

Lecture 19: Unix signals and Terminal management

- what is a signal
- signal handling
 - ◆ kernel
 - ◆ user
- signal generation
- signal example usage
- terminal management

1

What is signal

- signal is a mechanism of primitive communication, synchronization and notification
- two phases:
 - ◆ generation - when event occurs that requires the process to be notified of it
 - ◆ delivery (handling) - the process recognized the arrival of even and takes appropriate action
 - asynchronous - example: user sends an interrupt signal by pressing ctrl-C
 - synchronous - example: accessing illegal address
- between generation and delivery - signal is *pending*
- *Type* is associated with signal. It is an integer. It is different on different Unix flavors. Usually symbolic constants rather than integers are used to denote signals. Example: SIGHUP

2

Signal handling

- A program can specify how to handle a certain signal by specifying (installing) *signal handler* - a function that is called when signal is delivered
- default actions - if no specification is given:
 - ◆ abort - process terminates after dumping core
 - ◆ exit - process terminates, no coredump
 - ◆ ignore - ignores signal
 - ◆ stop - suspends process
 - ◆ continue - resumes process if suspended, otherwise - ignore
- the signal delivery (and default handling) is done by kernel
 - ◆ when there is a signal to be delivered - the kernel checks is process specified a handler for this type of signal
 - if yes - calls handler
 - if not - takes one of the default actions
- note that the process needs to be running to receive signals
- kernel delivers signals on the following three occasions:
 - ◆ before returning to user mode from a system call or interrupt
 - ◆ just before blocking on interruptible event
 - ◆ immediately after waking up from interruptible event

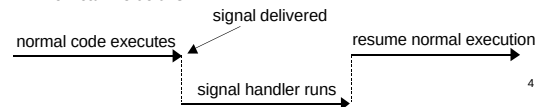
3

User program signal handling

- A program may elect to let the kernel handle the signal, install it's own signal handler or have it blocked (ignored)
- installing SIGINT handler


```
void sigint_handler (int sig){
    ... /* handles signal */
}

main(){
    /* installs handler */
    signal (SIGINT, sigint_handler);
    ...
}
```
- handler is executed "out of sequence"
- before switch to user mode kernel transfers control to signal handler
- and makes arrangements to return to previous point of execution
- how can we do this?



4

Signal generation

- Kernel generates signals to a process on various events.
- Some classes of events:
 - ◆ exceptions - attempt to execute illegal instruction, access memory out of range, division by zero, etc.
 - ◆ other processes - may send signals
 - ◆ terminal interrupts - certain keyboard characters generate signals (Ctrl-C)
 - ◆ notification - a process may request to be notified of certain event like device being ready for I/O

5

Signal example usage

- Asynchronous:
 - ◆ a user presses ctrl-C which generates keyboard interrupt
 - ◆ the interrupt is handled by terminal driver which generates SIGINT for the "foreground" process
 - ◆ when this process is scheduled to run, before it switches to user mode the kernel checks if the process installed signal handler
 - if yes - kernel makes the signal handler the first function to run in the user mode
 - if no - the kernel terminates the process
- Synchronous:
 - ◆ division by zero occurs - trap switches to kernel mode
 - ◆ trap handler recognizes the exception and either sends the signal to installed signal handler or aborts the process

6

Terminal management

- process group
 - ◆ each process has a *process group* identified by process group ID
 - ◆ process group is different from user group
 - ◆ each group has a leader; group ID is process ID of the leader.
Leader - process whose group ID is its process ID
 - ◆ when process is created it inherits group from parent
 - ◆ can leave group by executing a system call and becoming a new group leader
- controlling terminal
 - ◆ process may have controlling terminal - usually the terminal at which process was created
 - ◆ all processes of the same group share controlling terminal
 - ◆ process accepts input and sends output to controlling terminal
- /dev/tty
 - ◆ a special file which is the controlling terminal for each process
 - ◆ the terminal driver routes the requests to appropriate terminal
- controlling group - receives keyboard-generated signals (SIGINT ...) ⁷

Terminal management (cont.)

- When login shell *execs* a program at users request this process inherits a the shell's group ID
- this a user process usually
 - ◆ belongs to a controlling group of the terminal it was launched from - receives keyboard generated signals
 - ◆ has this terminal as its controlling terminal - receives keyboard input, outputs to terminal
- if a process executes the command to reset its process group it loses the controlling terminal and leaves controlling group becoming a *daemon* process