Linux For Embedded Systems

For Arabs

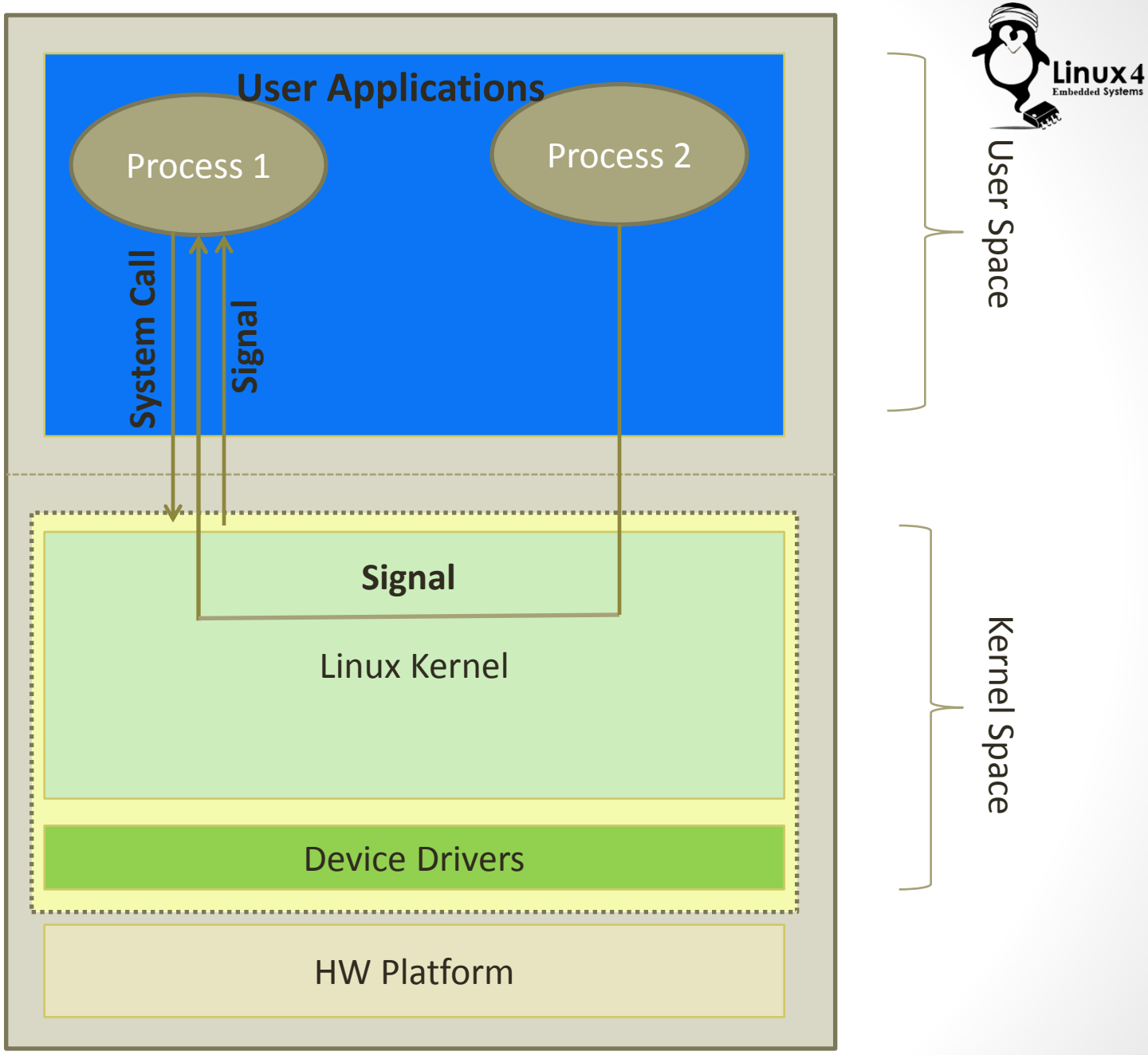
Course 102:

Understanding Linux

Ahmed ElArabawy

Linux4
Embedded Systems

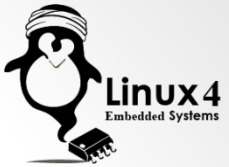
Lecture 19:
Using Signals


User Applications
Process 1
Process 2
System Call
Signal
Signal
Linux Kernel
Device Drivers
HW Platform
User Space
Kernel Space

What is a Signal ??

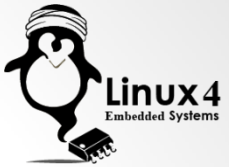
- Linux has two planes:
  - Kernel Plane: Has access to system and hardware resources
  - User Plane: For applications to run on
- User applications don't have direct access to the kernel
- For a user application to communicate with the kernel, it needs to use a **System Call**
- What about if the kernel needs to send a request/command/notification/… to the user application ??
- The used mechanism is **Signals**
- So, Signals are the mechanism that the kernel uses to communicate with the user applications
- Signals are also used between user applications (when one application needs to send a request/command to the other application)

What is a Signal ??



- Signal is the communication channel used by the Linux Kernel to communicate with a process running in the user plane
- This can be used by the kernel to kill a process, stop it, resume it, notify it of a timer expiry, etc…
- Signals can also be sent by another process running in the user plane. Examples,
  - When the user presses ***Ctrl-c*** to terminate the running application, or ***Ctrl-z*** to stop it
  - When the user uses the command "***bg***" to resume a stopped application in the background, or "***fg***" to resume it in the foreground
  - When the user uses the "***kill***" command to terminate a running application
- In all of these cases, a signal is sent to the destination application to achieve the required task
- When sent by another user application, it is first passed to the kernel in a system call, which in turn passes it back to the desired process

Signals



- Signals are described in both Unix Systems and POSIX Standard
- There are a table of signals. Each signal has,
  - Signal number
  - Signal name (starting with ‘SIG’)
- Signals with numbers (1-31) are normal signals
- Signals with numbers > 31 are called “**Real Time Signals**”

Passing a Signal to a Process (kill Command)

$ kill <pid>

$ kill -<Signal number> <pid>

$ kill <Signal name> <pid>

- The kill command passes a signal to the desired process
- Note that, despite that the command name is '***kill***', it is used for all types of signals even those that have different purposes and not just used to kill processes
- If no signal is specified, then the signal **SIGTERM** is assumed by default
- Examples:

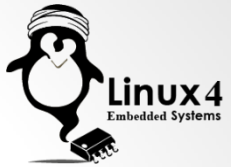
  $ kill -9 1185

  $ kill SIGKILL 1185

  } Identical commands

Other ways to “kill”

Selecting Processes by name (killall Command)



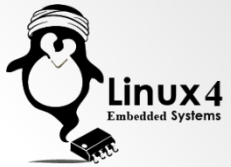
$ killall <Signal> <command name>

- The command ***killall*** sends the desired signal to all processes running the specified command

  $ killall chrome
- This is useful when you need to pick a process without searching for its pid
- Also useful when you need to send the signal to all instances of the running program
- If no signal is specified, **SIGTERM** is assumed
- Example:

  $ killall -9 gedit (this sends a SIGKILL signal to all “gedit” instances)

Selecting Processes by Other Criteria (pkill Command)



$ pkill <signal> -u <username>

$ pkill <siganal> -P <Parent process>

$ pkill <signal> <pattern for command>

$ pkill <signal> -t <Terminal name>

- The ***pkill*** command sends the desired signal to processes that match different criteria
- Examples:
  - Send a SIGTERM to all processes owned by user tom

    $ pkill -u tom
  - Send a SIGSTOP to all processes running under tty1 terminal

    $ pkill SIGSTOP -t tty1
  - Send SIGTERM to all children of process with pid 1107

    $ pkill –P 1107

Killing GUI Processes with the Mouse (the xkill Command)

$ xkill

- The xkill command allows the user to select the process to kill with the mouse
- Type the command, then click with the mouse on the UI of the process you want to kill

Dealing with Hanging GUI Apps

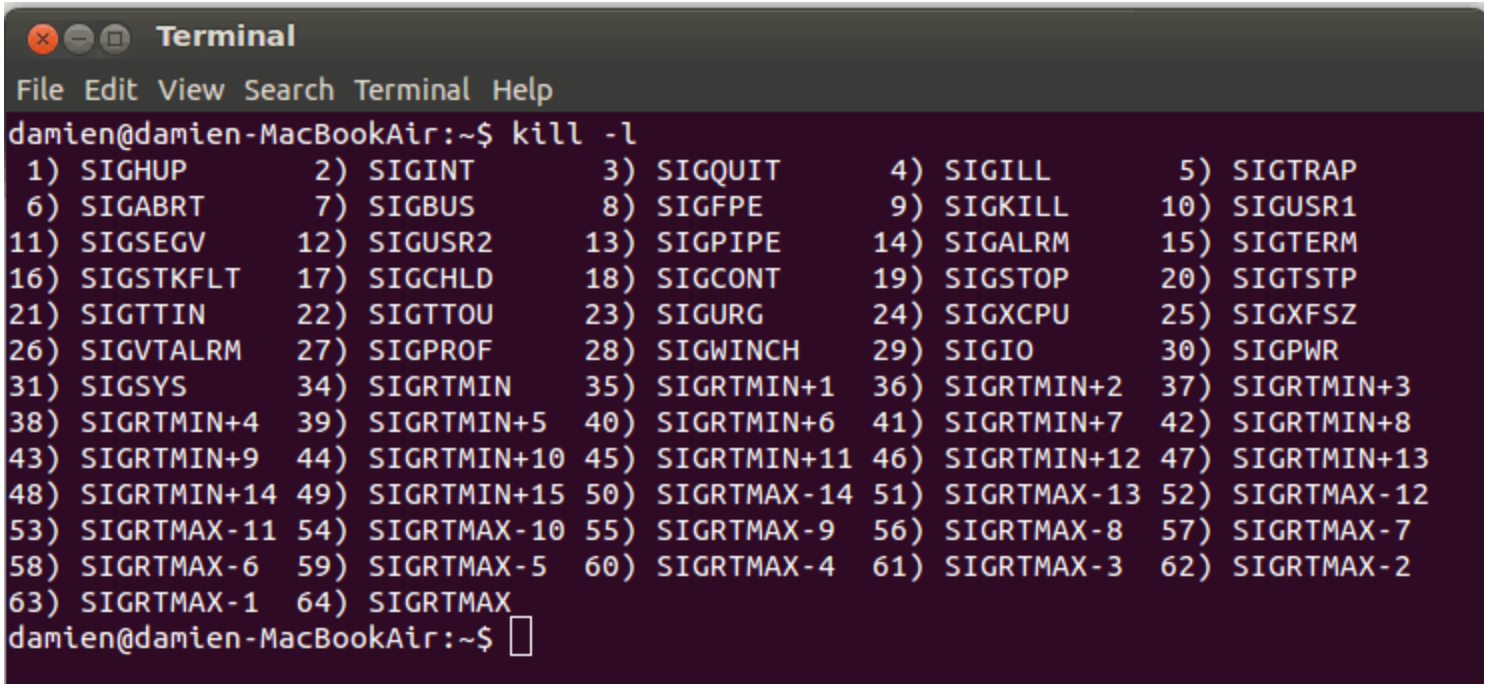
- If a GUI app is hanging, and you need to kill it but unable due to the hanging screen
  - get into a Linux virtual terminal via,

    [Ctrl] [Alt] [F1]
  - From the virtual terminal, issue the proper “kill” command

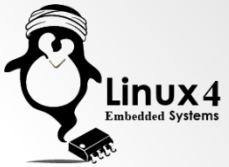
    $ killall -9 chrome
  - After you are done, get back to the tty-7 (for the X window)

    [Alt] [F7]

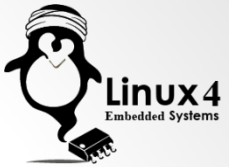
Listing Signal Names



Real time Signals



- Those are the signals with numbers > 31
- They don’t have a pre-defined function, and it is left to the user/developer to use
- Defined by the programmer
- Guaranteed order of delivery



Triggers for Sending Signals

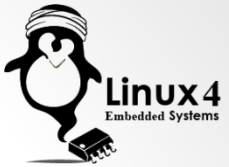
Keyboard Hot keys

- Some signals are triggered by the user by typing special hot-keys
- When the kernel detects these hot-keys, it sends the proper signal to the running process (or process group)
- Examples:
  - ***Ctrl-c*** triggers ***SIGINT***
  - ***Ctrl-z*** triggers ***SIGTSTP***
  - ***Ctrl-*** triggers ***SIGQUIT***

Request from another process

- A process can trigger the kernel to send a signal to another process by issuing a system call
- Examples for this is when the user issue the commands from the terminal
  - ***$ kill*** (triggers sending the desired signal, ***SIGTERM*** by default)
  - ***$ fg*** (triggers sending the signal ***SIGCONT***)
  - ***$ bg*** (triggers sending the signal ***SIGCONT***)
- A process can also trigger sending a signal to another process programmatically via some special function API

A Hardware Event

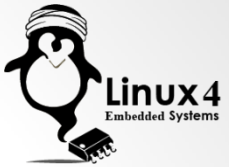


- Hardware events can trigger the kernel to send a signal to the proper process
- An example of that, when the kernel detects a modem disconnection, it sends the signal ***SIGHUP*** to all processes using this session

Death of a Related Process

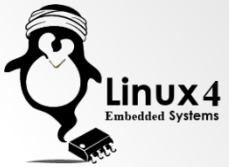
- When a process terminates, the kernel sends the following signals,
  - A ***SIGHUP*** is sent to all children processes to inform them about the death of their parent process
  - A ***SIGCHLD*** is sent to the parent of the terminated process to inform it about the death of a child process

Errors and Exceptions

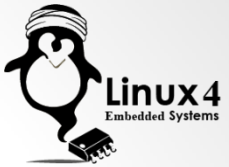


- Some exceptions within the process triggers the kernel to send signals to the process,
- Examples:
  - If the process tries to execute an illegal instruction, the kernel is triggered to send ***SIGILL*** signal to it
  - If the process suffers from a Floating Point Exception, the kernel sends it a ***SIGFPE***
  - If the process tries to write to a pipe while the other end is not listening, the kernel sends it a ***SIGPIPE***
  - If the process accesses a Null Pointer (or other memory exceptions), the kernel sends it ***SIGSEGV***
  - If a process running in the background tries to read/write to the standard input/output, the kernel will send it a ***SIGTTIN***/***SIGTTOU***
  - If the process exits on error using the ***abort()***, the kernel will send it ***SIGABRT***

Notification By the Kernel



- Sometimes, a process would ask the kernel for some notification when some event happens
- Normally, the process will be waiting for this notification
- The kernel delivers this notification via a signal
- Examples:
  - The kernel will use the signal ***SIGALRM*** and ***SIGVTALRM*** to notify the process of a timer expiry
  - A process can ask the kernel for a notification whenever a file handler has been accessed

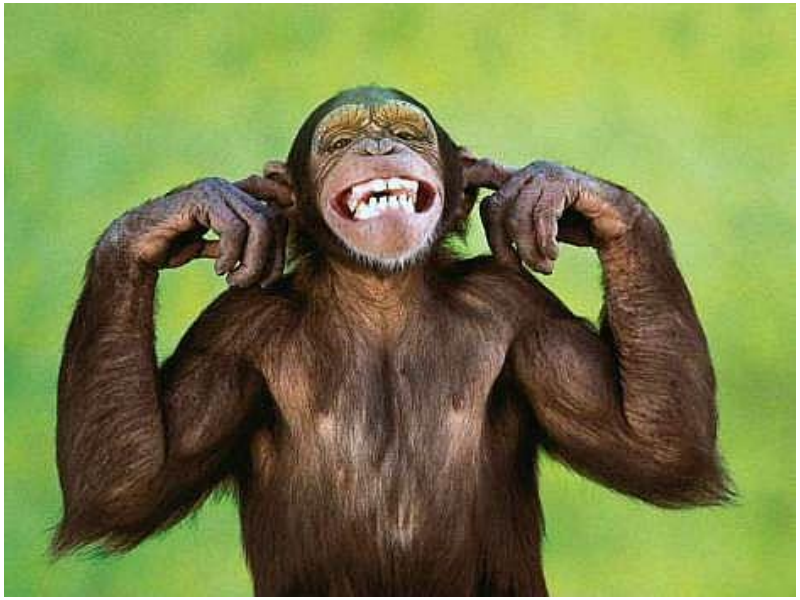


Process Response to Signals

Default Action

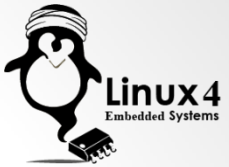
- When a process receives a signal, the normal behavior is to execute the default action
- Each signal has its own default action that is part of its description
- Examples:
  - Some signals have the default action of termination (graceful termination). This is applicable for several signals such as ***SIGTERM***, ***SIGPIPE***, ***SIGUSR1***, ***SIGUSR2***, …
  - Some signals have the default action of termination with generation of a core dump (stack trace for further debugging). This action is useful for signals that indicate some error in the process operation such as ***SIGSEGV***, ***SIGILL***, ***SIGABRT***, ***SIGQUIT*** …
  - Some signals have the default action of being ignored by the process such as ***SIGCHLD***
  - Some signals does not allow the process to change its default action. This means that the default action is the only allowed action. This includes the ***SIGSTOP*** which force the process to stop (suspend) and ***SIGKILL*** which force the process to terminate immediately

Ignoring a Signal



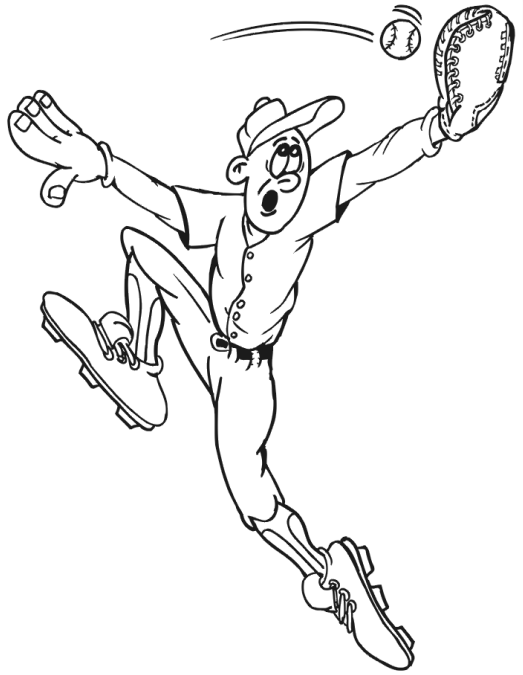
- A process can set itself to ignore certain incoming signals, accordingly no action is performed when these signals are received
- Some signals can not be ignored such as ***SIGSTOP***, and ***SIGKILL***

Blocking a Signal

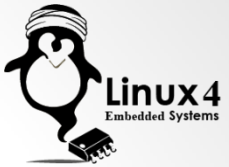


- A process can set itself to block a certain set of signals
- Blocking a signal means, keeping it in a queue for later handling (when the blocking is taken away)
- This is used when the process is executing some code that should not be interrupted by the incoming signals
- Every process has a mask of which signals to block
- Only one signal of a certain type will be blocked (other instances will be dropped).
- This last rule does not apply to real time signals (signals with numbers > 31)
- Some signals can not be blocked (***SIGSTOP*** and ***SIGKILL***)

Catching the Signal

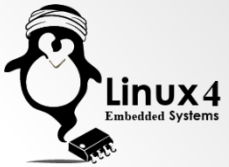


- A process can “catch the signal” before its default handler executes, and pass it to a special handler
- This is specially useful for some of the signals that have their default action of being ignored (such as ***SIGCHLD***)
- Some signals can not be caught (***SIGSTOP*** and ***SIGKILL***)



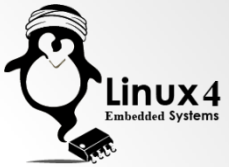
Popular Signals

Hang-Up Signal (1) SIGHUP



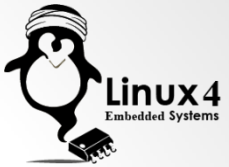
- The hang-up signal is used by the kernel to inform the processes running in a tty session when the session closes. This can be due to,
  - The modem for the session disconnects
  - The network connection used to connect to the machine breaks
- It is commonly used to request daemon processes to re-read their configuration file
- Default action for this signal for the process to terminate

Interrupt Signal (2) SIGINT



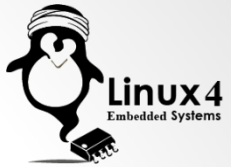
- Interrupt signal is triggered when the user uses the hot-key ***Ctrl-c***
- This causes the kernel to send ***SIGINT*** to the currently running process group
- The default action for this signal is for the process to terminate

Quit Signal (3) SIGQUIT



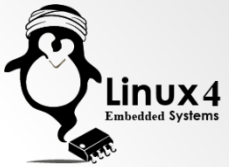
- The quit signal is triggered by the user when he uses the hot-keys ***Ctrl-***
- This triggers the kernel to send ***SIGQUIT*** to all processes in the currently running process group
- Default action is for the process to quit and generate a core dump file (to be used for debugging)

Illegal Command Signal (4) SIGILL



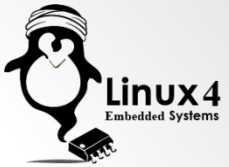
- The illegal command signal is triggered when the process program tries to execute an invalid instruction
- This may occur in the case of memory corruption of the code area or in stack memory
- Default action is to terminate and generate a core dump file

Abort Signal (6) SIGABRT



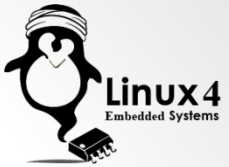
- The abort signal is sent by the kernel to the process when it calls the ***abort()*** function to indicate an erroneous exit of the program
- The default action is to terminate and generate a core dump file

Trap Signal (5) SIGTRAP



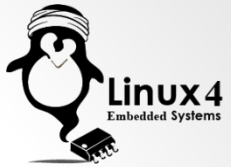
- The trap signal is normally used with debuggers and program tracers

Bus Error Signal (7) SIGBUS



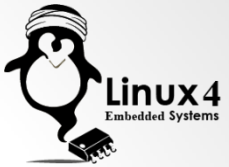
- The bus error signal is triggered by an error in the program in which an attempt was made to access memory incorrectly
- This can be caused by alignment errors in memory access
- The default action is to terminate and generate a core dump file

Floating Point Exception (8) SIGFPE



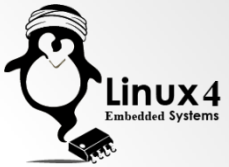
- The floating point exception signal is sent by the kernel to the process when it falls into an exception while running a floating point instruction
- The default action is to terminate and generate a core dump file

Kill Signal (9) SIGKILL



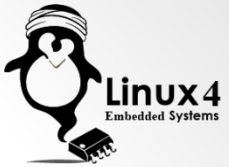
- The process was explicitly killed by a request from a different process
- The request is to kill the process immediately and forcibly without giving it a chance to clean-up
- A process can not ignore, block, or catch this signal

Stop Signals (17 &18) SIGSTOP & SIGTSTP



- ***SIGTSTP*** (terminal Stop) is triggered by pressing ***Ctrl-z***
- This signal causes the process (or process group) to suspend operation
- ***SIGSTOP*** has the same effect with the only difference that it can not be ignored/blocked/caught by the process while this is possible with ***SIGTSTP***
- Upon stopping due to either SIGSTOP & SIGTSTP, the process awaits ***SIGCONT*** to resume operation

Terminate Signal (15) SIGTERM



- The terminate signal is the default signal for the ***kill*** command when no signal is specified
- The default action for ***SIGTERM*** is to gracefully terminate the process (process group) after clean up

Linux 4 Embedded Systems

http://Linux4EmbeddedSystems.com