Linux Programming
10장. 메모리 관리

sisong@ut.ac.kr
한국교통대학교 컴퓨터공학전공
송석일

10.1 변수와 메모리

프로세스 구조

- 프로세스 구조

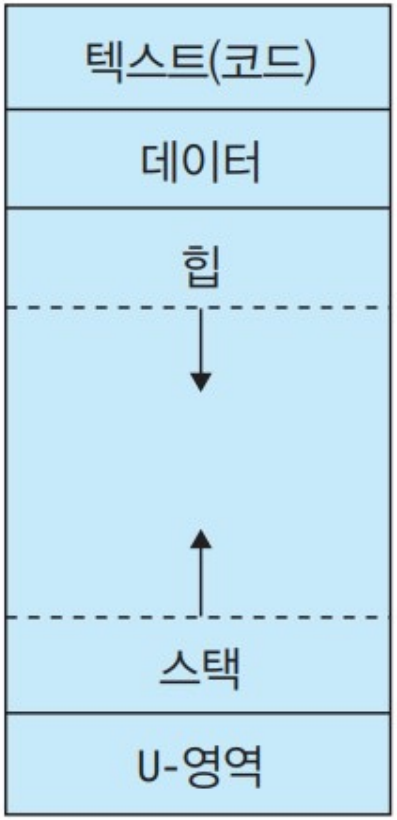


그림 10.1 프로세스 이미지

- 코드 세그먼트(code segment)
  - 기계어 명령어
- 데이터 세그먼트(data segment)

  int maxcount = 99; (initialized)

  long sum[1000]; (uninitialized)
- 스택(stack)
  - 지역 변수, 매개 변수,
  - 반환주소, 반환값, 등
- 힙(heap)
  - 동적 메모리 할당
  - malloc() in C,
  - new class() in Java

프로그램 시작할 때 메모리 영역

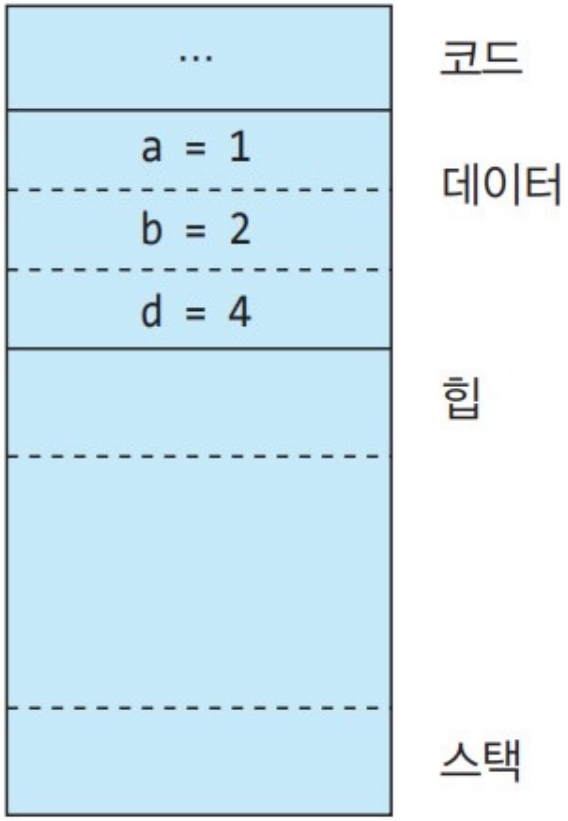


그림 10.2 프로그램 시작할 때 메모리 영역

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  int a = 1;
4.  static int b = 2;
5.
6.  int main() {
7.      int c = 3;
8.      static int d = 4;
9.      char *p;
10.
11.     p = (char *) malloc(40);
12.     fun(5);
13. }
14.
15. void fun(int n)
16. {
17.     int m = 6;
18.      ...
19. }
```

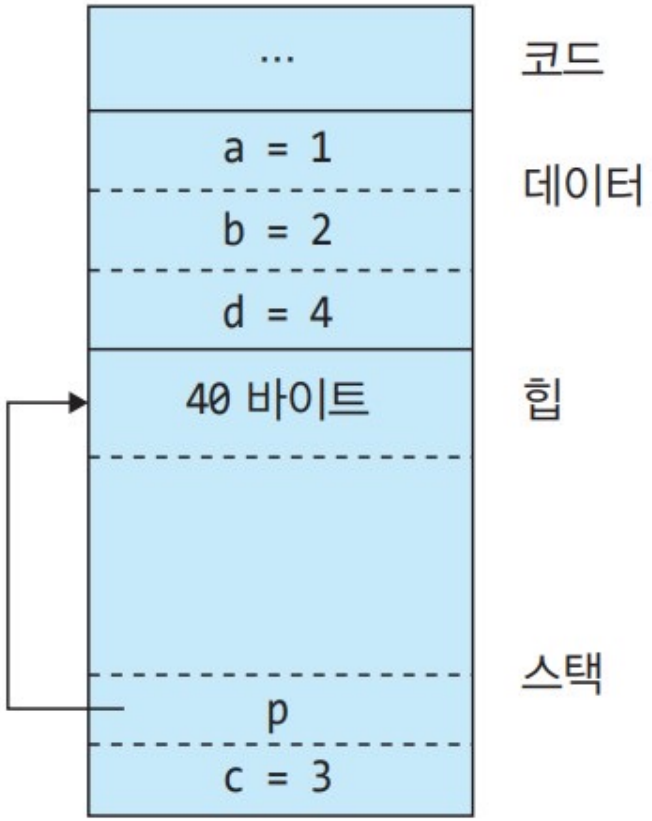


그림 10.3 main() 함수 실행할 때 메모리 영역

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  int a = 1;
4.  static int b = 2;
5.
6.  int main() {
7.      int c = 3;
8.      static int d = 4;
9.      char *p;
10.
11.     p = (char *) malloc(40);
12.     fun(5);
13. }
14.
15. void fun(int n)
16. {
17.     int m = 6;
18.      ...
19. }
```

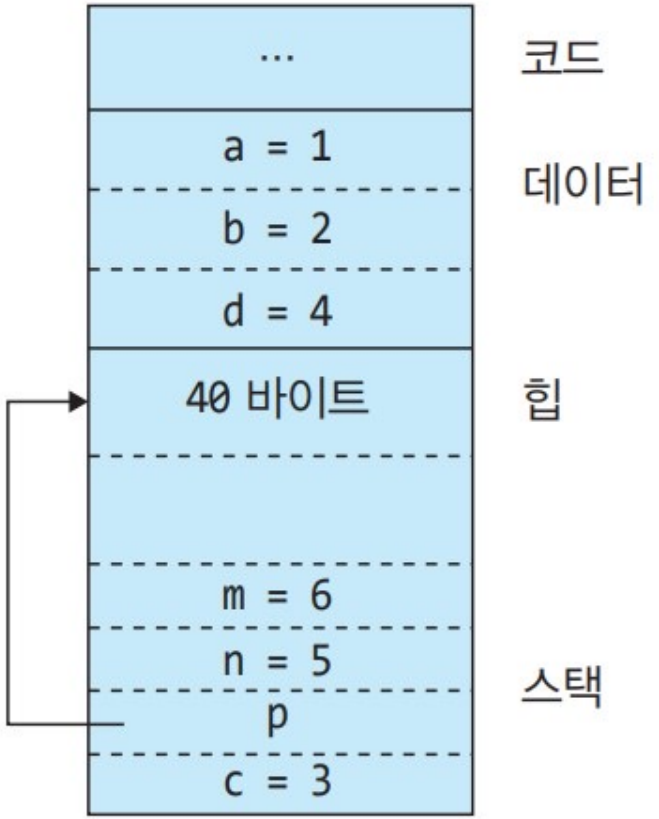


그림 10.4 함수 fun() 실행할 때 메모리 영역

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  int a = 1;
4.  static int b = 2;
5.
6.  int main() {
7.      int c = 3;
8.      static int d = 4;
9.      char *p;
10.
11.     p = (char *) malloc(40);
12.     fun(5);
13. }
14.
15. void fun(int n)
16. {
17.     int m = 6;
18.     int n = 5;
19.      ...
20. }
```

할당 방법에 따른 변수들의 분류

변수 구분	변수 종류
정적 변수	전역변수, static 변수
자동 변수	지역변수, 매개변수
동적 변수	힙 할당 변수

10.2 동적 메모리 할당

동적 메모리 할당

- 동적 할당 ?
  - 필요할 때 필요한 만큼만 메모리를 요청해서 사용
  - 메모리를 절약 가능
  - 하지만, 메모리 할당후 해제까지 직접 관리 필요

- 동적 할당 관련 시스템 호출
  - malloc( ), calloc( ), realloc( )
  - free( )

메모리 할당

#include <stdlib.h>

void *malloc(size_t *size*);

- size 크기의 메모리를 할당하며 그 시작주소를 void* 형으로 반환

void free(void **ptr*);

- 포인터 p가 가리키는 메모리 공간을 해제

- 힙에 동적 메모리 할당
- 라이브러리가 메모리 풀을 관리
- malloc() 함수는 메모리를 할당할 때 사용하고 free()는 할당한 메모리를 해제할 때 사용

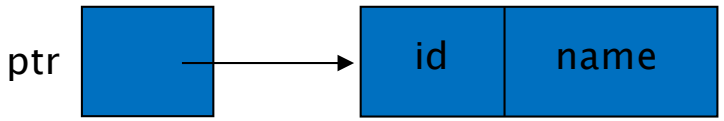
메모리 할당 예

```
char *ptr;
ptr = (char *) malloc(40);
```


```
int *ptr;
ptr = (int *) malloc(10 * sizeof(int));
```

구조체를 위한 메모리 할당 예

```
struct student {
        int id;
        char name[10];
};
struct student *ptr;
ptr = (struct student *) malloc(sizeof(struct student));
```



구조체 배열을 위한 메모리 할당 예

```
struct student *ptr;
ptr = (struct student *) malloc(n * sizeof(struct student));
```

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

struct student {
    int id;
    char name[20];
};

int main()
{
    struct student *p; int n, i;

    printf("몇 명의 학생을 입력하겠습니까? ");
    scanf("%d", &n);
    if (n <= 0) {
        fprintf(stderr, "오류: 학생 수 잘못 입력\n");
        fprintf(stderr, "프로그램 종료\n");
        exit(1);
    }

    p=(struct student *) malloc(n*sizeof(struct student));
    if (p == NULL) {
        perror("malloc");
        exit(2);
    }
    printf("%d 명의 학번과 이름을 입력하세요.\n", n);
    for (i = 0; i < n; i++)
        scanf("%d %s\n", &p[i].id, p[i].name);

    printf("\n* 학생 정보(역순) *\n");
    for (i = n-1; i >= 0; i--)
        printf("%d %s\n", p[i].id, p[i].name);

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

```
$ stud1
몇 명의 학생을 입력하겠습니까? 5
5 명의 학번과 이름을 입력하세요.
1001001 박연아
1001003 김태환
1001006 김현진
1001009 장샛별
1001011 홍길동
^D
* 학생 정보(역순) *
1001011 홍길동
1001009 장샛별
1001006 김현진
1001003 김태환
1001001 박연아
```

배열 할당 calloc()

```
#include <stdlib.h>
void *calloc(size_t n, size_t size);
```

- size 크기의 메모리를 n개 할당. 값을 모두 0으로 초기화
- 실패하면 NULL를 반환

```
#include <stdlib.h>
void *realloc(void *ptr, size_t newsize);
```

- ptr이 가리키는 이미 할당된 메모리의 크기를 newsize로 변경

malloc() vs calloc()

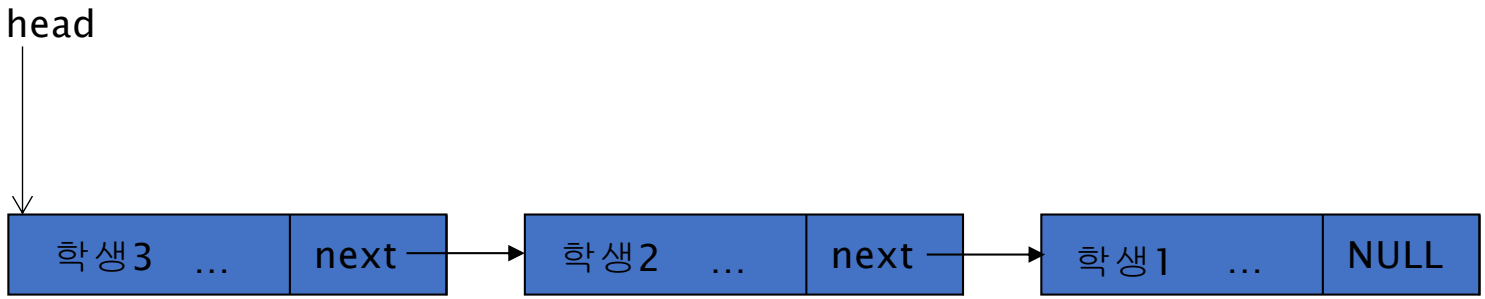
```
int *p,*q;
p = malloc(10*sizeof(int));
if (p == NULL)
        perror(“malloc”);

q = calloc(10, sizeof(int));
if (q == NULL)
        perror(“calloc”);
```

10.3 동적 할당과 연결 리스트

연결 리스트의 필요성

- 여러 학생들의 데이터를 저장 자료 구조
  - 구조체 배열
    - 배열의 크기를 미리 결정해야 함
    - 배열의 크기보다 많은 학생들은 처리할 수 없으며 이보다 적은 학생들의 경우에는 배열의 기억공간은 낭비
  - 동적 메모리 할당
    - 필요할 때마다 동적으로 메모리를 할당
    - 연결리스트(linked list)로 관리



10.4 공유 메모리

공유 메모리의 필요성

- 공유 메모리
  - 두개 이상 프로세스 사이에 메모리 영역을 공유해서 사용

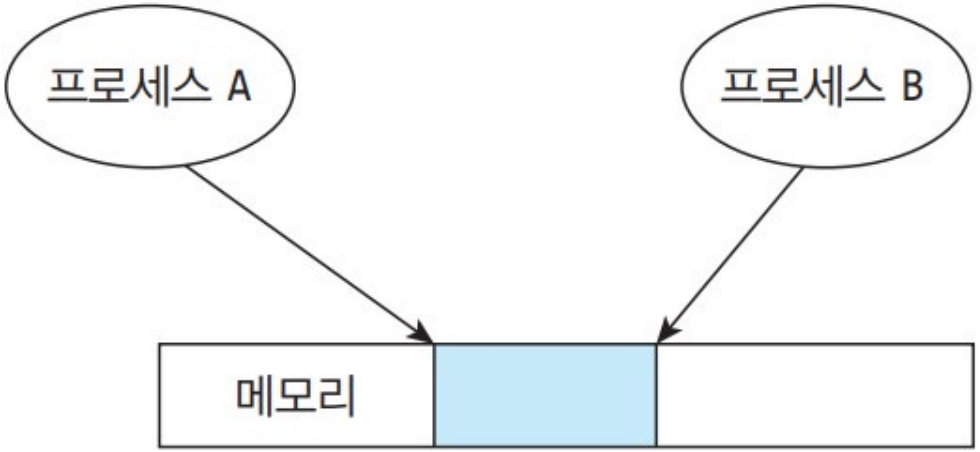


그림 10.11 공유 메모리 사용

공유 메모리 관련 함수 : shmget()

```
#include <sys/shm.h>
#include <sys/ipc.h>
int shmget(key_t key, size_t size, int shmflg);
```

- key를 사용하여 size 크기의 공유 메모리를 생성하고 생성된 공유 메모리의 ID 반환
  - IPC_PRIVATE : 항상 새로운 공유 메모리 생성
- shmflg
  - IPC_CREAT : 새로운 공유 메모리 생성. 생성할 공유 메모리의 접근권한을 함께 지정
  - IPC_EXCL : IPC_CREAT과 함께 사용되면 해당 공유 메모리가 이미 존재하면 실패

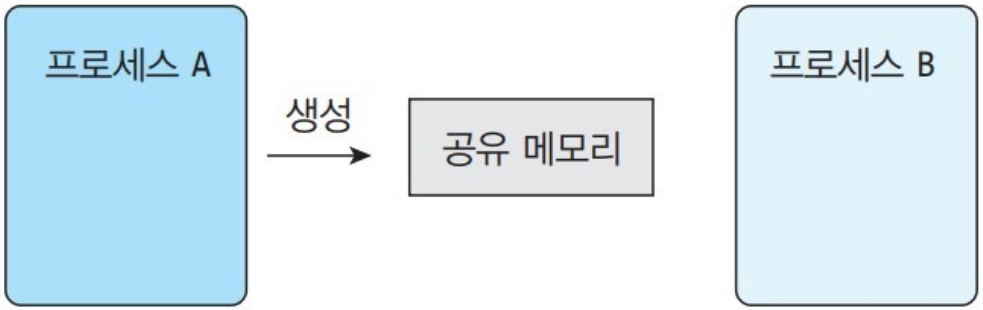


그림 10.12 공유 메모리 생성

공유 메모리 관련 함수 : shmat()

```
#include <sys/shm.h>
#include <sys/ipc.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- shmid 공유 메모리를 이 프로세스의 메모리 위치 shmaddr에 연결하고 그 주소 반환
- shmflg : 공유 메모리에 대한 읽기/쓰기 권한 지정

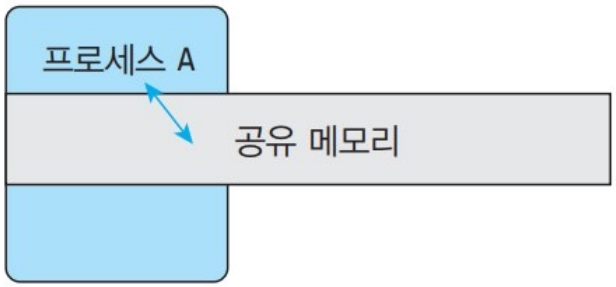


그림 10.13 공유 메모리에 연결

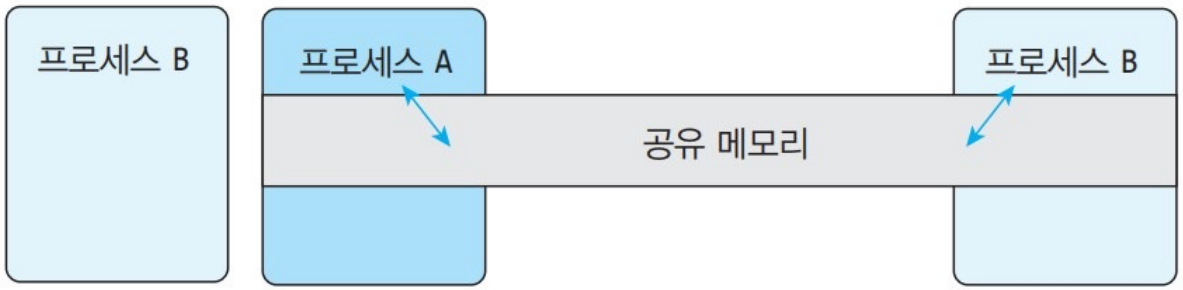


그림 10.14 다른 프로세스도 공유 메모리에 연결

공유 메모리 관련 함수

```
#include <sys/shm.h>
#include <sys/ipc.h>
int shmdt(const void *shmaddr);
```

- 공유 메모리에 대한 연결 주소 shmaddr를 연결 해제
- 성공 시 0, 실패 시 –1을 반환

```
#include <sys/shm.h>
#include <sys/ipc.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

- shmid 공유메모리를 cmd 명령어에 따라 제어
- cmd

 - IPC_RMID : 공유메모리 제거
 - IPC_SET : 공유메모리 정보를 buf에서 지정한 값으로 변환
 - IPC_STAT : 현재 공유메모리의 정보를 buf에 저장
 - SHM_LOCK/SHM_UNLOCK : 공유메모리 잠금/해제

공유 메모리 관련 함수

```
struct shmid_ds {
  struct    ipc_perm shm_perm;  /* 접근권한 */
  int  shm_segsz;            /* 세그먼트의 크기(bytes) */
  time_t    shm_atime;       /* 마지막 접근 시간 */
  time_t    shm_dtime;       /* 마지막 제거 시간 */
  time_t    shm_ctime;       /* 마지막 변경 시간 */
  unsigned short shm_cpid;     /* 생성자의 프로세스의 프로세스 id */
  unsigned short shm_lpid;     /* 마지막으로 작동한 프로세스의 프로세스 pid */
  short     shm_nattch;       /* 현재 접근한 프로세스의 수 */
  /* 다음은 개별적이다 */
  unsigned short   shm_npages;  /* 세그먼트의 크기(pages) */
  unsigned long   *shm_pages;
  struct shm_desc *attaches;    /* 접근을 위한 기술자들 */
  };
```

공유 메모리: shm1.c

```
1.  #include <sys/ipc.h>
2.  #include <sys/shm.h>
3.  #include <sys/types.h>
4.  #include <stdlib.h>
5.  #include <stdio.h>
6.  #include <string.h>
7.  int main()
8.  {
9.      int shmid;
10.     key_t key;
11.     char *shmaddr;
12.     key = ftok("helloshm", 1);
13.     shmid = shmget(key, 1024, IPC_CREAT|0644);
14.     if(shmid == -1) {
15.         perror("shmget");
16.         exit(1);
17.     }
18.     printf("shmid : %d", shmid);
19.     shmaddr = (char *) shmat(shmid, NULL, 0);
20.     strcpy(shmaddr, "hello shared memory");
21.     return(0);
22. }
```

```
$shm1
shmid : 17
$ ipcs -m
------ Shared Memory Segments --------
key            shmid  owner perms bytes nattch status
0x00000000  4      lect 600    16384 1       dest
0xffffffff     17     lect 644    1024  0
...
```

공유 메모리: shm2.c

```
1.  #include <sys/ipc.h>
2.  #include <sys/shm.h>
3.  #include <sys/types.h>
4.  #include <stdlib.h>
5.  #include <stdio.h>
6.  int main()
7.  {
8.      int shmid;
9.      key_t key;
10.     char *shmaddr;
11.     key = ftok("helloshm", 1);
12.     shmid = shmget(key, 1024, 0);
13.     if(shmid == -1) {
14.        perror("shmget");
15.        exit(1);
16.     }
17.     printf("shmid : %d\n", shmid);
18.     shmaddr = (char *) shmat(shmid, NULL, 0);
19.     printf("%s\n", shmaddr);
20.     return(0);
21. }
```

실행 결과

```
shmid : 17
hello shared memory
```

부모-자식 프로세스 사이의 메모리 공유: shm3.c

```
1.  #include <sys/ipc.h>
2.  #include <sys/shm.h>
3.  #include <sys/types.h>
4.  #include <sys/wait.h>
5.  #include <unistd.h>
6.  #include <stdlib.h>
7.  #include <stdio.h>
8.  int main()
9.  {
10.     int shmid;
11.     char *shmptr1, *shmptr2;
12.      shmid = shmget(IPC_PRIVATE,
13.                          10*sizeof(char),
14.                           IPC_CREAT|0666);
15.     if (shmid == -1) {
16.        printf("shmget failed\n");
17.        exit(0);
18.     }
```

```
19.     if (fork() == 0) {
20.        shmptr1=(char *) shmat(shmid, NULL, 0);
21.        for (int i=0; i<10; i++)
22.           shmptr1[i] = i*10;
23.        shmdt(shmptr1);
24.        exit(0);
25.     } else {
26.        wait(NULL);
27.        shmptr2=(char *) shmat(shmid, NULL, 0);
28.        for (int i=0; i<10; i++)
29.          printf("%d ", shmptr2[i]);
30.        shmdt(shmptr2);
31.        if (shmctl(shmid,IPC_RMID,NULL)==-1)
32.           printf("shmctl failed\n");
33.     }
34.     return 0;
35. }
```

실행 결과

```
0 10 20 30 40 50 60 70 80 90
```

10.5 메모리 관리 함수

메모리 관리 함수

- void *memset(void *s, int c, size_t n);
  - s에서 시작하여 n 바이트만큼 문자 c로 설정한 다음에 s를 반환
- int memcmp(const void *s1, const void *s2, size_t n);
  - s1과 s2에서 첫 n 바이트를 비교
  - 메모리 블록 내용이 동일하면 0을 반환
  - s1이 s2보다 작으면 음수를 반환
  - s1이 s2보다 크다면 양수를 반환
- void *memchr(const void *s, int c, size_t n);
  - s가 가리키는 메모리의 n 바이트 범위에서 문자 c를 탐색
  - c와 일치하는 첫 바이트에 대한 포인터를 반환
  - c를 찾지 못하면 NULL을 반환

메모리 관리 함수

- void *memmove(void *dst, const void *src, size_t n);
  - src에서 dst로 n 바이트를 복사하고, dst를 반환

- void *memcpy(void *dst, const void *src, size_t n);
  - src에서 dst로 n 바이트를 복사한다. 두 메모리 영역은 겹쳐지지 않음
  - 만일 메모리 영역을 겹쳐서 쓰길 원한다면 memmove() 함수를 사용
  - dst를 반환

mem.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <string.h>

4.  void main()
5.  {
6.      char str[32]="Do you like Linux?";
7.      char *ptr,*p;

8.      ptr = (char *) malloc(32);
9.      memcpy(ptr, str, strlen(str));
10.     puts(ptr);
11.     memset(ptr+12,'l',1);
12.     puts(ptr);

13.     p = (char *) memchr(ptr,'l',18);
14.     puts(p);
15.     memmove(str+12,str+7,10);
16.     puts(str);
17. }
```

```
$ mem
Do you like Linux?
Do you like linux?
like linux?
Do you like like Linux
```

메모리 맵핑

- 메모리 맵핑
  - 파일의 일부 영역에 메모리 주소를 부여
  - 마치 변수를 사용하는 것처럼 파일을 사용

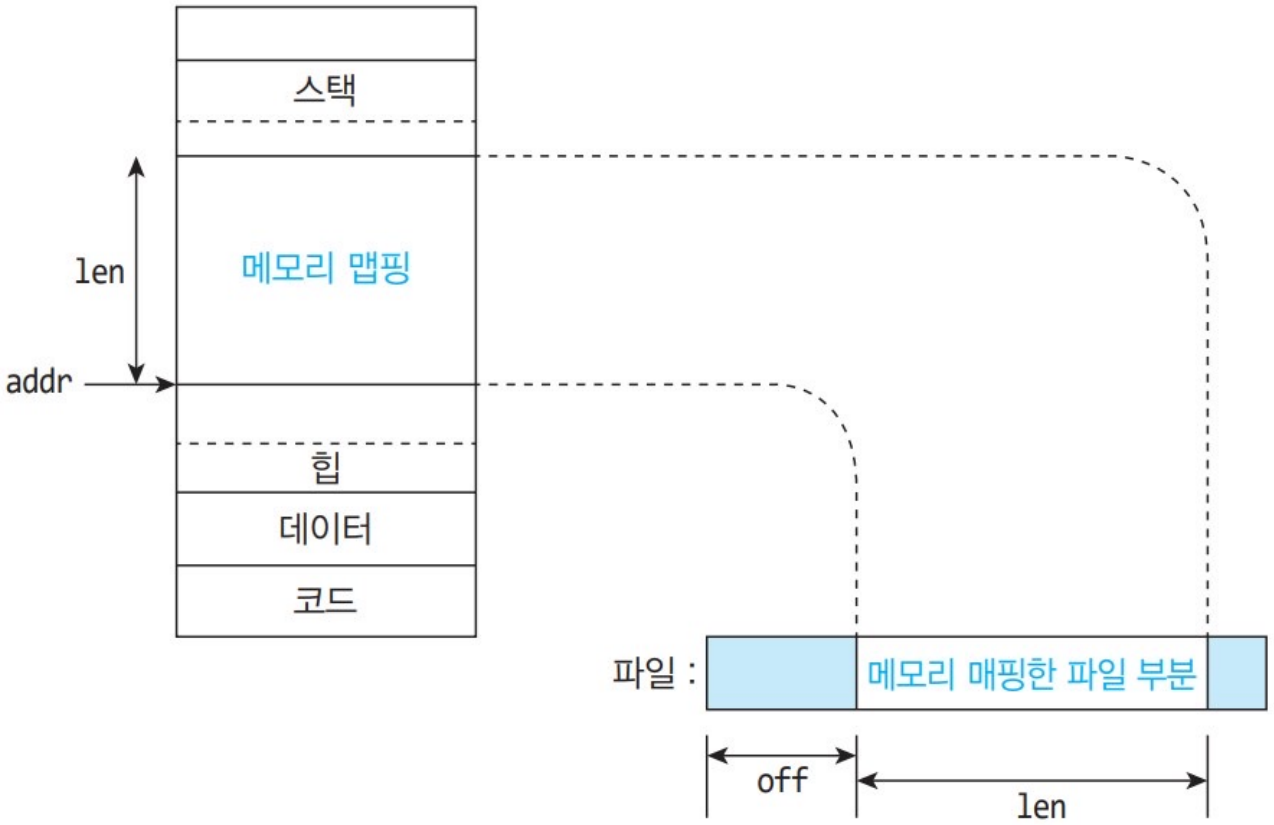


그림 10.15 메모리 맵핑

메모리 매핑 시스템 호출

```
#include <sys/types.h>
#include <sys/mman.h>
caddr_t mmap(caddr_t addr, size_t len, int prot, int flag, int fd, off_t off);
```

- fd가 나타내는 파일의 일부 영역(off부터 len 크기)에 메모리 주소를 부여하고 메모리 맵핑된 영역의 시작 주소(addr)를 반환

- addr: 메모리 맵핑에 부여할 메모리 시작 주소, 이 값이 NULL이면 시스템이 적당한 시작 주소를 선택
- len: 매핑할 파일 영역의 크기로 메모리 맵핑의 크기
- prot: 매핑된 메모리 영역에 대한 보호 정책
  - PROT_READ(읽기), PROT_WRITE(쓰기), PROT_EXEC(실행), PROT_NONE(접근 불가)
- flag :
  - MAP_SHARED: 매핑된 메모리를 여러 프로세스 간에 공유 하게 함
  - MAP_PRIVATE: 매핑된 메모리를 개인적으로 사용하기 위해 복제. 매핑된 메모리의 변경 사항은 현재 프로세스에만 영향을 미침
  - MAP_FIXED: 메모리 매핑을 원하는 특정 주소로 강제로 매핑. 일반적으로는 이 플래그를 사용하지 않고 0을 전달하여 시스템에게 매핑할 주소를 선택
  - MAP_ANONYMOUS: 실제 파일 대신에 익명의 메모리 매핑을 생성, 이 플래그는 파일이 아닌 메모리에 직접 매핑할 때 유용
  - MAP_FIXED_NOREPLACE: MAP_FIXED와 함께 사용되며, 주어진 주소에 이미 매핑이 존재하는 경우에도 새로운 매핑을 생성하지 않고 실패
- fd: 대상 파일의 파일 디스크립터
- off: 맵핑할 파일 영역의 시작 위치

메모리 매핑을 사용한 cat 명령어 구현:mmap.c

```
1.  #include <stdio.h>
2.  #include <sys/types.h>
3.  #include <sys/stat.h>
4.  #include <fcntl.h>
5.  #include <unistd.h>
6.  #include <stdlib.h>
7.  #include <sys/mman.h>
8.
9.  int main(int argc, char *argv[])
10. {
11.     struct stat sbuf;
12.     char *p;
13.     int fd;

14.     if (argc < 2) {
15.         fprintf(stderr, "사용법: %s 파일이름\n",
16.                 argv[0]);
17.         exit(1);
18.     }

19.     fd = open(argv[1], O_RDONLY);
20.     if (fd == -1) {
21.         perror("open");
22.         exit(1);
23.     }
```

```
24.     if (fstat(fd, &sbuf) == -1) {
25.         perror("fstat");
26.         exit(1);
27.     }

28.     p = mmap(0, sbuf.st_size, PROT_READ,
29.                     MAP_SHARED, fd, 0);
30.     if (p == MAP_FAILED) {
31.         perror("mmap");
32.         exit(1);
33.     }

34.     for (long l = 0; l < sbuf.st_size; l++)
35.         putchar(p[l]);

36.     close(fd);
37.     munmap(p, sbuf.st_size);
38.     return 0;
39. }
```