제목 : 운영체제 중간 과제

이름 : 이대호
학과: 항공소프트웨어공학과
학번: 202000969
교수명: 심종익 교수님
제출날짜: 20220518(수)

필수과제

1. 2주차 동영상강의에서 설명한 파일 복사 프로그램 2가지 작성

2. 교재 그림 3.8 프로세스 생성 프로그램 작성

3.4주차 동영상 강의 내용에 설명한 프로세스 생성 프로그램 작성

4. 교재 그림 3.16, 3.17 프로그램

1.2주차 동영상에서 설명한 파일복사 프로그램 2가지 작성

1 - 1(파일복사프로그램)

```c
//
//  main.c
//  필수과제1
//
//  Created by 이대호 on 2022/05/17.
//

#include<stdio.h>
#include<stdlib.h>
void main()
{
FILE *fin,*fout; //입력 및 출력파일의 주소를 받아주는 인스턴스
char c,infile[128],outfile[128];//파일에 글자를 받아 오는 변수 c, 입력 및 출력파일 이름을 받기 위한 배열
printf("입력 파일 이름: ");
gets(infile);//infile 배열에 입력파일 이름 저장.
printf("출력 파일 이름.  ");
gets(outfile);//outfile 배열에 출력파일 이름 저장.

fin=fopen(infile,"r");//fin변수에 fopen함수를 사용하여, infile배열에 있는 파일 읽기모드로 열기
if(fin==NULL)//만약 파일이 존재하지 않고, 값을 입력하지 않았다면
{
 printf("Error : 파일 %s을 열수 없습니다.\n",infile);//입력파일 이름과 함께 오류 메세지 출력
 exit(1);//강제종료
}

fout=fopen(outfile,"w");//위 제어문을 통과했다면 출력할 파일을 쓰기모드로 열기
if(fout==NULL)//만약 출력파일을 입력하지 않았다면
{
 printf("Error : 파일 %s을 열 수 없습니다.\n",outfile);//출력파일이름과 오류 메세지 출력
    exit(1);//강제종료
}
    while((c=fgetc(fin))!=EOF)//반복문을 사용하여 c라는 인스턴스에 입력파일의 끝까지 글자를 받아온다.
     fputc(c,fout);//출력파일에 입력파일 내용을 하나씩 넣어준다.
    fclose(fin);//반복문이 종료후, 입력파일을 닫아준다.
    fclose(fout);//모든과정이 끝난 후 출력파일을 닫아준다.
}
```

파일복사를 하는 프로그램이 2가지가 설명이 되었다. 위 파일은 프로그램 실행형식으로 돌아가는 파일이다. 먼저 FILE *fiin, *fout을 사용하여 입력 및 출력파일의 주소를 저장해준다. char 형으로 선언해준 c와, infile, oufile은 순서대로 파일에 글자를 받아 오는 인스턴스와 입력 및 출력파일의 이름을 받기 위한 문자열 배열을 선언해주었다. 최대 글자수는 128로 배열의 크기를 지정해주었다.

먼저 입력파일의 이름을 받기 위해 출력문을 사용하여 입력파일의 명을 입력하고 사용자에게 제시를 한다. 다음으로 표준입력에서 문자열을 받아 사용자가 전달한 메모리에 저장하는 함수인 gets함수를 이용하여, infile이라는 배열에 저장을 해준다. 같은 방법으로 출력파일의 이름과 outfile변수에 적어준다.

그 이후 fin변수에 파일을 여는 fopen함수를 사용하여 infile배열에 있는 파일을 읽기모드로 열기를 시도한다. 제어문을 사용하여 fin에 입력된 파일이 존재하지 않는다면 에러 메세지를 출력하며 exit(1)함수를 사용하여 강제종료를 해준다. 만약 파일이 존재한다면 출력파일을 담을 fout변수에 fopen함수를 사용하여 outfile를 읽기모드로 열어준다. 만약 fout 즉, 출력파일 명을 입력하지 않았을 때 파일을 열 수 없습니다. 라는 오류메세지를 출력하며 프로그램을 종료한다.

위 제어문을 모두 통과를 하였다면 while문을 사용하여 매개변수 c에 fgetc함수안에 읽기모드로 열린 입력파일의 내용을 파일의 내용이 끝날 때 까지 실행하도록 조건을 걸어주고 fputc함수를 사용하여 입력파일의 내용을 입력모드로 열린 출력파일에 하나하나 작성을 해준다.

위 조건문이 끝난 후, fin, fout파일을 fclose함수를 활용하여 파일을 닫아 준 후 프로그램을 종료한다.

위 코드를 실행한 결과는 다음과 같다.

- 프로그램 1-1 실행결과(입력파일이 존재하지 않을 때)

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./File1
입 력  파 일  이 름  : sample.txt
출 력  파 일  이 름  : Error : 파 일  을  열 수  없 습 니 다 .
```

- 프로그램 1-1 실행결과(입력파일이 존재 할 경우)

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./File1
입 력  파 일  이 름 : sample.txt
출 력  파 일  이 름 .  sample.bak
Error : 파 일  sample.txt을  열 수  없 습 니 다 .
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ vi sample.txt
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./File1
입 력  파 일  이 름 : sample.txt
출 력  파 일  이 름 .  sample.bak
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ls
File1  File1.c  sample.bak  sample.txt
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$
```

위 결과를 보았을 때 정상적으로 프로그램이 돌아갔다고 할 수 있다.

1 -2 (파일복사프로그램 명령어 형식)

1 - 2 코드

```c
//
//  1-2.c
//  필수과제1
//
//  Created by 이대호 on 2022/05/17.
//
#include<stdio.h>
#include<stdlib.h>

void main(int argc, char *argv[])
//int argc는 파일의 갯수를 받는다. 이 코드에서는 명령어처럼 사용할 파일의 이름, 입력,출력 파일 총 3개의 인자를 받는다.
//받은 이후 *argv배열 크기를 지정해준다.
{
 char c;//파일의 내용을 받아주는 변수
 FILE *inptr, *outptr;//입출력파일의 주소를 받아주는 파일형 변수.

 if((inptr=fopen(argv[1],"r"))==NULL)//만약 argv[1]에 저장되어있는 입력파일이 없다는 제어문
 {
  printf("ERROR:can't open file %s\n",argv[1]);//입력파일을 열 수 없다고 출력
  exit(1);//강제종료
 }
 if((outptr=fopen(argv[2],"w"))==NULL)//argv[2]에 저장된 출력파일 이름이 NULL값이라면
 {
  printf("ERROR:can't not open file %s\n",argv[2]);//파일을 열 수 없다고 출력
  exit(1);//강제종료
 }
 while((c=getc(inptr))!=EOF)//반복문을 사용하여 파일안에 값을 하나씩 받는다 입력파일의 값의 끝까지 실행
  putc(c,outptr);//입력파일에 있는 내용 하나하나 반복문이 끝날 때 까지 출력파일에 입력을 해준다.
 printf("File copied.\n");//파일이 복사가 끝났다는 메세지 출력
 fclose(inptr);//입력파일을 닫아준다.
 fclose(outptr);//출력파일을 닫아준다.
}
```

1-1번 코드와 같이 파일을 복사하는 프로그램이지만 1-2번 프로그램은 cp명령어와 같이 사용되도록 만든 프로그램이다. 먼저 main함수에 매개변수가 int argc와 char *argv[]가 있다. 명령어를 ./cp sample.txt sample.bak 위와 같은식으로 작성하게 되는데, 이 때 int argc에서 들어온 인자들의 개수를 받아 char *argv배열에 인자들의 갯수만큼 배열의 크기를 지정해준다. 이 부분이 1 - 1번과의 차이점이고 명령어처럼 사용할 수 있도록 하는 모습이다.

char c는 1- 1번과 같은 용도로 사용되는데 파일의 내용을 받아주는 변수이다. 똑같이 FILE변수를 선언하여 입출력 파일의 주소값을 저장해준다. 제어문을 통해 argv[1]즉, 입력파일(명령어 입력파일, 출력파일순으로 입력했을 때)과 읽기모드로 열었을 때 입력파일이 존재하지 않는다면 오류메세지와 함께 강제종료를 한다. 다음 제어문은 출력파일에 대한 제어문이다. fopen함수를 이용하여 읽기모드로 열린 argv[2]에 저장된 출력파일의 이름이 없을 때 오류메시지와 함께 프로그램을 종료한다.

위 제어문들을 모두 통과하면 입력파일이 존재하고 출력파일의 이름을 모두 적어준 경우이다. 위 경우를 통과했다면 입력파일은 읽기모드로 출력파일은 쓰기모드로 파일이 열린 상태이다. 1-1번과 동일하게 while문을 사용하여 c라는 변수에 입력파일의 내용을 받아오는데 파일의 끝까지 받아오는 조건을 달아 끝까지 while문을 실행해준다. while문은 입력파일에 입력을 해준다. 위 과정이 종료가 되었다면 파일이 복사되었다는 출력문과 함께 입력 및 출력파일을 닫아준다.

위 코드를 실행한 결과는 다음과 같다.

- 프로그램 1-2 실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./File2 sample.txt result.
txt
File copied.
```

프로그램 1-1을 돌리기 위해 미리 만들어 놓은 sample.txt를 사용하였고 존재하는 파일이기 때문에 result.txt라는 출력파일을 만들었다. 위 프로그램은 명령어 형식처럼 사용이 되었고 순서대로 프로그램 명 입력파일 출력파일 순이다.

프로그램 2

프로세스 생성 프로그램

코드

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

int main()
{
pid_t pid;//프로세스를 저장하는 타입인 pid_t형 pid를 선언
        /* fork a child process*/
        pid = fork();//자식 프로세스를 만드는 fork함수를 선언

        if(pid<0){/* error occurred */
         //pid
         fprintf(stderr,"Fork Failed\n");
         return 1;
        }
        else if(pid==0){/* pid값이 0이면 child process */
          execlp("/bin/ls","ls", NULL);
            //execlp함수는 경로에 등록된 디렉토리를 참고하여 다른 프로그램을 실행하고 종료한다.
            //자식프로세스가 생성되면 현재디렉토리에 있는 디렉토리를 출력한다.
        }
        else{/* pid값이 0보다 크면 parent process */
             wait(NULL);/*parent will wait for the child to complete */
             printf("Child Complete\n");//자식 프로세서가 종료되면 완료되었다고 출력한다.
        }

        return 0;

}
```

위 코드는 fork함수를 사용하여 프로세스가 생성되는 지 확인하는 프로그램이다. 프로세스를 저장하는 타입인 pid_t 자료형 pid를 선언해준다. 그 이후 pid는 fork()함수를 사용하여 자식 프로세스를 만든 함수를 선언해준다.

pid에 따라 프로세스 생성실패, 자식프로세스, 부모프로세스 종류가 달라지는데, 제어문을 통하여 구별해준다. 먼저 pid가 작을 땐 프로세스가 생성되지 않은 오류가 발생한 것이다, fprintf문을 사용하여 표준에러와 함께 실패를 했다는 출력문을 출력하고 종료를 해준다. 그 다음으로 pid가 0일 때, 자식프로세스이다. 이때 execlp함수를 사용하였다. 위 함수는 경로에 등록된 디렉토리를 참고하여 다른 프로그램을 실행하고 종료하는 함수이다. 그러므로 자식프로세스는 현재 디렉토리에 있는 파일들을 출력한다. pid가 0보다 클 때에는 부모프로세스이다. 부모프로세스는 자식프로세스가 종료될 때 까지 기다려 줘야하는데 wait(NULL)명령어는 자식프로세스를 기다려주는 명령을 사용해준다. 그 이후 자식프로세스가 종료가 되면 출력문을 통해 자식프로세스가 완료되었다고 출력해준다.

- 코드 실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ gcc MadeFork.c -o MadeFork
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./MadeFork
CP                File2       NPMO              PIDEX       outt1.txt
CS                File2.c     NPipeLine.c       Producer.c  outt2.txt
ChoiceCopy1       Fork.c      NPtest            Reader      pidEX.c
ChoicecCopy2      Hcopy1.c    NamedPipe.c       Reader.c    result.txt
Choicecopy2.c     MadeFork    NamedPipe1.c      Writer      sample.bak
Consumer.c        MadeFork.c  NamedPipeLineEx   Writer.c    sample.txt
CreateProcess.c   NP          NamedPipeWr       a.out       t1.bak
Fg3_1.c           NP1         NamedWriter.c     fork4       t1.txt
File1             NPE.c       OP                fork_ex     t2.txt
File1.c           NPMK        OrdinaryPipe.c    outt1.c
Child Complete
```

위 코드를 실행한 결과이다 위 ls 명령어를 실행해준 이유는 execlp함수의 특성과 매개변수로 "ls" 명령어를 넣어주었기 때문이다. 그 이후 마지막에 Child Complete라는 출력문을 확인 할 수있다. 반복문도 아닌데 어떻게 제어문을 통과하나 라는 의문을 가질 수 있지만 부모프로세스와 자식프로세는 다른 메모리에서 독립적으로 실행이 되기 때문에 반복문을 사용한 것처럼 보인 것이다. 부모프로세스는 자식프로세스보다 먼저 실행을 한 후 끝나게 되면 안되기 때문에 wait(NULL)함수에 의해 자식프로세스가 끝난 후 부모프로세스가 실행된 것을 알 수 있다.

프로그램 3

4주차 프로세스 생성프로그램

코드

```c
//
//  main.c
//  필수과제3
//
//  Created by 이대호 on 2022/05/17.
//
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
        pid_t pid;//프로세스 번호를 저장하는 타입의 pid_t형 인스턴스 pid를 선언
        pid = fork();//프로세스를 생성하는 fork함수
        if(pid == -1) {//프로세스가 생성되지 않으면 -1을 반환
                printf("Error: cannot fork\n");//프로세스를 생성할 수 없다는 메세지 출력
                exit(1);//강제종료
        }

        if(pid == 0) {//pid가 0이면 자식프로세서
                int j;
                for(j = 1; j <= 10; j++) {//1번부터 10번까지 자식프로세서 출력을 위한 반복문
                        printf("\nChild : pid = %d %d\n", getpid(), j);//getpid는 실행중인 프로세스 ID를 구하는 함수
                        sleep(1);//부모프로세서와 차이를 보여주기 위해 1초 딜레이
                }
        }
        else {//양수라면 부모프로세서
                int i;
            for(i = 1; i <= 10; i++) {//1부터 10번까지 부모프로세서 출력
                printf("\nParent : pid = %d %d\n", getpid(), i);////getpid는 실행중인 프로세스 ID를 구하는 함수
                sleep(2);//자식 프로세스와 차이를 보여주기 위해 2초 딜레이를 준다.
            }
            }
    return 0;//정상적인 종료를 위해 return 0을 해준다.
}
```

위 프로그램은 위에서 실행한 프로그램을 가시화 하기 위해서 반복문과 delay를 주어 부모 프로세서와 다르다는 것을 보여주는 프로그램이다. 먼저 pid_t자료형 변수 pid를 선언해준다. 다음로 pid에 fork함수를 사용하여 프로세스를 생성한다. 위와 같은 이유로 제어문을 통해 생성되지 않음, 자식 프로세스, 부모 프로세스를 구분해준다. 만약 생성되지 않았을 경우에는 생성이 되지 않았다는 경고문과 함께 프로그램을 종료해준다. 다음으로 자식프로세스의 경우에는 반복문을 통해 10개의 자식프로세스를 출력해주며 getpid를 사용하여 프로세스의 id를 식별해준다. 그 다음 자식프로세스의 번호를 j의 값만큼 붙혀주며 출력되는 것을 보기 위해 delay(1)을 통해 1초 간격으로 출력을 한다. 부모 프로세스 또한 자식프로세스와 동일한 방법을 이용한다. 다만 delay(2)초를 주어 독립적으로 실행되는 프로그램임을 보여준다. 마지막으로 정상적인 종료를 위하여 return 0을 선언해준다.

코드실행결과

```
[dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ vi CreateProcess.c
[dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ls
CS  Consumer.c  CreateProcess.c  File1  File1.c  File2  File2.c  Fork.c  PIDEX  Prod
[dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ vi CreateProcess.c
[dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ gcc CreateProcess.c -o CP
[dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./CP

Parent : pid = 19361 1

Child : pid = 19362 1

Child : pid = 19362 2

Parent : pid = 19361 2
```

프로그램 4

교재 그림 3.16 3.17

Producer & Consumer

코드

(Producer)

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

int main()
{
/* the size (in bytes) of shared memory object*/
    const int SIZE=4096;//공유 메모리의 크기
    /* name of the shared memory object */
const char *name="OS";//공유메모리 객체의 이름을 OS로 선언한다.
/* string written to shared memory */
const char *message_0 ="Hello";//공유메모리 0에 "Hello"를 넣어준다.
const char *message_1="World";//공유메모리 1에 "World"를 넣어준다.

/* Shared memory file descriptor */
int fd;//읽기 신호인지 쓰기신호인지 보내주는 신호연산자
/* pointer to shared memory object */
    char *ptr;//공유메모리 객체의 주소를 저장해주는 포인터를 선언한다.(공유메모리에 들어간 값들이 char형이기에 char형으로 선언)

            /* create the shared memory object */
            fd = shm_open(name,O_CREAT| O_RDWR,0666);//읽기 쓰기 신호에 공유메모리를 열겠다는 신호를 보내준다
                //shm_open함수의 인자로 공유메모리 객체, 객체 생성 및 객체 읽기쓰기,리눅스에서 권한을 지정하는 번호

            /* configure the size of the shared memory object */
            ftruncate(fd,SIZE);//파일 디스크립터로 파일 크기를 변경해는 함수다 매개인자로 파일 디스크립터와 공유메모리의 크기를 적어준다.

            /* memory map the shared memory object */
            ptr = (char  *) ///shared 메모리 위치
              mmap(0,SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,0);//mmap 0 ~ 공유메모리 사이즈만큼 ptr주소를 맞추어 준다.
                  READ, write함수를 쓰지 않고 공유메모리로 사용할 수 있게 한다. 처음의 0은 시작주소이다.

            /* write to the shared memory object */
            sprintf(ptr,"%s",message_0);
            /* sprintf는 출력하는 결과 값을 변수에 저장해주는는 기능이 있다. 그러므로 ptr 공유 메모리에 저장되어있는 message_0를 출력한다. */
            ptr+=strlen(message_0);
             /* message_0의 길이만큼 메모리 공간을 할당해주기 위해 strlen함수를 이용해준다. */
            sprintf(ptr,"%s",message_1);//위 message_0과 같다.
            ptr+=strlen(message_1);

            return 0;

    }
```

const int SIZE로 공유메모리의 값을 4096으로 지정한다.(중간에 변경 불가) 다음으로 const char을 통해 공유메모리의 이름을 OS로 명명해준다. 공유메모리에 message_0과 message_1을 넣어주어 각각 Hello 와 world를 넣어 출력을 할 때 Hello world가 나오도록 선언해준다. int fd를 선언하여 파일디스크립터를 선언하여 읽기 신호인지 쓰기 신호인지 구별을 해준다. 다음으로 ptr을 선언해주어 공유메모리 객체의 주소를 지정해주는 포인터를 선언한다. 이 때 공유메모리에 들어간 값들이. char형으로 선언 되었기 때문에 char형으로 선언해준다. 공유메모리를 열겠다는 신호로 shm_open함수를 사용해준다. 이 때 공유메모리에 객체, 객체 생성 및 객체 읽기쓰기, 리눅스의 권한을 제한하는 숫자를 매개인자로 받는다. 그 값들을 fd에 넣어준다. 다음으로 ftruncate함수를 사용하여 파일 디스크립터로 파일 크기를 변경해준다. 이 때 매개인자는 파일 디스크립터와 공유메모리의 크기를 적어준다. ptr은 (char *)를 해주어 공유 메모리의 위치를 넣어준다. mmap함수를 사용하여 공유메모리의 사이즈 만큼 ptr주소를 맞추어 넣어준다. 이 과정을 거친 후 출력하는 결과 값을 변수에 저장해주는 기능이 있는 sprintf문을 사용한다. 그러면 ptr 공유 메모리에 저장되어있는 message_0을 출력해준다. 그 다음으로 ptr==strlen(message_0)을 사용하여 message_0의 길이만큼 메모리 공간을 할당하여 공유메모리의 공간을 절약한다. 위와 같은 방법을 사용하여. message_1의 값을 출력해주고 모든 과정이 끝나면 return0을 통해 프로그램을 안전하게 종료한다.

코드(Consumer)

```c
//  Consumer.c
//  필수과제4
//
//  Created by 이대호 on 2022/05/17.
//
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>

int main()
{
        /* the size (in bytes) of shared memory object */
        const int SIZE = 4096;//공유 메모리의 크기 4096bytes로 설정해준다.
        /* name of the shared memory object */
        const char* name = "OS";//공유 메모리의 이름을 OS로 설정해준다.
        /* shared memory file descriptor */
        int fd;//공유메모리의 파일 디스크립터를 정수형 변수로 선언해준다 0은 표준입력 1은 표준 출력, 2는 표준에러이다
        //파일 디스크립터는 리눅스 혹은 유닉스 계열의 시스템에서 프로세스가 파일을 다룰 때 사용하는 개념으로 프로세스에서 특정 파일에 접근할 때 사용하는
            추상적인 값이다.
        /* pointer to shared memory obect */
        char* ptr;//공유메모리의 주소값을 저장해주는 변수

        /* open the shared memory object */
        fd = shm_open(name, O_RDONLY, 0666);//파일 디스크립터를 이용하여 읽기모드로 공유메모리를 열어준다.

        /* memory map the shared memory object */
    ptr = (char*)
            mmap(0, SIZE, PROT_READ, MAP_SHARED, fd, 0); //READ ONLY로 열 땐, PROT_READ인자만을 사용해야한다. Produce와 같은
                방식으로 사용하지만 읽기모드만 넣어준다.

    /* read from the shared memory object */
    printf("%s", (char*)ptr);//공유 메모리에 있는 값을 읽어온다.

    /* remove the shared memory object */
    shm_unlink(name);//공유메모리의 객체를 지워준다.

    return 0;
}
```

Consumer코드이다. 생산자와는 다른 코드 양상을 보인다. 먼저 공유 메모리의 크기 및 공유 메모리의 이름은 동일하게 선언해준다. 또한 파일디스크립터도 선언해준다. 여기서 다른 점은 fd=shm_open(name, O_RDONLY,0666)과 mmap(0, SIZE, PROT_READ,MAP_SHARED,fd,0)부분이다. 교재에서는 PROT_WRITE코드도 있었지만 위 코드 메뉴얼을 보면 O_RDONLY로 열어주면 PROT_READ만 매개변수로 있어야 프로그램이 실행이된다고 한다. 그러므로 필요없는 매개인자를 지워주고 위와 같은 코드로 수정을 해 주었다. fd는 파일 디스크립터를 이용하여 읽기모드로 공유메모리를 열어주고 Producer와 다른점은 읽기모드만 넣어준다. 그 다음에는 printf문을 사용하여 (char*)ptr로 공유메모리에 있는 값을 읽어온다. 이 값을 불러온 이후 shm_unlink(name)을 통해 공유메모리의 객체를 지워준다.

실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./CS ./PR
HellowWorld!dase202000969@dblab-HP-Z440-Workstation:~/middleExam$
```

위 명령어를 CS PR 즉 소비자부터 실행하고 생산자를 그 후에 선언을 하였는데 프로그램의 경우 뒤에서 부터 코드를 읽기 때문에 실제로는 생산자부터 실행이 되고 그 이후에 소비자가 실행이 되는 구조이다. 위 코드가 성공적으로 들어가 message_0,1에 들어있는 값인 HelloWorld! 가 정상적으로 실행된 것을 볼 수 있다.

선택과제

1. 파일 복사프로그램 read(), fread(), write(), fwrite()사용

2. 교재 그림 3.21 ~ 3.22 일반 파이프 프로그램

3. 네임드 파이프 프로그램

4. 교재 4.11 다중 스레드 프로그램

1.파일복사프로그램 함수 교체

코드 1-1

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

void main()                                          Return type of 'main' is not 'int'
{
    FILE *fin,*fout;
    char Infile[128],Outfile[128];
    char bf[5]={0,}; // int 4바이트 + 1 NULL바이트
    int TempFile;//fread를 출력한 내용의 크기가 들어가기 때문에 정수형 변수로 선언,fread함수만 바로 사용하면 쓰레기
        값이 들어가기 때문에 TempFile변수에 넣어줘 쓰레기값이 아닌 파일의 본 내용이 복사본에 저장되도록 해주었다.
    printf("입력 파일 이름 : ");
    scanf("%s",Infile);
    printf("출력 파일 이름 : ");
    scanf("%s",Outfile);

    fin=fopen(Infile,"r");
    if(fin==NULL)
    {
        printf("Error: 파일 %s을 열 수 없습니다\n",Infile);
        exit(1);
    }

    fout=fopen(Outfile,"w");
    if(fout==NULL)
    {
        printf("Error : 파일 %s을 열 수 없습니다.\n",Outfile);
        exit(1);
    }

    TempFile=fread(bf,sizeof(char),128,fin);//fread함수의 매개인자는 버퍼,해당 자료형의 크기만큼 불어오기
        위해 sizeof함수 사용, 128은 읽어올 크기이다. 그러므로 넉넉하게 주는 것이 좋다. 마지막 매개인자는 읽어올
        파일을 넣어준다. 위에서는 입력파일이 fin으로 선언되어있기 때문에 fin을 넣어주었다.
    fwrite(bf, sizeof(char), TempFile, fout);//출력파일은 입력파일의 내용을 복사해야한다. 위와 같이 처음엔 버퍼를
        적어주고, 다음으로 불러온 자료형의 크기를, TempFile은 128의 공간만큼 있지만 실제로 불러온 파일은 그보다 작은 값이
        될 수도 있다. 왜냐하면 EOD만큼 크기를 자르기 때문에 공간을 불러온 파일 크기만큼 하기 위해서 TempFile을
        넣어주었다.그리고 마지막 인자로 출력파일을 넣어준다.

    fclose(fin);//위 과정이 끝났기 때문에 입력 및 출력파일을 닫아준다.
    fclose(fout);
}
```

위 코드는 필수과제와 다른점은 함수를 달리 사용했다는 점이다. 그러므로 다른 점만 설명을 하겠다. 먼저 char bf[5]를 선언해주었다. 버퍼를 사용해서 읽어오기 위해서 선언을 하였고 크기를 5로 지정한 이유는 int형 은 4바이트이고 NULL값 1바이트를 더해 크기가 5인 배열을 선언하였다. 그 다음으로 int TempFile을 선언하였다. fread함수로 출력한 내용의 크기가 들어가기 때문에 정수형으로 선언을 하였고 fread함수만 바로 사용하면 쓰레기값 또한 같이 들어가기 때문에 위 와 같은 변수를 사용하였다. TempFile의 경우 fread함수를 사용하여 값을 불러왔으며 매개인자로는 버퍼, 해당 자료형의 크기만큼 불러오기 위해 sizeof()함수를 사용하였고, 파일을 넉넉히 불러오기 위해 위에 값과 같은 128로 읽어 올 크기를 정해주었다 상관이 없는 것이 파일의 끝까지 불러오기 때문에 128보다 작은 파일은 파일이 끝나는 시점까지만 불러온다. 마지막으로 불러올 파일은 입력파일이 들어간 포인터인 fin을 넣어주었다.

위와 같은 함수가 종료되었을 때 출력파일에 적어주기 위해 fwrite함수를 사용해주었다. 이 때 위와 같이 버퍼를 적어주고 불러올 자료형의 크기 및 TempFile에 들어간 만큼의 크기로 불러왔다. 마지막으로 출력할 파일의 주소가 저장된 fout을 입력해주었다. 반복문을 사용하지 않은 이유는 위 함수에서 입력된 크기만큼 불러오기 때문에 매우 큰 파일이 아니라면 주어진 크기만큼 불러온다. 그러므로 반복문을 사용하지 않았다.

파일을 다 불러왔으면 파일을 닫아주기 위해 fclose함수를 사용하여 입력파일 및 출력파일을 닫아 주었다.

실행결과(파일이 존재하지 않을 때)

```
[dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./ChoiceCopy1
[입 력  파 일  이 름  : hello.txt
[출 력  파 일  이 름  : bye.txt
 Error: 파 일  hello.txt을  열  수  없 습 니 다
```

실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./ChoiceCopy1
입 력  파 일  이 름  : t1.txt
출 력  파 일  이 름  : outt1.txt
```

1-2

명령어 형식으로 프로그램 실행

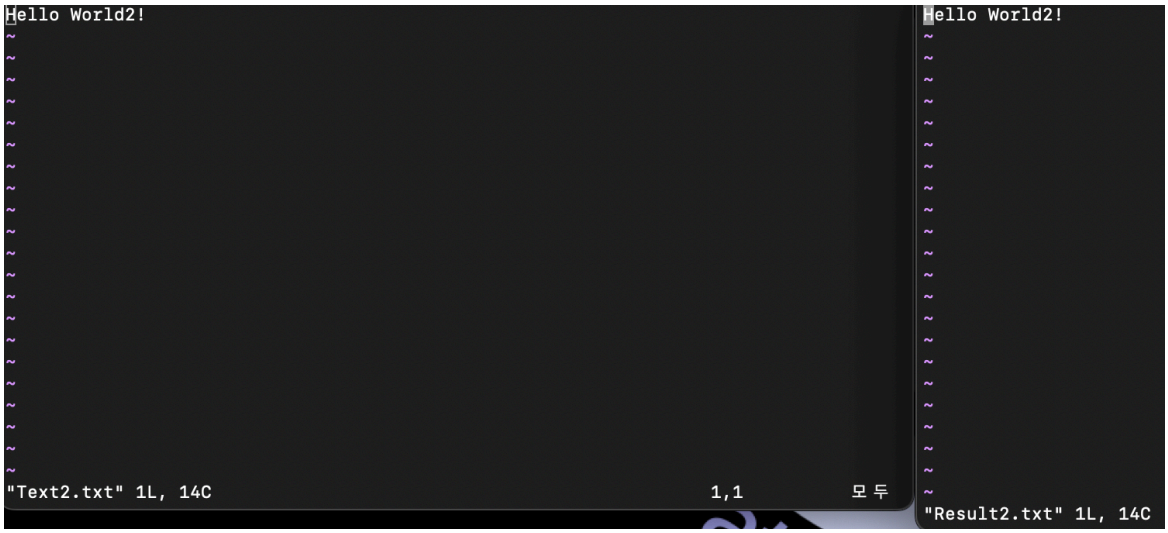
코드

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

void main(int argc,char *argv[])//필수과제와 같이 argc에서 매개인자의 갯수를 받고, 개수 만큼
    char *argv배열의 크기를 정해준다.
{
    FILE *inptr,*outptr;
    char Infile,Outfile,bf[5]={0,};                    2 ⚠ Unused variable 'Outfile'

    inptr=fopen(argv[1],"r");

    if(inptr==NULL){
        printf("ERROR : Can't open file %s\n",argv[1]);
        exit(1);
    }

    outptr=fopen(argv[2],"w");

    if(outptr==NULL){
        printf("ERROR : Can't open file %s\n",argv[2]);
        exit(1);
    }

    while((fread(bf,sizeof(char),1,inptr))!=EOF){//fread함수와 read함수는 용도는 똑같지만
        사용목적이 다르다, 내용을 하나씩 받아오기위해 while문을 사용하였다. 그 이유는 받아오는 크기를 1로
        지정하였기 때문에 파일에 내용이 없을 때 까지라는 조건을 두어서 하나씩 받아온다.
        fwrite(bf,sizeof(char),1,outptr);}//while문 조건에 맞게 fwrite함수를 사용하여
            출력파일에 값을 복사해준다.
    /*  */
    printf("File copied.\n");//출력파일이 완성되면 복사완료가 되었다는 문장을 출력해준다.
    fclose(inptr);//입력파일을 닫아준다.
    fclose(outptr);//출력파일을 닫아준다.

}
```

필수과제와 같은 부분은 생략하고 다른 부분부터 설명하겠다. while문의 조건식을 보면 fread함수를 사용하였다. 매개인자로는 버퍼, 자료형의 크기, 반복문을 사용하기 위해 하나씩 받아왔으며 읽는 파일은 입력파일이다. 실행은 파일의 끝이 아닐 때 까지 fwrite함수를 실행하며 이전 앞에선 fwrite함수에 넣어준 값만큼 읽어왔는데 1씩 넣어주었고 파일은 출력파일에 입력을 하도록 넣어 주었다. 위 반복문이 끝난 후에는 복사가 완료되었다는 출력문과 함께 열었던 파일들을 닫아주면서 프로그램을 종료한다.

실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./ChoiceCopy2 Text2.txt Resul
t2.txt
File copied.
```



위 결과를 보았을 때 정상적으로 실행된 것을 볼 수 있다.

2. 교재 그림 3.21 ~ 3.22 일반 파이프 프로그램

코드

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

#define BUFFER_SIZE 25//버퍼사이즈를 25로 설정
#define READ_END 0//fd표준입력 0
#define WRITE_END 1//fd표준출력 1

int main(void)
{
    char write_msg[BUFFER_SIZE] = "Greetings";//메세지를 써주기 위한 파이프라인 넣어준 데이터를 확인하기 위해서 넣어준 문장
    char read_msg[BUFFER_SIZE];//메세지 읽기를 위한 파이프라인

    int fd[2];//파일 디스크립터 읽기와 쓰기를 넣어주기 위해 크기가 2인 배열 생성
    pid_t pid;//프로세스 아이디를 구별하기 위해 선언한 변수

    if (pipe(fd) == -1) {//프로세스가 생성되지 않는 경우
        fprintf(stderr, "Pipe failed");//표준에러와 함께 파이프가 생성되지 않았음을 출력함
        return 1;//종료시킨다.
    }
    pid = fork();//fork를 사용하여 프로세스를 실행시킨다.

       if (pid < 0) {//프로세스 생성실패
           fprintf(stderr, "Fork Failed");//표준에러와 함께 프로세스가 생성되지 않았음을 출력
           return 1;//종료시킨다.
       }

       if (pid > 0) {//pid가 0보다 큰 경우는 부모 프로세스이다.
           close(fd[READ_END]);//쓰기를 위해서 읽기 파이프 라인을 닫아준다.
           write(fd[WRITE_END], write_msg, strlen(write_msg) + 1);//쓰기를 해준다. fd값을 표준 입력으로 하고, 메세지 입력
               버퍼에,메세지버퍼의 길이보다 1을 더해줘서 NULL값을 더해준다.
           close(fd[WRITE_END]);//쓰기가 완료되면 쓰기 파이프라인을 닫아준다.
       }
       else {//0보다 크지도 않고 작지도 않은 경우는 0일 때 즉, 자식프로세스 일 때이다.
           close(fd[WRITE_END]);//쓰기모드를 닫아준다. 자식프로세서는 Consumer의 개념이기 때문이다.
           read(fd[READ_END], read_msg, BUFFER_SIZE);//read함수를 사용하여 fd값을 읽기모드로 변경 후, 리드 메세지 파이프 라인을 사용하여
               버퍼 사이즈만큼 메세지를 받아온다.
           printf("read %s\n", read_msg);//받아온 메세지를 출력해준다.
           close(fd[READ_END]);//읽기가 끝나면 읽기파이프라인을 닫아준다.
       }

       return 0;
   }
```

위 프로그램은 파이프중에서 Ordinary Pipe프로그램이다. 먼저 버퍼의 크기를 25로 설정을 하고 fd신호값인 0과 1을 READ_END, WRITE_END를 전역변수로 선언을 해주었다. 이제 메인함수를 보자면 Char write_msg 입력 파이프라인의 크기를 버퍼의 크기만큼 만들어줬으며, 출력값을 확인하기 위해 “Greetings”라는 문장을 넣어주었다. 다음으로 읽는 파이프라인 또한 쓰기 파이프라인과 동일한 크기로 선언해주었다. 정수형으로 선언된 파일디스크립터는 배열을 사용하여 0과 1 입력과 출력의 값을 선언해준다. pid_t자료형을 사용 pid를 선언하여 프로세서 아이디를 구별하기 위해 선언을 해준다.

제어문을 통해 파이프의 디스크립터가 -1일 경우 프로세스가 생성되지 않은 경우이다. 이 경우 표준에러와 함께 파이프가 생성되지 않았음을 출력하고 종료시킨다. 그 다음 fork함수를 이용하여 pid에 넣어준다. 여기서 또한 제어문을 사용하여 프로세스가 생성되었는지 안되었는지 구분을 해준다. 만약 pid값이 음수면 프로세스 생성이 실패되었기 때문에 실패했다는 출력문과 함께 프로그램을 종료한다. 만약 0보다 크다면 부모 프로세스이다. 이때 쓰기를 위해 읽기 파이프라인을 닫아주기 위해 close(fd[READ_END]) 함수를 써준다. 그 이후 쓰기를 해주기 위해 fd값을 표준 입력으로 설정해주고 메세지를 입력한다. 마지막 인자에 메세지의 길이보다 +1을 해주어 NULL값이 들어가 원래 출력해야하는 메세지가 정상적으로 출력이 되도록 조절을 해준다. 출력이 될 것은 쓰기 파이프라인에 있는 값이다. 그 다음에 실행이 끝났으면 파이프라인을 close함수를 사용하여 fd값에 쓰기 값을 넣어준다.

else는 pid==0인 상황 즉, 자식프로세스를 뜻한다. 이것은 부모프로세스와 반대로 행해준다. 처음에 쓰기 파이프라인을 닫아주고 읽는 파이프라인을 여는데 읽어오는 값은 읽기 파이프라인에 있는 값이며, 버퍼의 사이즈 만큼 불러온다. 위 행위가 끝난 후 출력을 하는데 출력 파이프라인에서 받아

온 메세지를 출력해준다. 위 행위가 끝난 후 즉 읽기가 끝나면 읽기 파이프라인을 닫아주는 명령어로 프로그램을 종료한다. 위 프로그램의 실행은 아래와 같다

실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/middleExam$ ./OP
read Greetings
```

3. 네임드 파이프라인 프로그램

코드 (Writer)

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

int main()
{
    int fd;

    mkfifo("NamedPipeLineEx", 0666);

    if(mkfifo("NamedPipeLineEx",0666)==-1){
        perror("NamedPipe가 생성되지 않았습니다.");
        exit(1);
    }

    char mes[100];
    while (1)
    {
        fd = open("NamedPipeLineEx",O_WRONLY);
        gets(mes);
        write(fd, mes, 100);
        close(fd);
    }
    return 0;
}
```

네임드 파이프라인은 일반 파이프보다 더 강력한 통신도구로 사용된다. 먼저 부모자식 관계외 모든 프로세서와 통신이 가능하고 양방향 통신이 가능하다는 점에서 통신도구로써는 최적의 조건을 가지고 있다. 나는 네임드 파이프라인을 통해 양방향 통신은 구현하지 못했다. 대신 단방향 통신으로 네임드 파이프를 생성하고 사용을 해보았다. 먼저 Writer의 코드를 설명하겠다.

먼저 파일을 받을 변수 fd를 선언한다. 그 다음 네임드 파이프라인을 생성하는 mkfifo함수를 사용하여 생성할 파이프 라인의 이름을 작성하고 0666코드를 넣어준다. 이때 mkfifo함수가 리턴값이

-1이라면, 네임드파이프가 생성이 되지 않은 것이다. 그러므로 오류를 출력해주는 함수인 perror을 사용하여 파이프라인이 생성되지 않았다는 출력문과 함께 프로그램을 종료해준다.

정상적으로 파이프라인이 생성이 되었다면 네임드파이프 파일이 생성이 된다. 그 다음으로 Writer가 메세지를 작성할 문자열 배열을 선언해주고 그 크기는 100으로 해주었다. 다음으로 반복문을 사용하여 Writer는 네임드파이프를 통해 계속 입력을 할 수 있도록 하였다. 먼저 Writer이기 때문에 파이프를 쓰기모드로 열어주었고, while문을 사용한 것은 다른 값을 받아서 Reader에게 보내기 위함이였기 때문에 gets함수를 통해 문자열에 저장 후, 파이프라인에 입력받은 값을 쓴다. 그 이후 입력이 끝났다면 파이프 라인을 닫아주고 사용자가 강제종료를 하기 전 까지 while문을 반복한다. 단방향 통신이기 때문에 Writer는 쓰기만하고 Reader는 읽기만 한다.

코드(Reader)

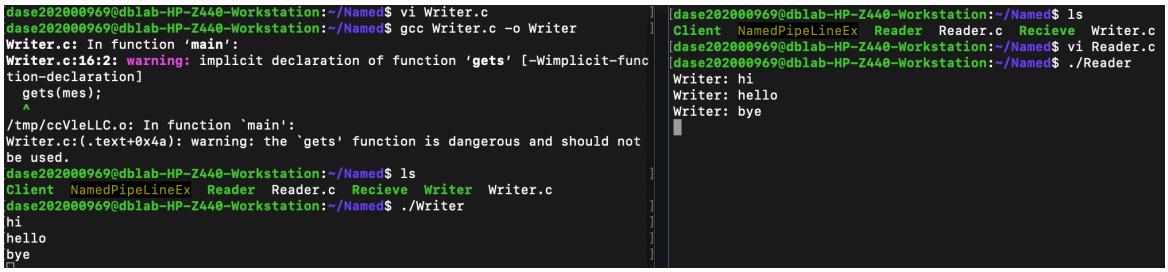
```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;
    char mes[100];
    while (1)
    {
        fd = open("NamedPipeLineEx",O_RDONLY);
        read(fd, mes, sizeof(mes));
        printf("Client: %s\n",mes);
        close(fd);
    }
    return 0;
}
```

위 코드는 Reader 코드이다. Writer가 먼저 실행이 된 후 Reader를 실행해야하기 때문에 네임드 파이프를 생성해주는 mkfifo함수와 그에 따른 예외처리문을 써주지 않았다. 그 이유는 네임드파이프는 생성이 된 후 파일이 생성이 되고 하나의 파이프라인을 이미 생성했기 때문에 위 코드에서 제거를 하였다. Reader함수는 실행이 되면 Wirter가 while문으로 메세지를 보내기 때문에 Reader도 while문을 사용하였고 반복문안에서 파이프라인을 읽기모드로 열고, 출력문을 통해 Writer가 보낸 내용을 볼 수 있도록 출력해준다. 읽기가 끝난 후 파이프라인을 닫아주는 방식으로 Writer는 메세지를 보내기만 Reader는 읽기만 하는 프로그램을 단방향 통신을 네임드파이프로 구현해보았다.

실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~/Named$ ls
Client  NamedPipeLineEx  Reader  Reader.c  Recieve  Writer  Writer.c
dase202000969@dblab-HP-Z440-Workstation:~/Named$
```

실행결과2



4. 교재 4.11 다중 스레드 프로그램

코드

```c
//  심화과제4
//
//  Created by 이대호 on 2022/05/17.
//
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>

int sum; /* this data is shared by the thread */
void *runner(void *param);

int main(int argc, char *argv[])
{
 pthread_t tid;//쓰레드의 식별자를 선언
 pthread_attr_t attr; //새로 생성되는 쓰레드의 속성을 설정한다.

 /* set the default attributes of the thread */
 pthread_attr_init(&attr);//쓰레드의 속성을 기본값으로 바꿔주는 함수를 사용하여 기본값으로 변경해준다.
 /* create the thread */
 pthread_create(&tid, &attr, runner, argv[1]);//쓰레드를 설정한다. 쓰레드의 식별자 주소, 러너 함수를 실행, 1부터 덧셈을 진행할 값을 char형 배열에 숫자를 받아준다.
/* wait for the thread to exit */
 pthread_join(tid,NULL);
//쓰레드가 정상종료할때 까지 기다리는 pthread_join을 사용한다. runner함수에서 pthread_exit에 0이 있어 종료처리가 되기 때문에 기다린다.
 printf("sum = %d\n",sum);
    //러너함수에서 나온 값을 출력해준다.
}

/* The thread will execute in this function */
void *runner(void *param)
{
    //char형으로 받았기 때문에 아스키코드값을 정수형으로 변경시켜주는 함수인 atoi함수를 사용하여 sum에 값을 더해준다.
 int i, upper = atoi(param);
 sum = 0;
 for (i =1; i<=upper; i++)
        sum+=i;

 pthread_exit(0);//쓰레드를 사용한 이유 사용한 쓰레드를 놓아준다.
}
```

먼저 메인함수를 실행할 스레드(부모스레드)가 할당이 된다. 이 함수 또한 프로그램이 명령어처럼 실행이 되기 때문에 argc는 들어오는 매개인자의 갯수를 카운트하고 들어오는 숫자까지의 합을 구하는 프로그램이기 때문에 argv[1]에 숫자가 들어온다. argv[0]에는 프로그램이 들어오게 된다. 그 이후 pthread_t tid 변수를 선언하여 식별자를 생성해준다. 그 이후 스레드의 속성을 설정하는 attr_t 자료형을 선언한다. 하지만 정확한 속성을 정해주지 않았기 때문에 기본적인 속성으로 바꾸어 주는 attr_init함수를 사용하여 attr의 속성을 기본으로 만들어준다. 그 이후 새로운 스레드를 생성시켜주는데 create함수에(&tid, &attr, runner, argv[1]) 매개인자를 넣어준다 먼저 tid는 스레드의 식별자이고, attr은 위에서 설정한 스레드이고 runner함수를 돌리기 위해 스레드를 새로 생성해주는 작업이다. argv[1]은 입력한 숫자까지의 합을 구하는 프로그램이기 때문에 입력한 숫자를 *parm의 값을 넣어준다. 그 다음에 새로 생성된 스레드(자식프로세스)는 runner 함수를 실행시킨다. char형으로 받았기 때문에 아스키코드를 정수형으로 바꾸어주는 atoi함수를 사용해서 upper라는 변수에 넣어주고 반복문에 upper 까지의 합을 구하는 로직을 작성해준다. 이 때 sum 변수에 합을 넣어주는데, 이 때 sum은 전역변수이며, 자식스레드와 부모스레드가 공유하는 데이터이다. 이 때 메인함수에 있는 부모스레드는 runner 함수를 실행하고 있는 자식스레드를 기다리는데 그 명령어가 pthread_join(tid,NULL);함수이다. 프로세스를 만드는 프로그램에서도 나온 함수이다. 그 다음에 자식스레드가 runner라는 함수를 다 실행하게 되면, pthread_exit함수를 사용하여 자식스레드

를 놓아준다. 부모스레드는 자식스레드가 종료가 된 이후 실행이 되기 때문에 runner 함수에서 반환과 동시에 다시 실행이되고 그 부모스레드는 메인함수를 실행하는 스레드이기 때문에 join이후에 명령어를 실행하게 된다. 자식스레드와 부모스레드는 전역변수인 sum을 공유했기 때문에 부모스레드가 sum이라는 값을 반환하고. 이 입력받은 숫자까지의 합을 출력하게 하는 프로그램이다.

즉 처음에는 단일 스레드 프로그램이였다가 함수를 실행하기 위해 자식스레드를 생성하여 멀티스레드 프로그램으로 변경되었다.그러므로 이 프로그램은 멀티스레드 프로그램이라고 부르는 것 이다.

실행결과

```
dase202000969@dblab-HP-Z440-Workstation:~$ gcc Multithreaded.c -lpthread -o MT
dase202000969@dblab-HP-Z440-Workstation:~$ ./MT 100
sum = 5050
```

과제를 하고 느낀점

이번학기 항공소프트웨어공학과의 수업중에는 코딩을 하는 과목이 없었다. 그래서 코딩에 대한 감각이 떨어지고 있었고 원래 이번학기에 리눅스를 배워야하는데 지난 학기에 배워서 리눅스 강의가 안열렸다는 점에서 아쉬웠다. 그래서 그런지 중요하지만 추상적인 과목이라고 생각을 했었다. 이번 과제를 통해 미루고미뤘던 서버에 원격접속도 해보고 계정도 만들어보고 또 거기안에서 내가 vi 편집기를 사용해서 코딩을 했다는 점에서 뭔가 뿌듯하다.

필수과제는 수업시간에 했었고 강의에서도 여러번 본 내용이라 어렵지 않았다. 컴파일 하는 작업과 코드에 에러가 발생했을 때 컴파일 도중 발견하는 것 때문에 귀찮은 정도만 있었다. 하지만 선택과제를 하는데 내가 헷갈려했던 개념과 멀티스레드 네임드파이프 그리고 부모 자식프로세스에 대한 내용을 내가 직접 코딩을 하고 출력을 하면서 추상적인 것들을 눈으로 직접 보았고 모르는 내용들은 교재와 구글링을 통해서 찾게되었다. 간만에 암기식 공부가 아니라 직접 모르는 내용을 찾아보고 새로운 것을 배우는 기회가 되었다. 아직도 네임드 파이프는 여러 방식이 있고 양방향 통신을 구현하지 못해서 아쉽다. 하지만 다른 프로그램을 사용하면서 C를 배울 때 많이 사용하지 않았던 함수들을 한번 더 보게 되었고 요즘 파이썬으로 사람들에게 교육을 하는데 file open close개념과 왜 파일을 열고 닫는지 잘 알고있어서 프로그램을 짜면서 어려움과 익숙함을 느끼는 과제였다.

물론 양도 많고 어려워서 밤을 정확히 4일 동안 216호와 218호를 넘나 들면서 잠을 못잤지만 중간고사를 준비하면서 소홀해진 운영체제에 대한 이해와 멀티스레드에 정확한 알고리즘이해 그리고 부모 자식 프로세스에 대한 함수와 독립적으로 실행이 된다는 점을 알게되어서 유익한 시간이 되었다.

* 구글링을 통해 많은 정보를 얻었지만, 그만큼 많은 사이트들을 거쳐오면서 모든 사이트를 저장해 놓지 못해서 참고 링크를 못단점 죄송합니다.*