Signal5 Demonstration

Files

The files for this demonstration can be found in the rlogin cluster in the directory

```
/web/courses/cs3214/spring2014/butta/examples/signal-demo/signal5
```

The files are `esh-sys-utils.c esh-sys-utils.h loop.c Makefile sleep.c watch.c`

The `“make” command` by default will create three executable files named `watch, sleep, and loop.` The `watch` program creates and prints the process ids of two child processes. One child process runs the `sleep` executable; it waits for 30 second and then terminates normally. The other child process runs the `loop` executable; it executes an infinite loop and will never terminate normally. The `watch` program reports any signals that it receives about the state of its child processes. The `watch` program terminates when both of its child processes have terminated.

Purpose

The purposes of this demonstration are

- to see how a parent process can detect changes in the execution state of a child process
- to see how to send signals to a process to change its execution state

Steps

1. Use the `Makefile` to create the executable programs `watch, sleep,` and `loop` by using the command `“make”`.
2. Have two different `xterm` windows available to use. One of these will be referred to as the “`run`” window and the other will be referred to as the “`command`” window.
3. At the shell prompt in the `run` window execute the `watch` program. Observe the process ids output by the program.
4. At the shell prompt in the command window use the command
   ```
   ps –s
   ```
   This command will show all the processes running on both the two `xterm` windows. Observe the `watch`, `sleep`, and `loop` processes are shown. The column labeled `STAT` shows a code for the current state of each process. A code that begins `R` is a running process. A code that begins `T` is a stopped process. A code that begins with `S` is a sleeping process. Observe the state of the `watch`, `sleep`, and `loop` processes.
5. Allow the watch program to execute. After approximately 30 seconds you should see that the `watch` program reports the termination of a child process. Compare the reported process id with the process id reported earlier. Which process terminated? Confirm you answer by using the “`ps –s`” command in the `command` window.

6. In the `command` window send a signal to the `loop` process that causes this process to stop by using the command "`kill –SIGSTOP <loop pid>`" where "`loop pid`" is the process id of the loop process. This pid was reported by the `watch` program when it began and is also part of the information given by the `ps` command.
7. In the command window send a signal to the `loop` process that causes this process to resume execution by using the command "`kill –SIGCONT <loop pid>`" where "`loop pid`" is the process id of the loop process. Observe the output of the `watch` program in the `run` window. Run the "`ps –s`" command in the `command` window and observe the state of the loop process.
8. Repeat steps 6 and 7 several times.
9. Repeat steps 6 and 7 using the `SIGTSTP` signal instead of the `SIGSTOP` signal. Observe the state of the `loop` process and the output of the `watch` program at each step.
10. In the `command` window send a signal to the loop process that causes this process to terminate by using the command "`kill –SIGKILL <loop pid>`". Observe the output of the watch program. Use the "`ps –s`" command and observe that the `watch` and `loop` processes no longer appear because they have both terminated.
11. Examine the code for the `watch` process in `watch.c`.

Questions

Based on your observations, answer these questions.

1. What effect do the `SIGSTOP`, `SIGTSTP`, and `SIGCONT` signals have on the execution state of the process receiving them?
2. What effect does the `SIGKILL` signal have on the execution state of the process receiving it?
3. How is a parent process informed when a child process changes execution state (e.g., from running to stopped or stopped to running)?
4. What is the role and meaning of the SIGCHLD signal?
5. What information is contained in the `status` value returned as an output parameter from the `waitpid` call?