Signals

What to Expect?

- W's of a Signal
- Signals in Linux
- Signal Handlers
- Signal related Functions
- Program Exit Codes

Asynchronous Communication

- In Kernel Space
  - Interrupts (Hardware to Software)
  - Soft Irqs (Software to Software)
- What about User Space?
- Moreover, between User Space & Kernel Space
- Though only software to software requirement
- Do System Calls take care of that?
- Communication: Yes, but they are Synchronous
- Asynchronous Solution is Signals

What is a Signal?

- Asynchronous Trigger from Kernel Space or a Process to another Process
- Mechanisms for communicating with and manipulating processes in Linux
- Handled in User Space like Interrupts in Kernel Space
  - Remains Pending on Masking. Once handled,
  - Immediate processing without finishing the current function or even the current line of code. Possible options
    - Ignore
    - Execute the Default Handler
    - Execute the User specified Handler

Signals in Linux

- Specified by a Number (currently 1-31)
- Also, referred by Human readable Names
- On Shell
  - Type 'kill -l'
  - Signal Operations: Using kill
- In Programming / C
  - Header: /usr/include/bits/signum.h
  - Signal Operations: Shall discuss further

Signals Originate from

- Kernel Space
  - Typically on an illegal operation. Namely,
    - SIGBUS (bus error),
    - SIGSEGV (segmentation violation)
    - SIGFPE (floating point exception)
  - Others like SIGCHLD
- A Process
  - Synchronously using kill system call

Signal Handler Registration

- Using signal(sig_no, handler)
  - Non-portable except for SIG_DFL & SIG_IGN as handlers
- Using sigaction(sig_no, action, old_action)
  - action & new_action are pointers to sigaction structures
  - action contains the desired disposition
  - old_action receives the previous disposition
  - Their sa_handler field takes one of the three values
    - SIG_DFL
    - SIG_IGN
    - A pointer to a signal handler function
- Signal Hanlder Prototype
  - void signal_handler(int sig_no);
- Exceptions to Registration: SIGKILL, SIGSTOP

Sensitivity of Signal Handlers

- Time Sensitive like Interrupt Handlers
- Perform the minimum work necessary &
  - Return control to the main program, Or
  - Terminate the program
- Mostly, they just record the signal occurrence
  - And the main program does the processing

Signal in a Signal

- Signal interrupting a signal handler
  - Rare Occurrence but could be a Problem
  - Difficult to diagnose & debug
  - Assigning to globals can be dangerous
- Solutions
  - Self is restricted by default
    - Unless SA_NODEFER flag is used
  - Disable other signals in signal handlers
    - By setting sa_mask using sigsetops during registration
  - Protect the global variables by using sig_atomic_t type

Signal related Functions

- raise
  - Sends a signal to calling thread
- kill, killpg
  - Sends a signal to a specified process or group of processes
- tkill, tgkill, pthread_kill
  - Sends a signal to a specified thread
- sigqueue
  - Sends a real-time signal with accompanying data to a specified process
- pause, sigsuspend
  - Suspends the calling thread until delivery of a signal whose action is either to execute a signal handler or to terminate the process
- sigwaitinfo, sigtimedwait, sigwait, signalfd
  - Accepts signals synchronously

Signal related Functions ...

- alarm(unsigned int secs);
  - Schedules an alarm signal (SIGALRM) to be sent to the calling process after “secs” seconds
  - Should not be mixed with setitimer / sleep
- abort();
  - Causes abnormal process termination by raising abort signal (SIGABRT)

Process Termination Revisited

- Normal: System Call 'exit' is invoked
  - With zero – Success Exit
  - With non-zero value – Error Exit
- Abnormal: A Signal has terminated it
  - From Kernel Space for a Bug – Fatal Exit
    - SIGBUS
    - SIGSEGV
    - SIGFPE
  - From a Process – Kill Exit
    - SIGINT (Ctrl + C)
    - SIGTERM
    - SIGABRT / abort()
    - ...

Exit Codes Decoded

- 8-bit unsigned integer
- Range: 0-255
- Normal Termination (using exit(value);)
  - “value” should be always between 0 and 127
  - Exit code is “value”
- Abnormal Termination (by signal)
  - Exit code is 128 + “signal”

What all have we learnt?

- W's of a Signal
- Signals in Linux
- Signal Handlers
- Signal related Functions
- Program Exit Codes

Any Queries?