실습 10주차

CPS LAB

메모리 매핑

목차

프로세스간 통신

- 실제 사용하는 응용 프로그램은 보통 한 개 이상의 프로세스 사이에서 서로 데이터를 주고 받는 경우가 일반적

- 프로세스간 데이터를 주고 받기 위해서는 통신 프로그래밍이 필수적
  - IPC (Inter Process Communication)

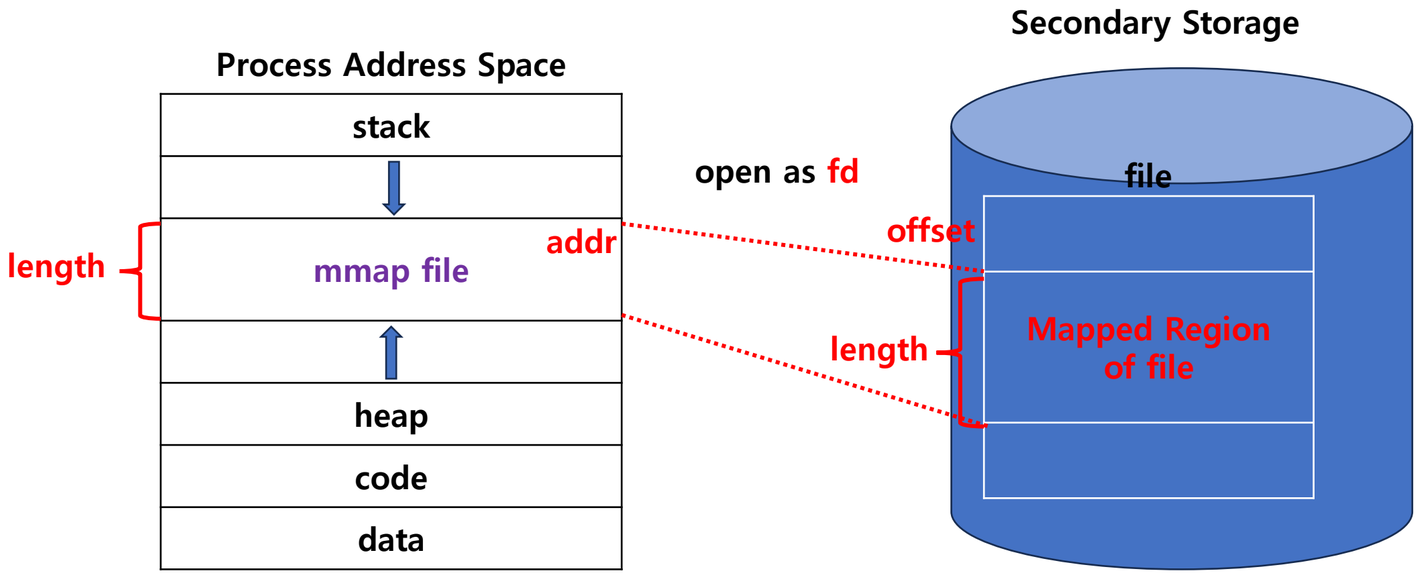
- 유닉스 시스템이 제공하는 IPC:
  - pipe, shared memory, message queue, signal 등

- 메모리 매핑을 이용한 통신 프로그래밍
  - mmap

mmap

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

Return Value: On success, mmap() returns a pointer to the mapped area.
On error, the value MAP_FAILED(that's (void *) -1) is returned



flags :

MAP_SHARED :
share this mapping.
The file may not actually be updated until msync() or munmap() is called.

MAP_PRIVATE :
Create a private copy-on-write mapping.
Updates to the mapping are not visible to other process mapping the same file

prot :

PROT_EXEC	Pages may be executed
PROT_READ	Pages may be read
PROT_WRITE	Pages may be written

예제풀이

예제 9-1

mmap()함수로 메모리 매핑하기

파일 전체 크기 알아내기

- lseek(fd, 0, SEEK_END);
- stat(argv[1], &statbuf);

질문: close하게 되면 매핑 내용이 없는 것인가?

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

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

        // mmap 받아온 포인터
        char *addr;
        // File Descriptor
        int fd;
        struct stat statbuf;

        if(argc != 2) {
                fprintf(stderr,
                        "Usage: %s filename\n", argv[0]);
                exit(1);
        }

        fd = open(argv[1], O_RDWR);
        if( fd < 0 ) {
                perror("open");
                exit(1);
        }

        // 매핑 내용의 크기를 알아내기
        /*
         * 두 번째, 방식
         * len = lseek(fd, 0, SEEK_END);
         *
         */
        // 첫 번째, 방식
        if( stat(argv[1], &statbuf) == -1) {
                perror("stat");
                exit(1);
        }

        // 함수 원형
        // void *mmap(void *addr, size_t length, int prot, int flags,
        //           int fd, off_t offset);
        addr = mmap(NULL, statbuf.st_size,
                        PROT_READ | PROT_WRITE,
                        MAP_SHARED,
                        fd,
                        (off_t)0);
        if( addr == MAP_FAILED ) {
                perror("mmap");
                exit(1);
        }

        close(fd);

        printf("%s", addr);

        return 0;
}
```

메모리 매핑 해제 :

```
#include <sys/mman.h>
```

```
int munmap (void *addr, size_t length);
```

- addr, 매핑된 메모리의 시작 주소
- length, 그 매핑된 영역의 크기

Return Value:

On success, munmap() returns 0.

On failure, it returns -1

```
/*
    The munmap() system call deletes the mappings for the specified address range,
    On the other hand, closing the file descriptor does not unmap the region.
*/
```

예제풀이

예제 9-2

munmap()함수로 메모리 매핑 해제하기

질문: 어디서부터 “segmentation fault” 나오는 것인가?

```
kyhooon@kyh:~/sysprogram_practice/p_ch9$ ./ch9_2.out hello
access already unmaped space
Segmentation fault (core dumped)
kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

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

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

        int fd;
        // 매핑되어던 주소
        char *addr;
        struct stat statbuf;

        if( argc != 2) {
                fprintf(stderr, "Usage: %s filename\n", argv[0]);
                exit(1);
        }

        // 인자 받는 첫번째 파일 정보 읽기
        if( stat(argv[1], &statbuf) == -1 ) {
                perror("stat");
                exit(1);
        }

        if( (fd = open(argv[1], O_RDWR)) == -1 ) {
                perror("open");
                exit(1);
        }

        addr = mmap(NULL, statbuf.st_size,
                        PROT_READ | PROT_WRITE,
                        MAP_SHARED,
                        fd,
                        (off_t) 0);
        if( addr == MAP_FAILED ) {
                perror("mmap");
                exit(1);
        }
        close(fd);

        if( munmap(addr, statbuf.st_size) == -1 ) {
                perror("munmap");
                exit(1);
        }
        printf("access already unmaped space \n");
        printf("%s\n", addr);

        return 0;
}
```

사용한 파일 크기 확장:

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

// 지정된 길이로 파일의 크기를 조정한다
// (파일의 크기를 늘리거나 줄일 때 사용함)
int truncate(const char *path, off_t length);
```

- path, 크기를 변경할 파일의 경로
- length, 변경하려는 크기

```
int ftruncate(int fd, off_t length);
```

- fd, 크기를 변경할 파일의 기술자

Return Value:

On success, returns 0.

On failure, it returns -1

예제풀이

질문: length가 크거나 작으면 ?

- hello -> h

```
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ echo "hello" > hello
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ gcc example.c -o example.out
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ cat hello
hello
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ ./example.out
파일이름 hello : 6 바이트(초기)
파일이름 hello : 56 바이트
파일이름 hello : 1 바이트
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ cat hello
hkyhooon@kyh:~/sysprogram_practice/p_ch9$
```

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

int main(void) {
    const char *filepath = "hello";
    struct stat statbuf;
    off_t length;

    if( stat(filepath, &statbuf) == -1 ) {
            perror("stat");
            exit(1);
    }
    length = statbuf.st_size;
    printf("파일이름 %s : %d 바이트(초기)\n",
                    filepath, (int)length);

    // 기준 파일 크기를 50바이트로 늘림
    if (truncate(filepath, length+50) == -1) {
        perror("truncate");
        return 1;
    }
    stat(filepath, &statbuf);
    printf("파일이름 %s : %d 바이트\n",
                    filepath, (int)statbuf.st_size);

    // 파일 크기를 1바이트로 정함
    if (truncate(filepath, 1) == -1) {
        perror("truncate");
        return 1;
    }
    stat(filepath, &statbuf);
    printf("파일이름 %s : %d 바이트\n",
                    filepath, (int)statbuf.st_size);

    return 0;
}
```

메모리 주소를 다시 매핑하기:

```
#define _GNU_SOURCE
#include <sys/types.h>
// 리눅스 특정 함수이므로, 다른 시스템이나
// 표준 C라이브러리에서 사용할 수 없음
void *mremap(void *old_address,
             size_t old_size,
             size_t new_size,
             int flags);
```

```c
/*
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
*/
// mmap으로 메모리 매핑 생성
void *addr = mmap(NULL,
                  old_size,
                  PROT_READ | PROT_WRITE,
                  MAP_ANONYMOUS | MAP_PRIVATE,
                  fd,
                  0);

// mremap으로 크기 조정
// 예) 동적 배열이 커져서 더 많은 공간이 필요할 때 유용하게 사용됨
void *new_addr = mremap(addr, old_size, new_size, MREMAP_MAYMOVE);

if( new_addr == MAP_FAILED ) {
        // 오류 처리
}
```

flags:

- **MREMAP_MAYMOVE**,
  기본적으로 현재 위치에 매핑을 확장할 공간이 충분하지 않으면 mremap()이 실패

- **MREMAP_FIXED**,
  매핑할 주소를 정확히 지정. 성공하면 해당 메모리 영역의 내용은 매핑된 내용으로 변경

Return Value:

- On success, returns a pointer to the new virtual memory area
- On failure, the value MAP_FAILED (that is, (void *) -1) is returned

매핑된 메모리 동기화:

```
#include <sys/mman.h>
int msync(void *addr,
          size_t length,
          int flags);
```

flags:

- **MS_ASYNC**,
  비동기 쓰기 작업, 적절한 타임에 동기화 작업수행
- **MS_SYNC**,
  동기화 완료할 때까지 리턴 한다
- **MS_INVALIDATE**,
  메모리에 있는 기존 내용을 무효화한다

Return Value:

- On success, zero is returned
- On failure, -1 is returned

```c
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {

    int fd = open("hello", O_RDWR);
    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    // 파일의 크기를 얻음
    off_t fileSize = lseek(fd, 0, SEEK_END);

    // 파일을 메모리에 매핑
    char *addr = mmap(NULL, fileSize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (addr == MAP_FAILED) {
        perror("Error mmaping the file");
        close(fd);
        return 1;
    }

    // 메모리에서 파일 내용을 변경
    addr[0] = 'J';  // 예를 들어, 첫 번째 문자를 변경

    // 변경 사항을 파일 시스템에 동기화
    if (msync(addr, fileSize, MS_SYNC) == -1) {
        perror("Could not sync the file to disk");
    }

    // 메모리 매핑 해제
    if (munmap(addr, fileSize) == -1) {
        perror("Error un-mmapping the file");
    }

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

```
kyhooon@kyh:~/sysprogram_practice/p_ch9$ echo "hello" > hello
kyhooon@kyh:~/sysprogram_practice/p_ch9$ vim posudo.c
kyhooon@kyh:~/sysprogram_practice/p_ch9$ gcc posudo.c -o posudo.out
kyhooon@kyh:~/sysprogram_practice/p_ch9$ ./posudo.out
kyhooon@kyh:~/sysprogram_practice/p_ch9$
kyhooon@kyh:~/sysprogram_practice/p_ch9$ cat hello
Jello
kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

"MS_INVALIDATE"는 다른 프로세스의 매핑이 최신 데이터를 반영하도록 보장함

매핑된 메모리 동기화:

```
#include <sys/mman.h>
int msync(void *addr,
          size_t length,
          int flags);
```

flags:

- **MS_ASYNC**,
  비동기 쓰기 작업, 적절한 타임에 동기화 작업수행
- **MS_SYNC**,
  동기화 완료할 때까지 리턴 한다
- **MS_INVALIDATE**,
  메모리에 있는 기존 내용을 무효화한다

Return Value:

- On success, zero is returned
- On failure, -1 is returned

```c
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {

    int fd = open("hello", O_RDWR);
    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    // 파일의 크기를 얻음
    off_t fileSize = lseek(fd, 0, SEEK_END);

    // 파일을 메모리에 매핑
    char *addr = mmap(NULL, fileSize, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (addr == MAP_FAILED) {
        perror("Error mmaping the file");
        close(fd);
        return 1;
    }

    // 메모리에서 파일 내용을 변경
    addr[0] = 'J';  // 예를 들어, 첫 번째 문자를 변경

    // 변경 사항을 파일 시스템에 동기화
    if (msync(addr, fileSize, MS_SYNC | MS_INVALIDATE) == -1) {
        perror("Could not sync the file to disk");
    }

    printf("MS_INVALIDATE 플래그 사용 후, 메모리 확인하기\n");
    printf("%s\n", addr);

    // 메모리 매핑 해제
    if (munmap(addr, fileSize) == -1) {
        perror("Error un-mmapping the file");
    }

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

```
kyhooon@kyh:~/sysprogram_practice/p_ch9$ vim posudo.c
kyhooon@kyh:~/sysprogram_practice/p_ch9$ gcc posudo.c -o posudo.out
kyhooon@kyh:~/sysprogram_practice/p_ch9$ echo "hello" > hello
kyhooon@kyh:~/sysprogram_practice/p_ch9$
kyhooon@kyh:~/sysprogram_practice/p_ch9$ ./posudo.out
MS_INVALIDATE 플래그 사용 후, 메모리 확인하기
Jello

kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

예제풀이

예제 9-5

데이터 교환하기

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

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

        int fd;
        pid_t pid;
        char *addr;
        struct stat statbuf;

        if( argc != 2 ) {
                fprintf(stderr, "Usages: %s filename\n", argv[0]);
                exit(1);
        }

        if( stat(argv[1], &statbuf) == -1 ) {
                perror("stat");
                exit(1);
        }

        if( (fd = open(argv[1], O_RDWR)) == -1 ) {
                perror("open");
                exit(1);
        }

        addr = mmap(NULL,
                    statbuf.st_size,
                    PROT_READ | PROT_WRITE,
                    MAP_SHARED,
                    fd,
                    (off_t) 0);
        if( addr == MAP_FAILED ) {
                perror("mmap");
                exit(1);
        }
        close(fd);
```

```c

        switch ( pid = fork() ) {

                case -1 :
                        perror("fork()");
                        exit(1);
                        break;

                case 0 :
                        // child process
                        printf("1. Child Process : addr = %s", addr);
                        sleep(1);
                        addr[0] = 'x';
                        printf("2. Child Process : addr = %s", addr);
                        sleep(2);
                        printf("3. Child Process : addr = %s", addr);
                        break;

                default:
                        // parent process
                        printf("1. Parent Process : addr = %s", addr);
                        sleep(2);
                        printf("2. Parent Process : addr = %s", addr);
                        addr[1] = 'y';
                        printf("3. Parent Process : addr = %s", addr);
                        break;
        }

        return 0;
}
```

```
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ echo "hello" > hello
[kyhooon@kyh:~/sysprogram_practice/p_ch9$
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ ./ch9_5.out hello
1. Parent Process : addr = hello
1. Child Process : addr = hello
2. Child Process : addr = xello
2. Parent Process : addr = xello
3. Parent Process : addr = xyllo
kyhooon@kyh:~/sysprogram_practice/p_ch9$ 3. Child Process : addr = xyllo

kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

연습 문제

실습1

작은 프로젝트:

메모리 매핑을 활용하는 프로젝트

1. 메모리 매핑을 사용하여 대용량 파일의 내용(전부 정수)
   정렬 알고리즘을 사용하여 정렬함

2. 정렬된 내용을 디스크 동기화함

실습1

작은 프로젝트:

해답:

1. 데이터 생성
   - shuf 명령어는 무슨 뜻인지 man로 찾도록 하자
   - 명령어 생성된 데이터의 앞 20개를 확인해보자

```
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ shuf -i 1-500 -n 200 > number
[kyhooon@kyh:~/sysprogram_practice/p_ch9$
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ cat -n number | head -n 20
     1  141
     2  231
     3  4
     4  216
     5  419
     6  80
     7  237
     8  83
     9  166
    10  369
    11  15
    12  221
    13  476
    14  94
    15  324
    16  432
    17  188
    18  292
    19  382
    20  238
 kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

실습1

작은 프로젝트:

해답:

2. 정렬 알고리즘

- 어떤 알고리즘 더 효율적인가?
- open나 mmap차이가 무엇인가?

3. 부분 코드 공개

```
41 int main() {
42     int fd = open("number", O_RDWR);
43     struct stat fileInfo;
44     // 파일 상태 검색
45
46
47
48
49     int *data = mmap(                                                fd, 0);
50     if (data == MAP_FAILED) {
51         perror("Error mmapping the file");
52         exit(1);
53     }
54
55     // 정렬 알고리즘 구현
56
57
58     if (msync(                        MS_SYNC) == -1) {
59         perror("Could not sync the file to disk");
60     }
61
62     if (munmap(data, fileInfo.st_size) == -1) {
63         perror("Error un-mmapping the file");
64     }
65
66     close(fd);
67     return 0;
68 }
```

```
kyhooon@kyh:~/sysprogram_practice/p_ch9$ cat -n number | head -n 20
     1  2
     2  4
     3  6
     4  8
     5  17
     6  18
     7  19
     8  21
     9  22
    10  23
    11  24
    12  29
    13  30
    14  33
    15  34
    16  35
    17  36
    18  37
    19  38
    20  40
kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

실습2

파일명이 test.txt 인 빈 파일을 생성하고, 이를 메모리에 매핑하려고 한다.
파일의 경로를 인자로 받아서 크기를 확장한 후, ”system programming”이라는 내용을 채운다.
그 이후에, 동기화 시킨다.

```
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ ls -l | grep test.txt
[kyhooon@kyh:~/sysprogram_practice/p_ch9$
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ gcc ch9_12.c -o ch9_12.out
[kyhooon@kyh:~/sysprogram_practice/p_ch9$
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ ./ch9_12.out test.txt
[kyhooon@kyh:~/sysprogram_practice/p_ch9$
[kyhooon@kyh:~/sysprogram_practice/p_ch9$ cat test.txt
 Linux System Programming
 kyhooon@kyh:~/sysprogram_practice/p_ch9$
```

```c
#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

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

        int fd, length;
        char *addr;

        // 확장된 파일크기 설정
        length = 1024;

        if( argc != 2 ) {
                fprintf(stderr, "Usage : %s filename\n", argv[0]);
                exit(1);
        }

        // argv[1] 파일을 생성하고
        // Owner:Group:Other -> 0666

        // argv[1] 파일의 크기를 조정함

        // 파일을 매핑한다(MAP_SHARED)

        close(fd);

        // 내용을 추가한다
        strcpy(addr, "Linux System Programming\n");

        // 동기화 시킴

        // 메모리 해체
        if( munmap(addr, length) == -1 ) {
                perror("munmap");
                exit(1);
        }

        return 0;
}
```

감사합니다.

CPS LAB