Linux System Programming #6-1
Lecture Notes

Spring 2020

School of Computer Science and Engineering,

Soongsil University, Seoul, Korea

Jiman Hong

jiman@acm.org

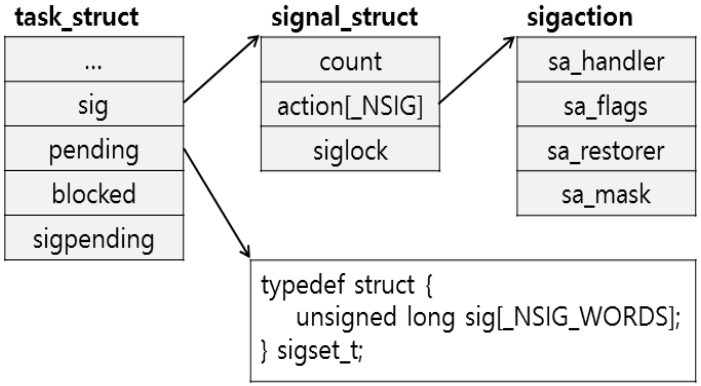
signal(2)

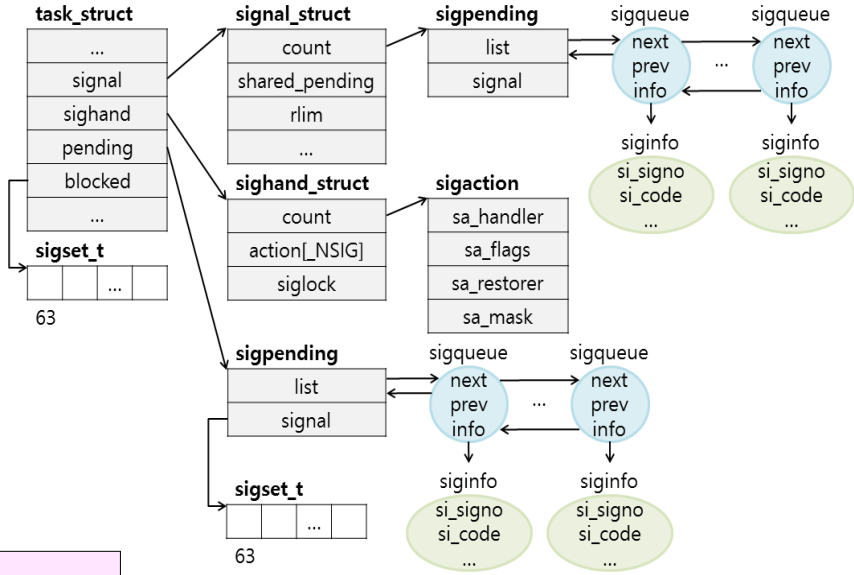
```
#include <signal.h>
sighandler_t signal (int signum, sighandler_t handler);
리턴 값: 성공 시 포인터, 에러 시 NULL
typedef void (*sighandler_t) (int);
```

- 지정한 시그널의 처리 방식을 등록하는 시스템호출 함수
- signum - 시그널
- handler <표 6-3>
  - SIG_IGN : signum으로 지정된 시그널 무시
  - SIG_DFL : 시그널 발생 시 시그널의 디폴트 액션을 실행
  - handler 주소 : 것은 프로세스가 해당 시그널을 캐치해서 어떻게 처리하겠다고 시스템에게 알려주는 것
- 호출에 의해 리턴되는 값
  - 그 시그널에 대한 이전 시그널 핸들러의 주소

시그널 호출과 관련된 자료 구조

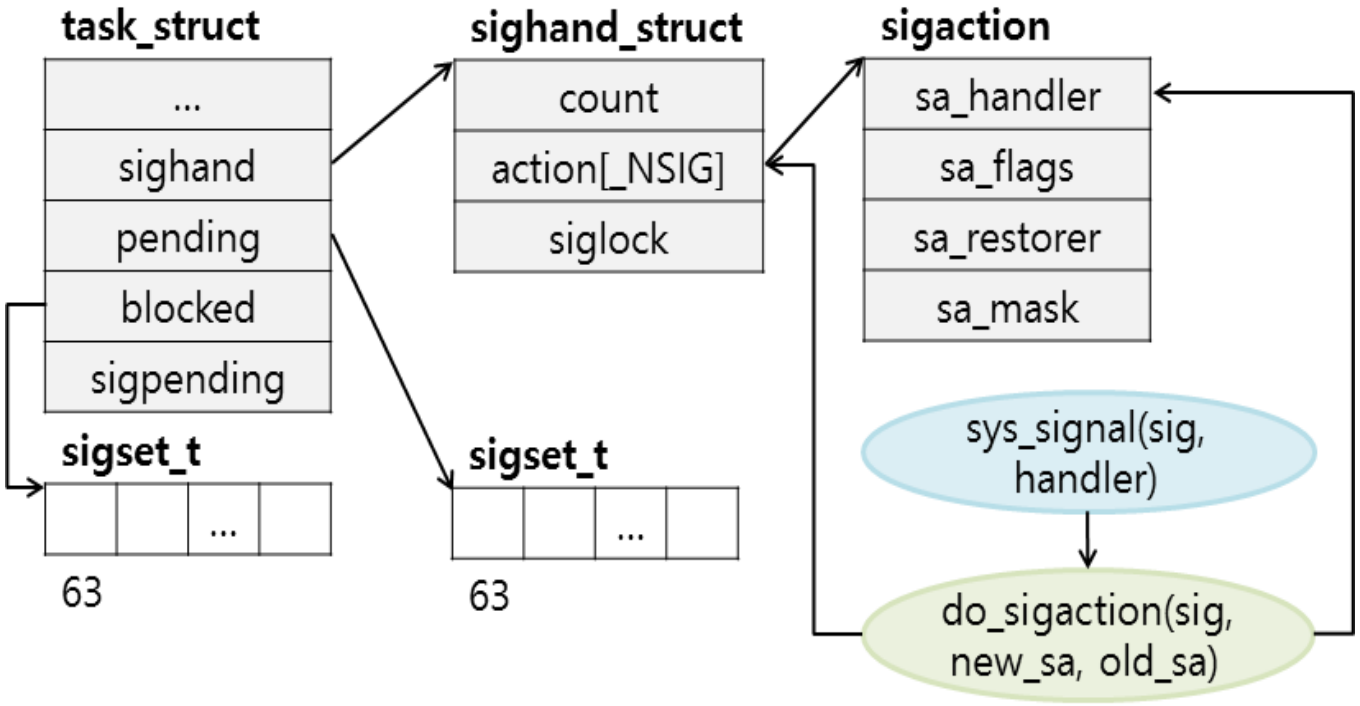
- 시그널이 발생
  - sig->action[_NSIG]->sa_handler를 통해 해당 시그널의 핸들러가 실행
    - 시그널이 발생하여 시스템에 들어오는 순간, 해당 시그널이 blocked에 설정



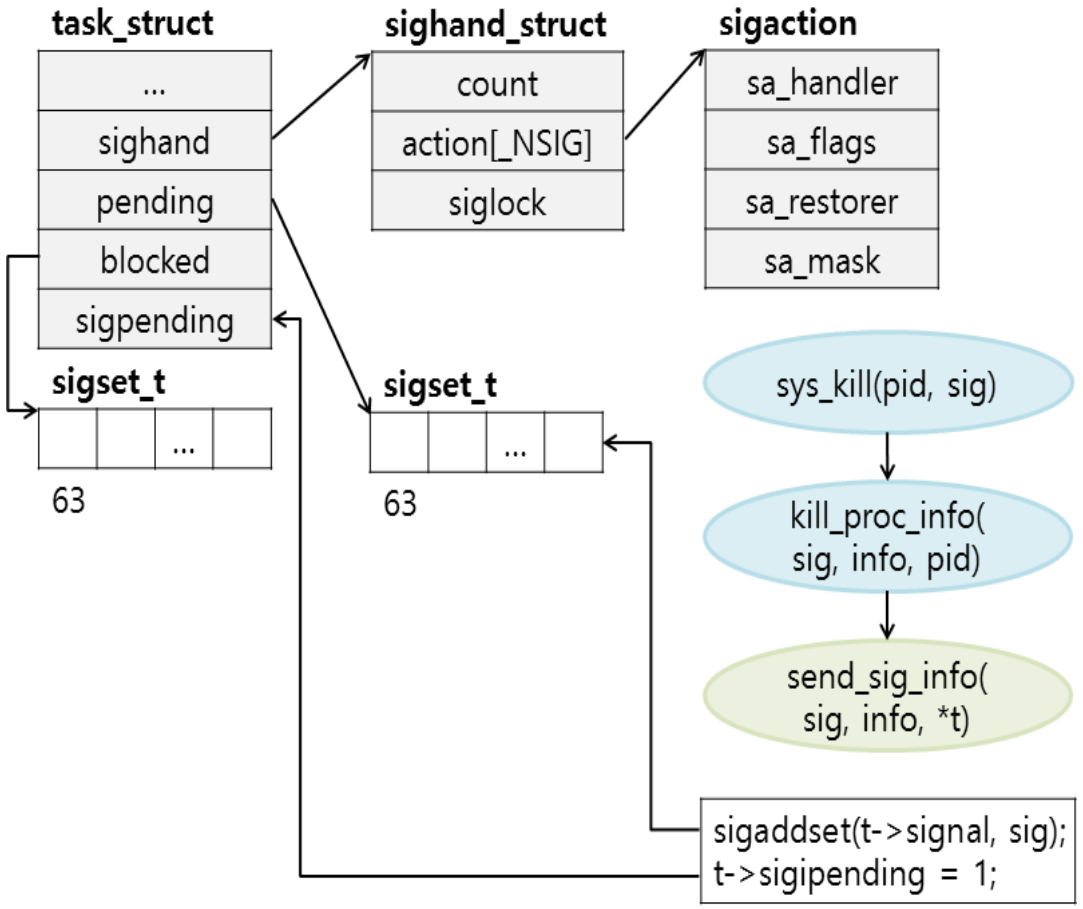


매크로	내용
pending	비트 별로 시그널을 구분하여, 시그널이 발생했을 때 팬딩하는 구조체
blocked	시그널 마스킹과 같은 역할. pending과 같이 비트 단위 시그널 구분
sigpending	시그널이 있으면 1, 없으면 0을 리턴해 시그널 처리를 보다 빨리 실행

Linux의 시그널 핸들러 등록 과정



Linux에서 시그널 전달 과정



Linux 커널 버전에 따른 시그널 전달 과정

```
/* Linux 커널 버전 2.4 */
sys_kill(pid, sig) {
struct task_struct *task = find_task_by_pid(pid);  // pid로 시그널 받을 task를 구함
task -> pending |= signo; // pending 구조체에 signo 비트를 설정
send_sig_info(sig, &info, task);
task -> sigpending = 1; // sigpending에 1을 할당
}
```

```
/* Linux 커널 버전 2.6 */
sys_kill(pid, sig) {
struct task_struct *task = find_task_by_pid(pid);  // pid로 시그널 받을 task를 구함
if (signal이 중첩되었을 경우)
count++;
task -> pending -> list = sig; // pending의 list에 시그널 추가
task -> sigpending = 1; // sigpending에 1을 할당
}
```

signal() 예제 1

<ssu_signal_1.c>

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

void ssu_signal_handler(int signo);
void (*ssu_func)(int);

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

   while (1) {
      printf("process running...\n");
      sleep(1);
   }

   exit(0);
}

void ssu_signal_handler(int signo) {
   printf("SIGINT 시그널 발생.\n");
   printf("SIGINT를 SIG_DFL로 재설정 함.\n");
   signal(SIGINT, ssu_func);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_signal_1
process running...
process running...
process running...
^CSIGINT 시그널 발생.
SIGINT를 SIG_DFL로 재설정 함.
process running...
process running...
^C
```

signal() 예제 2

<ssu_signal_2.c>

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

static void ssu_signal_handler(int signo);

int main(void)
{
   if (signal(SIGINT, ssu_signal_handler) == SIG_ERR) {
      fprintf(stderr, "cannot handle SIGINT\n");
      exit(EXIT_FAILURE);
   }

   if (signal(SIGTERM, ssu_signal_handler) == SIG_ERR) {
      fprintf(stderr, "cannot handle SIGTERM\n");
      exit(EXIT_FAILURE);
   }

   if (signal(SIGPROF, SIG_DFL) == SIG_ERR) {
      fprintf(stderr, "cannot reset SIGPROF\n");
      exit(EXIT_FAILURE);
   }
   if (signal(SIGHUP, SIG_IGN) == SIG_ERR) {
      fprintf(stderr, "cannot ignore SIGHUP\n");
      exit(EXIT_FAILURE);
   }

   while (1)
      pause();

   exit(0);
}

static void ssu_signal_handler(int signo) {
   if (signo == SIGINT)
      printf("caught SIGINT\n");
   else if (signo == SIGTERM)
      printf("caught SIGTERM\n");
   else {
      fprintf(stderr, "unexpected signal\n");
      exit(EXIT_FAILURE);
   }

   exit(EXIT_SUCCESS);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_signal_2
^Ccaught SIGINT
```

kill(2), raise(3)

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
리턴 값 : 성공 시 0, 에러 시 –1을 리턴하고 적절한 errno가 설정됨
#include <signal.h>
int raise(int sig);
리턴 값 : 성공 시 0, 에러 시 0이 아닌 값이 리턴됨
```

pid	Pid 인자에 따른 실행 방식
pid > 0	특정 프로세스(프로세스 ID가 pid인 프로세스)에 시그널을 보냄
pid == 0	프로세스 그룹 ID가 호출한 프로세스의 그룹 ID와 동일한 프로세스 그룹의 모든 프로세스 중 시그널을 보낼 권한을 가지고 있는 모든 프로세스에게 시그널을 보냄. 즉, 자신과 같은 그룹에 속한 모든 프로세스에 시그널 전달. 단, 시스템 프로세스들은 제외
pid < 0	프로세스 그룹 ID가 pid인 프로세스 그룹의 모든 프로세스 중 호출한 프로세스가 시그널을 보낼 권한을 가지고 있는 모든 프로세스에게 시그널을 보냄. 즉, 시그널 전달이 허용된 모든 프로세스에게 시그널 전달. 단, 시스템 프로세스들은 제외
pid == -1	호출한 프로세스가 시그널을 보낼 권한을 가지고 있는 모든 프로세스에 시그널을 보냄. 즉 프로세스 그룹 식별 번호가 –pid인 모든 프로세스에 시그널 전달. 단, 시스템 프로세스들은 제외

- kill()
  - pid로 지정된 하나의 프로세스 또는 프로세스 그룹에 시그널을 보내는 시스템호출 함수
  - 시그널은 sig 인자에 의해 지정
    - 시그널 이름에 대한 상수 값 또는 0
  - 프로세스가 존재하지 않을 경우 kill()는 –1을 리턴하고 errno를 ESRCH로 설정
- raise()
  - sig인자를 처리 중인 프로세스에 보내는 라이브러리 함수
  - 자기 자신에게 시그널을 보내는 함수
  - ANSI C에는 정의/ POSIX.1 미정의
  - raise(sig) = kill(getpid(), sig)

pid에 따른 kill()의 실행 방식

시그널	내용
pid > 0	특정 프로세스(프로세스 ID가 pid인 프로세스)에 시그널을 보냄
pid == 0	프로세스 그룹 ID가 호출한 프로세스의 그룹 ID와 동일한 프로세스 그룹의 모든 프로세스 중 시그널을 보낼 권한을 가지고 있는 모든 프로세스에게 시그널을 보냄. 즉, 자신과 같은 그룹에 속한 모든 프로세스에 시그널 전달. 단, 시스템 프로세스들은 제외
pid < 0	프로세스 그룹 ID가 pid인 프로세스 그룹의 모든 프로세스 중 호출한 프로세스가 시그널을 보낼 권한을 가지고 있는 모든 프로세스에게 시그널을 보냄. 즉, 시그널 전달이 허용된 모든 프로세스에게 시그널 전달. 단, 시스템 프로세스들은 제외
pid == -1	호출한 프로세스가 시그널을 보낼 권한을 가지고 있는 모든 프로세스에 시그널을 보냄. 즉 프로세스 그룹 식별 번호가 –pid인 모든 프로세스에 시그널 전달. 단, 시스템 프로세스들은 제외

kill() 예제

```
<ssu_kill_A.c>
#include <stdio.h>#include <stdlib.h>
#include <signal.h>

int main(int argc, char *argv[])
{
   if (argc != 2) {
      fprintf(stderr, "usage: %s [Process ID]\n", argv[0]);       exit(1);
   }
   else
      kill(atoi(argv[1]), SIGKILL);

   exit(0);
}

<ssu_kill_B.c>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
   while (1) {
      printf("\n[OSLAB]");
      sleep(5);
   }

   exit(0);
}
```

```
실행 결과
root@localhost:/home/oslab# ./ssu_signal_1
process running...
process running...
process running...
^CSIGINT 시그널 발생.
SIGINT를 SIG_DFL로 재설정 함.
process running...
process running...
^C
```

raise() 예제

```
<ssu_raise.c>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

void ssu_signal_handler1(int signo);
void ssu_signal_handler2(int signo);

int main(void)
{
   if (signal(SIGINT, ssu_signal_handler1) == SIG_ERR) {
      fprintf(stderr, "cannot handle SIGINT\n");
      exit(EXIT_FAILURE);
   }

   if (signal(SIGUSR1, ssu_signal_handler2) == SIG_ERR) {
      fprintf(stderr, "cannot handle SIGUSR1\n");
      exit(EXIT_FAILURE);
   }

   raise(SIGINT);
   raise(SIGUSR1);
   printf("main return\n");
   exit(0);
}
```

```
void ssu_signal_handler1(int signo) {
   printf("SIGINT 시그널 발생\n");
}

void ssu_signal_handler2(int signo) {
   printf("SIGUSR1 시그널 발생\n");
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_raise
SIGINT 시그널 발생
SIGUSR1 시그널 발생
main return
```

alarm(2), pause(2)

#include <unistd.h>
unsigned int alarm(unsigned int seconds);
리턴 값: 이전에 설정된 타이머가 없다면 0을 리턴, 그렇지 않다면 이전에 설정된 타이머가 만료될 때까지의 시간(초)
#include <unistd.h>
int pause(void);
리턴 값: 시그널이 캐치되거나 시그널 핸들러 함수가 리턴되었을 때, -1을 리턴하고 errno가 EINTR로 설정됨

- alarm()
  - 프로세스에 SIGALRM을 전달하는 시스템호출 함수
  - Seconds : 타이머로 사용될 초단위의 값
  - 타이머가 만료되면 SIGALRM 시그널이 발생 -> 디폴트 액션 : 프로세스를 종료
- pause()
  - 시그널이 발생할 때까지 실행 중이던 프로세스를 대기상태로 만드는 시스템호출
  - 액션이 프로세스의 종료이거나 시그널 핸들러의 호출인 시그널이 발생할 때까지 프로세스를 기다림
    - (1) 액션이 시그널 핸들러라면 시그널의 전달로 시그널 핸들러가 호출되고, 리턴한 후에 pause()가 리턴됨
    - (2) 액션이 프로세스의 종료라면 pause()는 리턴되지 않음

alarm() 예제 1

<ssu_alarm_1.c>

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

void ssu_signal_handler(int signo);

int count = 0;

int main(void)
{
   signal(SIGALRM, ssu_signal_handler);
   alarm(1);

   while(1);

   exit(0);
}

void ssu_signal_handler(int signo) {
   printf("alarm %d\n", count++);
   alarm(1);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_alarm_1
alarm 0
alarm 1
alarm 2
alarm 3
^C
```

alarm() 예제 2

<ssu_alarm_2.c>

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

#define LINE_MAX 2048

static void ssu_alarm(int signo);

int main(void)
{
   char buf[LINE_MAX];
   int n;

   if (signal(SIGALRM, ssu_alarm) == SIG_ERR) {
      fprintf(stderr, "SIGALRM error\n");
      exit(1);
   }

   alarm(10);

   if ((n = read(STDIN_FILENO, buf, LINE_MAX)) < 0) {
      fprintf(stderr, "read() error\n");
      exit(1);
   }

   alarm(0);
   write(STDOUT_FILENO, buf, n);
   exit(0);
}
```

```
static void ssu_alarm(int signo) {
   printf("ssu_alarm() called!\n");
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_alarm_2
Hello, OSLAB!
Hello, OSLAB!
```

pause() 예제

<ssu_pause.c>

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

void ssu_alarm(int signo);

int main(void)
{
   printf("Alarm Setting\n");
   signal(SIGALRM, ssu_alarm);
   alarm(2);

   while (1) {
      printf("done\n");
      pause();
      alarm(2);
   }

   exit(0);
}

void ssu_alarm(int signo) {
   printf("alarm..!!!\n");
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_pause
Alarm Setting
done
alarm..!!!
done
alarm..!!!
done
alarm..!!!
done
^C
```

sigemptyset(3), sigfillset(3), sigaddset(3), sigdelset(3), sigismember(3)

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

```
int sigismember(const sigset_t *set, int signo);
```
리턴 값 : signo가 set의 멤버이면 1, 그렇지 않다면 0, 에러 시 –1을 리턴하고 errno가 설정됨

- 시그널 집합을 처리하는 라이브러리 함수
- sigemptyset()
  - set 인자가 가리키는 시그널 집합을 공집합으로 만듬
  - => 시그널 집합에서 모든 시그널을 제외시킴
- sigfillset()
  - set 인자가 가리키는 시그널 집합에 모든 시그널 포함시킴
  - 시그널 집합을 사용하는 응용 프로그램은 반드시 시그널 집합을 사용하기 전에 sigemptyset()나 sigfillset()를 호출해서 시그널 집합을 초기화 해야 함
- sigaddset()
  - signo에 대한 시그널을 시그널 집합에 추가
- sigdelset()
  - signo에 대한 시그널을 시그널 집합에서 제거
- sigismember()
  - signo가 시그널 집합에 포함되어 있는지 확인

sigemptyset(), sigaddset(), sigismember() 예제

```
<ssu_sigset.c>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

int main(void)
{
   sigset_t set;

   sigemptyset(&set);
   sigaddset(&set, SIGINT);

   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");
   }
```

```
 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);
}
```

실행 결과

```
root@localhost:/home/oslab# ./ssu_sigset
SIGINT is included.
SIGSYS is not included.
```