Signals

Stanley Yao
Computer Science Department
University of Arizona

Problems

- There is a very complicated computation that takes a very very long time, in order to let the user know the program is working hard on the computation instead of hanging, we want to print a “.” on the screen every second while doing the computation.

Problems (cont.)

- In your program, you want to do some computation and also want to receive socket packages coming from the network. You don’t want your program stupidly sitting there solely listening to the socket port. Instead, you want it to do the computation while there is no socket packages coming in, and whenever a package comes in you put the computation aside for a while and process the package first.

Signal

- A UNIX mechanism to notify a process about program faults and external events.
  - Program faults: divided by zero, segmentation violation, bus error, etc.
  - External events: process termination, I/O available, broken pipe, etc.
- Signals: Many different signals identified by a unique integer and represent a particular event
- Signal handler: in a process, for each signal, there is a function that is called automatically by the system when the signal is received by the process. You can register your own singal handler, otherwise a default one will be used.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Number</th>
<th>Default Action</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGINT</td>
<td>2</td>
<td>Terminate</td>
<td>&quot;C</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>3</td>
<td>Terminate &amp; dump core</td>
<td>&quot;\</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>9</td>
<td>Terminate, can’t handle or ignore</td>
<td>&quot; Kill -9</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>11</td>
<td>Terminate</td>
<td>Segmentation violation</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>15</td>
<td>Terminate</td>
<td>Kill</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>Varies</td>
<td>Terminate</td>
<td>Program-defined meaning</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>14</td>
<td>Terminate</td>
<td>Alarm signal</td>
</tr>
<tr>
<td>SIGCHLD</td>
<td>Varies</td>
<td>Ignored</td>
<td>Child stopped or terminate</td>
</tr>
<tr>
<td>SIGIO</td>
<td>Varies</td>
<td>Terminate</td>
<td>I/O now available</td>
</tr>
</tbody>
</table>

Big Picture

- A signal is sent to the process
- The process receives the signal
- The SIGINT handler is called
- The process can then handle this signal
- Other signals can be handled similarly

OS

Process

SIGINT handler
SIGQUIT handler
SIGKILL handler
SIGIO handler
...
Setting Signal Handler

- typedef void (*sighandler_t)(int);
- sighandler_t signal(int signum, sighandler_t handler);
  - Install a new signal handler for the specified signal and returns the previous signal handler.
- handler
  - SIG_IGN: ignore
  - SIG_DFL: default action

Signal Mask

- int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
  - Change the signal mask and return the old mask by oset (if oset is not NULL)
- how
  - SIG_BLOCK: add set into the mask
  - SIG_UNBLOCK: remove set from the mask
  - SIG_SETMASK: replace the current mask with set

Signal-driven Socket

Further Readings

- Signals
  - Sample programs: http://www.kohala.com/start/apue.html

- Sockets
  - Sample programs: http://www.kohala.com/start/unpv12e.html

Acknowledgement