Signals (Chapter 10)

Software Interrupts

"Most nontrivial application programs need to deal with signals"

asyncronous events

Version 7 signals -- Not reliable
- Signal could "get lost"

BSD -- changes for reliable signals
- Changes were incompatible

POSIX -- also their concept of signals

Signal Basics

Signal Names:

- SIGINT - interrupt program
- SIGSEGV - segmentation violation
- SIGTSTP - stop signal from terminal
- SIGCHLD - child status has changed
- man 7 signal

Signal causes:

- Terminal generated
  - ^C - often SIGINT
  - ^Z - often SIGTSTP

- Harware generated
  - Divide by zero - SIGFPE (example divzero.c)
  - Bad pointer ref - SIGSEGV
  - Unaligned access - SIGBUS (example buserr.c)

Signal Basics (page 2)

More Signal causes:

- kill system call
  - int kill(pid_t pid, int sig);
  - pid > 0 => to that process
  - pid = 0 => to process group of sender
  - pid = -1 => All processes (except sender)
    - root -> all but system processes
    - !root -> all with same uid
  - root can signal any process
  - !root can only signal process with same uid

- kill user level command
  - Sometimes built into shells (bash)
  - Same as above

Signal Basics (page 3)

More Signal causes:

Other indications

- SIGURG -- Network related
- SIGPIPE -- Write to a pipe with no reader
- SIGALRM -- "Alarm Clock" went off
- SIGCHLD -- Child change of status

What happens at "signal time"?

Signal gets "Delivered" to the process

Actions ...

- Ignore the signal -- nothings happens
  - (Can’t ignore SIGKILL and SIGSTOP)
- Catch the signal
  - Starts a designated function
  - (Can’t catch SIGKILL and SIGSTOP)
- Default action
  - May ignore it
  - May terminate the process
  - May dump core and terminate process

Again ... look at "man 7 signal"

How to use:

Simple version (unreliable):

```
void (*signal(int sig, void (*func)(int));)(int)
```

- func -> function name OR
  - SIG_DFL
  - SIG_IGN
- sig -> Signal Name
- return -> previous function pointer (or SIG_DFL or SIG_IGN)

Example: sig.c

Other issues:

system calls may be interrupted by signals
EINTR is an error code for an interrupted system call

Other signal related calls

- raise(3)
- alarm(3) / setitimer(2)
- pause(3) / sigsuspend(2)
- abort(3)

Use of system calls in handler!

- Save errno at least!
- Don’t use routines like malloc!
- How about printf?
  - Not a good idea!

"Advanced" signal interface

```
#include <signal.h>
struct sigaction {
        void    (*sa_handler)(int);
        sigset_t sa_mask;
        int     sa_flags;     };

int sigaction(int sig, const struct sigaction *act,
              struct sigaction *oact);
```

- sa_mask -- a "set" of signals to "block" during handler running.
  - Routines to make signal sets:
    - sigemptyset, sigfillset, sigaddset, sigdelset, sigismember
- sa_flags -- Controls other things
  - SA_RESTART -- restart system calls that can be restarted
  - Others ... not that important here

"Advanced" signal interface (page 2)

Example: sigaction.c

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

- Block/unblock the current set of signals from being delivered.

```
int  sigpending(sigset_t *set);
```

- Returns set of signals waiting (blocked) to be delivered

```
int sigsuspend(const sigset_t *sigmask);
```

- Wait for a signal to be delivered. sigmask normally empty.

sigsetjmp / siglongjmp

- setjmp and longjmp that deals with signals.

Signal Set operations

From "man sigsetops"

```
#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);

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