컴퓨터학부 20201841 박세연

<sigaction1>

1. 소스코드

**[sigaction\_1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

// 시그널 핸들러 함수 정의

void ssu\_signal\_handler(int signo) {

// 시그널 핸들러가 호출되었음을 알리는 메시지 출력

printf("ssu\_signal\_handler control\n");

}

int main(void) {

struct sigaction sig\_act; // 시그널 액션 구조체 정의

sigset\_t sig\_set; // 시그널 집합 정의

// sig\_act 구조체 초기화

sigemptyset(&sig\_act.sa\_mask); // sa\_mask를 비운다

sig\_act.sa\_flags = 0; // 플래그 초기화

sig\_act.sa\_handler = ssu\_signal\_handler; // 시그널 핸들러 설정

// SIGUSR1 시그널에 대해 sig\_act를 설정

sigaction(SIGUSR1, &sig\_act, NULL);

// 첫 번째 kill() 호출 전 메시지 출력

printf("before first kill()\n");

// 현재 프로세스에 SIGUSR1 시그널을 보낸다

kill(getpid(), SIGUSR1);

// sig\_set 집합 초기화

sigemptyset(&sig\_set);

// sig\_set에 SIGUSR1 시그널 추가

sigaddset(&sig\_set, SIGUSR1);

// SIGUSR1 시그널을 블록(차단)하도록 시그널 마스크 설정

sigprocmask(SIG\_SETMASK, &sig\_set, NULL);

// 두 번째 kill() 호출 전 메시지 출력

printf("before second kill()\n");

// 현재 프로세스에 SIGUSR1 시그널을 보낸다 (하지만 차단되어 있음)

kill(getpid(), SIGUSR1);

// 두 번째 kill() 호출 후 메시지 출력

printf("after second kill()\n");

// 프로그램 정상 종료

exit(0);

}

1. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<sigaction2>

1. 소스코드

**[sigaction\_2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

// 시그널이 대기 중인지 확인하는 함수

void ssu\_check\_pending(int signo, char \*signame);

// 시그널 핸들러 함수 정의

void ssu\_signal\_handler(int signo);

int main(void) {

struct sigaction sig\_act; // 시그널 액션 구조체 정의

sigset\_t sig\_set; // 시그널 집합 정의

// sig\_act 구조체 초기화

sigemptyset(&sig\_act.sa\_mask); // sa\_mask를 비운다

sig\_act.sa\_flags = 0; // 플래그 초기화

sig\_act.sa\_handler = ssu\_signal\_handler; // 시그널 핸들러 설정

// SIGUSR1 시그널에 대해 sig\_act를 설정, 실패 시 에러 메시지 출력 후 종료

if (sigaction(SIGUSR1, &sig\_act, NULL) != 0) {

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

exit(1);

} else {

// sig\_set 집합 초기화 및 SIGUSR1 시그널 추가

sigemptyset(&sig\_set);

sigaddset(&sig\_set, SIGUSR1);

// SIGUSR1 시그널을 블록(차단)하도록 시그널 마스크 설정, 실패 시 에러 메시지 출력 후 종료

if (sigprocmask(SIG\_SETMASK, &sig\_set, NULL) != 0) {

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

exit(1);

} else {

// SIGUSR1 시그널이 블록되었음을 알리는 메시지 출력

printf("SIGUSR1 signals are now blocked\n");

// 현재 프로세스에 SIGUSR1 시그널을 보낸다

kill(getpid(), SIGUSR1);

printf("after kill()\n");

// SIGUSR1 시그널이 대기 중인지 확인

ssu\_check\_pending(SIGUSR1, "SIGUSR1");

// 시그널 집합 초기화 및 시그널 마스크 해제

sigemptyset(&sig\_set);

sigprocmask(SIG\_SETMASK, &sig\_set, NULL);

// SIGUSR1 시그널이 더 이상 블록되지 않음을 알리는 메시지 출력

printf("SIGUSR1 signals are no longer blocked\n");

// SIGUSR1 시그널이 대기 중인지 다시 확인

ssu\_check\_pending(SIGUSR1, "SIGUSR1");

}

}

exit(0);

}

// 시그널이 대기 중인지 확인하는 함수

void ssu\_check\_pending(int signo, char \*signame) {

sigset\_t sig\_set; // 시그널 집합 정의

// 현재 대기 중인 시그널들을 sig\_set에 저장

if (sigpending(&sig\_set) != 0)

printf("sigpending() error\n");

else if (sigismember(&sig\_set, signo))

// sig\_set에 signo 시그널이 포함되어 있으면 대기 중임을 알리는 메시지 출력

printf("a %s signal is pending\n", signame);

else

// sig\_set에 signo 시그널이 포함되어 있지 않으면 대기 중이지 않음을 알리는 메시지 출력

printf("%s signals are not pending\n", signame);

}

// 시그널 핸들러 함수 정의

void ssu\_signal\_handler(int signo) {

printf("in ssu\_signal\_handler function\n");

}

2. 실행결과

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<sigaction3>

1. 소스코드

**[sigaction\_3.c]**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

// SIGINT 시그널 핸들러 함수

static void ssu\_signal\_handler1(int signo);

// SIGQUIT 시그널 핸들러 함수

static void ssu\_signal\_handler2(int signo);

int main(void) {

struct sigaction act\_int, act\_quit; // 시그널 액션 구조체 정의

// SIGINT 시그널 핸들러 설정

act\_int.sa\_handler = ssu\_signal\_handler1; // SIGINT 시그널에 대한 핸들러 설정

sigemptyset(&act\_int.sa\_mask); // 시그널 마스크 초기화

sigaddset(&act\_int.sa\_mask, SIGQUIT); // 시그널 마스크에 SIGQUIT 추가 (SIGINT 핸들러가 실행 중일 때 SIGQUIT 블록)

act\_quit.sa\_flags = 0; // 플래그 초기화

// SIGINT 시그널에 대한 액션 설정, 실패 시 에러 메시지 출력 후 종료

if (sigaction(SIGINT, &act\_int, NULL) < 0) {

fprintf(stderr, "sigaction(SIGINT) error\n");

exit(1);

}

// SIGQUIT 시그널 핸들러 설정

act\_quit.sa\_handler = ssu\_signal\_handler2; // SIGQUIT 시그널에 대한 핸들러 설정

sigemptyset(&act\_quit.sa\_mask); // 시그널 마스크 초기화

sigaddset(&act\_quit.sa\_mask, SIGINT); // 시그널 마스크에 SIGINT 추가 (SIGQUIT 핸들러가 실행 중일 때 SIGINT 블록)

act\_int.sa\_flags = 0; // 플래그 초기화

// SIGQUIT 시그널에 대한 액션 설정, 실패 시 에러 메시지 출력 후 종료

if (sigaction(SIGQUIT, &act\_quit, NULL) < 0) {

fprintf(stderr, "sigaction(SIGQUIT) error\n");

exit(1);

}

// 시그널이 발생할 때까지 대기

pause();

exit(0);

}

// SIGINT 시그널 핸들러 함수 정의

static void ssu\_signal\_handler1(int signo) {

printf("Signal handler of SIGINT : %d\n", signo);

printf("SIGQUIT signal is blocked : %d\n", signo);

printf("sleeping 3 sec\n");

sleep(3); // 3초 대기

printf("Signal handler of SIGINT ended\n");

}

// SIGQUIT 시그널 핸들러 함수 정의

static void ssu\_signal\_handler2(int signo) {

printf("Signal handler of SIGQUIT : %d\n", signo);

printf("SIGINT signal is blocked : %d\n", signo);

printf("sleeping 3 sec\n");

sleep(3); // 3초 대기

printf("Signal handler of SIGQUIT ended\n");

}

2. 실행결과

텍스트, 스크린샷, 멀티미디어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<sigsetjmp1>

1. 소스코드

**[sigsetjmp\_1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <setjmp.h>

// 시그널 핸들러 함수 선언

void ssu\_signal\_handler(int signo);

// 전역 점프 버퍼 선언

jmp\_buf jump\_buffer;

int main(void) {

// SIGINT 시그널에 대해 ssu\_signal\_handler를 핸들러로 설정

signal(SIGINT, ssu\_signal\_handler);

// 무한 루프

while (1) {

// setjmp로 점프 지점 설정

if (setjmp(jump\_buffer) == 0) {

// 초기 호출 시 메시지 출력 및 pause로 대기

printf("Hit Ctrl-c at anytime ... \n");

pause();

}

}

exit(0);

}

// 시그널 핸들러 함수 정의

void ssu\_signal\_handler(int signo) {

char character;

// SIGINT 시그널을 일시적으로 무시

signal(signo, SIG\_IGN);

printf("Did you hit Ctrl-c?\n" "Do you really want to quit? [y/n] ");

// 사용자 입력 받기

character = getchar();

// 사용자가 'y' 또는 'Y'를 입력하면 프로그램 종료

if (character == 'y' || character == 'Y')

exit(0);

else {

// 그렇지 않으면 SIGINT 시그널 핸들러를 다시 설정하고 longjmp로 점프

signal(SIGINT, ssu\_signal\_handler);

longjmp(jump\_buffer, 1);

}

}

2. 실행결과

텍스트, 스크린샷, 멀티미디어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<sigsetjmp2>

1. 소스코드

**[sigsetjmp\_2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <setjmp.h>

#include <signal.h>

#include <time.h>

// 함수 프로토타입 선언

static void ssu\_alarm(int signo);

static void ssu\_func(int signo);

void ssu\_mask(const char \*str);

// 전역 변수

static volatile sig\_atomic\_t can\_jump; // 점프가 가능한지 확인하는 플래그

static sigjmp\_buf jump\_buf; // longjmp를 위해 호출 환경을 저장하는 버퍼

int main(void) {

// SIGUSR1 신호에 대한 핸들러 설정

if (signal(SIGUSR1, ssu\_func) == SIG\_ERR) {

fprintf(stderr, "SIGUSR1 error");

exit(1);

}

// SIGALRM 신호에 대한 핸들러 설정

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

fprintf(stderr, "SIGALRM error");

exit(1);

}

// 초기 신호 마스크를 출력

ssu\_mask("starting main: ");

// 나중에 longjmp로 돌아올 호출 환경을 저장

if (sigsetjmp(jump\_buf, 1)) {

// siglongjmp로 돌아올 때, 신호 마스크를 출력하고 종료

ssu\_mask("ending main: ");

exit(0);

}

can\_jump = 1; // 점프가 가능함을 나타내는 플래그 설정

// 신호를 기다리는 무한 루프

while (1)

pause();

exit(0);

}

// 현재 신호 마스크를 출력하는 함수

void ssu\_mask(const char \*str) {

sigset\_t sig\_set;

int err\_num;

err\_num = errno; // 현재 errno를 저장

// 현재 신호 마스크를 가져옴

if (sigprocmask(0, NULL, &sig\_set) < 0) {

printf("sigprocmask() error");

exit(1);

}

printf("%s", str);

// 어떤 신호가 차단되어 있는지 확인하고 출력

if (sigismember(&sig\_set, SIGINT))

printf("SIGINT ");

if (sigismember(&sig\_set, SIGQUIT))

printf("SIGQUIT ");

if (sigismember(&sig\_set, SIGUSR1))

printf("SIGUSR1 ");

if (sigismember(&sig\_set, SIGALRM))

printf("SIGALRM ");

printf("\n");

errno = err\_num; // errno를 복원

}

// SIGUSR1 신호에 대한 핸들러

static void ssu\_func(int signo){

time\_t start\_time;

// 점프가 허용되지 않은 경우 반환

if (can\_jump == 0)

return;

// 신호 마스크를 출력하고 알람 설정

ssu\_mask("starting ssu\_func: ");

alarm(3);

start\_time = time(NULL);

// 5초 동안 바쁜 대기

while (1)

if (time(NULL) > start\_time + 5)

break;

// 신호 마스크를 출력하고, 플래그를 재설정하며, 저장된 환경으로 점프

ssu\_mask("ending ssu\_func: ");

can\_jump = 0;

siglongjmp(jump\_buf, 1);

}

// SIGALRM 신호에 대한 핸들러

static void ssu\_alarm(int signo){

ssu\_mask("in ssu\_alarm");

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어이(가) 표시된 사진

자동 생성된 설명

<sigsetjmp3>

1. 소스코드

**[sigsetjmp\_3.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <setjmp.h>

#include <signal.h>

// 함수 프로토타입 선언

static void ssu\_signal\_handler1(int signo);

static void ssu\_signal\_handler2(int signo);

// 전역 변수로 점프 버퍼 선언

sigjmp\_buf jmp\_buf1;

sigjmp\_buf jmp\_buf2;

int main(void){

struct sigaction act\_sig1;

struct sigaction act\_sig2;

int i, ret;

// 현재 프로세스 ID 출력

printf("My PID is %d\n", getpid());

// 첫 번째 점프 버퍼 설정

ret = sigsetjmp(jmp\_buf1, 1);

if (ret == 0) {

// sigsetjmp에서 처음 호출된 경우

act\_sig1.sa\_handler = ssu\_signal\_handler1;

sigaction(SIGINT, &act\_sig1, NULL); // SIGINT에 대한 핸들러 설정

}

else if (ret == 3){

// SIGINT로 인해 siglongjmp로 돌아온 경우

printf("----------------\n");

}

printf("Starting\n");

// 두 번째 점프 버퍼 설정

sigsetjmp(jmp\_buf2, 2);

act\_sig2.sa\_handler = ssu\_signal\_handler2;

sigaction(SIGUSR1, &act\_sig2, NULL); // SIGUSR1에 대한 핸들러 설정

// 20번 반복하면서 1초마다 i를 출력

for(i = 0; i < 20; i++) {

printf("i = %d\n", i);

sleep(1);

}

exit(0);

}

// SIGINT 신호에 대한 핸들러

static void ssu\_signal\_handler1(int signo) {

fprintf(stderr, "\nInterrupted\n"); // 인터럽트 메시지 출력

siglongjmp(jmp\_buf1, 3); // 첫 번째 점프 버퍼로 점프, 3을 반환값으로 설정

}

// SIGUSR1 신호에 대한 핸들러

static void ssu\_signal\_handler2(int signo) {

fprintf(stderr, "\nSIGUSR1\n"); // SIGUSR1 메시지 출력

siglongjmp(jmp\_buf2, 2); // 두 번째 점프 버퍼로 점프, 2를 반환값으로 설정

}

2. 실행결과

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

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<sigsuspend1>

1. 소스코드

**[sigsuspend\_1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

int main(void) {

sigset\_t old\_set;

sigset\_t sig\_set;

// 빈 신호 집합 초기화

sigemptyset(&sig\_set);

// SIGINT 신호를 신호 집합에 추가

sigaddset(&sig\_set, SIGINT);

// 현재 신호 마스크에 SIGINT 신호를 블록하여 설정, 이전 신호 마스크를 old\_set에 저장

sigprocmask(SIG\_BLOCK, &sig\_set, &old\_set);

// old\_set에 의해 지정된 신호가 수신될 때까지 대기 (SIGINT 블록 해제)

sigsuspend(&old\_set);

exit(0);

}

2. 실행결과

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<sigsuspend2>

1. 소스코드

**[sigset.c]**

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <signal.h>

// SIGUSR1 신호 핸들러

static void ssu\_func(int signo);

// 현재의 시그널 마스크를 출력하는 함수

void ssu\_print\_mask(const char \*str);

int main(void)

{

// 변수 선언

sigset\_t new\_mask, old\_mask, wait\_mask;

// 프로그램 시작 시 시그널 마스크 출력

ssu\_print\_mask("program start: ");

// SIGINT에 대한 시그널 핸들러 설정

if (signal(SIGINT, ssu\_func) == SIG\_ERR) {

fprintf(stderr, "signal(SIGINT) error\n");

exit(1);

}

// wait\_mask를 초기화하고 SIGUSR1을 추가

sigemptyset(&wait\_mask);

sigaddset(&wait\_mask, SIGUSR1);

// new\_mask를 초기화하고 SIGINT를 추가하여 블록

sigemptyset(&new\_mask);

sigaddset(&new\_mask, SIGINT);

// 현재 시그널 마스크를 new\_mask로 변경하고 이전 마스크(old\_mask)를 저장

if (sigprocmask(SIG\_BLOCK, &new\_mask, &old\_mask) < 0) {

fprintf(stderr, "SIG\_BLOCK() error\n");

exit(1);

}

// 크리티컬 영역 진입 시 시그널 마스크 출력

ssu\_print\_mask("in critical region: ");

// SIGUSR1 시그널이 올 때까지 대기

if (sigsuspend(&wait\_mask) != -1) {

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

exit(1);

}

// sigsuspend에서 리턴 후 시그널 마스크 출력

ssu\_print\_mask("after return from sigsuspend: ");

// 이전 시그널 마스크(old\_mask)로 변경

if (sigprocmask(SIG\_SETMASK, &old\_mask, NULL) < 0) {

fprintf(stderr, "SIG\_SETMASK() error\n");

exit(1);

}

// 프로그램 종료 시 시그널 마스크 출력

ssu\_print\_mask("program exit: ");

exit(0);

}

// 현재의 시그널 마스크를 출력하는 함수

void ssu\_print\_mask(const char \*str) {

sigset\_t sig\_set;

int err\_num;

err\_num = errno;

// 현재 시그널 마스크를 가져와서 출력

if (sigprocmask(0, NULL, &sig\_set) < 0) {

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

exit(1);

}

printf("%s", str);

if (sigismember(&sig\_set, SIGINT))

printf("SIGINT ");

if (sigismember(&sig\_set, SIGQUIT))

printf("SIGQUIT ");

if (sigismember(&sig\_set, SIGUSR1))

printf("SIGUSR1 ");

if (sigismember(&sig\_set, SIGALRM))

printf("SIGALRM ");

printf("\n");

errno = err\_num;

}

// SIGUSR1 시그널 핸들러

static void ssu\_func(int signo) {

ssu\_print\_mask("\nin ssu\_func: ");

}

2. 실행결과

텍스트, 스크린샷, 멀티미디어, 폰트이(가) 표시된 사진

자동 생성된 설명

<sigsuspend3>

1. 소스코드

**[sigprocmask.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <time.h>

// SIGALRM 시그널 핸들러 함수

void ssu\_signal\_handler(int signo);

// 현재 시간을 출력하는 함수

void ssu\_timestamp(char \*str);

int main(void)

{

struct sigaction sig\_act; // 시그널 액션 구조체

sigset\_t blk\_set; // 블록할 시그널 집합

// 블록할 시그널 집합 설정 (모든 시그널을 포함)

sigfillset(&blk\_set);

// SIGALRM은 블록하지 않음

sigdelset(&blk\_set, SIGALRM);

// 시그널 액션 구조체 초기화

sigemptyset(&sig\_act.sa\_mask);

sig\_act.sa\_flags = 0;

sig\_act.sa\_handler = ssu\_signal\_handler;

// SIGALRM 시그널에 대한 핸들러 등록

sigaction(SIGALRM, &sig\_act, NULL);

// 현재 시간 출력

ssu\_timestamp("before sigsuspend()");

// 5초 후에 SIGALRM 시그널이 발생하도록 설정

alarm(5);

// SIGALRM을 제외한 다른 모든 시그널을 블록하고 대기

sigsuspend(&blk\_set);

// sigsuspend 종료 후 현재 시간 출력

ssu\_timestamp("after sigsuspend()");

exit(0);

}

// SIGALRM 시그널 핸들러 함수

void ssu\_signal\_handler(int signo) {

printf("in ssu\_signal\_handler() function\n");

}

// 현재 시간을 출력하는 함수

void ssu\_timestamp(char \*str) {

time\_t time\_val;

// 현재 시간을 구하고 출력

time(&time\_val);

printf("%s the time is %s\n", str, ctime(&time\_val));

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<abort>

1. 소스코드

**[abort.c]**

#include <stdio.h>

#include <stdlib.h>

int main(void) {

// 프로그램이 강제 종료될 것임을 알리는 메시지를 출력

printf("abort terminate this program\n");

// 프로그램을 즉시 강제 종료. 이후의 코드는 실행되지 않음

abort();

// 이 줄은 절대 실행되지 않음

printf("this line is never reached\n");

// 정상적인 프로그램 종료. 그러나 이 코드는 실행되지 않음

exit(0);

}

2. 실행결과

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<sleep>

1. 소스코드

**[sleep.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

// 현재 시간을 출력하는 함수

void ssu\_timestamp(char \*str);

int main(void)

{

unsigned int ret; // sleep 함수의 반환값을 저장할 변수

// sleep 호출 전에 현재 시간 출력

ssu\_timestamp("before sleep()");

// sleep 함수를 호출하여 10초 동안 프로세스를 대기

ret = sleep(10);

// sleep 호출 후에 현재 시간 출력

ssu\_timestamp("after sleep()");

// sleep 함수의 반환값 출력

printf("sleep() returned %d\n", ret);

exit(0);

}

// 현재 시간을 출력하는 함수

void ssu\_timestamp(char \*str) {

time\_t time\_val;

// 현재 시간을 구하고 출력

time(&time\_val);

printf("%s the time is %s\n", str, ctime(&time\_val));

}

2. 실행결과

텍스트, 스크린샷, 멀티미디어, 폰트이(가) 표시된 사진

자동 생성된 설명