Recitation 13
12-7-2015
Today

- Signals
  - Signal handlers
  - Alarms
  - Exercises

- Clarification and questions about PA4
Signals

What are they?
Notifications sent from the OS or programs

Why are they useful?
Listen and catch signals to do something (alarms)

What should we do with them?
Send, listen, ignore, handle
Send a signal

- Signals can be sent to a process with:
  ```c
  int kill(int pid, int sig)
  ```
- Each signal is identified by a number.
- When a signal is received the following can happen:
  1) default action
  2) signal is ignored or blocked
  3) signal is caught and handled
# Some signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
<th>Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGHUP</td>
<td>1</td>
<td>Term</td>
<td>Hangup detected on controlling terminal or death of controlling process</td>
</tr>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>Term</td>
<td>Interrupt from keyboard</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>3</td>
<td>Core</td>
<td>Quit from keyboard</td>
</tr>
<tr>
<td>SIGILL</td>
<td>4</td>
<td>Core</td>
<td>Illegal Instruction</td>
</tr>
<tr>
<td>SIGABRT</td>
<td>6</td>
<td>Core</td>
<td>Abort signal from abort(3)</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>8</td>
<td>Core</td>
<td>Floating point exception</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>Term</td>
<td>Kill signal (cannot be caught, blocked, or ignored)</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>Core</td>
<td>Invalid memory reference</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>13</td>
<td>Term</td>
<td>Broken pipe: write to pipe with no readers</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>14</td>
<td>Term</td>
<td>Timer signal from alarm(2)</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>Term</td>
<td>Termination signal</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>30,10,16</td>
<td>Term</td>
<td>User-defined signal 1</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>31,12,17</td>
<td>Term</td>
<td>User-defined signal 2</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Blocking Signals

- Each process has a signal mask, which indicates which signals are blocked.
- The signal mask can be set with: `int sigprocmask(int how, sigset_t *set, sigset_t *old)`.
- You can specify which signals to block, unblock or just reset the whole mask.
- There are functions that you can use to manipulate the set: `sigemptyset, sigaddset, sigismember`. 
Handle a signal

- First, define a handler function to execute when the signal arrives:
  ```c
  void handler(int signum)
  {
    printf("Signal has arrived!\n")
    ...
  }
  ```
- Once a signal handler function is defined, you can install a signal handler using the sigaction call.
sigaction signature

```c
struct sigaction new_action;
new_action.sa_handler = handler;
new_action.sa_flags = 0; //special options
sigsetempty(&new_action.sa_mask);
//signals to block
sigaction(SIGQUIT, &new_action, NULL);
```
sigaction alternative

`sa_handler` only takes a signal number as argument

`sa_sigaction` allows you to specify a handle function which takes in more than just the signal number (takes signal and 2 pointers)

Do not assign to both `sa_handler` and `sa_sigaction`.

(check out “man sigaction”)
Signal Handler Example

Example handler.c
SIGINT Exercise

- Write a program (longCount.c) that catches the SIGUSR1 & SIGUSR2 signals.
- In another process SIGUSR1 & SIGUSR2 are thrown.
SIGALRM

- SIGALRM is a signal sent to a process once a time limit has elapsed.
- Defined in the signal.h header.
- Can be used to make a long running action time out or to perform an action periodically.
- Can be scheduled from alarm(int seconds) function.
- SIGALRM and alarm are used to implement Sleep().
SIGALRM Exercise

- Write a signal handler that handles SIGALRM and prints “Woke up for the # time”
- Call alarm(5) three times.
PA4 Clarifications

- Print the following messages to the screen

Thread 1 is handling client 127.0.0.1,XXXXX
Thread 1 finished handling client 127.0.0.1,XXXXX
- XXXXX should be the port number which is allocated by OS after accepting the connection.

- Print the following messages to the screen to monitor your program behavior

server listens
client connects
server accepts connection
server sends handshaking: (100,0,)
client sends handshake response: (101,0,)
client sends twitterTrend request: (102,11,"Minneapolis")
server sends twitterTrend response: (103,14,"UMN,Lakes, Snow")
server sends end of response: (105,0,)
client sends end of request: (104,0,)
server closes the connection