Linux System Programming #6-3
Lecture Notes

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

sigprocmask(2)

```
#include <signal.h>
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- 현재의 시그널 마스크를 검사하거나 변경하는 시스템호출 함수
- 프로세스의 시그널 마스크
  - 시그널이 마스크(블록)되어 있어 현재 프로세스에게 전달되지 않는 시그널들의 집합
- how 인자
  - 시그널 마스크의 구체적인 변경 방법을 정의하는 상수
  - 시그널 마스크를 설정 => set 인자에 NULL 포인터가 아닌 포인터를 넣고 시그널 마스크의 구체적인 변경 방법을 how 인자에 정의
- set 인자
  - 마스크하거나 마스크를 해제할 시그널 집합의 주소
  - set 인자가 NULL이라면, 시그널 마스크는 변경되지 않고 how 인자가 어떤 값을 가지고 있어도 무시
- oldset 인자
  - 이전 시그널 설정 값을 저장할 시그널 집합의 주소, NULL. 사용 가능
  - 시그널 마스크를 검사할 때 oldset 인자에 NULL 포인터가 아닌 값을 넣으면 프로세스의 현재 시그널 마스크가 oldset 인자가 가리키는 곳에 저장

how 인자에 따른 sigprocmask()의 실행

how 인자	설명
SIG_BLOCK	프로세스의 현재 시그널 마스크와 set이 가리키는 시그널 집합의 합집합이 프로세스의 새 프로세스 마스크가 됨. 주로 블록할 시그널들을 더 추가할 때 유용
SIG_UNBLOCK	프로세스의 현재 시그널 마스크와 set이 가리키는 시그널 집합의 교집합이 프로세스의 새 프로세스 마스크가 됨. 주로 블록된 시그널들에 대해 블록을 해제할 때 유용
SIG_SETMASK	이전 블록된 시그널 집합을 모두 지우고 set이 가리키는 시그널 집합의 값이 프로세스의 새 시그널 마스크로 설정

sigprocmask() 예제

```
<ssu_sigprocmask.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main(void)
{
   sigset_t sig_set;
   int count;

   sigemptyset(&sig_set);
   sigaddset(&sig_set, SIGINT);
   sigprocmask(SIG_BLOCK, &sig_set, NULL);

   for (count = 3 ; 0 < count ; count--) {
      printf("count %d\n", count);
      sleep(1);
   }

   printf("Ctrl-C에 대한 블록을 해제\n");
   sigprocmask(SIG_UNBLOCK, &sig_set, NULL);
   printf("count중 Ctrl-C입력하면 이 문장은 출력 되지 않음.\n");

   while (1);

   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigprocmask
count 3
count 2
count 1
Ctrl-C에 대한 블록을 해제
count중 Ctrl-C입력하면 이 문장은 출력 되지 않음.
^C
root@localhost:/home/oslab# ./ssu_sigprocmask
count 3
^Ccount 2
^Ccount 1
Ctrl-C에 대한 블록을 해제
```

sigpending(2)

```
#include <signal.h>
int sigpending(sigset_t *set);
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- 팬딩 중이거나, 블록되었을 때 발생한 시그널 집합을 가져오는 시스템호출 함수
  - 시그널을 블록된 상태에서 어떤 시그널이 발생해서 블럭되었는지 확인
  - set 인자가 가리키는 곳에 호출한 프로세스에 시그널이 블록되어 있으며 현재 팬딩 중인 시그널들의 집합을 저장

sigpending() 예제

<ssu_sigpending.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int main(void)
{
   sigset_t pendingset;
   sigset_t sig_set;
   int count = 0;

   sigfillset(&sig_set);
   sigprocmask(SIG_SETMASK, &sig_set, NULL);

   while (1) {
      printf("count: %d\n", count++);
      sleep(1);

      if (sigpending(&pendingset) == 0) {
         if (sigismember(&pendingset, SIGINT)) {
            printf("SIGINT가 블록되어 대기 중. 무한 루프를 종료.\n");
            break;
         }
      }
   }

   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigpending
count: 0
count: 1
count: 2
count: 3
^Zcount: 4
^Zcount: 5
count: 6
count: 7
count: 8
^CSIGINT가 블록되어 대기 중. 무한 루프를 종료.
```

sigaction(2)

```
#include <signal.h>
int sigaction(int signo, const struct sigaction *act, struct sigaction *oldact);
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 errno가 설정됨
```

- sigpending()는 팬딩 중이거나, 블록되었을 때 발생한 시그널 집합을 가져오는 시스템호출 함수
- set 인자가 가리키는 곳에 호출한 프로세스에 시그널이 블록되어 있으며 현재 팬딩 중인 시그널들의 집합을 저장
- act : 액션 수정
- oldact : 이전 액션

```
struct sigaction{
void (*sa_handler)(int);          //SIG_IGN나 SIG_DFL의 시그널 핸들러 주소
sigset_t sa_mask;                 //추가적으로 블록할 시그널
int sa_flags;                     //시그널 옵션, <표 6-10>참조
void (*sa_sigaction)(int, siginfo_t *, void *); //대안 핸들러
};
```

각 시그널 처리에 대한 옵션 플래그들(sa_flags)

옵션	설명
SA_INTERRUPT	이 시그널에 의해 가로채인 시스템 호출이 자동으로 재시작 되지 않음
SA_NOCLDSTOP	signo가 SIGCHLD인 경우, 자식 프로세스가 중단되었을 때 이 시그널을 발생하지 않도록 함
SA_NOCLDWAIT	signo가 SIGCHLD인 경우, 이 옵션은 호출한 프로세스의 자식 프로세스들이 종료되었을 때 시스템이 좀비 프로세스들을 만들지 못하게 함
SA_NODEFER	이 시그널에 의해 시그널 핸들러가 실행되는 도중에 시스템이 같은 시그널을 자동으로 블록되지 않게 함
SA_ONSTACK	sigaltstack(2)로 대안 스택이 선언되어 있다면 시그널이 대안 스택의 프로세스에 전달되게 함
SA_RESETHAND	시그널 핸들러에 진입할 때 시그널의 처분 방식을 SIG_DFL로 재설정하고 SA_SIGINFO플래그를 해제함
SA_RESTART	이 시그널에 의해 가로채인 시스템 호출이 자동으로 재시작하게 함
SA_SIGINFO	시그널 핸들러에 추가적인 정보 두 개를 전달하게 함

sigaction() 예제 1

<ssu_sigaction_1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void ssu_signal_handler(int signo) {
   printf("ssu_signal_handler control\n");
}

int main(void) {
   struct sigaction sig_act;
   sigset_t sig_set;

   sigemptyset(&sig_act.sa_mask);
   sig_act.sa_flags = 0;
   sig_act.sa_handler = ssu_signal_handler;
   sigaction(SIGUSR1, &sig_act, NULL);
   printf("before first kill()\n");
   kill(getpid(), SIGUSR1);
   sigemptyset(&sig_set);
   sigaddset(&sig_set, SIGUSR1);
   sigprocmask(SIG_SETMASK, &sig_set, NULL);
   printf("before second kill()\n");
   kill(getpid(), SIGUSR1);
   printf("after second kill()\n");
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigaction_1
before first kill()
ssu_signal_handler control
before second kill()
after second kill()
```

sigaction() 예제 2

```
<ssu_sigaction_2.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

void ssu_check_pending(int signo, char *signame);
void ssu_signal_handler(int signo);

int main(void)
{
   struct sigaction sig_act;
   sigset_t sig_set;

   sigemptyset(&sig_act.sa_mask);
   sig_act.sa_flags = 0;
   sig_act.sa_handler = ssu_signal_handler;

   if (sigaction(SIGUSR1, &sig_act, NULL) != 0) {
      fprintf(stderr, "sigaction() error\n");
      exit(1);
   }
   else {
      sigemptyset(&sig_set);
      sigaddset(&sig_set, SIGUSR1);

      if (sigprocmask(SIG_SETMASK, &sig_set, NULL) != 0) {
         fprintf(stderr, "sigprocmask() error\n");
         exit(1);
      }
      else {
         printf("SIGUSR1 signals are now blocked\n");
```

```
 kill(getpid(), SIGUSR1);
         printf("after kill()\n");
         ssu_check_pending(SIGUSR1, "SIGUSR1");
         sigemptyset(&sig_set);
         sigprocmask(SIG_SETMASK, &sig_set, NULL);
         printf("SIGUSR1 signals are no longer blocked\n");
         ssu_check_pending(SIGUSR1, "SIGUSR1");
      }
   }
   exit(0);
}
void ssu_check_pending(int signo, char *signame) {
   sigset_t sig_set;

   if (sigpending(&sig_set) != 0)
      printf("sigpending() error\n");
   else if (sigismember(&sig_set, signo))
      printf("a %s signal is pending\n", signame);
   else
      printf("%s signals are not pending\n", signame);
}
void ssu_signal_handler(int signo) {
   printf("in ssu_signal_handler function\n" );
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigaction_2
SIGUSR1 signals are now blocked
after kill()
a SIGUSR1 signal is pending
in ssu_signal_handler function
SIGUSR1 signals are no longer blocked
SIGUSR1 signals are not pending
```

sigaction() 예제 3

<ssu_sigaction_3.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
static void ssu_signal_handler1(int signo);
static void ssu_signal_handler2(int signo);

int main(void)
{
   struct sigaction act_int, act_quit;

   act_int.sa_handler = ssu_signal_handler1;
   sigemptyset(&act_int.sa_mask);
   sigaddset(&act_int.sa_mask, SIGQUIT);
   act_quit.sa_flags = 0;

   if (sigaction(SIGINT, &act_int, NULL) < 0) {
      fprintf(stderr, "sigaction(SIGINT) error\n");
      exit(1);
   }
   act_quit.sa_handler = ssu_signal_handler2;
   sigemptyset(&act_quit.sa_mask);
   sigaddset(&act_quit.sa_mask, SIGINT);
   act_int.sa_flags = 0;
   if (sigaction(SIGQUIT, &act_quit, NULL) < 0) {
      fprintf(stderr, "sigaction(SIGQUIT) error\n");
      exit(1);
   }
   pause();
   exit(0);
}
```

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

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

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigaction_3
^CSignal handler of SIGINT : 2
SIGQUIT signal is blocked : 2
sleeping 3 sec
^CSignal handler of SIGINT ended
Signal handler of SIGINT : 2
SIGQUIT signal is blocked : 2
sleeping 3 sec
Signal handler of SIGINT ended
```

sigsetjmp(3), siglongjmp(3)

```
#include <setjmp.h>
int sigsetjmp(sigjmp_buf env, int savesigs);
리턴 값 : 직접 호출되었을 때에는 0, siglongjmp() 호출에서 리턴된 경우에는 0이 아닌 값

void siglongjmp(sigjmp_buf env, int val);
```

- sigsetjmp()/ siglongjmp()
  - 프로세스의 현재 상태를 저장하고, 프로세스의 저장된 위치로 복귀
  - setjmp()와 longjmp()를 시그널에서 사용하기 힘듦
  - longjmp()의 문제점
    - 시그널이 전달되어 시그널 핸들러가 실행되면 해당 시그널은 자동적으로 시그널 마스크에 추가되며, 블록됨

sigsetjmp() 예제 1

<ssu_sigsetjmp_1.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <setjmp.h>

void ssu_signal_handler(int signo);

jmp_buf jump_buffer;

int main(void)
{
   signal(SIGINT, ssu_signal_handler);

   while (1) {
      if (setjmp(jump_buffer) == 0) {
         printf("Hit Ctrl-c at anytime ... \n");
         pause();
      }
   }

   exit(0);
}
```

```
void ssu_signal_handler(int signo) {
   char character;

   signal(signo, SIG_IGN);
   printf("Did you hit Ctrl-c?\n" "Do you really want to quit? [y/n] ");
   character = getchar();
   if (character == 'y' || character == 'Y')
      exit(0);
   else {
      signal(SIGINT, ssu_signal_handler);
      longjmp(jump_buffer, 1);
   }
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigsetjmp_1
Hit Ctrl-C at anytime ...
^CDid you hit Ctrl-C?
Do you really want to quit? [y/n] y
root@localhost:/home/oslab# ./ssu_sigsetjmp_1
Hit Ctrl-C at anytime ...
^CDid you hit Ctrl-C?
Do you really want to quit? [y/n] n
Hit Ctrl-C at anytime ...
^C^C^C^C^C^C^C
```

sigsetjmp() 예제 2

page 332

sigsetjmp() 예제

page 334

sigsuspend(2)

```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
리턴 값 : 항상 –1을 리턴, errno는 EINTR로 설정
```

- 블록시킬 시그널을 재설정함과 함께 동시에 캐치할 수 있는 시그널이 도착하거나 종료(SIGINT) 시그널이 도착할 때까지 프로세스를 잠시 블록시키는 액션을 원자적으로 실행하는 시스템호출
- sigprocmask(SIG_SETMAST, &set, NULL);와 pause();를 원자적으로 실행하여 sigprocmask()와 pause() 사이에 발생할 수 있는 시그널을 잃어버리지 않게 함

- 해당 시그널을 처리하는 시그널 핸들러가 있다면 해당 시그널 핸들러를 실행하고, 해당 시그널 핸들러가 리턴하면 sigsuspend()는 –1을 리턴하고 sigsuspend()를 호출하기 전 마스크된 시그널 집합이 복원됨
- 주의할 점
  - 프로세스를 종료시키는 액션을 하는 시그널을 받는 경우 sigsuspend()는 리턴되지 않음
  - SIGKILL이나 SIGSTOP 시그널은 마스크에 포함시켜도 블록되지 않으며 기존 마스크에 영향을 주지 않음
  - 일반적으로 sigsuspend()는 프로그램의 특정 코드가 실행되는 영역에서 특정 시그널을 블록시키기 위해 sigprcokmask()와 함께 사용함

sigsuspend() 예제 1

<ssu_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;

   sigemptyset(&sig_set);
   sigaddset(&sig_set, SIGINT);
   sigprocmask(SIG_BLOCK, &sig_set, &old_set);
   sigsuspend(&old_set);
   exit(0);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigsuspend_1
^C
root@localhost:/home/oslab#
```

sigsuspend() 예제 2

page 338

sigsuspend() 예제 3

<ssu_sigsuspend_3.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>

void ssu_signal_handler(int signo);
void ssu_timestamp(char *str);

int main(void)
{
   struct sigaction sig_act;
   sigset_t blk_set;

   sigfillset(&blk_set);
   sigdelset(&blk_set, SIGALRM);
   sigemptyset(&sig_act.sa_mask);
   sig_act.sa_flags = 0;
   sig_act.sa_handler = ssu_signal_handler;
   sigaction(SIGALRM, &sig_act, NULL);
   ssu_timestamp("before sigsuspend()");
   alarm(5);
   sigsuspend(&blk_set);
   ssu_timestamp("after sigsuspend()");
   exit(0);
}

void ssu_signal_handler(int signo) {
   printf("in ssu_signal_handler() function\n");
}
```

```
void ssu_timestamp(char *str) {
   time_t time_val;

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

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigsuspend_3
before sigsuspend() the time is Wed Jan 11 21:21:57 2017

in ssu_signal_handler() function
after sigsuspend() the time is Wed Jan 11 21:22:02 2017
```

abort(3)

```
#include <stdlib.h>
void abort(void);
```

- 프로세스 자신에게 SIGABRT 시그널을 발생시키며, 프로그램을 비정상적으로 종료시키는 라이브러리 함수
- 프로세스는 이 시그널을 무시할 수 없음
  - SIGABRT에 대한 별도의 시그널 핸들러를 두고 프로세스가 종료하기 전에 여러 가지 정리 작업을 실행할 수 있음

- (참고)
  - POSIX.1 : 이 시그널 핸들러에서 스스로 종료하지 않는 경우에는 시그널 핸들러가 리턴한 후 abort()에 의해서 프로세스가 종료된다고 정의
  - ANSI C : abort()의 호출에 따른 출력 스트림의 플러시(flush)와 같은 임시 파일의 삭제 등의 실행 여부는 시스템에 종속적이라고 되어 있는 반면에 POSIX.1에서는 abort()가 프로세스를 종료시킬 때 모든 표준 입출력 스트림에 대해 fclose()를 호출한 효과를 가진다고 정의

abort() 예제

<ssu_abort.c>

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   printf("abort terminate this program\n");
   abort();
   printf("this line is never reached\n");
   exit(0);
}
```

실행 결과 [Ubuntu, Fedora]

```
root@localhost:/home/oslab# ./ssu_abort
abort terminate this program
Aborted (core dumped)
```

실행 결과 [macOS]

```
ssu-ui-MacBook-Air:oslab root# ./ssu_abort
abort terminate this program
Abort trap: 6
```

sleep(3)

```
#include <unistd.h>
unsigned int sleep(unsigned int seconds);
리턴 값 : 0 또는 덜 잠든 시간(초)
```

- 호출한 프로세스를 seconds 인자로 지정된 시간이 경과하거나, 시그널이 전달되어 시그널 핸들러가 리턴될 때까지 일시정지 상태로 만드는 라이브러리 함수
- 지정된 시간이 경과되어 리턴하게 되는 경우에는 리턴 값이 0이고, 시그널 핸들러가 리턴되서 sleep()가 리턴되는 경우에는 리턴 값이 seconds 인자로 지정한 시각까지 남은 시간(초)이 리턴
- sleep()는 alarm()를 이용해서 구현할 수 있음

sleep() 예제

<ssu_sleep.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

void ssu_timestamp(char *str);

int main(void)
{
   unsigned int ret;

   ssu_timestamp("before sleep()");
   ret = sleep(10);
   ssu_timestamp("after sleep()");
   printf("sleep() returned %d\n", ret);
   exit(0);
}

void ssu_timestamp(char *str) {
   time_t time_val;

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

실행 결과

```
root@localhost:/home/oslab# ./ssu_sleep
before sleep() the time is Wed Jan 11 21:25:59 2017

after sleep() the time is Wed Jan 11 21:26:09 2017

sleep() returned 0
```