컴퓨터학부 20201841 박세연

<read1>

1. 소스코드

**[ssu\_employee.h]**

#define NAME\_SIZE 64

// 회원 정보를 담을 구조체 선언

struct ssu\_employee {

char name[NAME\_SIZE];

int pid;

int salary;

};

**[ssu\_read.c]**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include "ssu\_employee.h"

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

{

struct ssu\_employee record; // 구조체 변수를 선언

int fd; // 파일 디스크립터

int record\_num; // 출력할 회원의 번호를 입력받을 변수

if (argc < 2) // 명령어에 들어온 인자의 개수가 2개 이상이면 에러 처리

{

fprintf(stderr, "Usage : %s file\n", argv[0]); // 에러 출력

exit(1); // 비정상 종료

}

if ((fd = open(argv[1], O\_RDONLY)) < 0) // 명령줄에서 받은 파일을 open하고 에러 처리

{

fprintf(stderr, "open error for %s\n", argv[1]); // 에러 출력

exit(1); // 비정상 종료

}

// 사용자가 음수를 입력하지 않는 이상 계속해서 회원 정보를 출력

while (1)

{

printf("Enter record number : ");

scanf("%d", &record\_num); // 사용자에게 출력할 회원의 번호를 입력받음

if (record\_num < 0) // 음수를 입력받으면 break문을 통해 반복문을 탈출하고 프로그램 종료

break;

// lseek 함수로 출력하고자 하는 회원정보를 담고있는 부분으로 커서를 이동시킴

// 이때, 가장 처음부터 ssu\_employee 구조체의 크기 \* 번호 만큼 커서를 이동시켜야 하므로,

// whence인자로는 0, offset인자로는 회원 번호 \* 구조체 크기 를 가짐

if (lseek(fd, (long)record\_num \* sizeof(record), 0) < 0) // 에러처리

{

fprintf(stderr, "lseek error\n");

exit(1); // 비정상 종료

}

if (read(fd, (char \*)&record, sizeof(record)) > 0)

printf("Employee : %s Salary : %d\n", record.name, record.salary);

else

printf("Record %d not found\n", record\_num);

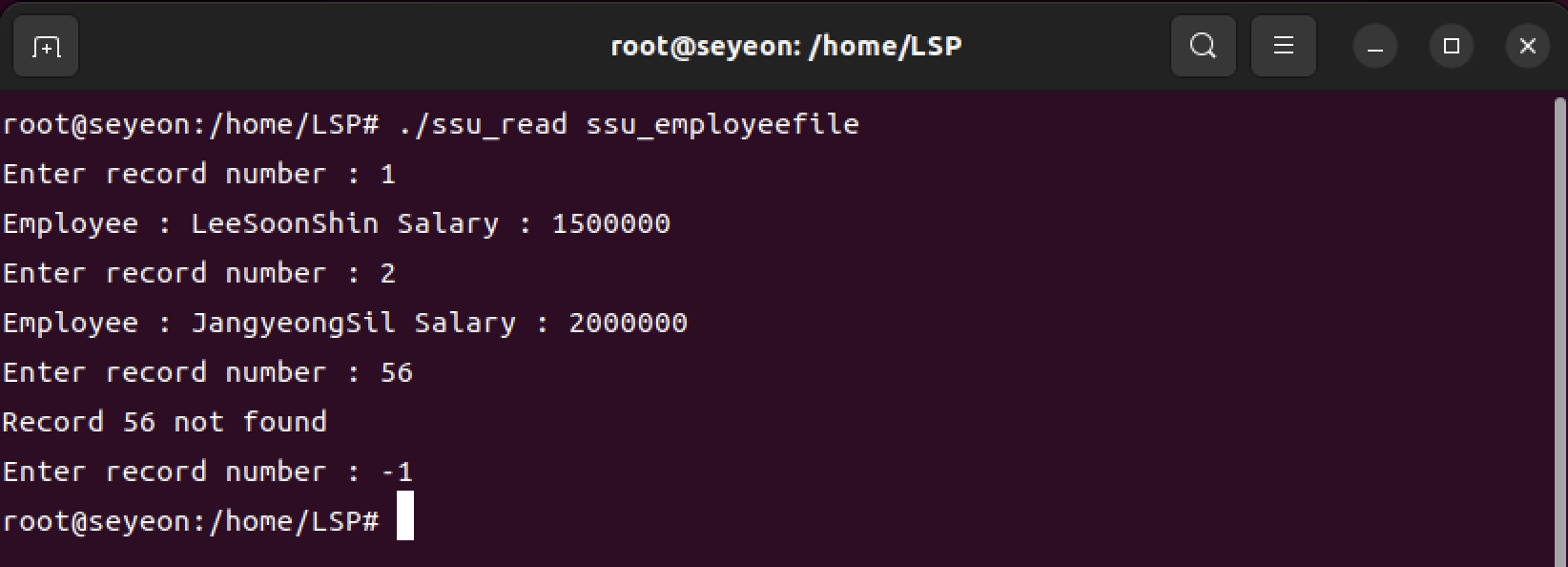
}

close(fd);

exit(0);

}

1. 실행결과



<read2>

1. 소스코드

**[ssu\_read\_2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#define BUFFER\_SIZE 1024

int main(void)

{

char buf[BUFFER\_SIZE]; // 문자를 읽어들이기 위한 buf 배열 선언

char \*fname = "ssu\_test.txt"; // 파일 이름

int count; // 읽은 바이트 수를 저장할 변수 선언

int fd1, fd2; // 파일 디스크립터

// open 함수를 이용하여 파일 디스크립터 값 받음

fd1 = open(fname, O\_RDONLY, 0644);

fd2 = open(fname, O\_RDONLY, 0644);

if (fd1 < 0 || fd2 < 0) // 파일 디스크립터 반환 값이 음수라면 에러 처리

{

fprintf(stderr, "open error for %s\n", fname); // 에러 출력

exit(1); // 비정상 종료

}

count = read(fd1, buf, 25); // buf를 이용하여 25byte를 읽어옴

// 읽어온 문자는 buf에 저장되고, read는 읽어온 byte수를 반환하기 때문에 count는 읽은 byte수를 가짐

buf[count] = 0; // 가장 끝의 문자열에 0을 넣어 문자열의 끝을 나타냄

printf("fd1's first printf : %s\n", buf); // buf를 출력하여 저장한 문자열을 확인함

lseek(fd1, 1, SEEK\_CUR); // lseek 함수를 이용하여 커서를 이동, 1칸 이동

// 현재 read로 읽은 곳을 커서가 가리키고 있고 1칸을 이동시켜 개행을 건너 띔

count = read(fd1, buf, 24); // buf를 이용하여 24byte를 읽어옴

buf[count] = 0; // 가장 끝의 문자열에 0을 넣어 문자열의 끝을 나타냄

printf("fd1's second printf : %s\n", buf); // buf를 출력하어 저장한 문자열을 확인함

count = read(fd2, buf, 25); // buf를 이용하여 25byte를 읽어옴

buf[count] = 0; // 가장 끝의 문자열에 0을 넣어 문자열의 끝을 나타냄

printf("fd2's first printf : %s\n", buf); // buf를 출력하여 저장한 문자열을 확인함

lseek(fd2, 1, SEEK\_CUR); // lseek 함수를 이용하여 커서를 이동, 1칸 이동

count = read(fd2, buf, 24); // 현재 read로 읽은 곳을 커서가 가리키고 있고 1칸을 이동시켜 개행을 건너 뜀

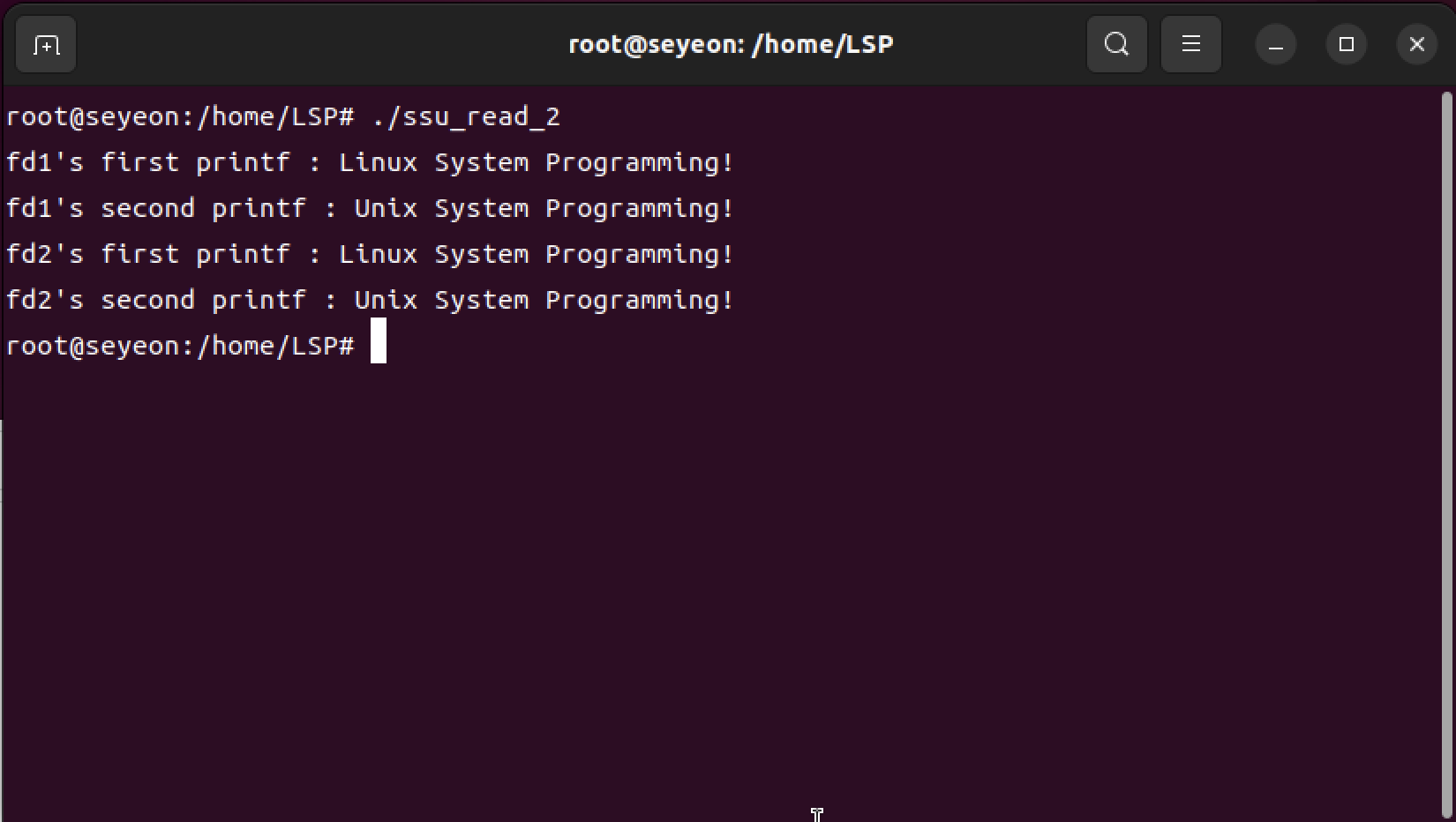
buf[count] = 0; // 가장 끝의 문자열에 0을 넣어 문자열의 끝을 나타냄

printf("fd2's second printf : %s\n", buf); // buf를 출력하여 저장한 문자열을 확인함

exit(0); // 정상종료

}

2. 실행결과



<read3>

1. 소스코드

**[ssu\_read\_3A.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main(void)

{

char c; // 문자를 한 문자씩 읽어오기 위한 변수

int fd; // 파일 디스크립터

// 파일을 읽기 전용으로 open하고 fd를 변환받음

if ((fd = open("ssu\_test.txt", O\_RDONLY)) < 0)

{ // 에러처리

fprintf(stderr, "open error for %s\n", "ssu\_test.txt"); // 에러 출력

exit(1); // 비정상 종료

}

// while 반복문을 통해 내용을 한 글자씩 읽어서 표준 출력으로 출력함

while (1)

{

if (read(fd, &c, 1) > 0) // read 함수를 통해 파일의 내용을 1글자씩 읽어서 c에 저장함

putchar(c); // putchar 함수는 매개변수로 들어온 문자를 표준출력하여 문자로 출력함

else

break; // 문자열이 모두 끝나면 else문으로 이동하여 break가 실행되고 반복문을 빠져나감

}

exit(0);

}

**[ssu\_read\_3B.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main(void)

{

char character; // 문자를 한 문자씩 읽어오기 위한 변수

int fd; // 파일 디스크립터

int line\_count = 0; // 개행의 개수를 셀 변수

// 파일을 읽기 전용으로 open하고 fd를 변환받음

if ((fd = open("ssu\_test.txt", O\_RDONLY)) < 0)

{ // 에러처리

fprintf(stderr, "open error for %s\n", "ssu\_test.txt"); // 에러 출력

exit(1); // 비정상 종료

}

// while 반복문을 통해 개행 문자의 개수를 세어서 몇 줄인지 출력하는 함수

while (1)

{

if (read(fd, &character, 1) > 0) // read 함수를 통해 파일의 내용을 1글자씩 읽어서 character에 저장

{

if (character == '\n') // 만약, 읽은 문자가 개행문자인 경우 개행 개수를 세는 변수를 +1

line\_count++;

}

else

break; // 문자열이 모두 끝나면 else문으로 이동하여 break가 실행되고 반복문을 빠져나감

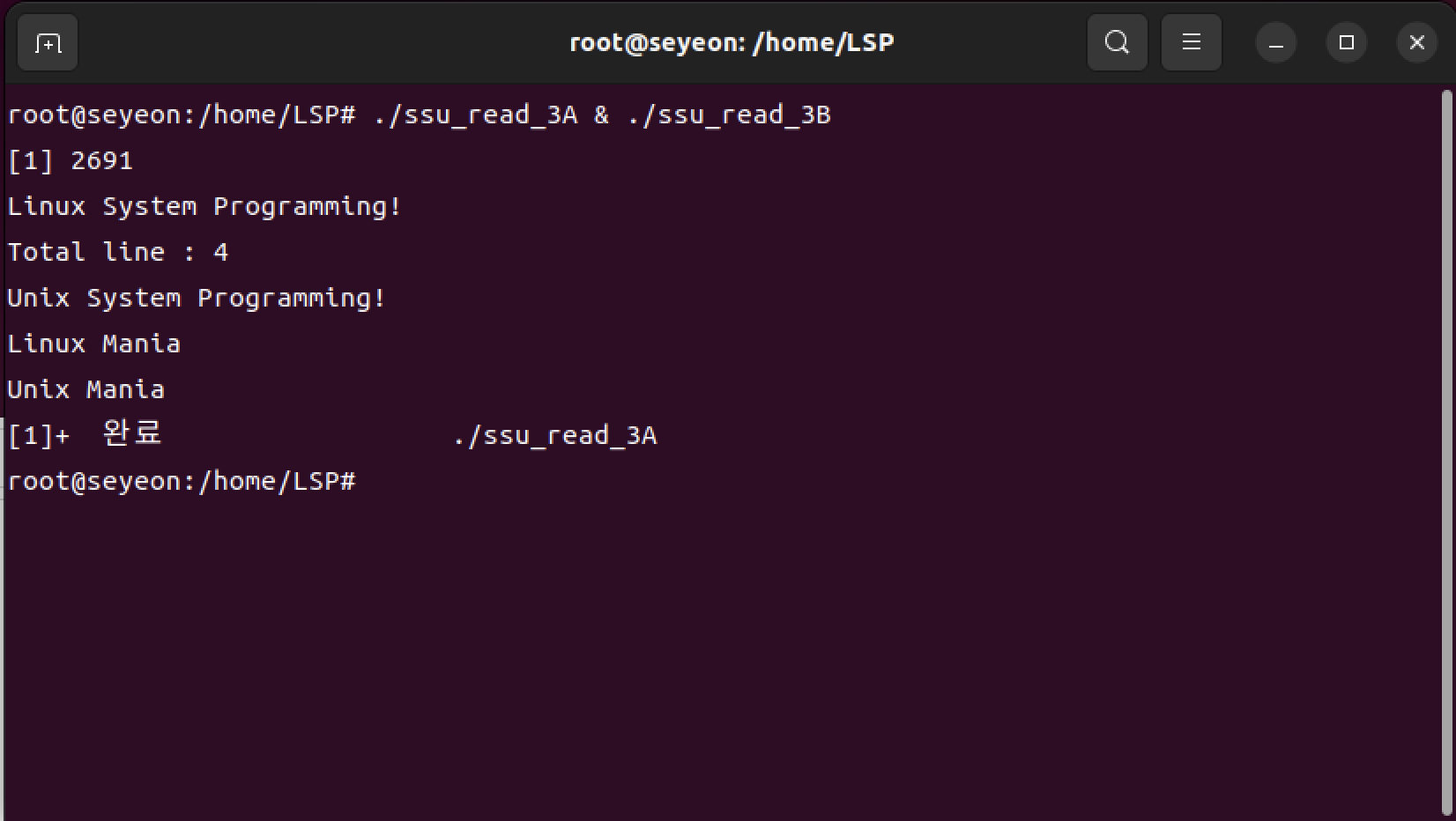
}

printf("Total line : %d\n", line\_count); // 최종 개행문자개수(몇 줄인지)를 출력함

exit(0);

}

2. 실행결과



<read4>

1. 소스코드

**[ssu\_read\_4.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#define TABLE\_SIZE 128

#define BUFFER\_SIZE 1024

// 주어진 파일에서 특정한 라인을 읽어오는 함수

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

{

// 파일 내 각 라인의 정보를 저장하는 데 사용하는 구조체

static struct

{

long offset; // 라인의 시작 위치를 저장하는 변수

int length; // 라인의 길이를 저장하는 변수

} table[TABLE\_SIZE];

char buf[BUFFER\_SIZE]; // 파일을 읽을 때 사용할 buf 변수

long offset; // 현재 문자의 위치를 나타내는 변수

int entry; // 현재 처리 중인 라인(몇번째 줄인지) 변수

int i; // for문에 사용할 변수

int length; // 읽어온 byte수를 저장하기 위한 변수

int fd; // 파일 디스크립터

// 파일명이 주어지지 않은 경우 (명령줄에서 인자가 2개들어와야 정상 작동함) 에러 처리

if (argc < 2)

{

fprintf(stderr, "usage : %s <file>\n", argv[0]); // 에러출력

exit(1); // 비정상종료

}

// 인자로 받은 파일을 open 함수를 이용하여 읽기 전용으로 파일 디스크립터를 반환받음

if ((fd = open(argv[1], O\_RDONLY)) < 0) // 에러처리

{

fprintf(stderr, "open error for %s\n", argv[1]); // 에러 출력

exit(1); // 비정상 종료

}

// 0으로 우선 초기화

entry = 0;

offset = 0;

// while문을 통해 읽으려는 파일을 BUFFER\_SIZE만큼 buf로 읽고 저장함

// 실제로 읽어온 byte수는 length에 저장함

while ((length = read(fd, buf, BUFFER\_SIZE)) > 0)

{

for (i = 0; i < length; i++) // 읽은 데이터의 길이만큼 반복문 수행

{

table[entry].length++; // 현재 처리중인 라인의 길이를 증가시킴

offset++; // 현재 파일 내에서 offset을 증가시킴

if (buf[i] == '\n') // 개행문자를 만나면 새로운 라인이 시작되었다는 의미

table[++entry].offset = offset; // entry를 증가시켜 다음 라인을 가리키도록 하고, 다음 라인의 정보를 저장 (다음라인의 시작위치는 현재 offset)

}

}

// DEBUG 매크로가 정의되어 있는 경우에만 실행

#ifdef DEBUG

// 테이블에 저장된 각 라인의 offset과 length를 출력

for (i = 0; i < entry; i++)

printf("%d : %ld, %d\n", i + 1, table[i].offset, table[i].length);

#endif

// 사용자로부터 출력할 라인의 번호를 입력받음

while (1)

{

printf("Enter line number : ");

scanf("%d", &length);

// 음수를 입력하면 반복문을 탈출하고 프로그램 종료

if (--length < 0)

break;

// 사용자가 올바른 번호를 입력한 경우

lseek(fd, table[length].offset, 0); // lseek을 통해 그 줄에 해당하는 테이블로 커서를 이동

if (read(fd, buf, table[length].length) <= 0) // 커서를 이동 후, buf를 통해 테이블의 내용을 length 만큼 읽음

continue;

buf[table[length].length] = '\0';

printf("%s", buf); // 입력한 라인의 내용을 출력

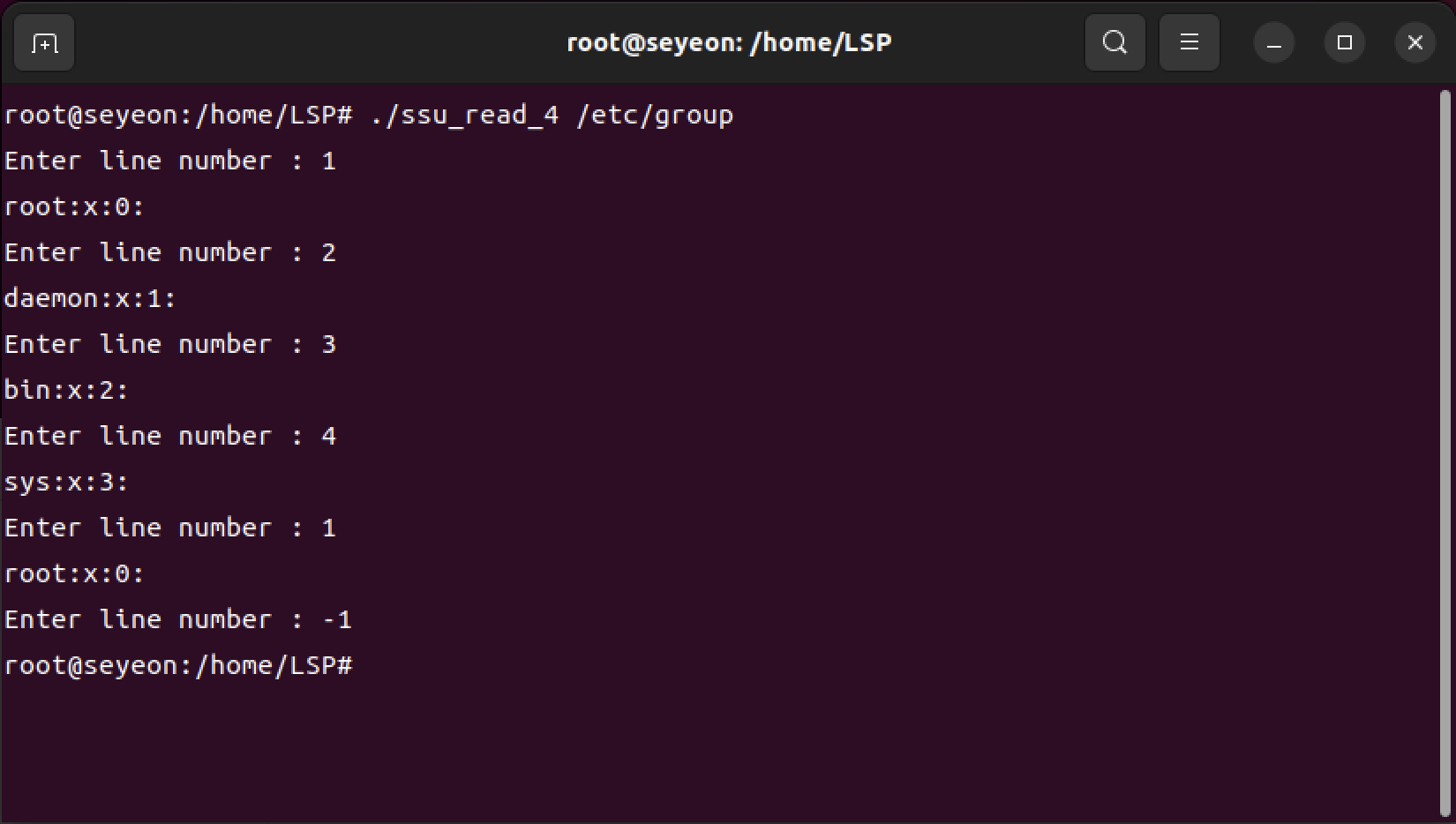
}

close(fd); // 파일 닫기

exit(0);

}

2. 실행결과



<write1>

1. 소스코드

**[ssu\_write\_1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define BUFFER\_SIZE 1024

// printf 함수를 구현함

int main(void)

{

char buf[BUFFER\_SIZE]; // 문자를 읽을 버퍼를 생성

int length; // 읽은 문자열의 길이를 저장하는 변수 생성

length = read(0, buf, BUFFER\_SIZE); // read 함수에서는 buf 변수로 BUFFER\_SIZE 만큼의 문자열을 읽음

// 이떄 fd를 0으로 지정하면, 이는 표준 입력이기 때문에 사용자가 입력한 문자열을 읽는다는 의미

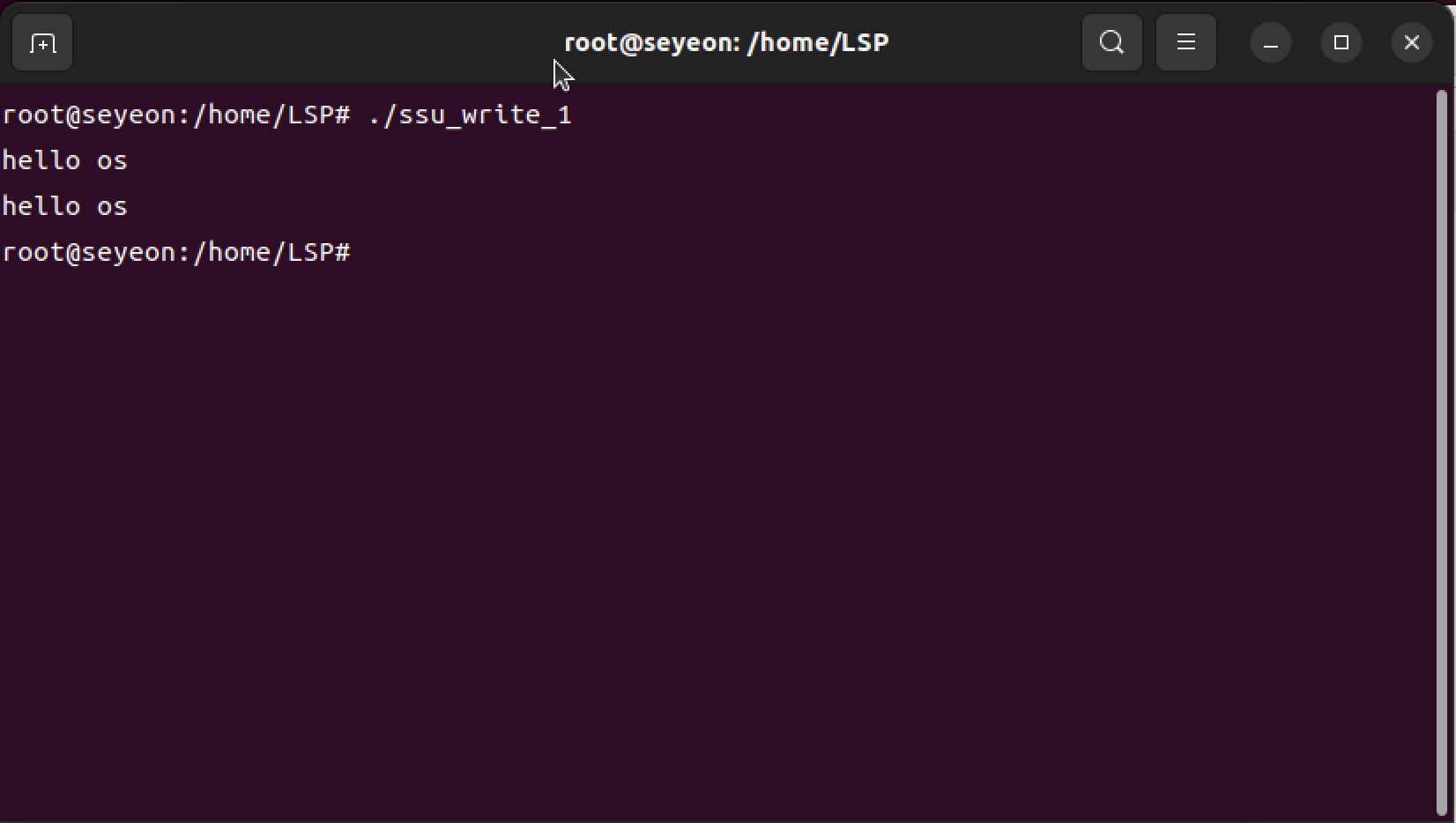
// 읽은 문자열의 byte수를 length에 저장

write(1, buf, length); // fd가 1이므로 표준 출력을 통해 length길이 만큼 buf에 저장되어있는 문자열을 출력

exit(0);

}

2. 실행결과



<write2>

1. 소스코드

**[ssu\_write\_2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#define S\_MODE 0644

#define BUFFER\_SIZE 1024

// cp 함수를 구현함

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

{

char buf[BUFFER\_SIZE]; // 문자를 읽을 버퍼를 생성

int fd1, fd2; // 파일 디스크립터

int length; // 읽은 문자열의 길이를 저장하는 변수 생성

// 명령어와 2개의 파일을 입력받아야 함 (총 3개의 인자를 명령줄에서 받아와야함)

if (argc != 3) // 그 이외의 겨우는 에러처리

{

fprintf(stderr, "Usage : %s filein fileout\n", argv[0]); // 에러출력

exit(1); // 비정상 종료

}

// open함수로 복사할 파일의 파일 디스크립터를 읽기 전용으로 반환받음

if ((fd1 = open(argv[1], O\_RDONLY)) < 0) // 에러 처리

{

fprintf(stderr, "open error for %s\n", argv[1]); // 에러 출력

exit(1); // 비정상 종료

}

// open함수로 복사하여 생성할 파일을 파일 디스크립터를 반환받음

// 이때 파일에 문자열을 출력하기 위해 0644 mode로 open함

if ((fd2 = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, S\_MODE)) < 0) // 에러처리

{

fprintf(stderr, "open error for %s\n", argv[2]); // 에러 출력

exit(1); // 비정상 종료

}

// while 반복문을 이용하여 fd1의 내용을 fd2에 복사함

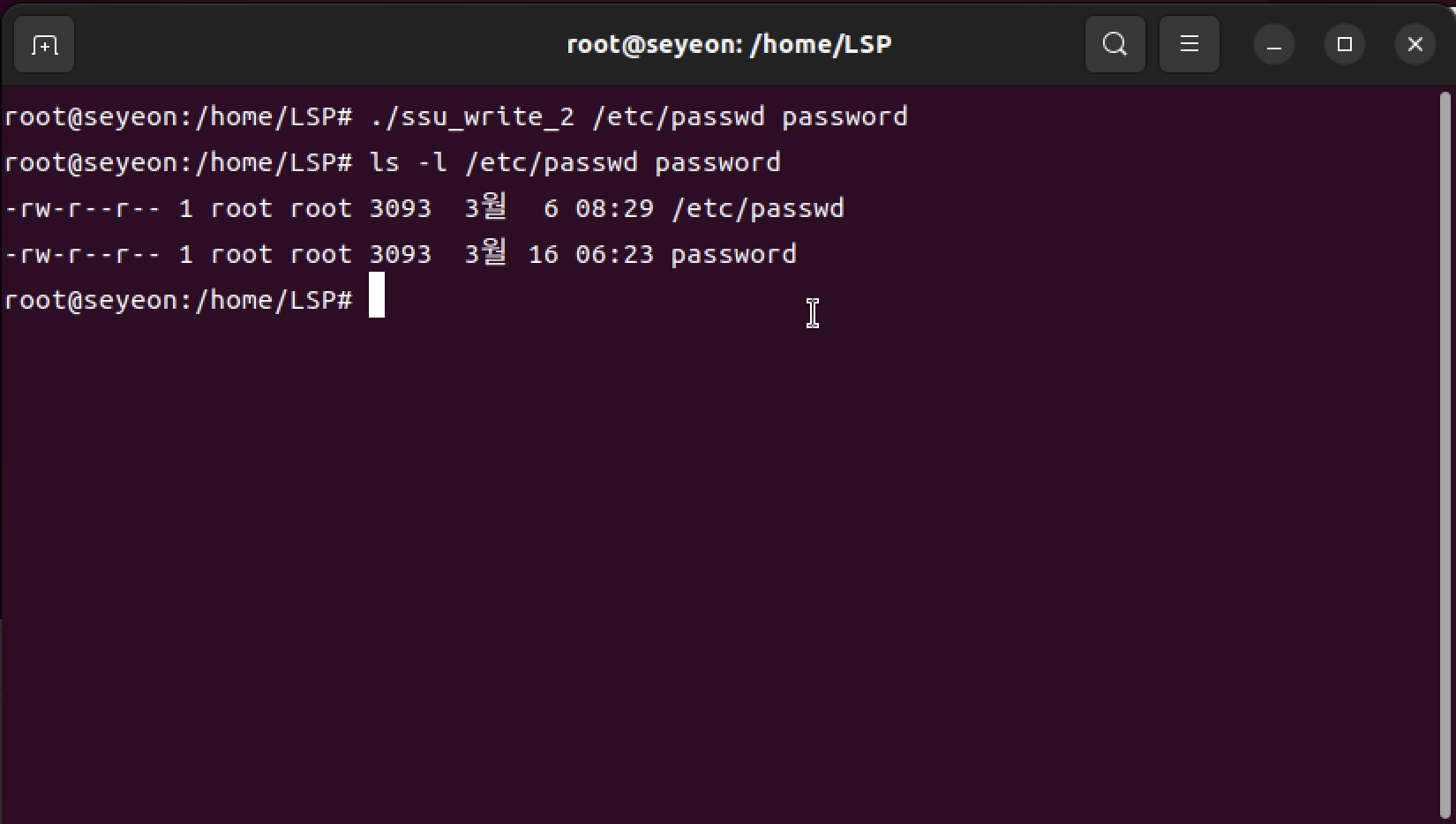
while ((length = read(fd1, buf, BUFFER\_SIZE)) > 0) // read 함수로 fd1의 내용을 읽어와서 buf에 저장하고 읽은 byte수를 length에 저장함

write(fd2, buf, length); // fd2에 buf의 내용을 length만큼 출력함

exit(0);

}

2. 실행결과



<write3>

1. 소스코드

**[ssu\_employee.h]**

#define NAME\_SIZE 64

// 회원 정보를 담을 구조체 선언

struct ssu\_employee {

char name[NAME\_SIZE];

int pid;

int salary;

};

**[ssu\_write\_3.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include "ssu\_employee.h"

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

{

struct ssu\_employee record; // 구조체 변수 선언

int fd; // 파일 디스크립터

// 명령줄에서 명령어와 파일명을 받아야하기 때문에 총 2개의 인자를 입력받아야 함

// 파일명을 입력받지 못한 경우 예외 처리

if (argc < 2)

{

fprintf(stderr, "usage : %s file\n", argv[0]); // 에러출력

exit(1); // 비정상 종료

}

// 인자로 받은 파일을 open 함수를 이용하여 파일 디스크립터를 반환받음

if ((fd = open(argv[1], O\_WRONLY | O\_CREAT | O\_EXCL, 0640)) < 0)

{ // 에러 처리

fprintf(stderr, "open error for %s\n", argv[1]); // 에러 출력

exit(1); // 비정상 종료

}

// while 반목문을 통해 사용자가 '.'를 입력할 때까지 계속해서 회원의 정보를 입력받게 함

while (1)

{

printf("Enter employee name <SPACE> salary : "); // 회원의 이름과 수익을 공백 기준으로 입력받음

scanf("%s", record.name);

if (record.name[0] == '.') // '.'를 입력하면 반복문을 탈출하고 프로그램 종료

break;

scanf("%d", &record.salary);

record.pid = getpid(); // getpid() : 함수를 호출한 프로세스의 ID를 반환받음

write(fd, (char \*)&record, sizeof(record)); // 구조체의 내용을 파일에 씀

}

close(fd);

exit(0);

}

2. 실행결과

