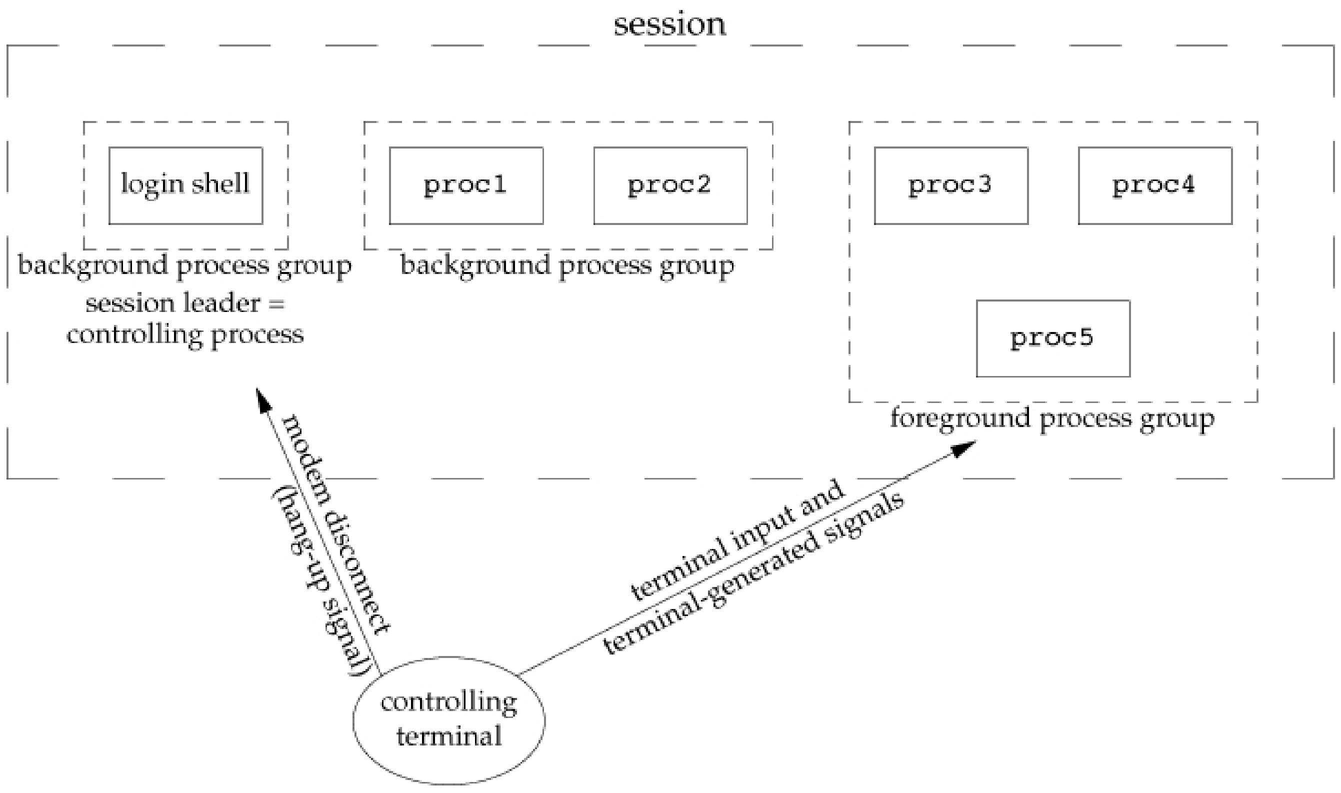
Signals

Process groups, sessions, controlling terminal

After issuing the following commands:

```
proc1 | proc2 &
proc3 | proc4 | proc5
```

You have:



Sending signals

```
#include <signal.h>

int kill(pid_t pid, int signo);

int raise(int signo);

        Both return: 0 if OK, –1 on error
```

- if pid < 0, the signal is sent to the process group with pgid == | pid |

SIGALRM

alarm() and pause() functions

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
        Returns: 0 or number of seconds until previously set alarm

int pause(void);
        Returns: –1 with errno set to EINTR
```

- `alarm(0)` cancels the previous alarm if one was set
- alarm() & pause() can be used to implement `sleep()`, but there are many subtleties

Waking up slow system calls using alarm()

```
#include "apue.h"

static void sig_alrm(int);

int
main(void)
{
    int     n;
    char    line[MAXLINE];

    if (signal(SIGALRM, sig_alrm) == SIG_ERR)
        err_sys("signal(SIGALRM) error");

    alarm(10);
    if ((n = read(STDIN_FILENO, line, MAXLINE)) < 0)
        err_sys("read error");
    alarm(0);

    write(STDOUT_FILENO, line, n);
    exit(0);
}

static void
sig_alrm(int signo)
{
    /* nothing to do, just return to interrupt the read */
}
```

Two problems:

1. A race condition: the alarm can be missed between `alarm(10)` and `read()`
2. This doesn't work if slow system calls are automatically restarted

Using setjmp & longjmp to solve the two problems (Optional material)

```
#include "apue.h"
#include <setjmp.h>

static void     sig_alrm(int);
static jmp_buf  env_alrm;

int
main(void)
{
    int     n;
    char    line[MAXLINE];

    if (signal(SIGALRM, sig_alrm) == SIG_ERR)
        err_sys("signal(SIGALRM) error");
    if (setjmp(env_alrm) != 0)
        err_quit("read timeout");

    alarm(10);
    if ((n = read(STDIN_FILENO, line, MAXLINE)) < 0)
        err_sys("read error");
    alarm(0);

    write(STDOUT_FILENO, line, n);
    exit(0);
}

static void
sig_alrm(int signo)
{
    longjmp(env_alrm, 1);
}
```

Signal sets

```
#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);

        All four return: 0 if OK, –1 on error

int sigismember(const sigset_t *set, int signo);

        Returns: 1 if true, 0 if false, –1 on error
```

A possible implementation:

```
#define sigemptyset(ptr)  (*(ptr) = 0)
#define sigfillset(ptr)   (*(ptr) = ~(sigset_t)0, 0)

/*
 * <signal.h> usually defines NSIG to include signal number 0.
 */
#define SIGBAD(signo)   ((signo) <= 0 || (signo) >= NSIG)

int sigaddset(sigset_t *set, int signo)
{
    if (SIGBAD(signo)) {
        errno = EINVAL;
        return(-1);
    }
    *set |= 1 << (signo - 1);       /* turn bit on */
    return(0);
}

int sigdelset(sigset_t *set, int signo)
{
    if (SIGBAD(signo)) {
        errno = EINVAL;
        return(-1);
    }
    *set &= ~(1 << (signo - 1));    /* turn bit off */
    return(0);
}

int sigismember(const sigset_t *set, int signo)
{
    if (SIGBAD(signo)) {
        errno = EINVAL;
        return(-1);
    }
    return((*set & (1 << (signo - 1))) != 0);
}
```

sigprocmask() & sigpending() functions

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *restrict set,
                sigset_t *restrict oset);

        how: SIG_BLOCK, SIG_UNBLOCK, or SIG_SETMASK

        Returns: 0 if OK, −1 on error

int sigpending(sigset_t *set);

        Returns: 0 if OK, −1 on error
```

Example:

```
#include "apue.h"

static void sig_quit(int);

int
main(void)
{
    sigset_t    newmask, oldmask, pendmask;

    if (signal(SIGQUIT, sig_quit) == SIG_ERR)
        err_sys("can´t catch SIGQUIT");

    /*
     * Block SIGQUIT and save current signal mask.
     */
    sigemptyset(&newmask);
    sigaddset(&newmask, SIGQUIT);
    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
        err_sys("SIG_BLOCK error");

    sleep(5);   /* SIGQUIT here will remain pending */

    if (sigpending(&pendmask) < 0)
        err_sys("sigpending error");
    if (sigismember(&pendmask, SIGQUIT))
        printf("\nSIGQUIT pending\n");

    /*
     * Restore signal mask which unblocks SIGQUIT.
     */
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
        err_sys("SIG_SETMASK error");
    printf("SIGQUIT unblocked\n");

    sleep(5);   /* SIGQUIT here will terminate with core file */
    exit(0);
}

static void
sig_quit(int signo)
{
    printf("caught SIGQUIT\n");
    if (signal(SIGQUIT, SIG_DFL) == SIG_ERR)
        err_sys("can´t reset SIGQUIT");
}
```

sigaction() function

```
#include <signal.h>

int sigaction(int signo, const struct sigaction *restrict act,
              struct sigaction *restrict oact);

        Returns: 0 if OK, −1 on error

struct sigaction {
  void     (*sa_handler)(int);  /* addr of signal handler, */
                                /* or SIG_IGN, or SIG_DFL */
  sigset_t sa_mask;             /* additional signals to block */
  int      sa_flags;            /* signal options, Figure 10.16 */
  /* alternate handler */
  void     (*sa_sigaction)(int, siginfo_t *, void *);
};
```

An implementation of `signal()` using sigaction:

```
#include "apue.h"

/* Reliable version of signal(), using POSIX sigaction().  */
Sigfunc *
signal(int signo, Sigfunc *func)
{
    struct sigaction    act, oact;

    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    if (signo == SIGALRM) {
#ifdef  SA_INTERRUPT
        act.sa_flags |= SA_INTERRUPT;
#endif
    } else {
        act.sa_flags |= SA_RESTART;
    }
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```

1. Signal handler remains in place
2. Interrupted system calls automatically restart, except for SIGALRM

signal_intr() – an alternate version that does not restart system calls:

```
#include "apue.h"

Sigfunc *
signal_intr(int signo, Sigfunc *func)
{
    struct sigaction    act, oact;

    act.sa_handler = func;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
#ifdef SA_INTERRUPT
    act.sa_flags |= SA_INTERRUPT;
#endif
    if (sigaction(signo, &act, &oact) < 0)
        return(SIG_ERR);
    return(oact.sa_handler);
}
```

Last updated: 2014–02–13