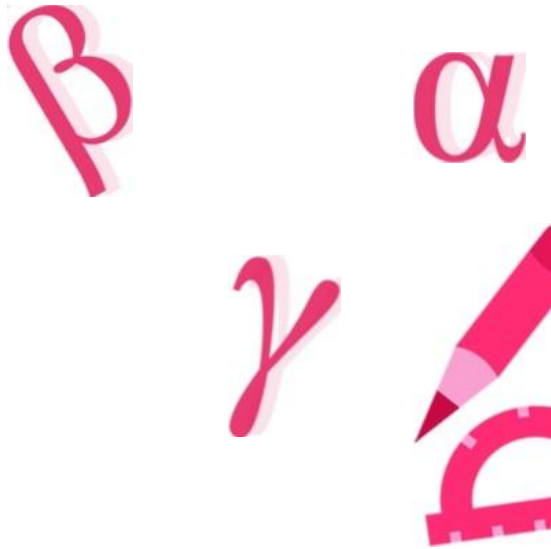


MODULE 11 :
Signals

Signals

- Signals are various notifications sent to a process in order to notify it of various "important" events.
- Signals are one of the oldest inter-process communication methods used by Unix systems.
- By their nature, they interrupt whatever the process is doing at that moment, and forces to handle them immediately.
- Each signal has an integer number that represents it (1, 2 and so on), as well as a symbolic name that is usually defined in the file /usr/include/signal.h
- A signal could be generated by a keyboard interrupt or an error condition such as the process attempting to access a non-existent location in its virtual memory.

Signals

There are a set of defined signals that the kernel can generate or that can be generated by other processes in the system

SIGINT	SIGQUIT
SIGTERM	SIGKILL
SIGSTOP	SIGCONT
SIGALRM	SIGCHLD
SIGFPE	SIGHUP
SIGPIPE	SIGSEGV

Signals

- Each signal in linux has a default action.
- If a signal's handler is set to the default action then the kernel will handle it.
- The SIGSTOP signal's default handler will change the current process's state to Stopped and then run the scheduler to select a new process to run.
- Alternatively, the process can specify its own signal handler.
- Processes can block all the signals, with the exception of SIGSTOP and SIGKILL.
- If a blocked signal is generated, it remains pending until it is unblocked.

Signals

- Signals have no relative priorities.
- The number of supported signals is limited to the word size of the processor.
- Processes with a word size of 32 bits can have 32 signals whereas 64 bit processors like the Alpha AXP may have up to 64 signals.
- Not every process in the system can send signals to every other process, the kernel can and super users can.
- Signals are generated by setting the appropriate bit in the task_struct's signal field.

System Calls for Signals

Handling Signals: signal()

To give a signal a new action, you can use the signal() system call:

#include <signal.h>

sighandler_t signal(int signum, sighandler_t handler);

Data Type: sighandler_t

This is the type of signal handler functions. Signal handlers take one integer argument specifying the signal number, and have return type void.

void handler (int signum)

{

...

}

Pre-defined Signal Handlers

- **For convenience, there are two pre-defined signal handler functions that can be used instead of writing our own:**

- **SIG_IGN:**

  Causes the process to ignore the specified signal. For example, in order to ignore Ctrl-C completely (useful for programs that must NOT be interrupted in the middle, or in critical sections):

  ```
  signal(SIGINT, SIG_IGN);
  ```

- **SIG_DFL:**

  Causes the system to set the default signal handler for the given signal (i.e. the same handler the system would have assigned for the signal when the process started running):

  ```
  signal(SIGTSTP, SIG_DFL);
  ```

System Calls for Signals

- **Sending Signals: kill()**

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

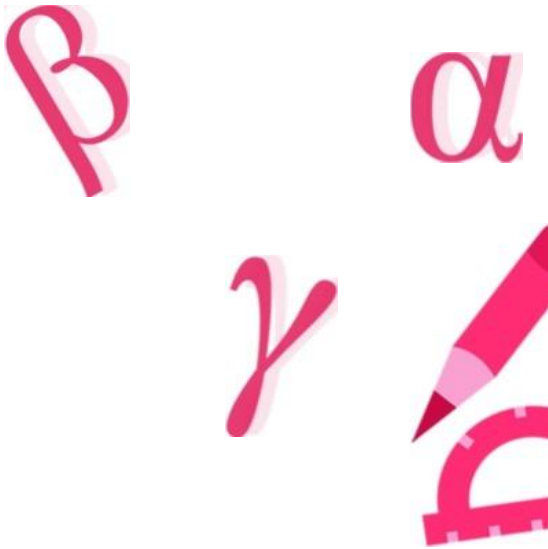
- The kill system call can be used to send signal to any other process specified by pid.
  - If pid is positive, then signal sig is sent to pid.
  - If pid equals 0, then sig is sent to every process in the process group of the current process.

System Calls for Signals (Contd.)

- A process can send itself a signal with the raise function. This function is declared in signal.h.

int raise (int signum)

- The raise function sends the signal signum to the calling process. It returns zero if successful and a nonzero value if it fails. About the only reason for failure would be if the value of signum is invalid.



Thank You !!