returns : nonnull pointer if OK, NULL on error
void free(void *ptr);
```

- dynamic allocation of memory from heap
- provide suitable alignment
  - ex) doubles must start at the addresses that are multiples of 8
- library manages memory pool

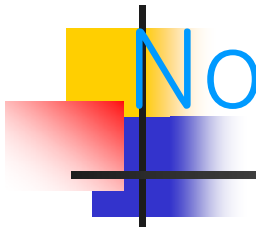


Memory Allocation

- malloc()
  - allocates specified number of bytes,
  - initial value of memory is indeterminate
- calloc()
  - allocates specified number of objects of specified size,
  - initialized to all 0 bits
- realloc()
  - changes size of previously allocated memory,
  - initial value of new area is indeterminate



비지역 점프(Nonlocal jumps)



Nonlocal Jumps: `setjmp/longjmp`

- Transfer control to an arbitrary location.
  - Direct nonlocal goto: setjmp/longjmp
  - a way to break the procedure call/return discipline
  - Useful for error/exception recovery and signal handling

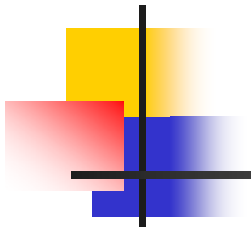
- **`int setjmp(jmp_buf env)`**
  - Must be called before longjmp
  - Identifies a return site for a subsequent longjmp.
  - Called once, returns one or more times

- **`void longjmp(jmp_buf env, int val)`**
  - Called after `setjmp`
  - Nonlocal goto the return site set by setjmp
  - Called once, but never returns

setjmp(), longjmp()

```
#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 store information at *env* for return to setjmp point
  - returns 0 if called directly
  - returns nonzero(*val*) if returning from a call to longjmp
- longjmp use *env* to jump to setjmp point and *val* as a return value.
  - Several longjmp can use the same setjmp location with different *val.*
- normally *env* is a global variable since it can be referenced from another function.



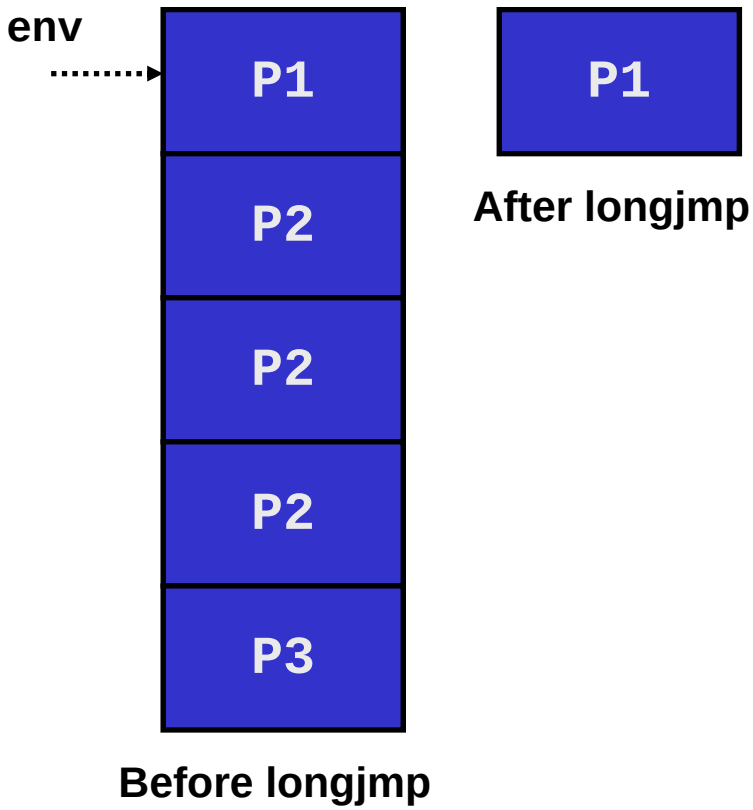
setjmp/longjmp Example

```
#include <setjmp.h>
jmp_buf buf;

main() {
   if (setjmp(buf) != 0) {
       printf("back in main due to an error\n");
       exit(0);
   }
   else
      printf("first time through\n");
   p1(); /* p1 calls p2, which calls p3 */
}
...
p3() {
   <error checking code>
   if (error)
      longjmp(buf, 1)
}
```

Limitations of Nonlocal Jumps

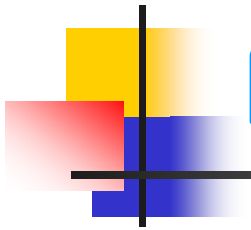
- Works within stack discipline
  - Can only long jump to environment of function that has been called but not yet completed

```
jmp_buf env;

P1()
{
  if (setjmp(env)) {
    /* Long Jump to here */
  } else {
    P2();
  }
}

P2()
{  . . . P2(); . . . P3(); }

P3()
{
  longjmp(env, 1);
}
```



결과: testjmp.c

```
#include  <setjmp.h>  /* testjmp.c */
static void  f1(int, int, int);                  static void  f2(void);
static jmp_buf  jmpbuffer;

int main(void) {
  int        count;
  register int  val;
  volatile int  sum;

  count = 2; val = 3; sum = 4;
  if (setjmp(jmpbuffer) != 0) {
    printf("after longjmp: count = %d, val = %d, sum = %d\n", count, val, sum);
    exit(0);
  }
  count = 97; val = 98; sum = 99; /* changed after setjmp, before longjmp */
  f1(count, val, sum);     /* never returns */
}

static void f1(int i, int j, int k) {
  printf("in f1(): count = %d, val = %d, sum = %d\n", i, j, k);
  f2();
}

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



Results: testjmp.c

```
$ cc testjmp.c
$ a.out
in f1(): count = 97, val = 98, sum = 99
after longjmp: count = 97, val = 98, sum = 99

$ cc -O testjmp.c
$ a.out
in f1(): count = 97, val = 98, sum = 99
after longjmp: count=2, val=3, sum=99
```