시스템프로그래밍(가) 과제3

소프트웨어학부 20192800 권대현

1. 개발 환경
   - 운영체제: Windows 11 Home
   - 하위 시스템: GNU/Linux 5.15.146.1-microsoft-standard-WSL2 x86_64
   - 리눅스 버전: Ubuntu 22.04.2 LTS
2. 소스코드 설명
   - mystdio.h

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

#define BUFSIZE 1024
#define SEEK_SET 0
#define SEEK_CUR 1
#define SEEK_END 2


typedef struct myFile {
    int fd;
    int pos;
    int size;
    int mode;
    int flag;
    char lastop;
    int eof;    //bool
    char *buffer;
    int bufferPos;
    int bufferEnd;
} myFile;

myFile  *myfopen(const char *pathname, const char *mode);
int     myfread(void *ptr, int size, int nmemb, myFile *stream);
int myfwrite(const void *ptr, int size, int nmemb, myFile *stream);
int myfflush(myFile *stream);
int myfseek(myFile *stream, int offset, int whence);
int myfeof(myFile *stream);
int myfclose(myFile *stream);

myFile *myfopen(const char *pathname, const char *mode)
{
```

```
    myFile *file = (myFile *)malloc(sizeof(myFile));    //myFile 구조체 선언 및 메모리 할당
    int flag;

    if (strcmp(mode, "r") == 0)
        flag = O_RDONLY;
    else if (strcmp(mode, "r+") == 0)
        flag = O_RDWR;
    else if (strcmp(mode, "w") == 0)
        flag = O_WRONLY | O_CREAT | O_TRUNC;
    else if (mode == "w+")
        flag = O_RDWR | O_CREAT | O_TRUNC;
    else if (mode == "a")
        flag = O_WRONLY | O_CREAT | O_APPEND;
    else if (mode == "a+")
        flag = O_RDWR | O_CREAT | O_APPEND;
    else
    {
        printf("Failed to open file (Wrong mode)\n");
        return NULL;
    }

    file->fd = open(pathname, flag, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
    if(file->fd < 0)
    {
        printf("Failed to open file (Open failed)\n");
        return NULL;
    }

    if(flag & O_APPEND)
        file->pos = lseek(file->fd, 0, SEEK_END);   //만약 file open mode가 a 또는 a+라면 마지막 커서 위치로 파일 커서 옮기기
    else
        file->pos = lseek(file->fd, 0, SEEK_SET);

    struct stat st; //file 크기를 구하기 위한 stat 구조체 선언
    if (fstat(file->fd, &st) < 0)
    {
        printf("Failed to open file (Size error)\n");
        close(file->fd);
        free(file);
    }

    //myFile 구조체 초기화
    file->size = st.st_size;
    file->mode = flag;
    file->flag = 0;
    file->lastop = '\0';
```

```
    file->eof = 0;
    file->buffer = (char *)malloc(sizeof(char) * BUFSIZE);
    file->bufferPos = 0;
    file->bufferEnd = 0;

    return file;
}

int myfread(void *ptr, int size, int nmemb, myFile *stream)
{
    //파일 스트림 예외 처리
    if (stream == NULL || stream->mode & O_WRONLY)
    {
        printf("Failed to read file (Wrong stream)\n");
        return 0;
    }

    size_t totalSize = size * nmemb;    //read할 크기 총량
    size_t readSize = read(stream->fd, ptr, totalSize); //read한 크기를 담기
위한 변수

    stream->pos += readSize;    //읽어들인 크기만큼 pos값 증가

    if (readSize == totalSize)  //만약 읽어들인 크기와 read할 크기 총량이 같다면
eof 처리
        stream->eof = 1;

    return (int)readSize / size;
}

int myfwrite(const void *ptr, int size, int nmemb, myFile *stream)
{
    if (stream == NULL || stream->mode & O_RDONLY)
    {
        printf("Failed to write file (Wrong stream\n");
        return 0;
    }
    size_t totalSize = size * nmemb;
    size_t writeSize = 0;

    //버퍼링 수행
    while(writeSize < totalSize)    //writeSize가 totalSize와 같거나 커질
때까지 반복
    {
        //버퍼 공간의 크기를 담기 위한 변수 선언
        size_t bufferSpace = BUFSIZE - stream->bufferPos;   //전체 버퍼 크기
대비 남은 공간 계산
```

```
        if(bufferSpace == 0)
        {
            if(myfflush(stream) == EOF)
                return writeSize / size;
            bufferSpace = BUFSIZE;
        }

        size_t copySpace = totalSize - writeSize;
        if(copySpace > bufferSpace)
            copySpace = bufferSpace;

        //stream의 버퍼로 ptr 메모리 복사
        memcpy(stream->buffer + stream->bufferPos, (const char *)ptr +
writeSize, copySpace);
        stream->bufferPos += copySpace;
        writeSize += copySpace;
    }

    stream->pos += writeSize;

    return (int)writeSize / size;

}

int myfflush(myFile *stream)
{
    if(stream == NULL)
    {
        printf("Failed to flush (Wrong stream)\n");
        return EOF;
    }

    //만약 stream의 버퍼 커서가 0보다 크다면 (적어도 한번 옮겨졌다면)
    if (stream->bufferPos > 0)
    {
        size_t flushSize = write(stream->fd, stream->buffer, stream-
>bufferPos);
        if(flushSize != stream->bufferPos)
        {
            printf("Failed to flush\n");
            return EOF;
        }
        stream->bufferPos = 0;
    }

    return 0;
}
```

```c
int myfseek(myFile *stream, int offset, int whence)
{
    if(stream == NULL)
        return -1;

    //seek 하기 전에 flush 수행
    if(myfflush(stream) == EOF)
        return -1;

    //seek 수행하여 커서 이동
    off_t pos = lseek(stream->fd, offset, whence);
    if(pos < 0)
        return -1;

    //커서 관련 변수 초기화
    stream->pos = pos;
    stream->eof = 0;
    stream->bufferPos = 0;
    stream->bufferEnd = 0;

    return 0;
}

int myfeof(myFile *stream)
{
    if(stream == NULL)
        return -1;

    return stream->eof;
}

int myfclose(myFile *stream)
{
    if(stream == NULL)
        return EOF;

    //close 하기 전 flush 수행
    if(myfflush(stream) == EOF)
        return EOF;

    //close 수행
    if(close(stream->fd) < 0)
    {
        printf("Failed to close file\n");
        return EOF;
    }

    //메모리 해제
```

```
    free(stream->buffer);
    free(stream);

    return 0;
}
```

- Assignment-3 pdf의 과제 명세에 맞게 IO wrapper 함수를 구현했다.
- 먼저 파일 구조체를 위한 myFile 구조체를 정의했다.
  - 파일 디스크립터 주소를 나타내는 fd
  - 파일의 커서를 나타내는 pos
  - 파일의 크기를 나타내는 size
  - 파일의 open mode를 나타내는 mode
  - 파일의 flag를 나타내는 flag
  - 파일의 최근 명령어를 나타내는 lastop
  - 파일의 EOF 여부를 나타내는 eof
  - 파일의 버퍼링을 위한 buffer
  - 파일의 버퍼 커서를 나타내는 bufferPos
  - 파일의 버퍼 끝을 나타내는 bufferEnd
- 파일을 열기 위한 myfopen()함수를 구현했다.
  - 먼저 myFile 구조체로 *file 변수를 선언하고 메모리를 동적 할당해준다.
  - 인자로 const char *mode를 받고 mode의 형태에 따라 open() flag값을 설정해준다.
  - 앞서 선언한 file내 fd에 open() 함수를 호출하여 리턴 값을 파일 디스크립터 주소를 넣어준다.
  - Mode의 append 여부에 따라 lseek함수를 통해 파일의 커서 위치를 변경시킨다.
  - File 크기를 구하기 위해 stat을 사용하여 fstat()을 수행한다.
  - myFile 구조체의 변수를 초기화한다.
- 파일로부터 데이터를 읽기 위한 myfread()함수를 구현했다.
  - 인자로 받은 파일 스트림이 NULL이거나 읽기 모드가 쓰기일 경우, 에러를 출력하고 0을 리턴한다.
  - 읽을 크기 총량을 담을 totalSize 변수를 선언한다.
  - 읽은 크기 총량을 담을 readSize 변수를 선언하고 read()를 수행한다.
  - readSize만큼 파일 스트림의 pos값을 증가시킨다.
  - 만약 읽은 크기와 읽을 크기 총량이 같다면 eof를 1로 설정한다.
  - 최종적으로 읽어들인 크기를 리턴한다.
- 파일로 데이터를 쓰기 위한 myfwrite()함수를 구현했다.
  - myfread와 동일하게 파일 스트림에 대한 예외 처리를 한다.

- ◆ 파일에 쓸 크기의 총량을 담을 totalSize를 선언한다.
- ◆ 쓴 크기의 총량을 담을 writeSize를 선언한다.
- ◆ writeSize가 totalSize와 같거나 커질 때까지 반복하는 while문을 통해 버퍼링을 수행한다.
  - 전체 버퍼 크기 대비 남은 공간을 담을 bufferSpace를 선언한다.
  - 만약 bufferSpace가 0, 즉 버퍼가 꽉 찼다면, flush를 통해 버퍼를 비워준다.
  - memcpy함수를 호출하여 ptr로 받아온 쓸 메모리를 파일 스트림의 buffer에 저장한다.
  - 파일 스트림의 bufferPos를 copy한 크기만큼 증가시킨다.
- ◆ 파일 스트림의 pos를 writeSize만큼 증가시킨다.
- ◆ 파일로 쓴 크기를 리턴한다.

■ 버퍼를 비우기 위한 myfflush()함수를 구현했다.
- ◆ 만약 파일 스트림의 버퍼 커서 bufferPos가 0보다 크다면
- ◆ Write()를 통해 버퍼에 저장된 데이터를 파일에 작성한다.
- ◆ 작성한 뒤 bufferPos를 0으로 초기화한다.

■ 파일의 커서를 이동시키기 위한 myfseek()함수를 구현했다.
- ◆ Seek를 수행하기 전에 flush를 통해 버퍼를 비워준다.
- ◆ lseek함수를 통해 파일 커서를 이동시킨다.
- ◆ 파일의 위치와 관련된 값들을 전부 초기화시켜준다.

■ 파일의 eof 여부를 체크하기 위한 myfeof()함수를 구현했다.
- ◆ 인자의 파일 스트림의 eof에 접근하고 값을 리턴한다.

■ 파일을 닫기 위한 myfclose()함수를 구현했다.
- ◆ 닫기 전에 flush를 통해 버퍼를 비워준다.
- ◆ close()함수를 호출하여 파일을 닫는다.
- ◆ 스트림의 버퍼와 스트림의 메모리를 해제한다.

3. 문제점 및 해결 방법

- myfopen() 구현 문제
  - ■ open을 통한 파일 디스크립터 번호를 myFile 구조체 내 fd에 넣는 작업을 수행하려 하는데, Segmentation Fault가 뜨는 오류가 발생했다.
  - ■ myFile 구조체 변수를 선언하면서 malloc을 통한 메모리 할당을 하지 않아서 생긴 간단한 문제였다.
- 버퍼링 구현 문제
  - ■ 처음엔 read 함수와 write 함수를 단순 wrapper 함수로 구현했다. 그러나, 과제 명세에서는 버퍼링을 구현할 것을 요구하기에 구현 방법에 대해서 고민하게 됐다.

- 버퍼링이란 데이터 입출력 시 여러 번의 작은 작업 대신 한 번의 큰 작업으로 성능을 최적화하는 기술을 의미한다. 이를 구현하기 위해 입출력 함수에서 while문을 돌아서 쓰기 데이터로 들어온 크기만큼 write를 수행하도록 변경했다. 먼저 쓸 데이터 총량을 계산하고, while문 내에서 버퍼의 남은 공간을 지속적으로 확인하며 버퍼에 데이터를 복사하게 된다. 만약, 버퍼가 가득 찰 경우, 버퍼 내용을 파일에 기록하고 버퍼를 초기화시킨다.

- myfread() 버퍼링 구현 문제
  - myfread()를 버퍼링으로 구현하였더니, myfread()를 수행할 때 파일에 write이 수행되는 버그가 발생했다.
  - 이는 버퍼에 남아있는 데이터가 read 작업을 수행할 때에도 남아있어서 생긴 문제라 판단하였고, myfwrite()함수에 myfflush를 호출하여 버퍼를 강제로 비워줬고, myfread()를 wrapper 함수로 구현하여 문제를 해결했다.