Signals

Haviland – Ch. 6
Signals

• Unexpected/unpredictable asynchronous events
  – floating point error
  – death of a child
  – interval timer expired (alarm clock)
  – control-C (termination request)
  – control-Z (suspend request)

• Events are called interrupts

• When the kernel recognizes an event, it sends a signal to the process.

• Normal processes may send signals.
What are signals for?

- When a program forks into 2 or more processes, rarely do they execute independently.
- The processes usually require some form of synchronization, often handled by signals.
- To transfer data between processes, we will use pipes and sockets (coming soon).
- Signals are generated by
  - machine interrupts
  - the program itself, other programs or the user.
Software Interrupts

- `<sys/signal.h>` lists the signal types on CDF.
- “man 7 signal” (“man 5 signal” on Solaris) gives some description of various signals
  - SIGTERM, SIGABRT, SIGKILL
  - SIGSEGV, SIGBUS
  - SIGSTOP, SIGCONT
  - SIGCHLD
  - SIGPIPE
  - SIGUSR1, SIGUSR2
Signal handlers

• When a C program receives a signal, control is immediately passed to a function called a signal handler.

• The signal handler function can execute some C statements and exit in 3 different ways:
  – return control to the place in the program which was executing when the signal occurred.
  – return control to some other point in the program.
  – terminate the program by calling exit.
Default actions

- Each signal has a default action:
  - terminate
  - stop
  - ignore
- The default action can be changed for most signal types using the `sigaction()` function. The exceptions are SIGKILL and SIGSTOP.
Signal table

- For each process, Unix maintains a table of actions that should be performed for each kind of signal.
- Here are a few...

<table>
<thead>
<tr>
<th>Signal</th>
<th>Default Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGINT</td>
<td>Terminate</td>
<td>Interrupt from keyboard</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>Terminate/Dump core</td>
<td>Invalid memory reference.</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>Terminate</td>
<td>Kill</td>
</tr>
<tr>
<td></td>
<td>(cannot ignore)</td>
<td></td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>Ignore</td>
<td>Child stopped or terminated.</td>
</tr>
<tr>
<td>SIGSTOP</td>
<td>Stop (cannot ignore)</td>
<td>Stop process.</td>
</tr>
<tr>
<td>SIGCONT</td>
<td></td>
<td>Continue if stopped.</td>
</tr>
</tbody>
</table>
**sigaction()**

- Install a signal handler, `act`, for the signal `sig`.
  ```c
  int sigaction(int sig,
               const struct sigaction *act,
               struct sigaction *oldact);
  ```

- Struct defined in `<signal.h>` to fill in to pass in for `act`.
  ```c
  struct sigaction {
      /* SIG_DFL, SIG_IGN, or pointer to function */
      void (*sa_handler)(int);
      sigset_t sa_mask; /* Signals to block during handler */
      int sa_flags; /* flags and options */
  };
  ```

- You may come across various extensions, including another field in the `sigaction` struct for a function to catch signals.
• Run the program and try sending different signals to it.
Sending a signal

- From the command line use
  \texttt{kill [-signal] pid [pid]...}
- If no signal is specified, \texttt{kill} sends the \texttt{TERM} signal to the process.
- Signal can be specified by the number or name without the \texttt{SIG}.
- Examples:
  \texttt{kill -QUIT 8883}
  \texttt{kill -STOP 78911}
  \texttt{kill -9 76433} \quad (9 \text{ == KILL})
Signalling between processes

• One process can send a signal to another process using the misleadingly named function call.

        kill(int pid, int sig);

• This call sends the signal \texttt{sig} to the process \texttt{pid}

• Signalling between processes can be used for many purposes:
  – kill errant processes
  – temporarily suspend execution of a process
  – make a process aware of the passage of time
  – synchronize the actions of processes.
Timer signals

• Three interval timers are maintained for each process:
  – SIGALRM (real-time alarm, like a stopwatch)
  – SIGVTALRM (virtual-time alarm, measuring CPU time)
  – SIGPROF (used for profilers)
• Useful functions to set and get timer info:
  – sleep() – cause calling process to suspend.
  – usleep() – like sleep() but at a finer granularity.
  – alarm() – sets SIGALRM
  – pause() – suspend until next signal arrives
  – setitimer(), getitimer()
• sleep() and usleep() are interruptible by other signals.
Blocking Signals

• Signals can arrive at any time.
• To temporarily prevent a signal from being delivered we block it.
• The signal is held until the process unblocks the signal.
• When a process ignores a signal, it is thrown away.
Groups of signals

• Signal masks are used to store the set of signals that are currently blocked.

• Operations on sets of signals:

```c
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(const sigset_t *set, int signo);
```
The function `sigprocmask()` modifies the signal mask and has the following signature:

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

- **how** indicates how the signal will be modified:
  - `SIG_BLOCK`: add to those currently blocked
  - `SIG_UNBLOCK`: delete from those currently blocked
  - `SIG_SETMASK`: set the collection of signals being blocked

- **set** points to the set of signals to be used for modifying the mask

- **oset** on return holds the set of signals that were blocked before the call.