컴퓨터학부 20201841 박세연

<signal1>

1. 소스코드

**[signal\_1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

// 함수 선언

void signal\_handler(int signo);

// 함수 포인터 선언

void (\*func)(int);

int main(void){

// SIGINT 시그널에 대한 핸들러 함수를 func에 할당하고, 현재 등록된 핸들러 함수를 func에 저장

func = signal(SIGINT, signal\_handler);

while(1) {

printf("processing running...\n");

sleep(1);

}

exit(0);

}

// SIGINT 시그널에 대한 핸들러 함수

void signal\_handler(int signo) {

printf("SIGINT 시그널 발생.");

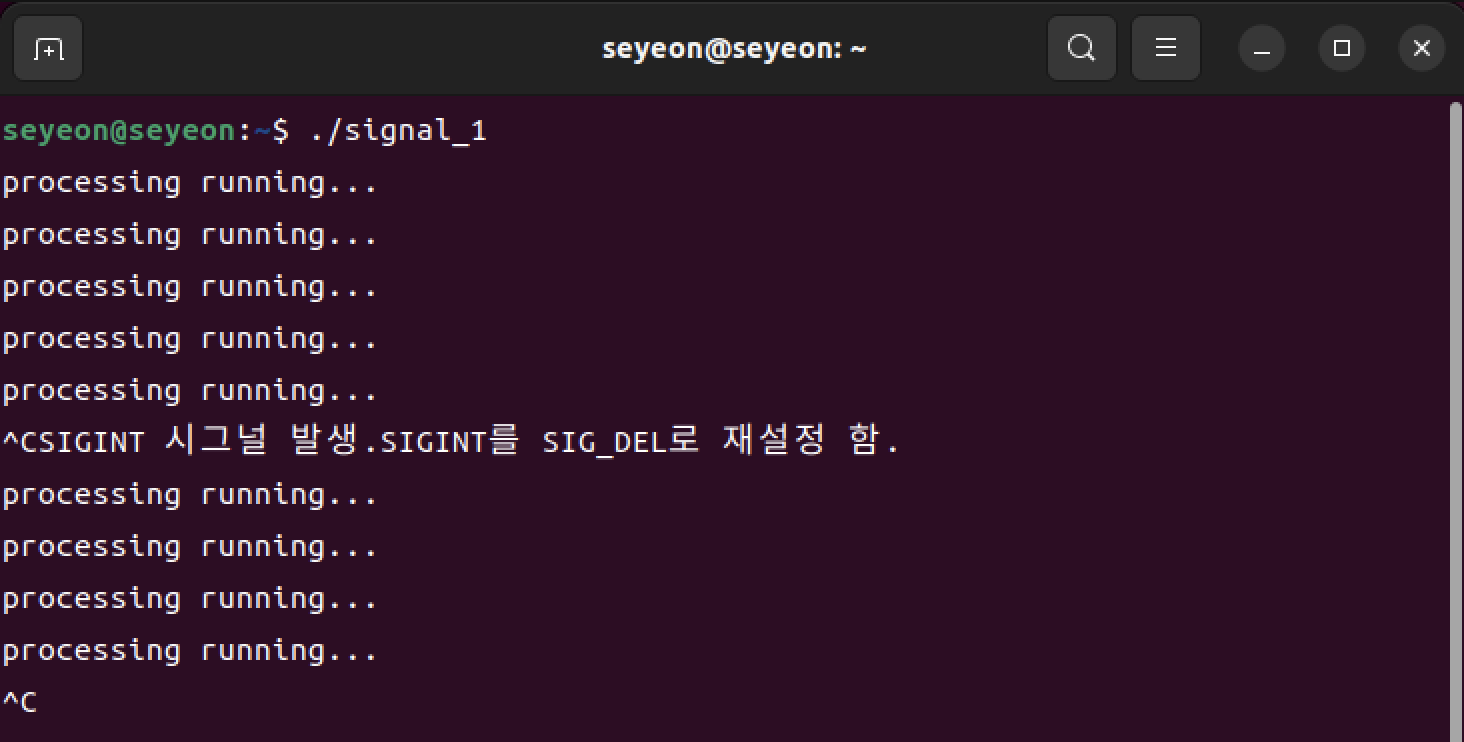
printf("SIGINT를 SIG\_DEL로 재설정 함.\n");

// 이전에 등록된 SIGINT 시그널 핸들러 함수를 func에 저장된 함수로 재설정

signal(SIGINT, func);

}

1. 실행결과



<signal2>

1. 소스코드

**[signal\_2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

// SIGINT, SIGTERM 시그널을 처리하는 핸들러 함수의 선언

static void signal\_handler(int signo);

int main(void){

// SIGINT 시그널에 대한 핸들러 함수 등록 및 오류 처리

if (signal(SIGINT, signal\_handler) == SIG\_ERR) {

fprintf(stderr, "cannot handler SIGINT\n");

exit(EXIT\_FAILURE);

}

// SIGTERM 시그널에 대한 핸들러 함수 등록 및 오류 처리

if (signal(SIGTERM, signal\_handler) == SIG\_ERR) {

fprintf(stderr, "cannot handler SIGTERM\n");

exit(EXIT\_FAILURE);

}

// SIGPROF 시그널의 핸들러 함수를 기본값으로 재설정 및 오류 처리

if (signal(SIGPROF, SIG\_DFL) == SIG\_ERR) {

fprintf(stderr, "cannot reset SIGPROF\n");

exit(EXIT\_FAILURE);

}

// SIGHUP 시그널을 무시하도록 설정 및 오류 처리

if (signal(SIGHUP, SIG\_IGN) == SIG\_ERR) {

fprintf(stderr, "cannot ignore SIGHUP\n");

exit(EXIT\_FAILURE);

}

// 무한 루프에서 pause()를 호출하여 프로그램을 대기 상태로 진입

while(1) {

pause();

}

exit(0);

}

// SIGINT, SIGTERM 시그널을 처리하는 핸들러 함수의 정의

static void signal\_handler(int signo) {

// SIGINT 시그널 처리

if (signo == SIGINT)

printf("caught SIGINT\n");

// SIGTERM 시그널 처리

else if (signo == SIGTERM)

printf("caught SIGTERM\n");

// 예상치 못한 시그널 처리

else {

fprintf(stderr, "unexpected signal\n");

exit(EXIT\_FAILURE);

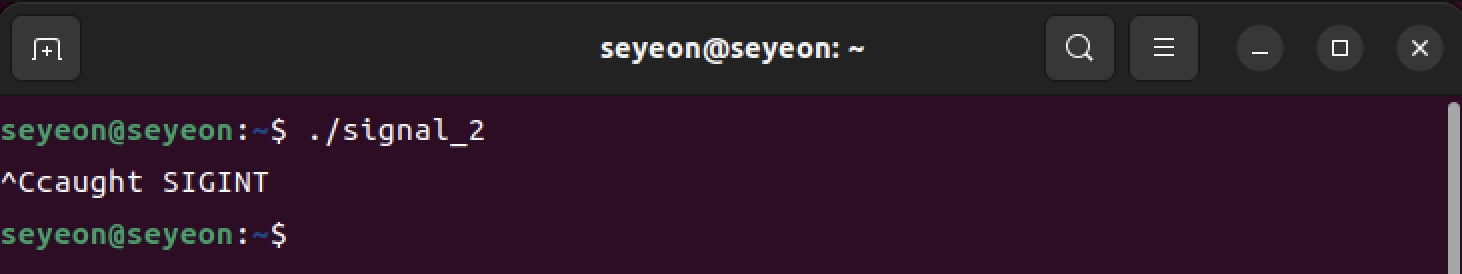
}

// 프로그램 정상 종료

exit(EXIT\_SUCCESS);

}

2. 실행결과



<kill>

1. 소스코드

**[kill\_A.c]**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

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

// 명령행 인자가 2개가 아닌 경우 오류 메시지 출력 후 종료

if (argc != 2){

fprintf(stderr, "usage: %s [Process ID]\n", argv[0]);

exit(1);

}

else

// 명령행 인자로 전달된 프로세스 ID에 SIGKILL 시그널을 보냄

kill(atoi(argv[1]), SIGKILL);

exit(0);

}

**[kill\_B.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(void){

// 무한 루프 시작

while (1) {

// 5초마다 "[OSLAB]"을 출력

printf("\n[OSLAB]");

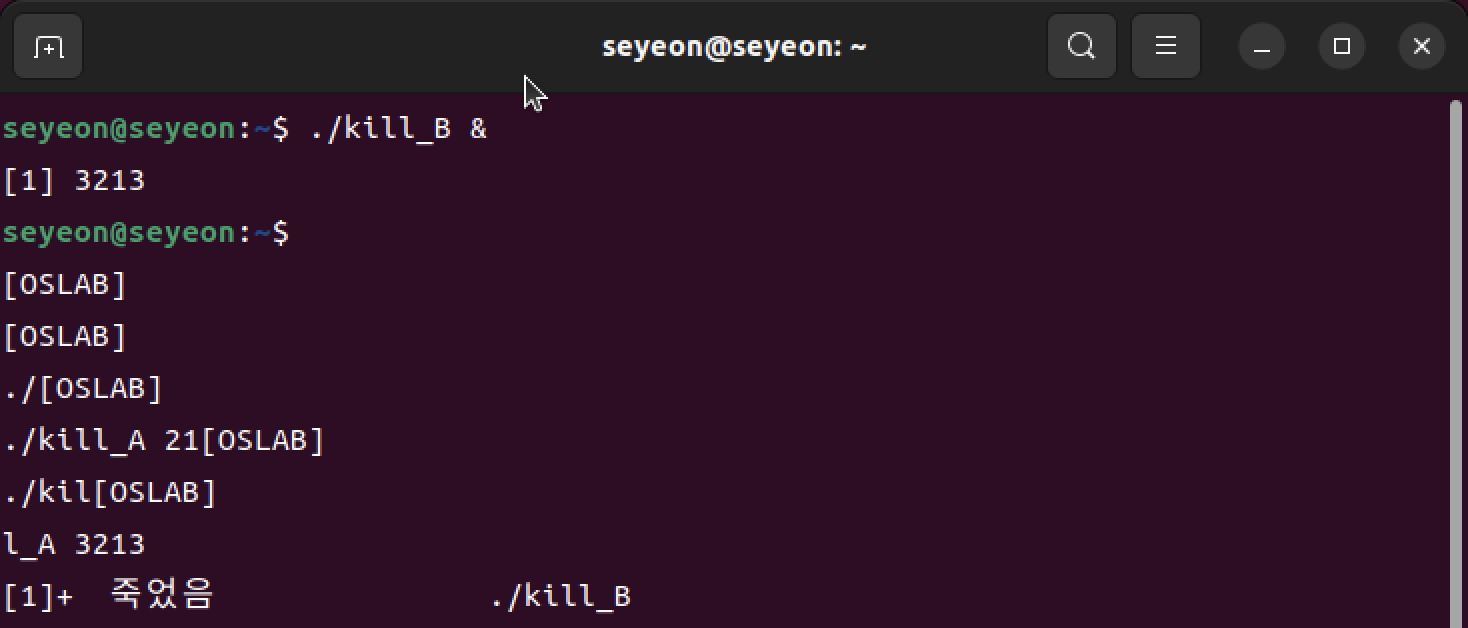
sleep(5); // 5초 대기

}

exit(0); // 프로그램 종료

}

2. 실행결과



<raise>

1. 소스코드

**[raise.c]**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

// SIGINT 시그널을 처리하는 핸들러 함수의 선언

void signal\_handler1(int signo);

// SIGUSR1 시그널을 처리하는 핸들러 함수의 선언

void signal\_handler2(int signo);

int main(void){

// SIGINT 시그널에 대한 핸들러 함수 등록 및 오류 처리

if (signal(SIGINT, signal\_handler1) == SIG\_ERR) {

fprintf(stderr, "cannot hanler SIGINT\n");

exit(EXIT\_FAILURE);

}

// SIGUSR1 시그널에 대한 핸들러 함수 등록 및 오류 처리

if (signal(SIGUSR1, signal\_handler2) == SIG\_ERR) {

fprintf(stderr, "cannot handle SIGUSR1\n");

exit(EXIT\_FAILURE);

}

// SIGINT 시그널 발생

raise(SIGINT);

// SIGUSR1 시그널 발생

raise(SIGUSR1);

// "main return" 출력

printf("main return\n");

exit(0);

}

// SIGINT 시그널을 처리하는 핸들러 함수의 정의

void signal\_handler1(int signo) {

printf("SIGINT 시그널 발생\n");

}

// SIGUSR1 시그널을 처리하는 핸들러 함수의 정의

void signal\_handler2(int signo) {

printf("SIGUSR1 시그널 발생\n");

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<alarm1>

1. 소스코드

**[alarm1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

void signal\_handler(int signo);

int count = 0; // 알람 횟수를 저장하기 위한 변수

int main(void)

{

signal(SIGALRM, signal\_handler); // SIGALRM 시그널에 대한 시그널 핸들러 등록

alarm(1); // 1초 후에 SIGALRM 시그널을 발생시키도록 알람 설정

while(1); // 무한 루프를 통해 프로그램을 실행하는 동안 대기

exit(0); // 프로그램 종료

}

void signal\_handler(int signo){

printf("alarm %d\n", count++); // 알람이 발생할 때마다 알람 횟수를 출력하고 증가시킴

alarm(1); // 1초 후에 다음 알람을 설정

}

2. 실행결과

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<alarm2>

1. 소스코드

**[alarm2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#define LINE\_MAX 2048

static void ssu\_alarm(int signo); // SIGALRM 시그널을 처리할 함수 선언

int main(void)

{

char buf[LINE\_MAX]; // 입력 데이터를 저장할 버퍼

int n; // read() 함수의 반환값을 저장할 변수

// SIGALRM 시그널에 대한 시그널 핸들러를 등록하고, 등록 실패 시 오류 메시지 출력 후 프로그램 종료

if (signal(SIGALRM, ssu\_alarm) == SIG\_ERR) {

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

exit(1);

}

alarm(10); // 10초 후에 SIGALRM 시그널 발생

// 표준 입력으로부터 데이터를 읽어들임, 실패 시 오류 메시지 출력 후 프로그램 종료

if ((n = read(STDIN\_FILENO, buf, LINE\_MAX)) < 0) {

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

exit(1);

}

alarm(0); // SIGALRM 알람 해제

write(STDOUT\_FILENO, buf, n); // 읽어들인 데이터를 표준 출력에 씀

exit(0); // 프로그램 종료

}

// SIGALRM 시그널을 처리하는 함수

static void ssu\_alarm(int signo) {

printf("ssu\_alarm() called!\n");

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<pause>

1. 소스코드

**[pause.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

void ssu\_alarm(int signo); // SIGALRM 시그널을 처리할 함수의 선언

int main(void){

printf("Alarm Setting\n"); // 알람 설정 메시지 출력

signal(SIGALRM, ssu\_alarm); // SIGALRM 시그널을 처리할 함수 등록

alarm(2); // 2초 후에 SIGALRM 시그널 발생

while(1) {

printf("done\n");

pause(); // 시그널이 발생할 때까지 대기

alarm(2); // 2초 후에 다음 알람 설정

}

exit(0);

}

// SIGALRM 시그널을 처리하는 함수

void ssu\_alarm(int signo) {

printf("alarm..!!!\n"); // 알람 메시지 출력

}

2. 실행결과

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

자동 생성된 설명

<sigset>

1. 소스코드

**[sigset.c]**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

int main(void){

sigset\_t set; // 시그널 집합을 저장할 변수

sigemptyset(&set); // 시그널 집합을 초기화

sigaddset(&set, SIGINT); // SIGINT 시그널을 시그널 집합에 추가

// SIGINT가 시그널 집합에 포함되어 있는지 확인하고 결과에 따라 메시지 출력

switch (sigismember(&set, SIGINT))

{

case 1:

printf("SIGINT is included. \n");

break;

case 0 :

printf("SIGINT is not included. \n");

break;

default :

printf("failed to call sigismember() \n");

}

// SIGSYS가 시그널 집합에 포함되어 있는지 확인하고 결과에 따라 메시지 출력

switch(sigismember(&set, SIGSYS))

{

case 1:

printf("SIGSYS is included. \n");

break;

case 0:

printf("SIGSYS is not included. \n");

break;

default :

printf("failed to call sigismember() \n");

}

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<sigprocmask>

1. 소스코드

**[sigprocmask.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int main(void)

{

sigset\_t sig\_set; // 시그널 집합을 저장할 변수

int count;

sigemptyset(&sig\_set); // 시그널 집합을 초기화

sigaddset(&sig\_set, SIGINT); // 시그널 집합에 SIGINT를 추가

sigprocmask(SIG\_BLOCK, &sig\_set, NULL); // 시그널 집합에 포함된 시그널을 블록

// 3부터 카운트를 시작하여 0이 될 때까지 반복

for(count = 3; 0 < count; count--) {

printf("count %d\n", count); // 카운트 출력

sleep(1); // 1초 쉬기

}

printf("Ctrl-C에 대한 블록을 해제\n");

sigprocmask(SIG\_UNBLOCK, &sig\_set, NULL); // 시그널 집합에 포함된 시그널 블록 해제

printf("count중 Ctrl-C입력하면 이 문장은 출력 되지 않음.\n");

while(1); // 무한 루프로 프로그램이 종료되지 않도록 함

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<sigpending>

1. 소스코드

**[sigpending.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int main(void)

{

sigset\_t pendingset; // 대기 중인 시그널 집합을 저장할 변수

sigset\_t sig\_set; // 시그널 집합을 저장할 변수

int count = 0; // 카운트를 위한 변수

sigfillset(&sig\_set); // 시그널 집합을 모두 채움 (모든 시그널을 포함)

sigprocmask(SIG\_SETMASK, &sig\_set, NULL); // 현재 시그널 마스크를 시그널 집합으로 설정

// 무한 루프

while (1) {

printf("count: %d\n", count++); // 카운트 출력

sleep(1); // 1초 쉬기

// 대기 중인 시그널 확인

if (sigpending(&pendingset) == 0) {

// SIGINT가 대기 중인지 확인

if (sigismember(&pendingset, SIGINT)) {

printf("SIGINT가 블록되어 대기 중. 무한 루프를 종료.\n"); // SIGINT가 블록되어 있으면 메시지 출력

break; // 무한 루프 종료

}

}

}

exit(0); // 프로그램 종료

}

2. 실행결과

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

자동 생성된 설명