Inter-Process Communications using Signals

What is a signal?

- Program must sometimes deal with unexpected or unpredictable events, such as :
  - a floating point error
  - a power failure
  - an alarm clock “ring”
  - the death of a child process
  - a termination request from a user (i.e., Control+C)
  - a suspend request from a user (i.e., Control+Z)
- These kind of events are sometimes called interrupts, as they must interrupt the regular flow of a program in order to be processed.
- When UNIX recognizes that such an event has occurred, it sends the corresponding process a signal.

Signals

- A signal is a software generated interrupt that is sent to a process by the OS because:
    - it did something (oops)
    - the user did something (pressed ^C)
    - another process wants to tell it something (SIGUSR?)

- There are a fixed set of signals that can be sent to a process.

Therefore a signal is:

- a software interrupt delivered to a process by the OS because:
  - it did something (segmentation fault, FPE)
  - the user did something (pressed Ctrl+C)
  - another process wants to tell it something (SIGUSR?)
- one way a process can communicate with other processes
- Signal Types:
  - Some signals are asynchronous, they may be raised at any time (user pressing Ctrl+C)
  - Some signals are directly related to hardware (illegal instruction, arithmetic exception, such as attempt to divide by 0) - synchronous signals
- Signals are defined in “/usr/include/sys/signal.h” following the header chain actual constants are defined in “/usr/include/bits/signum.h”

Some Signal Numbers

- Signals are identified by integers.
- Signal numbers have symbolic names. For example, SIGCHLD is the number of the signal to a parent when a child terminates.

```
#define SIGHUP      1     /* Hangup. */
#define SIGINT      2     /* Interrupt. */
#define SIGQUIT     3     /* Quit. */
#define SIGILL      4     /* Illegal instruction. */
#define SIGTRAP     5      /* Trace trap. */
#define SIGABRT  6     /* Abort. */
```

List of Signals (Linux)

Macro	Signal Code	Default Action	Description
SIGHUP	1	Quit	Hang up - sent to a process when its controlling terminal has disconnected
SIGINT	2	Quit	Interrupt – Ctrl+C pressed by user, terminate the process after saving the work, can be trapped
SIGQUIT	3	Dump	Quit
SIGILL	4	Dump	Illegal instruction (bad opcode)
SIGTRAP	5	Dump	Trace trap (used by debuggers)
SIGABRT	6	Dump	Abort process – Ctrl+\ pressed by user, terminate immediately
SIGBUS	7	Dump	bus error (bad format address or unaligned memory access)
SIGFPE	8	Dump	Floating Point (Arithmetic) execution bad argument
SIGKILL	9	Quit	shell command Kill –SIGKILL or kill -9 or system call kill() to send SIGKILL signal by another process (unblockable)

List of Signals (BSD)

Macro	Signal Code	Default Action	Description
SIGUSR1	10	Quit	user signal 1
SIGSEGV	11	Dump	segmentation violation (out-of-range address)
SIGUSR2	12	Quit	user signal 2
SIGPIPE	13	Quit	write on a pipe or other socket with no one to read it.
SIGALRM	14	Quit	alarm clock timeout
SIGTERM	15	Quit	software termination signal(default signal sent by kill)
SIGSTKFLT	16	Quit	Stack fault
SIGCHLD	17	Ignore	child status changed
SIGCONT	18	Ignore	continued after suspension
SIGSTOP	19	Quit	Suspend process by signal (unblockable)
SIGTSTP	20	Quit	Keyboard store, Stopped by user (Ctrl+Z pressed which suspend/pause the process)

List of Signals (BSD)

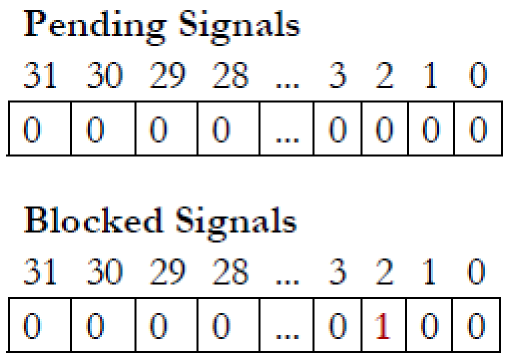
Macro	Signal Code	Default Action	Description
SIGTTIN	21	Quit	Background read from tty
SIGTTOU	22	Quit	Background write from tty output
SIGURG	23	Ignore	Urgent condition on socket
SIGXCPU	24	Dump	CPU time limit exceeded
SIGXFSZ	25	Dump	file size limit exceeded
SIGVTALRM	26	Quit	virtual timer expired
SIGPROF	27	Quit	profiling timer expired
SIGWINCH	28	Ignore	window size change
SIGIO	29	Ignore	I/O is now possible
SIGINFO	29	Ignore	Status request from keyboard
SIGPWR	30	Quit	Power failure restart
SIGSYS	31	Dump	bad argument to system call or non-existence system call

OS Structures for Signals

- For each process, the operating system maintains 2 integers with the bits corresponding to a signal numbers.
- The two integers keep track of:
  - pending signals
  - blocked signals
- With 32 bit integers, up to 32 different signals can be represented.

Example

- In the example below, the SIGINT (= 2) signal is blocked and no signals are pending.



Pending Signals

31	30	29	28	...	3	2	1	0
0	0	0	0	...	0	0	0	0

Blocked Signals

31	30	29	28	...	3	2	1	0
0	0	0	0	...	0	1	0	0

Sending, Receiving Signals

- A signal is sent to a process setting the corresponding bit in the pending signals integer for the process.
- Each time the OS selects a process to be run on a processor, the pending and blocked integers are checked.
- If no signals are pending, the process is restarted normally and continues executing at its next instruction.

Sending, Receiving Signals

- If 1 or more signals are pending, but each one is blocked, the process is also restarted normally but with the signals still marked as pending.
- If 1 or more signals are pending and NOT blocked, the OS executes the routines in the process's code to handle the signals.

Default Signal Handlers

- There are several default signal handler routines.
- Each signal is associated with one of these default handler routine.
- The different default handler routines typically have one of the following actions:
    - ignore the signal; i.e., do nothing, just return - Ign
    - terminate the process - Term
    - unblock a stopped process - Cont
    - block the process - Stop

Actions when Signal is received

- The default handler usually performs one of the following actions:
  - Abort: terminate the process and generate a core file (dump)
  - Quit/Exit: terminate the process without generating a core image file (quit)
  - Ignore: ignore and discard the signal (ignore)
  - Stop: suspend the process (suspend)
  - Continue: resume the process (resume)

User Defined Signal Handlers

- A process can replace the default signal handler for almost all signals (but not SIGKILL and SIGSTOP) by its own handler function.
- A signal handler function can have any name, but must have return type void and have one int parameter.
- Example, you might choose the name sigchld_handler for a signal handler for the SIGCHLD signal (termination of a child process). Then the declaration would be:

```
void sigchld_handler(int sig);
```

User Defined Signal Handlers

- When a signal handler executes, the parameter passed to it is the number of the signal.
- A programmer can use the same signal handler function to handle several signals. In this case the handler would need to check the parameter to see which signal was sent.
- On the other hand, if a signal handler function only handles one signal, it isn't necessary to bother examining the parameter since it will always be that signal number.

Sending signals from keyboard

- Process suspended using Ctrl+Z (SIGSTSTP) can be brought back to life using fg command which sends (SIGCONT) signal to resume
- For background process cant use Ctrl+C, Ctrl+Z etc hence kill command is used (how one can run process in background ?)

kill [options] pid

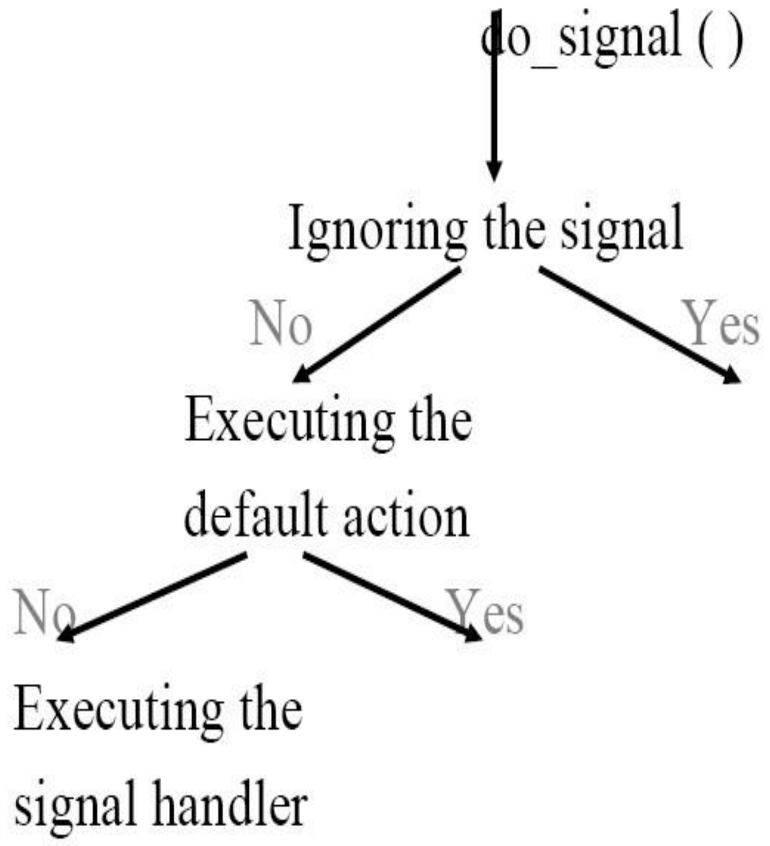
--l lists all signals you can send

--signal number

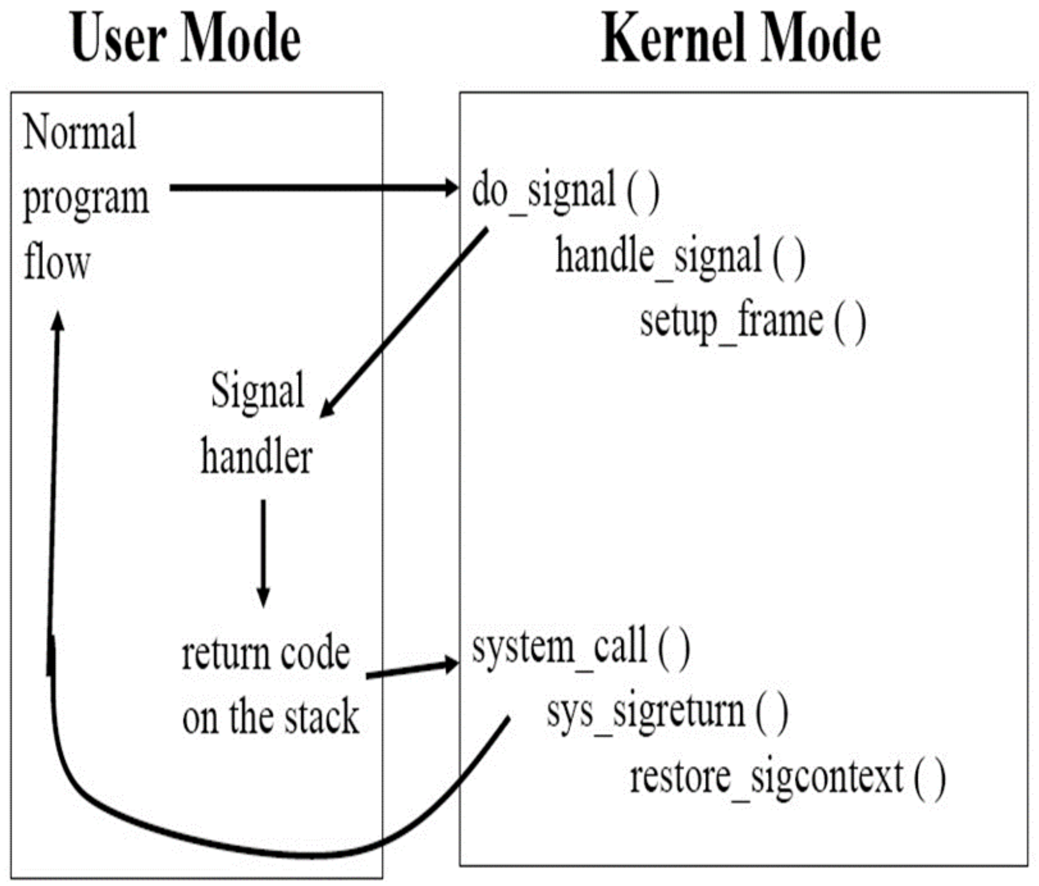
kill command can send SIGKILL signal upon pressing Ctrl+C to the foreground process which cannot be ignored

Signal Handling

- Most signals can be ignored except SIGKILL and SIGSTOP
- Application can trap the signal (e.g. SIGINT when Ctrl+C pressed) and handle signal disposition via signal handler
- Signal handler can perform default action (usually process termination) or customized actions such as blocking the signal



Catching Signal


User Mode
Kernel Mode
Normal program flow
do_signal ( )
handle_signal ( )
setup_frame ( )
Signal handler
return code on the stack
system_call ( )
sys_sigreturn ( )
restore_sigcontext ( )