컴퓨터학부 20201841 박세연

<fcntl1>

1. 소스코드

**[fddup.c]**

#include <fcntl.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

int main(void)

{

int testfd; // 새로운 파일 디스크립터

int fd; // 원본 파일 디스크립터

// "test.txt" 파일을 생성 (이미 존재하면 열기)

fd = open("test.txt", O\_CREAT);

// 파일 디스크립터 fd를 복사하여 5 이상 중 가장 작은 번호의 파일 디스크립터 할당

testfd = fcntl(fd, F\_DUPFD, 5);

printf("testfd : %d\n", testfd); // 새로운 파일 디스크립터 번호 출력

// 다시 파일 디스크립터 fd를 복사하여 5 이상 중 가장 작은 번호의 파일 디스크립터 할당

testfd = fcntl(fd, F\_DUPFD, 5);

printf("testfd : %d\n", testfd); // 새로운 파일 디스크립터 번호 출력

// 사용자 입력을 기다림

getchar();

}

1. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl2>

1. 소스코드

**[fdcopy.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <string.h>

#include <fcntl.h>

#define MSG "message will be written to Terminal\n" // 출력할 메시지

int main(void) {

int new\_fd; // 새로운 파일 디스크립터

// STDOUT\_FILENO(표준 출력)를 복사하여 새로운 파일 디스크립터 생성

if ((new\_fd = fcntl(STDOUT\_FILENO, F\_DUPFD, 3)) == -1) {

fprintf(stderr, "Error : Copying File Descriptor\n"); // 복사 실패 시 에러 메시지 출력

exit(1); // 프로그램 종료

}

// 표준 출력 파일 디스크립터 닫기

close(STDOUT\_FILENO);

// 파일 디스크립터 3을 통해 메시지 출력

write(3, MSG, strlen(MSG));

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl3>

1. 소스코드

**[closeonexec\_3.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <fcntl.h>

int main(void) {

int flag;

// 표준 출력(STDOUT\_FILENO)에 대한 파일 디스크립터 플래그를 가져옴

if ((flag = fcntl(STDOUT\_FILENO, F\_DUPFD)) == -1) {

// 플래그를 가져오는 데 실패할 경우 에러 메시지를 출력하고 프로그램 종료

fprintf(stderr, "Error : Checking CLOSE\_ON\_EXEC\n");

exit(1);

}

// 플래그 값을 출력

printf("CLOSE ON EXEC flag is = %d\n", flag);

exit(0);

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl4>

1. 소스코드

**[nonblock1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <fcntl.h>

// 파일 디스크립터에 플래그 설정 함수 선언

void set\_flags(int fd, int flags);

// 파일 디스크립터에서 플래그 해제 함수 선언

void clr\_flags(int fd, int flags);

char buf[500000]; // 버퍼 크기 선언

int main(void){

int ntowrite, nwrite; // 남은 바이트 수와 쓰여진 바이트 수

char \*ptr; // 버퍼 포인터

// 표준 입력(STDIN\_FILENO)으로부터 데이터를 읽어 버퍼에 저장

ntowrite = read(STDIN\_FILENO, buf, sizeof(buf));

fprintf(stderr, "reading %d bytes\n", ntowrite); // 읽은 바이트 수를 출력

set\_flags(STDOUT\_FILENO, O\_NONBLOCK); // 표준 출력(STDOUT\_FILENO)에 O\_NONBLOCK 플래그 설정

ptr = buf; // 버퍼 포인터 초기화

while (ntowrite > 0) { // 남은 바이트 수가 0보다 큰 동안 반복

errno = 0; // 에러 번호 초기화

nwrite = write(STDOUT\_FILENO, ptr, ntowrite); // 데이터를 표준 출력에 쓰기

fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno); // 쓰여진 바이트 수와 에러 번호 출력

if (nwrite > 0) { // 쓰여진 바이트 수가 0보다 크면

ptr += nwrite; // 포인터를 앞으로 이동

ntowrite -= nwrite; // 남은 바이트 수 감소

}

}

clr\_flags(STDOUT\_FILENO, O\_NONBLOCK); // 표준 출력에서 O\_NONBLOCK 플래그 해제

exit(0); // 프로그램 종료

}

// 파일 디스크립터에 플래그 설정 함수 정의

void set\_flags(int fd, int flags)

{

int val;

if ((val = fcntl(fd, F\_GETFL, 0)) < 0) { // 현재 플래그 가져오기

fprintf(stderr, "fcntl F\_GETFL failed"); // 실패 시 에러 메시지 출력

exit(1); // 프로그램 종료

}

val |= flags; // 새로운 플래그 추가

if (fcntl(fd, F\_SETFL, val) < 0) { // 플래그 설정

fprintf(stderr, "fcntl F\_SETFL failed"); // 실패 시 에러 메시지 출력

exit(1); // 프로그램 종료

}

}

// 파일 디스크립터에서 플래그 해제 함수 정의

void clr\_flags(int fd, int flags)

{

int val;

if ((val = fcntl(fd, F\_GETFL, 0)) < 0){ // 현재 플래그 가져오기

fprintf(stderr, "fcntl F\_GETFL failed"); // 실패 시 에러 메시지 출력

exit(1); // 프로그램 종료

}

val &= ~flags; // 플래그 해제

if (fcntl(fd, F\_SETFL, val) < 0) { // 플래그 설정

fprintf(stderr, "fcntl F\_SETFL failed"); // 실패 시 에러 메시지 출력

exit(1); // 프로그램 종료

}

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<fcntl5>

1. 소스코드

**[nonblock2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <fcntl.h>

#define MAX\_CALL 100 // 호출 기록의 최대 개수

#define MAX\_BUF 20000 // 버퍼의 최대 크기

void serror(char \*str)

{

fprintf(stderr, "%s\n", str); // 에러 메시지 출력 함수

exit(1); // 프로그램 종료

}

int main(void){

int nread, nwrite, val, i = 0; // 읽은 바이트 수, 쓰여진 바이트 수, fcntl 리턴 값, 반복 횟수

char \*ptr; // 버퍼 포인터

char buf[MAX\_BUF]; // 입력 버퍼

int call[MAX\_CALL]; // 호출 기록을 저장할 배열

// 표준 입력(STDIN\_FILENO)에서 데이터를 읽어 버퍼에 저장

nread = read(STDIN\_FILENO, buf, sizeof(buf));

fprintf(stderr, "read %d bytes\n", nread); // 읽은 바이트 수 출력

// 표준 출력(STDOUT\_FILENO)의 플래그를 가져와서 O\_NONBLOCK 플래그 추가

if ((val = fcntl(STDOUT\_FILENO, F\_GETFL, 0)) < 0)

serror("fcntl F\_GETFL error");

val |= O\_NONBLOCK;

if (fcntl(STDOUT\_FILENO, F\_SETFL, val) < 0)

serror("fcntl F\_SETFL error");

// 버퍼에 있는 데이터를 표준 출력에 쓰기

for (ptr = buf; nread > 0; i++){

errno = 0;

nwrite = write(STDOUT\_FILENO, ptr, nread);

// 호출 기록에 쓰여진 바이트 수 저장

if (i < MAX\_CALL)

call[i] = nwrite;

fprintf(stderr, "nwrite = %d, errno = %d\n", nwrite, errno); // 쓰여진 바이트 수와 에러 번호 출력

if (nwrite > 0){

ptr += nwrite;

nread -= nwrite;

}

}

// 표준 출력(STDOUT\_FILENO)의 플래그를 가져와서 O\_NONBLOCK 플래그 해제

if ((val = fcntl(STDOUT\_FILENO, F\_GETFL, 0)) < 0)

serror("fcntl F\_GETFL error");

val &= ~O\_NONBLOCK;

if (fcntl(STDOUT\_FILENO, F\_SETFL, val) < 0)

serror("fcntl F\_SETFL error");

// 호출 기록 출력

for(i = 0; i < MAX\_CALL; i++)

fprintf(stdout, "call[%d] = %d\n", i, call[i]);

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<fcntl6>

1. 소스코드

**[open\_on\_exec.c]**

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

int main(void)

{

int fd;

int val;

fd = open("exec\_copy.txt", O\_CREAT);

execl("/home/seyeon/loop", "./loop", NULL);

exit(0);

}

**[close\_on\_exec.c]**

#include <stdio.h>

#include <stdlib.h>

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl7>

1. 소스코드

**[fcntl\_7.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

int main(void)

{

int fd;

int val;

if ((fd = open("exec\_copy.txt", O\_CREAT)) < 0) {

fprintf(stderr, "open error for %s\n", "exec\_copy.txt");

exit(1);

}

val = fcntl(fd, F\_GETFD, 0); // fd의 파일 디스크립터 플래그를 가져옴

// FD\_CLOEXEC 비트가 설정되어 있는지 확인

if (val & FD\_CLOEXEC)

printf("close-on-exec bit on\n");

else

printf("close-on-exec bit off\n");

val |= FD\_CLOEXEC; // FD\_CLOEXEC 비트를 설정

// FD\_CLOEXEC 비트가 설정되어 있는지 다시 확인

if(val & FD\_CLOEXEC)

printf("close-on-exec bit on\n");

else

printf("close-on-exec bit off\n");

fcntl(fd, F\_SETFD, val); // 변경된 파일 디스크립터 플래그를 fd에 설정

execl("/home/seyeon/loop", "./loop", NULL);

// "/home/seyeon/loop" 프로그램을 실행, 이때 현재 프로세스의 메모리 공간이 대체됨

// execl 호출 이후에는 코드가 실행되지 않음

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl8>

1. 소스코드

**[fcntl\_8.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include "ssu\_employee.h" // 구조체 ssu\_employee의 정의를 포함

#define DUMMY 0 // fcntl 함수 호출 시 사용되는 더미 값

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

{

struct ssu\_employee record; // 직원 정보를 저장할 구조체 변수

int fd; // 파일 디스크립터

int flags; // 파일 상태 플래그

int length; // 기록할 데이터의 길이

int pid; // 현재 프로세스 ID

// 프로그램 인자가 부족할 경우 사용법을 출력하고 종료

if (argc < 2) {

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

exit(1);

}

// argv[1] 파일을 읽기/쓰기 모드로 엶

if ((fd = open(argv[1], O\_RDWR)) < 0) {

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

exit(1);

}

// 파일 상태 플래그를 가져옴

if ((flags = fcntl(fd, F\_GETFL, DUMMY)) == -1) {

fprintf(stderr, "fcntl F\_GETFL error\n");

exit(1);

}

// O\_APPEND 플래그를 설정

flags |= O\_APPEND;

// 설정된 플래그를 파일 디스크립터에 적용

if (fcntl(fd, F\_SETFL, flags) == -1) {

fprintf(stderr, "fcntl F\_GETFL error\n");

exit(1);

}

pid = getpid(); // 현재 프로세스 ID를 가져옴

while(1) {

printf("Enter employee name : ");

scanf("%s", record.name); // 직원 이름 입력

// 이름이 '.'이면 루프 종료

if (record.name[0] == '.')

break;

printf("Enter employee salary : ");

scanf("%d", &record.salary); // 직원 급여 입력

record.pid = pid; // 현재 프로세스 ID를 저장

length = sizeof(record); // 기록할 데이터의 길이 설정

// 파일에 직원 정보를 기록, 기록 길이가 맞지 않으면 오류 출력 후 종료

if (write(fd, (char \*)&record, length) != length) {

fprintf(stderr, "record write error\n");

exit(1);

}

}

close(fd); // 파일 디스크립터 닫기

exit(0);

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl9>

1. 소스코드

**[fcntl\_lock1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <error.h>

#include <errno.h>

#define NAMESIZE 50 // 이름의 최대 크기

#define MAXTRIES 5 // 최대 재시도 횟수

// 직원 정보를 저장할 구조체 정의

struct employee {

char name[NAMESIZE];

int salary;

int pid;

};

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

{

struct flock lock; // 파일 잠금을 위한 구조체

struct employee record; // 직원 정보를 저장할 구조체 변수

int fd, sum = 0, try = 0; // 파일 디스크립터, 급여 합계, 시도 횟수

// 프로그램 시작 시 10초 대기

sleep(10);

// 읽기 전용으로 파일 열기

if ((fd = open(argv[1], O\_RDONLY)) == -1) {

perror(argv[1]); // 파일 열기에 실패하면 에러 메시지 출력 후 종료

exit(1);

}

// 파일 잠금 설정

lock.l\_type = F\_RDLCK; // 읽기 잠금

lock.l\_whence = 0; // 파일 시작 위치 기준

lock.l\_start = 0L; // 시작 오프셋

lock.l\_len = 0L; // 전체 파일 잠금

// 파일 잠금을 설정하고, 실패 시 최대 MAXTRIES만큼 재시도

while (fcntl(fd, F\_SETLK, &lock) == -1) {

if (errno == EACCES) { // 접근 권한 문제일 경우

if (try++ < MAXTRIES) { // 최대 재시도 횟수 미만일 경우

sleep(1); // 1초 대기 후 재시도

continue;

}

// 최대 재시도 횟수를 초과하면 에러 메시지 출력 후 종료

printf("%s busy -- try later\n", argv[1]);

exit(2);

}

perror(argv[1]); // 다른 에러 발생 시 에러 메시지 출력 후 종료

exit(3);

}

// 파일에서 직원 정보를 읽어와 출력하고 급여 합계를 계산

sum = 0;

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

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

sum += record.salary;

}

printf("\nTotal salary: %d\n", sum); // 총 급여 합계 출력

// 파일 잠금 해제

lock.l\_type = F\_UNLCK;

fcntl(fd, F\_SETLK, &lock);

close(fd); // 파일 디스크립터 닫기

}

2. 실행결과

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

<fcntl10>

1. 소스코드

**[fcntl\_lock2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <fcntl.h>

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

{

struct flock lock; // 파일 잠금을 위한 구조체

int fd; // 파일 디스크립터

char command[100]; // 시스템 명령어 문자열

// 읽기/쓰기 모드로 파일 열기

if ((fd = open(argv[1], O\_RDWR)) == -1) {

perror(argv[1]); // 파일 열기에 실패하면 에러 메시지 출력 후 종료

exit(1);

}

// 파일 잠금 설정

lock.l\_type = F\_WRLCK; // 쓰기 잠금

lock.l\_whence = 0; // 파일 시작 위치 기준

lock.l\_start = 0l; // 시작 오프셋

lock.l\_len = 0l; // 전체 파일 잠금

// 파일 잠금을 설정하고, 실패 시 에러 처리

if (fcntl(fd, F\_SETLK, &lock) == -1) {

if (errno == EACCES) { // 접근 권한 문제일 경우

printf("%s busy -- try later\n", argv[1]); // 파일이 바쁨을 알림

exit(2);

}

perror(argv[1]); // 다른 에러 발생 시 에러 메시지 출력 후 종료

exit(3);

}

// 파일을 vim으로 열기 위한 명령어 문자열 생성

sprintf(command, "vim %s\n", argv[1]);

system(command); // 시스템 명령어 실행

// 파일 잠금 해제

lock.l\_type = F\_UNLCK;

fcntl(fd, F\_SETLK, &lock);

close(fd); // 파일 디스크립터 닫기

return 0;

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<fcntl11>

1. 소스코드

**[fcntl\_lock3.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#define NAMESIZE 50 // 이름의 최대 크기

// 직원 정보를 저장할 구조체 정의

struct employee {

char name[NAMESIZE];

int salary;

int pid;

};

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

{

struct flock lock; // 파일 잠금을 위한 구조체

struct employee record; // 직원 정보를 저장할 구조체 변수

int fd, recnum, pid; // 파일 디스크립터, 레코드 번호, 프로세스 ID

long position; // 파일에서의 위치

// 읽기/쓰기 모드로 파일 열기

if ((fd = open(argv[1], O\_RDWR)) == -1) {

perror(argv[1]); // 파일 열기에 실패하면 에러 메시지 출력 후 종료

exit(1);

}

pid = getpid(); // 현재 프로세스 ID를 가져옴

for(;;) { // 무한 루프 시작

printf("\nEnter record number: ");

scanf("%d", &recnum); // 레코드 번호 입력

if (recnum < 0) // 음수 입력 시 루프 종료

break;

position = recnum \* sizeof(record); // 레코드 번호에 따른 파일 위치 계산

// 파일 잠금 설정

lock.l\_type = F\_WRLCK; // 쓰기 잠금

lock.l\_whence = 0; // 파일 시작 위치 기준

lock.l\_start = position; // 시작 오프셋

lock.l\_len = sizeof(record); // 잠글 데이터의 길이

// 파일 잠금을 설정하고, 실패 시 에러 처리

if (fcntl(fd, F\_SETLKW, &lock) == -1) {

perror(argv[1]);

exit(2);

}

// 파일 위치를 이동하고 레코드 읽기

lseek(fd, position, 0);

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

// 레코드를 찾지 못하면 메시지 출력 후 잠금 해제

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

lock.l\_type = F\_UNLCK;

fcntl(fd, F\_SETLK, &lock);

continue;

}

// 직원 정보 출력

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

record.pid = pid; // 현재 프로세스 ID를 기록

// 새로운 급여 입력

printf("Enter new salary: ");

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

// 파일 위치를 이동하고 레코드 쓰기

lseek(fd, position, 0);

write(fd, (char\*)&record, sizeof(record));

// 파일 잠금 해제

lock.l\_type = F\_UNLCK;

fcntl(fd, F\_SETLK, &lock);

}

close(fd); // 파일 디스크립터 닫기

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<fcntl12>

1. 소스코드

**[fcntl\_lock4.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#define NAMESIZE 50

// 구조체 정의: 직원 정보를 저장하는 구조체

struct employee {

char name[NAMESIZE]; // 직원 이름

int salary; // 직원 급여

int pid; // 프로세스 ID

};

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

{

// 파일 잠금 구조체 및 기타 변수 선언

struct flock lock;

struct employee record;

int fd, recnum, pid;

long position;

char ans[5];

// 파일 열기

if ((fd = open(argv[1], O\_RDWR)) == -1) {

perror(argv[1]); // 에러 출력

exit(1); // 프로그램 종료

}

// 현재 프로세스 ID 가져오기

pid = getpid();

// 무한 루프: 사용자가 음수 값을 입력할 때까지 반복

for (;;) {

printf("\nEnter record number : ");

scanf("%d", &recnum); // 사용자로부터 레코드 번호 입력 받음

// 음수가 입력되면 루프를 종료

if (recnum < 0)

break;

// 레코드의 위치 계산

position = recnum \* sizeof(record);

// 파일 잠금 설정

lock.l\_type = F\_RDLCK; // 읽기 잠금 설정

lock.l\_whence = 0; // 파일의 시작부터 오프셋 계산

lock.l\_start = position; // 잠금 시작 위치

lock.l\_len = sizeof(record); // 잠금 길이

// 파일 잠금 설정 요청

if (fcntl(fd, F\_SETLKW, &lock) == -1) {

perror(argv[1]); // 에러 출력

exit(2); // 프로그램 종료

}

// 파일 오프셋 이동 및 레코드 읽기

lseek(fd, position, 0);

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

printf("record %d not found\n", recnum); // 레코드를 찾지 못한 경우 메시지 출력

lock.l\_type = F\_UNLCK; // 잠금 해제

fcntl(fd, F\_SETLK, &lock); // 파일 잠금 설정 해제 요청

continue; // 반복문의 다음 반복으로 이동

}

// 직원 정보 출력

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

// 급여를 업데이트할 지 여부 확인

printf("Do you want to update salary (y or n)? ");

scanf("%s", ans);

// 사용자 입력에 따라 처리

if (ans[0] != 'y') {

lock.l\_type = F\_UNLCK; // 잠금 해제

fcntl(fd, F\_SETLK, &lock); // 파일 잠금 설정 해제 요청

continue; // 반복문의 다음 반복으로 이동

}

// 쓰기 잠금 설정

lock.l\_type = F\_WRLCK; // 쓰기 잠금 설정

if (fcntl(fd, F\_SETLKW, &lock) == -1) {

perror(argv[1]); // 에러 출력

exit(3); // 프로그램 종료

}

// 프로세스 ID와 새로운 급여 입력 받음

record.pid = pid;

printf("Enter new salary: ");

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

// 파일 오프셋 이동 및 레코드 쓰기

lseek(fd, position, 0);

write(fd, (char \*)&record, sizeof(record));

// 파일 잠금 설정 해제

lock.l\_type = F\_UNLCK; // 잠금 해제

fcntl(fd, F\_SETLK, &lock); // 파일 잠금 설정 해제 요청

}

// 파일 닫기

close(fd);

// 프로그램 종료

return 0;

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명