강원혁신플랫폼

리눅스프로그래밍

시그널

Linux에서 프로세스와 통신하고 동작을 관리하는 데 사용되는
소프트웨어 인터럽트를 무엇이라고 하나요?

시그널(Signal)

학습목표

학습 내용

1. 리눅스(Linux)의 주요 시그널
2. 시그널의 주요 함수

학습 목표

- 리눅스(Linux)의 주요 시그널에 대해 설명할 수 있다.
- 시그널의 주요 함수를 파악할 수 있다.

리눅스(Linux)의 주요 시그널

Signal 시그널

Linux에서 시그널(signal)은 프로세스와 통신하고
동작을 관리하는 데 사용되는 소프트웨어 인터럽트
커널, 운영 체제 또는 기타 프로세스에 의해 프로세스로 전송

시그널(Signal)

목적

시그널

- 프로세스 종료 요청
- 예외 조건 처리
- 이벤트에 대한 프로세스 알림

Signal & 응답성

응답성(Responsiveness)

- 시그널은 응답성과 관련하여 중요한 역할
- 시스템의 안정성과 응답성을 유지하는 데 도움
- 다른 프로세스나 시스템에 영향을 미치는 오류를 최소화

Signal & 응답성

응답성(Responsiveness)

시스템 부하

프로세서, 메모리, 디스크, 네트워크 등의
시스템 자원 사용량이 얼마나 높은지에 따라
응답성이 달라질 수 있음

프로세스 우선순위

운영 체제는 실행되는 프로세스들에게 우선순위를 부여하여
리소스 할당을 조정. 우선순위가 높은 프로세스는
더 많은 자원을 할당받고, 더 빠르게 실행될 수 있음

Signal & 응답성

응답성(Responsiveness)

01 02 03 04

입출력 처리

- 디스크 또는 네트워크와 같은 입출력 작업은 일반적으로 CPU 작업보다 느림
- 입출력 작업이 많은 시스템에서는 해당 작업들이 응답성에 영향을 미칠 수 있음

Signal & 응답성

응답성(Responsiveness)

01 02 03 04

프로세스 스케줄링

- 리눅스는 여러 프로세스들을 실행하기 위해 스케줄러를 사용
- 스케줄러는 프로세스들이 얼마나 오래 실행될지,
  언제 실행될지를 결정
- 효율적인 스케줄링은 응답성을 향상

Signal & 응답성

응답성(Responsiveness)

01 02 03 04

하드웨어 성능

- 하드웨어의 성능도 응답성에 영향
- 높은 성능의 CPU, 빠른 메모리, 고속 디스크 등이 응답성을 향상시키는 데 도움

Signal & 응답성

응답성(Responsiveness)

01 02 03 04

프로그램 설계

- 프로그램의 효율적인 설계되어 있는지도 응답성에 영향

소프트웨어 이벤트 처리와 시그널

소프트웨어 이벤트 처리(Event Handling)

- 사용자 입력, 외부 장치 상태 변화, 또는 다른 소프트웨어 컴포넌트로부터 발생하는 이벤트에 대해 대응하는 메커니즘
- 이벤트 처리는 사용자 인터페이스 (UI)에서 매우 중요. 사용자 입력에 반응하여 적절한 동작을 수행하는 것

시그널(Signal)

- 시그널은 운영 체제나 다른 프로세스로부터 프로세스에게 특정 이벤트를 알리는 메커니즘
- 시그널은 비동기적으로 발생. 시그널은 일반적으로 예외 상황을 처리하거나 프로세스 동작을 변경하는 데 사용

리얼타임

Real-Time Operating System, RTOS 리눅스

- 실시간 시스템에서 사용되는 운영체제
- 실시간 시스템은 높은 신뢰성과 정확성을 요구
- 실시간 작업들을 정확한 시간 안에 처리하는 데 중점을 둠
- 실시간 제어 시스템, 자동차 제어, 항공기 제어, 로봇 시스템 등

하드 리얼타임(Hard Real-Time) 응답성

- 모든 작업이 정해진 시간 안에 완료
- 정해진 시간 안에 완료되지 않으면 치명적인 결과 초래

소프트 리얼타임(Soft Real-Time) 응답성

- 작업이 시간 내에 완료되지 않더라도 큰 문제가 발생하지 않음

리눅스의 주요 시그널

시그널명	번호	내용	비고
SIGHUP	1	터미널의 연결이 끊어질 때 전달(HangUP)	POSIX
SIGINT	2	인터럽트(INTerrupt)를 위해 사용자가 INTR 문자(Ctrl + C)를 입력했을 때 전달	ANSI
SIGQUIT	3	터미널에서 사용자가 QUIT 키(Ctrl + \)를 누르면 전달(QUIT)	POSIX
SIGILL	4	비정상적인 기계어 명령어를 실행하는 경우에 발생(ILLegal instruction)	ANSI
SIGTRAP	5	디버그에서 트레이스 트랩(Trace tRAP)을 위해 사용한 시그널	POSIX
SIGABRT	6	abort( ) 함수나 프로세스가 비정상 종료로 인해서 발생되는 에러 의미(ABoRt)	ANSI
SIGIOT	6	SIGABRT와 유사한 작업을 수행시 발생하는 시그널(IOT trap)	4.2 BSD
SIGBUS	7	유용하지 않은 포인터가 비참조되어 BUS 에러가 발생했을 때 발생(BUS 에러)	4.2 BSD

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

리눅스의 주요 시그널

시그널명	번호	내용	비고
SIGFPE	8	나눗셈에서 0으로 나누는 등의 심각한 산술적 에러 보고(Floating-Point Exception)	ANSI
SIGKILL	9	즉각적인 프로그램 종료를 일으키기 위해 사용(KILL, unblockable)	POSIX
SIGUSR1	10	응용 프로그램에서 사용자가 정의하여 사용할 수 있는 시그널1 (USeR-defined signal 1)	POSIX
SIGSEGV	11	할당된 메모리의 범위를 벗어나는 곳에서 읽거나, 쓰기를 시도할 때 발생 (SEGmentation Violation)	ANSI
SIGUSR2	12	응용 프로그램에서 사용자가 정의하여 사용할 수 있는 시그널2 (USeR-defined signal 2)	POSIX
SIGPIPE	13	파이프로 연결된 프로세스는 이미 종료되었지만 파이프로 계속 데이터를 전송할 때 발생(broken PIPE)	POSIX
SIGALRM	14	alarm 함수에 의해 설정된 타이머에 의해 발생(ALaRM clock)	POSIX
SIGTERM	15	kill 명령어 의해서 프로그램을 종료하는 데 사용하는 시그널(TERMination)	ANSI

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

리눅스의 주요 시그널

시그널명	번호	내용	비고
SIGSTKFLT	16	보조 프로세서에서 스택 오류(STacK FauLT)가 발생했다는 것을 의미	
SIGCLD	SIGCHLD	(SIGCHLD과 동일)	System V
SIGCHLD	17	exit( )나 wait( ) 함수에 의해서 자식 프로세스가 종료하거나 정지한 경우 부모 프로세스에게 전달(CHiLD status has changed)	POSIX
SIGCONT	18	정지된 프로세스의 실행을 재개하는 작업 제어 시그널(CONTinue)	POSIX
SIGSTOP	19	프로세스를 정지시키는 작업 제어 시그널(STOP, unblockable)	POSIX
SIGTSTP	20	suspend 키(Ctrl + z)를 누르면 발생하는 정지 시그널(Terminal/keyboard SToP)	POSIX
SIGTTIN	21	백그라운드에서 작업하는 프로세스가 터미널에 읽기를 시도하면 발생 (Background read from TTy INput)	POSIX
SIGTTOU	22	백그라운드에서 작업하는 프로세스가 터미널에 출력을 시도하거나 그 터미널 모드를 설정하려 시도할 때 발생(Background write to TTy OUtput)	POSIX

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

리눅스의 주요 시그널

시그널명	번호	내용	비고
SIGURG	23	소켓에서 긴급(urgent) 메시지가 도착하면 발생(URGent condition on socket)	4.2 BSD
SIGXCPU	24	CPU 제한을 초과할 때 발생(CPU limit eXceeded)	4.2 BSD
SIGXFSZ	25	시스템에서 지원하는 파일의 크기를 넘을 때 발생(File SiZe limit eXceeded)	4.2 BSD
SIGVTALRM	26	현재 프로세스에 의해 사용된 CPU 시간을 계산하는 타이머의 경과를 알려줌 (VirTual ALaRM clock)	4.2 BSD
SIGPROF	27	프로파일 타이머가 끝날 때 발생(PROFiling alarm clock)	4.2 BSD
SIGWINCH	28	X 윈도우상에서 윈도우(터미널)의 크기가 변경될 때 발생(WINdow size CHange)	4.2 BSD, Sun
SIGPOLL	SIGIO	입출력에서 폴링이 가능해지면 발생(POLLable event occurred)	System V
SIGIO	29	입출력이 가능해질 때 발생(I/O now possible)	4.2 BSD

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

리눅스의 주요 시그널

시그널명	번호	내용	비고
SIGPWR	30	전원 공급에 문제가 발생했을 때 호출(PoWeR failure restart)	System V
SIGSYS	31	시스템 호출에 잘못된 인자가 전달될 때 발생(Bad SYStem call)	
SIGUNUSED	31	사용되지 않는 시그널(UNUSED)	

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

시그널의 주요 함수

시그널의 주요 함수

함수	내용
signal()	특정 신호가 수신될 때 프로세스의 동작을 정의
pause()	신호를 받을 때까지 호출 프로세스를 휴면 상태로
sigaction()	특정 신호와 관련된 작업을 검사하고 수정
sigprocmask()	신호 이름 및 해당 설명 메시지를 포함하여 신호 메시지를 표준 오류로 인쇄하는 데 사용

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

시그널의 주요 함수

함수	내용
strsignal()	주어진 신호 번호를 설명하는 문자열을 반환
alarm()	프로세스의 알람 시계설정
kill()	프로세스 또는 프로세스 그룹에 신호를 보냄
raise()	호출 프로세스 자체에 신호를 보냄

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

미리 정의된 시그널 함수의 포인터 값

상수	내용	비고
SIG_DFL	기본 설정(Default)으로 처리함	((void (*)( )) 0)
SIG_IGN	지정된 설정 방법을 무시(Ignore)함	((void (*)( )) 1)
SIG_ERR	에러가 발생함	Error

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

struct sigaction sa_flags의 값


```
struct sigaction {
        void     (*sa_handler)(int);
        void     (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t   sa_mask;
        int        sa_flags;
        void     (*sa_restorer)(void);
    };
```

struct sigaction sa_flags의 값

상수	내용	비고
SA_NOCLDSTOP	시그널 번호(signum)가 SIGCHILD일 경우, 자식 프로세스가 멈추었을 때, 부모 프로세스에 SIGCHILD가 전달되지 않음	
SA_NOCLDWAIT	시그널 번호(signum)가 SIGCHLD일 경우, 자식 프로세스가 좀비가 되지 않도록 함	리눅스 커널 2.6 이상
SA_NODEFER	시그널을 처리하는 동안에 전달되는 시그널은 블록되지 않음	
SA_ONSTACK	sigaltstack( ) 함수에 의해서 제공되는 시그널 핸들러를 호출함	
SA_RESETHAND	시그널을 받으면 설정된 행동을 취하고 시스템 기본 설정인 SIG_DFL로 재설정됨	
SA_RESTART	시그널 처리에 의해 방해받은 시스템 호출은 시그널 처리가 끝나면 재시작함	
SA_SIGINFO	a_handler 대신에 sa_sigaction이 동작되며, 시그널 번호, 시그널의 생성 이유, 시그널을 받는 프로세스의 정보 등 보다 다양한 인수를 받을 수 있음	리눅스 커널 2.2 이상, siginfo_t 등의 구조체를 사용

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

시그널 마스크와 sigprocmask()함수

sigprocmask() 함수

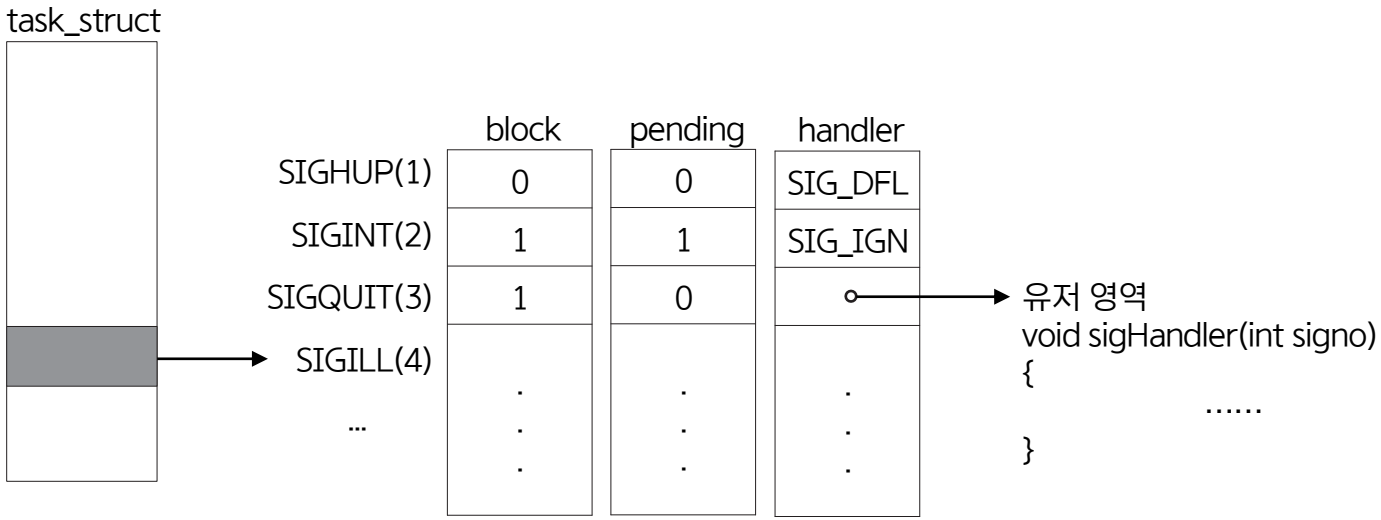
호출 프로세스의 시그널 마스크를 검사하거나
수정하는 데 사용

시그널 마스크

현재 차단되어 프로세스에 전달할 수 없는
신호 집합

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

시그널 마스크와 sigprocmask()함수

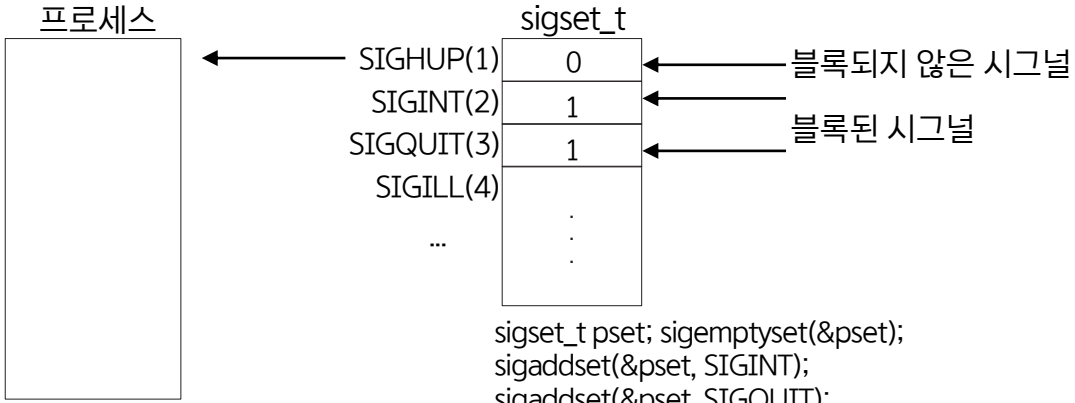


출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

시그널 셋(sigset_t)

sigset_t

어떤 시그널이 세트에 포함되거나 제외되는지에 대한 정보를 보유하는 데이터 구조




```
sigset_t pset; sigemptyset(&pset);
sigaddset(&pset, SIGINT);
sigaddset(&pset, SIGQUIT);
...
sigprocmask(SIG_BLOCK, &pset, NULL);
```

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

kill( ) 함수의 첫 번째 인자(pid)의 지정 방법


```
#include <sys/stypes.h>
#include <signal.h>

 int kill(pid_t pid, int signum);
 int raise(int signum);
```

kill( ) 함수의 첫 번째 인자(pid)의 지정 방법

값	내용
pid > 0	프로세스 ID가 pid인 프로세스에게 시그널을 전달함
pid == 0	호출한 프로세스와 같은 프로세스 그룹 ID의 모든 프로세스에게 시그널을 전달함
pid < 0	pid의 절댓값에 해당하는 프로세스 그룹 ID의 모든 프로세스에게 시그널을 전달함

따라하기 handleSignal.c


```
#include <stdio.h>
#include <signal.h>          /* signal( ) 함수를 위해 사용 */
#include <stdlib.h>          /* exit( ) 함수를 위해 사용 */
#include <string.h>          /* strsignal() 함수를 위해 사용 */
#include <unistd.h>

static void printSigset(sigset_t *set);       /* 현재 sigset_t에 설정된 시그널 표시 */
static void sigHandler(int);                  /* 시그널 처리용 핸들러 */

int main(int argc, char **argv)
{
    sigset_t pset;           /* 블록할 시그널을 등록할 sigset_t 형 */
```

```
sigemptyset(&pset);         /* 모두 0으로 설정 */
sigaddset(&pset, SIGQUIT);  /* SIGQUIT와 SIGRTMIN을 설정 */
sigaddset(&pset, SIGRTMIN);
sigprocmask(SIG_BLOCK, &pset, NULL);        /* 현재의 시그널 마스크에 추가 */

printSigset(&pset);         /* 현재 설정된 sigset_t를 화면으로 출력 */
```

따라하기 handleSignal.c


```
if(signal(SIGINT, sigHandler) == SIG_ERR) {       /* SIGINT의 처리를 위한 핸들러 등록 */
    perror("signal() : SIGINT");
    return -1;
}
if(signal(SIGUSR1, sigHandler) == SIG_ERR) {      /* SIGUSR1 처리를 위한 핸들러 등록 */
    perror("signal() : SIGUSR1");
    return -1;
}
if(signal(SIGUSR2, sigHandler) == SIG_ERR) {      /* SIGUSR2 처리를 위한 핸들러 등록 */
    perror("signal() : SIGUSR2");
    return -1;
}
```

따라하기 handleSignal.c


```
if(signal(SIGPIPE, SIG_IGN) == SIG_ERR) {          /* SIGPIPE 처리를 위한 핸들러 등록 */
        perror("signal() : SIGPIPE");
        return -1;
    }
    while(1) pause();          /* 시그널 처리를 위해 대기 */

    return 0;
}
```

따라하기 handleSignal.c


```
static void sigHandler(int signo)            /* 시그널 번호를 인자로 받는다. */
{
    if(signo == SIGINT) {                    /* SIGINT 시그널이 발생했을 때 처리 */
        printf("SIGINT is catched : %d\n", signo);
        exit(0);
    } else if(signo == SIGUSR1) {            /* SIGUSR1 시그널이 발생했을 때 처리 */
        printf("SIGUSR1 is catched\n");
    } else if(signo == SIGUSR2) {            /* SIGUSR2 시그널이 발생했을 때 처리 */
        printf("SIGUSR2 is catched\n");
    } else if(signo == SIGQUIT) {
        printf("SIGQUIT is catched\n");
        sigset_t uset;
```

```
        sigemptyset(&uset);
        sigaddset(&uset, SIGINT);
        sigprocmask(SIG_UNBLOCK, &uset, NULL);
    } else {
        fprintf(stderr, "Catched signal : %s\n", strsignal(signo));
    }
}
```

따라하기 handleSignal.c


```
static void printSigset(sigset_t *set)
{
    int i;
    for(i = 1; i < NSIG; ++i) {                /* sigset_t에 설정된 전체 비트를 출력 */
        printf((sigismember(set, i))?"1":"0");
    }
    putchar('\n');
}
```

따라하기 handleSignal.c

실행결과

```
$ gcc -o handleSignal handleSignal.c

^C 시그널에 반응
$ ./handleSignal
00100000000000000000000000000010000000000000000000000000000
^CSIGINT is catched : 2

$ ./handleSignal &
[1] 2584
$
00100000000000000000000000000010000000000000000000000000000
```

따라하기 handleSignal.c

실행결과

SIGUSR1 시그널에 반응

$ kill -USR1 2584

$ SIGUSR1 is catched

SIGUSR2 시그널에 반응

$ kill -USR2 2584

$ SIGUSR2 is catched

SIGPIPE, SIGQUIT 시그널에 무반응

$ kill -s SIGPIPE 2584

$ kill -s SIGQUIT 2584

따라하기 handleSignal.c

실행결과

```
SIGTERM 시그널에 반응, 프로그램 종료
$ kill -s SIGTERM 2584
$
[1]+  Terminated              ./handleSignal
```

요약정리

01 • 리눅스의 주요 시그널

02 • 시그널의 주요 함수