Signals and Signal Handling – Part 2

Some Other System Calls

- kill( ), raise( )
  - kill( ) sends a signal to a process or a group of process.
  - raise( ) sends a signal to the calling process itself.
- alarm( )
  - Be used to set a timer that will expire at a specified time in the future.
- pause( )
  - Be used to suspend the calling process until a signal is received.
- sigsetjmp( ), siglongjmp( )

kill (1)

- #include <sys/types.h>

  #include <signal.h>

  int kill(pid_t *pid*, int *sig*);
  - Used to send any signal to any process group or process.

kill (2)

- *pid*
  - *pid* > 0: signal *sig* is sent to *pid* (*individual*).
  - 0: *sig* is sent to all processes that belong to the same process group of the sender.
  - -1: if the effective user-id of the process is superuser, then *sig* is sent to every process except for the system processes. If not, sig is sent to all processes with a real user-id equal to the effective user-id of the sender.
  - *pid* < -1: *sig* is sent to every process with a process group-id equal to the absulute value of *pid*
- *sig* is 0
  - No signal is sent, but error checking is still performed.
- Return value
  - On success, zero is returned.
  - On error, -1 is returned.

Example #10: kill (1)

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

int ntimes = 0;


main()
{
        pid_t pid, ppid;
        void p_action (int), c_action (int);
        static struct sigaction pact, cact;


        /* 부모를 위해 SIGUSR1 행동을 지정한다. */
        pact.sa_handler = p_action;
        sigaction (SIGUSR1, &pact, NULL);
```

Example #10: kill (2)

```
switch (pid = fork()){
   case 1:            /* 오류 */
        perror ("synchro");
        exit (1);
  case 0:              /* 자식 */
        /* 자식을 위해 행동을 지정 */
        cact.sa_handler = c_action;
        sigaction (SIGUSR1, &cact, NULL);
        /* 부모의 프로세스 식별번호를 얻음. */
        ppid = getppid();
        for (;;)
        {
              sleep (1);
              kill (ppid, SIGUSR1);
              pause();
       }
        /* 결코 퇴장(exit) 않음. */
```

Example #10: kill (3)

```
    default:            /*  부모 */
        for(;;)
        {
            pause();
            sleep (1);
            kill (pid, SIGUSR1);
        } /* 결코 퇴장(exit) 않음 */
    }
}

void p_action (int sig)
{
  printf ("Parent caught signal #%d□n", ++ntimes);
}

void c_action (int sig)
{
  printf ("Child caught signal #%d□n", ++ntimes);
}
```

raise

- #include <signal.h>

  int raise (int *sig*);
  - Sends a signal to the current process.
  - Semantically equals to kill(getpid( ), sig).
  - Return value
    - 0 on success, nonzero for failure.

alarm

- #include <unistd.h>

  unsigned int alarm(unsigned int *seconds*);
  - Arranges for a SIGALRM signal to be delivered to the process in *seconds* seconds.
  - If *seconds* is zero, no new alarm is scheduled
  - In any event any previously set alarm is cancelled.
  - Return value
    - The number of seconds remaining until any previously scheduled alarm was due to be delivered.
    - Zero if there was no previously scheduled alarm.

pause

- #include <unistd.h>

  int pause(void);
  - Causes the invoking process to sleep until a signal is received.
  - Return value
    - Always returns -1.

Example #11: alarm (1)

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

#define TIMEOUT 5      /* in seconds */
#define MAXTRIES 5
#define LINESIZE 1024
#define CTRL_G '\007'  /* ASCII Bell */
#define TRUE 1
#define FALSE 0

/* used to see if timeout has occured */
static int timed_out;

/* will hold input line */
static char in_line[LINESIZE];
```

Example #11: alarm (2)

```
char *quickreply(char *prompt)
{
    void (*was)(int);
    void catch(int);
    int ntries;
    char *answer;

    /* catch SIGALRM + save previous action */
    was = signal(SIGALRM, catch);

    for(ntries = 0; ntries < MAXTRIES; ntries++) {
        timed_out = FALSE;
        printf("\n%s > ", prompt);

        /* set alarm clock */
        alarm(TIMEOUT);

        /* get input line */
        answer = gets(in_line);
```

Example #11: alarm (3)

```
        /* turn off alarm */
        alarm(0);

        /* if timed_out TRUE, then no reply */
        if(!timed_out)
            break;
    }

    /* restore old action */
    signal(SIGALRM, was);

    /* return appropriate value */
    return (ntries == MAXTRIES ? ((char *) 0) : answer);
}
```

Example #11: alarm (4)

```
/* executed when SIGALRM received */
void catch(int signo)
{
    /* set timeout flag */
    timed_out = TRUE;

    /* ring a bell */
    putchar(CTRL_G);

}
```

Example #11: alarm (5)

```
int main(void)
{
    char *name;

    if ((name = quickreply("Please enter your name")) != NULL)
        printf("Thank you, your name is %s\n", name);
    else
        printf("I give up!\n");
}
```

Example #12: pause (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#define TRUE   1
#define FALSE  0
#define BELLS  "□007□007□007"     /*ASCII bells */

int alarm_flag = FALSE;

/* SIGALRM을 처리할 루틴 */
void setflag (int sig)
{
      alarm_flag = TRUE;
}

main (int argc, char **argv)
{
      int nsecs, j;
      pid_t pid;
      static struct sigaction act;
```

Example #12: pause (2)

```
if ( argc<=2 )
{
        fprintf (stderr, "Usage:  tml #minutes message□n");
        exit (1);
}
if ((nsecs=atoi (argv[1] *60) <= 0)
{
       fprintf (stderr, " tml : invalid time□n");
       exit (2);
}
/* 백그라운드 프로세스를 생성하기 위해 fork한다. */
switch (pid = fork()){
   case -1:          /* 오류 */
       perror ("tml");
       exit (1);
   case 0:            /* 자식 */
       break;
   default:           /* 부모 */
       printf ("tml processid %d□n", pid);
       exit (0);
}
```

Example #12: pause (3)

```
    /* 얼람을 위한 행동을 지정한다. */
    act.sa_handler = setflag;
    sigaction (SIGALRM, &act, NULL);
    /* 얼람 시계를 켠다. */
    alarm (nsecs);
    /* 시그널이 올 때까지 중단(pause) ... */
    pause();
    /* 만일 시그널이 SIGALRM이면 메시지를 프린트하라. */
    if (alarm_flag == TRUE)
    {
        printf (BELLS);
        for (j = 2; j < argc; j++)
            printf ("%s", argv[j]);
        printf ("□n");
    }


    exit (0);
}
```

abort

- #include <stdlib.h>

  void abort(void);
  - Send the SIGABORT signal to the calling process, causing abnormal termination, i.e. a core dump.
  - If the abort( ) function causes program termination, all open streams are closed and flushed.
  - If the SIGABORT signal is ignored, then the ISO C implementation will still terminate the process, but other implementations may allow the abort( ) function to return to the caller.
  - Return value
    - The abort( ) function never returns.

sigsetjmp and siglongjmp

- #include <setjmp.h>

  int sigsetjmp(sigjmp_buf *env*, int *savemask*);

  void siglongjmp(sigjmp_buf *env*, int *val*);
  - Used for jumping from signal handler.
  - If *savemask* is non-zero, the current signal mask will be saved in *env* and restored when siglongjmp( ) is called from the signal handler.
  - Return value
    - sigsetjmp( ) returns 0 if called directly, non-zero (*val*) if returning from a call to siglongjmp( ).
    - siglongjmp( ) never return.

Example #13: siglongjmp (1)

```
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <time.h>
#include <sys/types.h>

static void sig_usr1(int), sig_alrm(int);
static void sigjmp_buf jmpbuf;
static volatile sig_atomic_t canjump;

void pr_mask(const char *str)
{
    sigset_t sigset;

    if (sigprocmask(0, NULL, &sigset) < 0) {
        perror("sigprocmask error");
        exit(1);
    }
```

Example #13: siglongjmp (2)

```
    printf("s%", str);
    if (sigismember(&sigset, SIGINT))
        printf("SIGINT");
    if (sigismember(&sigset, SIGQUIT))
        printf("SIGQUIT");
    if (sigismember(&sigset, SIGUSR1))
        printf("SIGUSR1");
    if (sigismember(&sigset, SIGALRM))
        printf("SIGALRM");

    /* remaining signals can go here */

    printf("\n");
}
```

Example #13: siglongjmp (3)

```
int main(void)
{
    if (signal(SIGUSR1, sig_usr1) == SIG_ERR) {
        perror("signal (SIGUSR1) error");
        exit(1);
    }
    if (signal(SIGALRM, sig_alrm) == SIG_ERR) {
        perror("signal (SIGALRM) error");
        exit(1);
    }
    pr_mask("starting main: ");

    if (sigsetjmp(jmpbuf, 1)) {
        pr_mask("ending main: ");
        exit(0);
    }
    canjump = 1;     /* now sigsetjmp() is OK */
    for (;;)
        pause();
}
```

Example #13: siglongjmp (4)

```
static void sig_usr1(int signo)
{
    time_t starttime;

    if (canjump == 0)
        return;     /* unexpected signal, ignore */

    pr_mask("starting sig_usr1: ");

    alarm(3);     /* SIGALRM in 3 seconds */

    starttime = time(NULL);
    for (;;)     /* busy wait for 5 seconds */
        if (time(NULL) > starttime + 5)
            break;

    pr_mask("finishing sig_usr1: ");
```

Example #13: siglongjmp (5)

```
    canjump = 0;
     siglongjmp(jmpbuf, 1); /* jump back to main, don't return */
}

static void sig_alrm(int signo)
{
     pr_mask("in sig_alrm: ");
     return;
}

dhlee@kde:~/Course/SP/example>a.out &
starting main:
[1] 21040
dhlee@kde:~/Course/SP/example>kill -USR1 21040
dhlee@kde:~/Course/SP/example>starting sig_usr1: SIGUSR1
in sig_alrm: SIGUSR1SIGALRM
finishing sig_usr1: SIGUSR1
ending main:

[1]    Done                          a.out
dhlee@kde:~/Course/SP/example>
```

More Examples #1: signal (1)

```
/* signal.c
   This program illustrate the use of the signal() system call.

   Three child processes are created in this program.
   In the first child, all signals are to be ignored except for SIGINT
   and SIGALRM. Each is handled by a different signal handler.
   After setting up the signals, this child process will go asleep to wait
   for a signal to wake it up, and then go back to sleep again.
   Note: Since it is an infinite loop, the child process will live forever
   till it is terminated manually or system reboot.
   In the second child, all signals are to be caught and handled by the same
   signal handler. After setting up the signals, this child process will go
   asleep waiting for a signal, and then this process will terminate after
   the signal has been handled.
   In the 3rd child, only SIGUSR2 will be caught. A new environment is setup
   in this child, which will later be used in its predefined signal handler to
   pass into a new program as its environment.
   In the parent process, various signals are sent to these three child
   processes.
*/
#include <stdio.h>
#include <signal.h>
```

More Examples #1: signal (2)

```
int pid;
static char *chdargv[] = {"p1", "p2", (char *)0};
static char *chdenv[4];

int main(int argc, char *argv[])
{
    int i, cpid1, cpid2, cpid3;
    void wakeup(int), handler(int), trapper(int), parent(int);

    if (!(cpid1 = fork())) {            /* 1st child */
        pid = cpid1 = getpid();         /* get pid for child 1 */
        printf("\nCPID1 = %d", cpid1);
        for (i = 1; i < NSIG; i++)
            signal(i, SIG_IGN);         /* ignore all signals */
        signal(SIGINT, handler);        /* except for interrupt */
        signal(SIGALRM, wakeup);        /* and alarm clock */
        alarm((unsigned)2);             /* set alarm for 2 secs */
        for (;;)
            pause();                    /* wait for signals */
        printf(" -- CPID1 (%d) terminates\n", cpid2); /* never gets here */
        exit(0);
    }
```

More Examples #1: signal (3)

```
else {
    if (!(cpid2 = fork())) {       /* 2nd child */
        pid = cpid2 = getpid();    /* get pid for child 2 */
        printf("\n\tCPID2 = %d", cpid2);
        for (i = 1; i < NSIG; i++)
            signal(i, trapper);    /* to trap signals */
        pause();
        printf(" -- CPID2 (%d) terminates\n", cpid2);
        exit(0);
    }
    else {
        if (!(cpid3 = fork())) {         /* 3rd child */
            pid = cpid3 = getpid();      /* get pid for child 3 */
            printf("\n\t\tCPID3 = %d ", cpid3);
            signal(SIGUSR2, trapper);

            chdenv[0] = "HOME=here";
            chdenv[1] = "PATH=/bin";
            chdenv[2] = "MAILX=/usr/lib/xxx";
            chdenv[3] = (char *)0 ;

            pause();
```

More Examples #1: signal (4)

```
    printf(" -- CPID3 (%d) terminates\n");
    exit(0);
}
                        /* parent process */
pid = getpid();           /* pid for parent */
sleep(3);                /* to let child run first */
printf("\nThis is parent process (pid = %d)\n", pid);

for (i = 1; i < NSIG; i++)
    signal(i, parent);    /* catch all signals */

printf("\n\tSend SIGBUS(%d) to CPID1 (%d)", SIGBUS, cpid1);
kill(cpid1, SIGBUS);

printf("\n\tSend SIGINT(%d) to CPID1 (%d)", SIGINT, cpid1);
kill(cpid1, SIGINT);
printf("\n\t\tSend SIGBUS(%d) to CPID2 (%d)", SIGBUS, cpid2);
kill(cpid2, SIGBUS);

printf("\n\t\tSend SIGTERM(%d) to CPID2 (%d)", SIGTERM, cpid2);
kill(cpid2, SIGEMT);
```

More Examples #1: signal (5)

```
            printf("\n\t\tSend SIGUSR1(%d) to CPID2 (%d)", SIGUSR1, cpid2);
            kill(cpid2, SIGUSR1);

            printf("\n\t\t\tSend SIGUSR2(%d) to CPID3 (%d)", SIGUSR2, cpid3);
            kill(cpid3, SIGUSR2);
            wait((int *)0);
        }
    }
    return(0);
} /* main */

void wakeup(int dummy)
{
    printf("\nI (pid = %d) am up now\n", pid);
}  /* wakeup */

void handler(int dummy)
{
    printf("\nI (pid = %d) got an interrupt; will continue\n", pid);
}  /* handler */
```

More Examples #1: signal (6)

```
void trapper(int i)
{
    signal(i, trapper);    /* old style; don't reset to default */
    if (i == SIGUSR1) {
        printf("\n\tGot SIGUSR1(%d); process(%d) will terminate\n", i, pid);
        exit(2);
    }
    else {
        if (i == SIGUSR2) {
            printf("\n\t\t(%d) got SIGUSR2(%d); execs a new program", i, pid);
            execve("./sigexec", chdargv, chdenv);
            perror("\nTHIS LINE SHOULDN'T BE HERE\n");
        }
        else
            printf("\n\tGot a signal(%d); process(%d) continues\n", i, pid);
    }
} /* trapper */

void parent(int sig)
{
    printf("\nSignal (%d) received by parent (%d)", sig, pid);
} /* parent */
```

More Examples #1: signal (7)

```
/* sigexec.c
   This program is used to print the commandline arguments' and current
   environment variables' vaules. Note: It is called from signal.c
*/

extern char **environ;

main(int argc, char * argv[])
{
    char **env;
    int i;

    printf("\nParameters are:\n");
    for (i=0; i<argc; i++)
        printf("%2d: %s\n", i, argv[i]);

    printf("\nEnviroment variable are:\n");
    for (env = environ; *env != (char *)0; env++)
        printf(" %s\n", *env);
    return(0);
} /* sigexec */
```

More Examples #2: sigaction (1)

```
/* sigaction.c
   This program is similar to the previous signal.c program; however, the new
   system call sigaction() is used instead.
   There is a new structure named sigaction (to confuse users). It is composed
   of three fields:
      struct sigaction {
        int      sa_flags;          ** signal options **
        void     (*sa_handler)();   ** addr of signal handler **
        sigset_t sa_mask;           ** additional signals to block **

   When the program is first started, it will create and initialize a signal
   mask to unblock all signals. It then will create a sigaction structure
   to be used for handling individual signals.

   Then, three child processes will be created. The function of each child
   process is similar to that in the previous signal.c program. Also the
   function of the parent is similar too.
*/
#include <stdio.h>
#include <signal.h>
```

More Examples #2: sigaction (2)

```
int pid;
static char *chdargv[] = {"p1", "p2", (char *)0};
static char *chdenv[4];

int main (int argc, char *argv[])
{
    int i, cpid1, cpid2, cpid3;
    struct sigaction action, old_action;
    void wakeup(), handler(), trapper(int), parent(int);
    sigset_t sigmask;                   /* a process-wide signal mask */

    sigemptyset(&sigmask);              /* to empty bits in sigmask */
    sigprocmask(SIG_SETMASK, &sigmask, 0);  /* to initialize signal mask */
    action.sa_flags   = 0;              /* to init signal flags */
    action.sa_handler = SIG_IGN;        /* to ignore signal */
    sigemptyset(&action.sa_mask);       /* zero out sa_mask */

    if (!(cpid1 = fork())) {            /* 1st child */
        pid = cpid1 = getpid();         /* get pid for child 1 */
        printf("\nCPID1 = %d", cpid1);
        for (i = 1; i < NSIG; i++)
            sigaction(i, &action, &old_action);  /* ignore all signals */
```

More Examples #2: sigaction (3)

```
     sigaddset(&action.sa_mask, SIGINT);   /* add SIGINT to sa_mask **
        ** implicitly done, not needed, just for illustration purpose */
    action.sa_flags  |= SA_RESTART;        /* sys calls auto restart */
    action.sa_handler = handler;           /* user-defined sig handler */
    sigaction(SIGINT, &action, &old_action); /* SIGINT is also blocked */

    sigaddset(&action.sa_mask, SIGALRM);     /* again, for illustration */
    action.sa_handler = wakeup;              /* another user-defined */
    sigaction(SIGALRM, &action, &old_action);/* SIGALRM is also blocked */
    alarm((unsigned)2);              /* set alarm for 2 secs */
    for (;;)
        pause();                     /* wait for signals */
    printf(" -- CPID1 (%d) terminates\n", cpid2); /* never gets here */
    exit(0);
}
else {
    if (!(cpid2 = fork())) {       /* 2nd child */
        pid = cpid2 = getpid();    /* get pid for child 2 */
        printf("\n\tCPID2 = %d", cpid2);
        action.sa_handler = trapper;        /* one more user-defined */
        for (i = 1; i < NSIG; i++)
            sigaction(i, &action, &old_action);
```

More Examples #2: sigaction (4)

```
    pause();

    printf(" -- CPID2 (%d) terminates\n", cpid2);
    exit(0);
}
else {
    if (!(cpid3 = fork())) {         /* 3rd child */
        pid = cpid3 = getpid();      /* get pid for child 3 */
        printf("\n\t\tCPID3 = %d ", cpid3);

        sigaddset(&action.sa_mask, SIGUSR2);  /* block sig SIGUSR2 */
        action.sa_handler = trapper;
        sigaction(SIGUSR2, &action, &old_action);

        chdenv[0] = "HOME=here";
        chdenv[1] = "PATH=/bin";
        chdenv[2] = "MAILX=/usr/lib/xxx";
        chdenv[3] = (char *)0 ;

        pause();

        printf(" -- CPID3 (%d) terminates\n");
        exit(0);
```

More Examples #2: sigaction (5)

```
                        /* parent process */
    pid = getpid();     /* pid for parent */
    sleep(3);           /* to let child run first
    printf("\nThis is parent process (pid = %d)\n", pid);

    sigfillset(&action.sa_mask);      /* to block all signals */
    action.sa_handler = parent;       /* all goes to parent */
    for (i = 1; i < NSIG; i++)
        sigaction(i, &action, &old_action); /* catch all signals */
    printf("\n\tSend SIGBUS(%d) to CPID1 (%d)", SIGBUS, cpid1);
    kill(cpid1, SIGBUS);
    printf("\n\tSend SIGINT(%d) to CPID1 (%d)", SIGINT, cpid1);
    kill(cpid1, SIGINT);
    printf("\n\t\tSend SIGBUS(%d) to CPID2 (%d)", SIGBUS, cpid2);
    kill(cpid2, SIGBUS);
    printf("\n\t\tSend SIGTERM(%d) to CPID2 (%d)", SIGTERM, cpid2);
    kill(cpid2, SIGTERM);
    printf("\n\t\tSend SIGUSR1(%d) to CPID2 (%d)", SIGUSR1, cpid2);
    kill(cpid2, SIGUSR1);
    printf("\n\t\t\tSend SIGUSR2(%d) to CPID3 (%d)", SIGUSR1, cpid3);
    kill(cpid3, SIGUSR2);
    wait((int *)0);
}
```

More Examples #2: sigaction (6)

```
    }
    return(0);
} /* main */

void wakeup()
{
    printf("\nI (pid = %d) am up now\n", pid);
}  /* wakeup */

void handler()
{
    printf("\nI (pid = %d) got an interrupt; will continue\n", pid);
}  /* handler */

void trapper(int i)
{
    /* No reset is needed any more */
    if (i == SIGUSR1) {
        printf("\n\tGot SIGUSR1(%d); process(%d) will terminate\n", i, pid);
        exit(2);
    }
```

More Examples #2: sigaction (7)

```
    else {
        if (i == SIGUSR2) {
            printf("\n\t\t(%d) got SIGUSR2(%d); execs a new program", i, pid);
            execve("./sigexec", chdargv, chdenv);
            perror("\nTHIS LINE SHOULDN'T BE HERE\n");
        }
        else
            printf("\n\tGot a signal(%d); process(%d) continues\n", i, pid);
    }
} /* trapper */

void parent(int sig)
{
    printf("\nSignal (%d) received by parent (%d)", sig, pid);
} /* parent */
```

Lab Assignment #1

- For each sample program, briefly describe what it does, how it does, and why.
- Identify any programming error if it exists, and show how the error can be fixed.