Linux System Programming

by ProgCoach4U

시그널

시그널이란?

- 정의
  - 비동기 이벤트를 처리하기 위한 메커니즘
  - 소프트웨어 인터럽트
- 언제 쓰이나?
  - Ctrl + C
  - Child process termination
  - Alarm
  - divide by zero
  - Inter-process communication
  - ...
- 어떤 정보를 담고 있나?
  - **시그널 번호** + 추가적인 정보 + (아주 작은) 사용자 정의 데이터

시그널의 처리

- 무시
  - 아무런 동작도 하지 않음
  - SIGKILL, SIGSTOP은 무시 불가능
- 처리
  - 시그널 별 처리 함수를 수행
- 기본 동작
  - 시그널 종류 별 기본 동작 수행
    - 프로세스 종료
    - 코어덤프 생성 후 종료
    - 무시
    - 정지

주요 시그널 번호

시그널번호	기본 동작	의미
SIGHUP	종료	프로세스의 제어 터미널 닫힘(사용자 로그아웃), 설정 리로드
SIGINT	종료	사용자가 Ctrl + C 발생
SIGKILL	종료	붙잡을 수 없는 프로세스 종료
SIGSEGV	코어 덤프	메모리 접근 위반
SIGALARM	종료	알람 발생
SIGTERM	종료	붙잡을 수 있는 프로세스 종료
SIGUSR1/2	종료	사용자 정의 시그널
SIGCHLD	종료	자식 프로세스 종료
SIGCONT	진행	프로세스를 정지했다가 계속 수행함
SIGSTOP	정지	프로세스 실행 보류

시그널의 실행과 상속

- fork() → 자식 프로세스는 부모 프로세스의 시그널 동작 상속 받음
- exec() → 부모 프로세스가 붙잡아 처리하는 시그널은 기본동작으로 변경

시그널 동작	fork() 수행 후	exec() 수행 후
무시	상속됨	상속됨
기본 동작	상속됨	상속됨
붙잡아 처리	상속됨	상속되지 않음(기본 동작 처리)
대기 중인 시그널	상속되지 않음	상속됨

시그널 처리 설정

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

파라미터
- signum: 처리 대상 시그널 번호
- handler: 시그널 핸들러
  - SIG_IGN: 해당 시그널을 무시 처리 한다.
  - SIG_DFL: 해당 시그널을 기본 동작 처리 한다.
  - 그외 사용자 정의 시그널 핸들러

반환값
- 성공시 이전 시그널 핸들러
- 실패시 SIG_ERR

재진입성(reentrant)

- Reentrant function이란?
  - 실행이 끝나기 전에 중단되었다가 재개되었을 때에도 정상적으로 수행을 마치는 함수
- 시그널 핸들러와 reentrant와의 관계
  - 시그널 핸들러가 호출되는 시점에 어떤 코드를 실행 중이었는지 알 수 없다.
  - 특히, 재진입이 불가능한 API를 사용하면 데이터가 망가질 수 있다.
- 그래서 시그널 핸들러는 ...
  - 재진입이 가능한 함수만 사용해야 한다.
  - 글로벌 데이터를 수정할 때 조심해야 한다.
  - 필요한 경우 시그널 블록 처리 → 데이터 처리 → 시그널 언블록 처리 하여 데이터를 안전하게 다뤄야 한다.

시그널 보내기

```
int kill(pid_t pid, int sig);
```

파라미터
- pid: 시그널 송신 대상 지정
  - 1 이상: 프로세스 ID
  - 0: 프로세스 그룹 전체
  - -1: 권한 내의 모든 프로세스
  - -1 미만: 프로세스 그룹 아이디가 (-pid)인 프로세스 그룹 전체
- sig: 보낼 시그널 번호
  - 단, 0인 경우 보내지는 않고, process 유무 및 권한 판단 수행

반환값
- 하나 이상의 프로세스에게 송신 시 0 리턴
- 실패시 -1

시그널 블록

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

파라미터
- how
  - SIG_BLOCK: blocked signal에 추가
  - SIG_UNBLOCK: 현재 blocked signal에서 제외
  - SIG_SETMASK: set 변수를 blocked signal로 설정
- set: 설정할 sigset
- oldset: 기존 설정되어 있던 sigset

반환값
- 성공시 0
- 실패시 -1

시그널 모음 설정

int sigemptyset(sigset_t *set);
- sigset 변수를 empty set으로 설정(아무런 시그널 설정되어 있지 않음)

int sigfillset(sigset_t *set);
- sigset 변수에 모든 시그널을 설정

int sigaddset(sigset_t *set, int signum);
- sigset 변수에 특정 시그널을 추가

int sigdelset(sigset_t *set, int signum);
- sigset 변수에서 특정 시그널을 삭제

int sigismember(const sigset_t *set, int signum);
- sigset 변수에서 특정 시그널이 포함되어 있는지 확인
- 포함시 1 리턴

고급 시그널 관리

```
int sigaction(int signum, const struct sigaction *act,
                          struct sigaction *oldact);
```

파라미터
- signum: 대상 시그널 번호
- act: 시그널 처리 액션
- oldact: 기존 시그널 처리 액션

반환값
- 성공시 0
- 실패시 -1

고급 시그널 관리

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

- sa_flags에 SA_SIGINFO 포함 시 sa_sigaction이 호출됨
- sa_mask는 시그널 핸들러 실행하는 동안 블록해야 할 시그널 모음 정의

고급 시그널 관리

- sa_flags

flags	meaning
SA_SIGINFO	sa_sigaction()이 사용됨
SA_NODEFER	설정시 시그널 핸들러가 실행 중인 시그널에 대해 블록하지 않음
SA_NOCLDSTOP	signo가 SIGCHLD인 경우 자식이 정지해도 SIGCHLD를 보내지 않도록 함
SA_NOCLDWAIT	signo가 SIGCHLD인 경우 자식에 대한 자동처리 활성화(wait() 필요 없음)
SA_RESETHAND	시그널 핸들러가 1회면 동작한 후 default로 초기화 된다.
SA_RESTART	read() 등의 시스템콜이 자동으로 재시작된다.

고급 시그널 관리

- siginfo_t 주요 필드(man sigaction(2) 참조)

field	meaning
si_signo	시그널 번호
si_errno	0이 아닌경우 시그널과 관련된 에러 코드
si_code	시그널을 일으킨 원인
si_int, si_ptr	sigqueue()를 통해 보낸 시그널 페이로드
si_addr	SIGBUS, SIGFPE, SIGILL, SIGSEGV, SIGTRAP의 경우 장애를 일으킨 주소
si_fd	SIGPOLL의 경우 작업을 완료한 파일 디스크립터

고급 시그널 관리

- 모든 시그널 번호에 대해 유효한 si_code

si_code	meaning
SI_ASYNCIO	비동기식 입출력의 완료 표시
SI_KERNEL	커널이 보낸 시그널
SI_MESGQ	POSIX 메시지 큐의 상태 변화 표시
SI_QUEUE	sigqueue()로 보낸 시그널
SI_TIMER	POSIX 타이머 만료를 표시
SI_USER	kill() 또는 raise()로 보낸 시그널

고급 시그널 관리

- SIGCHLD인 경우에만 유효한 si_code

si_code	meaning
CLD_CONTINUED	자식이 정지되었다가 재시작됨
CLD_DUMPED	자식이 비정상적으로 종료됨
CLD_EXITED	자식이 exit()로 정상 종료됨
CLD_KILLED	자식이 종료됨
CLD_STOPPED	자식이 정지됨
CLD_TRAPPED	자식이 트랩에 걸림

페이로드와 함께 시그널 전송

```
union sigval {
    int sival_int;
    void *sival_ptr;
};
int sigqueue(pid_t pid, int signo, const union sigval value);
```

파라미터
- pid: 대상 프로세스
- signo: 시그널 번호
- value: 페이로드

반환값
- 성공시 0
- 실패시 -1

해당 시그널을 sigaction()으로 받으면...
- siginfo_t.si_code: SI_QUEUE로 설정됨
- siginfo_t.si_int: 페이로드 설정됨

시간

현재 시간 얻기

```
time_t time(time_t *tloc);
```

파라미터
- tloc: 현재 시간값을 저장할 버퍼(NULL 입력 가능)

반환값
- 1970년 1월 1일 0시 0분 0초(UTC)로부터 현재 시각까지 지난 초
- 실패시 -1

현재 시간 얻기

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

파라미터
- tv: 현재 시간값을 저장할 버퍼
  - 1970년 1월 1일 0시 0분 0초(UTC)로부터 현재 시각까지 지난 시간
- tz: 사용되지 않음

반환값
- 성공시 0
- 실패시 -1

시간 정보 다루기

```
struct tm *localtime(const time_t *timep);
struct tm *localtime_r(const time_t *timep, struct tm *result);
```
- timezone 기반 시간 정보

파라미터
- timep: 변환할 시간 정보
- result: thread-safe 지원을 위한 버퍼

반환값
- 성공시 세분화 된 시간 정보
- 실패시 NULL

```
struct tm {
        int tm_sec;    /* Seconds (0-60) */
        int tm_min;    /* Minutes (0-59) */
        int tm_hour;   /* Hours (0-23) */
        int tm_mday;   /* Day of the month (1-31) */
        int tm_mon;    /* Month (0-11) */
        int tm_year;   /* Year - 1900 */
        int tm_wday;   /* Day of the week (0-6, Sunday = 0) */
        int tm_yday;   /* Day in the year (0-365, 1 Jan = 0) */
        int tm_isdst;  /* Daylight saving time */
};
```

시간 정보 다루기

```
struct tm *gmtime(const time_t *timep);
struct tm *gmtime_r(const time_t *timep, struct tm *result);
```
- UTC 기반 시간 정보

파라미터
- timep: 변환할 시간 정보
- result: thread-safe 지원을 위한 버퍼

반환값
- 성공시 세분화 된 시간 정보
- 실패시 NULL

시간 정보 다루기

time_t mktime(struct tm *tm);
- localtime 기준으로 세분화된 시간 정보를 초단위 시간정보로 변경

파라미터
- tm:세분화된 시간 정보

반환값
- 성공시 초단위 현재 시각 정보
- 실패시 -1

단순 타이머

unsigned int alarm(unsigned int seconds);
- 일정 시간 후 SIGALARM 보내줌
- SIGALARM에 대한 signal 설정 및 처리 필요

파라미터
- seconds: 타임아웃 값(초단위)
  - 단, 0인 경우 이전 타이머를 해제

반환값
- 성공시 이전 타이머 설정이 만료까지 남은시간
- 이전 타이머가 없는 경우 0 리턴함

인터벌 타이머

```
int getitimer(int which, struct itimerval *curr_value);
int setitimer(int which, const struct itimerval *new_value,
                      struct itimerval *old_value);
```

파라미터
- which: 타이머의 종류
  - ITIMER_REAL: 실제 시간(SIGALARM 전송)
  - ITIMER_VIRTUAL: 사용자 영역 코드가 수행되는 동안에만 타이머가 흐름(SIGVTALRM 전송)
  - ITIMER_PROF
  - ITIMER_REAL
- new_value: 설정할 타임아웃 정보
- old_value: 기존 타임아웃 정보

반환값
- 성공시 0
- 실패시 -1

```
struct itimerval {
    struct timeval it_interval;    /* 인터벌 타임아웃 값 */
    struct timeval it_value;       /* 초기 타임아웃 값 */
};
```

감사합니다.