Unix Signals (CS422S)

Ken Wong
Washington University
kenw@wustl.edu
www.arl.wustl.edu/~kenw

THE SIGNAL CONCEPT

- Signals are software interrupts
- Provides a way of handling asynchronous events
  - e.g., User presses interrupt key to stop a program
  - e.g., A process in a pipeline terminates prematurely
- Every signal has a name and a positive integer
  - `<signal.h>` defines mapping between names and numbers
  - `SIGINT`: User presses DELETE (interrupt) key
  - `SIGPIPE`: Process writes to a pipe after reader has quit
  - `SIGALRM`: Alarm clock expires
  - `SIGCHLD`: Child process terminates
  - `SIGTERM`: Terminate (kill) process (See also `SIGKILL`)
  - `SIGSTOP`: Suspend (Ctrl-Z) process (See also `SIGSTK`)
**CODE EXAMPLE 1**

```java
static void handle_alarm (int signo) {
    ... code to handle alarm ...
    return;  // or exit
}
int main (int argc, char *argv[]) {
    struct sigaction action, oaction;
    sigemptyset (&action.sa_mask);   // block all signals
    action.sa_flags = 0;             // no signal options
    action.sa_handler = handle_alarm;  // the handler code
    if (sigaction(SIGALRM, &action, &oaction) < 0) Err...
    for (int i=0; i<5; i++) {
        alarm(2);         // send SIGALRM in 2 sec
        pause();          // suspend until get signal
        printf("I woke up\n");
    }
    return 0;
}
```

**SIGNAL RELATED FUNCTIONS**

- **kill**: Send a signal to a process
- **raise**: Send a signal to itself
- **alarm**: Set a timer that will expire in the future
- **pause**: Suspend calling process until a signal is delivered to process
- **sigemptyset**: Exclude all signals in signal set
- **sigismember**: Return 1 if a signal is in a signal set
- **sigprocmask**: Examine/modify which signals are blocked from deliver to a process
- **sigpending**: Return blocked/pending signal set
- **sigaction**: Examine/modify action for a signal

**SIGNAL SETS**

- **A signal set** is a data type (sigset_t) used to represent multiple signals
  - Each signal is represented by a single bit
  - Solaris uses 128 bits
- **Initialize signal set by first turning all bits either off or on (use sigemptyset or sigfillset)
- **Then add and delete specific signals in the set (use sigaddset or sigdelset)**

**sigprocmask FUNCTION**

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

- Returns 0 if OK, -1 otherwise
- If oset is nonnull, current signal mask is retured through oset
- If set is nonnull, how indicates how signal mask is modified
  - SIG_BLK: New mask is union of old and new mask
  - SIG_UNBLOCK: set indicates signals to unblock
  - SIG_SET: New mask will be in set
- One of the pending, unblocked signals will be delivered after return from sigprocmask
**sigprocmask EXAMPLE**

```c
void printMask(void) {
    sigset_t sigset;
    int old_errno;

    old_errno = errno; // if called by sig hndlr
    if (sigprocmask(0, NULL, &sigset) < 0) Err...
    if (sigismember(&sigset, SIGALRM))
        printf("SIGALRM will be handled\n");
    if (sigismember(&sigset, SIGVTALRM))
        printf("SIGVTALRM will be handled\n");
    if (sigismember(&sigset, SIGUSR1))
        printf("SIGUSR1 will be handled\n");
    errno = old_errno; // restore
}
```

**sigaction FUNCTION**

```c
int sigaction (int signo,
               const struct sigaction *action,
               struct sigaction *oaction);
```

- Return 0 if OK; -1 otherwise
- signo: Signal number to be examined/modified
- action: If nonnull, we are modifying the action
- oaction: If nonnull, return previous signal action
- struct sigaction members
  - void (*sa_handler)(): Signal handler, SIG_IGN or SIG_DFL
  - sigset_t *sa_mask: Additional signals to block before calling signal handler (restore old mask after return)
  - int sa_flags: Signal options (e.g., SA_RESTART)