CS354: Machine Organization and Programming

Lecture 28

Friday the November 06th 2015

Section 2

Instructor: Leo Arulraj

Class Announcements

Monday's (11/9) lecture will be a review lecture for Midterm 2

General tip for Midterm prep:

1) Prioritize:

- Get thorough on the basic concepts first
- Get thorough on the stuff covered in lecture first
- If you have time left, focus on the extra material from the text book

2) Don't spend time memorizing stuff (e.g. EEPROM, DDRSDRAM, etc. Just remember basic details.)

Lecture Overview

1. Signal Handling
2. Sending and Receiving Signals
3. Other details about Signals

Signals

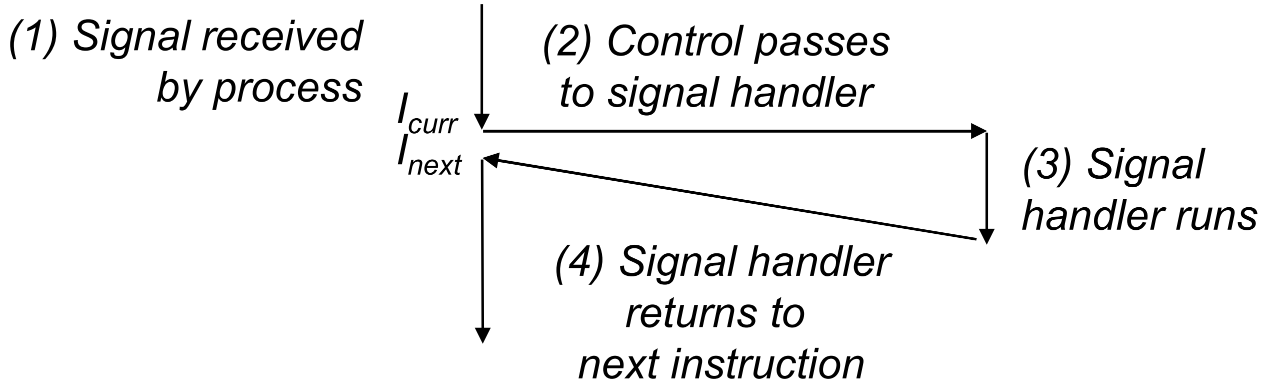
Unix Signal is a higher level software form of exceptional control flow.

A Signal is a small message that notifies a process that an event of some type has occurred in the sytem.

Processes and the Operating System can interrupt other processes using Signals.

Signal Handling

Control flow while handling signals.



Signals

Low level exception we discuss in last lecture are handled by the Operating System's exception handlers and are not visible to user level processes.

Signals provide a mechanism for exposing these low level exceptions to user processes.

E.g. - If a process executes an illegal instruction, then OS kernel sends a SIGILL signal.

- If a process divides by zero, then the OS kernel sends the process a SIGFPE signal.

Signals

List of Linux Signals in “man 7 signal”

The transfer of a signal occurs in two distinct steps:

1) Sending a signal

2) Receiving a signal

Sending Signals

OS Kernel sends/delivers a signal to a destination process by updating the process context.

A signal can be sent in two ways:

1) Kernel has detected an event like divide-by-zero or termination of a child process
2) A process has invoked the kill function to explicitly request the kernel to send a signal to the destination process.

Sending Signals – Process Groups

All mechanisms for sending signals to processes in Linux rely on the notion of process groups.

pid_t getpgrp(void);

int setpgid(pid_t pid, pid_t pgid);

Four ways of Sending Signals

1. With /bin/kill program:
   a. "/bin/kill -9 pid" sends signal 9 (SIGKILL) to process 15213
   b. "/bin/kill -9 -pid" sends signal 9 to all processes in process group 15213
2. Sending signals from the keyboard:
   a. Typing Ctrl-C on shell sends SIGINT signal to every process in the foreground process group.
   b. Typing Ctrl-Z sends SIGTSTP to every foreground process and the result is to suspend them.

Four ways of Sending Signals

3. Sending signals with the kill function:

   int kill(pid_t pid, int sig);

   - positive pid sends signal to that process
   - negative pid sends signal to every process in process group abs(pid)

4. Sending signals with the alarm function:

   unsigned int alarm(unsigned int secs)

   - A process can send SIGALRM signals to itself by calling the alarm function.

Example Programs for Sending Signals

1) Using kill function

2) Using alarm function

Pending Signals

A signal that has been sent bug not yet received is called a pending signal.

There is at most one pending signal of type k at any point in time.

Repetitive signals of same type are discarded and not queued.

Blocked Signals

A process can selectively block the receipt of certain signals.

When a signal is blocked, it can be delivered but the resulting pending signal will not be received until the process unblocks the signal.

A pending signal is received at most once.

Pending bit vector and block bit vectors maintained by the OS kernel for each process.

Receiving Signals

Before kernel returns control to a process after executing a exception handler, it checks the set of unblocked pending signals.

- If the set is empty(the usual case), then control goes to the next instruction.
- If the set is not empty, then OS kernel chooses one of the pending signals and forces the process to receive the signal.

Receiving Signals

Each signal has a predefined default action which is one of:

1) The process terminates
2) The process terminates and dumps core
3) The process stops until restarted by a SIGCONT signal
4) The process ignores the signal

Receiving Signals

However, a process can choose to install its own modified default action for all signal except SIGSTOP and SIGKILL using:

sighandler_t signal(int signum, sighandler_t handler);

Signal handlers are yet another example of concurrency.

Receiving Signals

The signal function can change the action associated with a signal in one of three ways:

1) If handler is SIG_IGN, then signals of type signum are ignored.
2) If handler is SIG_DFL, then the action for signals of type signum reverts to the default action.
3) Otherwise, handler is the address of a user defined function called signal handler that will be invoked whenever the process receives a signal of type signum.

Example program for user defined signal handler function.

Signal Handing Issues

- Pending signals are blocked: Unix signal handlers block pending signals of the type currently being processed by the handler.
- Pending signals are not queued: There can be atmost one pending signal of any particular type.
- System calls can be interrupted: In some systems, interrupted system calls will return immediately to user with an error condition.

Signal Handing Issues

- Example Programs illustrating Signal Handling Issues from the CSAPP textbook

Portable Signal Handling

Signal Handling Semantics differ from System to System (E.g. Linux vs Solaris)

Use sigaction() to specify the semantics that application wants.

Explicitly Blocking and Unblocking Signals

Applications can explicitly block and unblock selected signals using the sigproc-mask function.

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);

First parameter "how" can be:

- SIG_BLOCK: Add the signals in set to blocked
- SIG_UNBLOCK: Remove the signals in set from blocked
- SIG_SETMASK: blocked = set

Avoiding Concurrency Bugs

Tricky race scenarios can occur with signal handling if programmer is not careful.

Example programs illustrating concurrency bugs with signal handling and a technique to avoid the bug.

Unix Tools for Manipulating Processes

strace: trace system calls and signals

top: display linux tasks

ps: report a snapshot of current processes

pmap: report memory map of a process

/proc : read kernel state regarding processes from userspace