

Linux Internals & Networking

System programming using Kernel interfaces

Team Emertxe



Contents



Linux Internals & Networking

Contents

- .Introduction
- .Transition to OS programmer
- .System Calls
- .Process
- .IPC
- .Signals
- .Networking
- .Threads
- .Synchronization
- .Process Management
- .Memory Management



Signals

A decorative horizontal bar at the bottom of the slide. It features a gradient from bright pink on the left to deep purple on the right. On the far right, there is a graphic of two overlapping chevrons pointing to the right, with the front chevron in a darker purple and the back one in a lighter pinkish-purple.



- Signals are software interrupts that provide a mechanism for handling asynchronous events.
- Events can originate from
 - outside the system
 - such as when the user generates the interrupt character (usually via Ctrl-C)
 - from activities within the program or kernel
 - such as when the process executes code that divides by zero



- Signals are raised(sent / generated)
- The kernel then stores the signal until it is able to deliver it.
- once it is free to do so, the kernel handles the signal as appropriate.
- The kernel can perform one of three actions upon receiving the signal,
 - Ignore the signal
 - Catch and handle the signal
 - Perform the default action



• Every signal has a symbolic name that starts with the prefix SIG.

• For example,

– **SIGINT** - The signal sent when the user presses Ctrl-C

– **SIGABRT** - The signal sent when the process calls the abort() function

– **SIGKILL** - The signal sent when a process is forcefully terminated.

• Signals are all defined in a header file included from **<signal.h>**

kill -l : This command will list the signals

Get Basics Right

Function pointers



- What is function pointer?
 - `Datatype *ptr ;` normal pointer
 - `Datatype (*ptr)(datatype,...);` Function pointer
- How it differs from normal data pointer?

Function Pointer

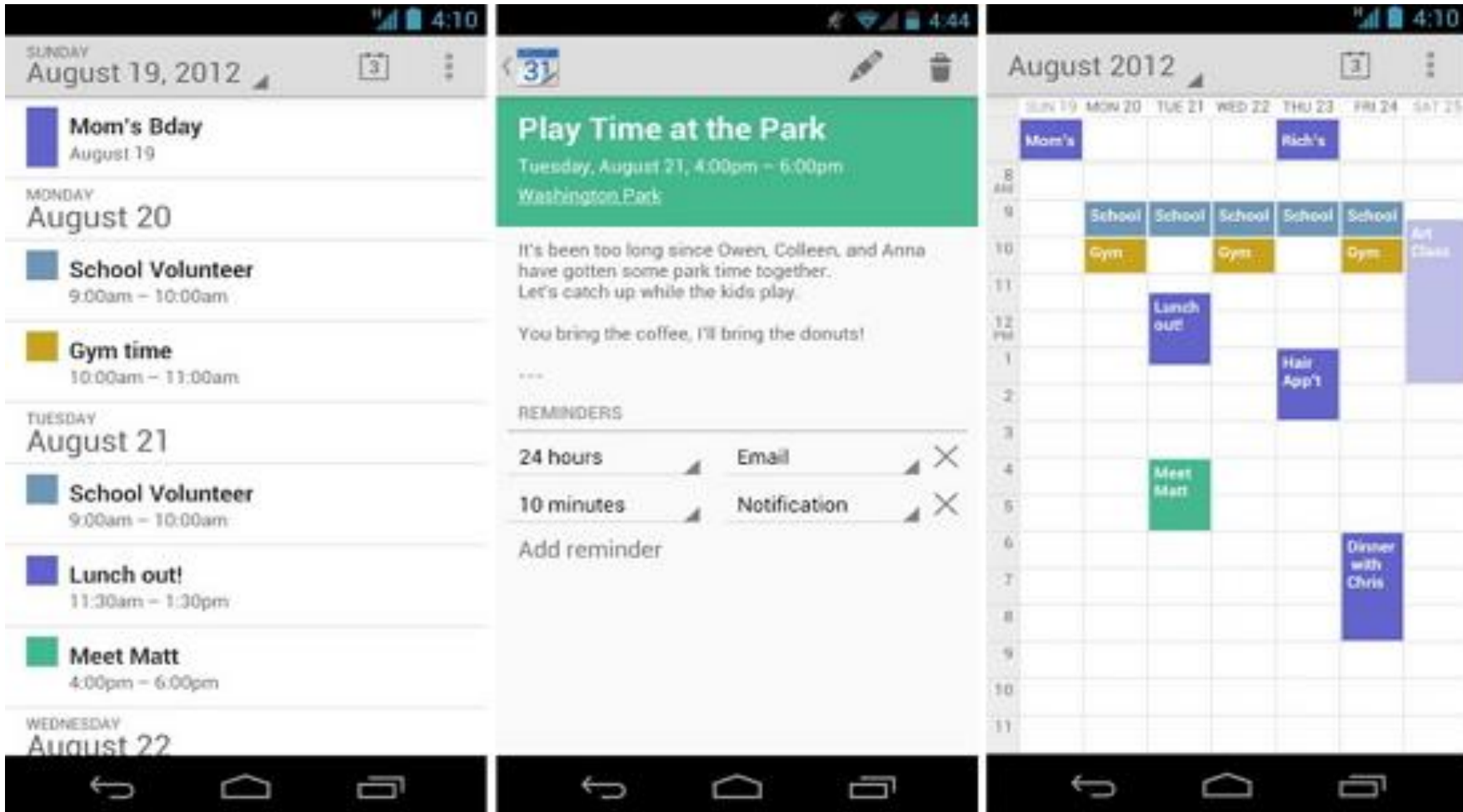
- Holds address of function
- Pointing to a address from code segment.
- Dereference to execute the function
- Pointer arithmetic not valid

Data Pointer

- Holds address of an object
- Pointing to a address from stack/heap/data
- Dereference to get value from address
- Pointer arithmetic is valid

Get Basics Right

Call back functions



Registering an event for later use

Get Basics Right

Call back functions



- In computer programming, a callback is a reference to executable code, or a piece of executable code, that is passed as an argument to other code.
- This allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.

Team Emertxe





- .The kernel
- .A Process may also send a Signal to another Process
- .A Process may also send a Signal to itself
- .User can generate signals from command prompt:
 - .'kill' command:
 - .\$ kill <signal_number> <target_pid>
 - .\$ kill -KILL 4481
 - .Sends kill signal to PID 4481
 - .\$ kill -USR1 4481
 - .Sends user signal to PID 4481



- When a process receives a signal, it processes by handling immediately.
- For all possible signals, the system defines a default disposition or action to take when a signal occurs
- There are four possible default dispositions:
 - **Exit:** Forces process to exit
 - **Core:** Forces process to exit and create a core file
 - **Stop:** Stops the process
 - **Ignore:** Ignores the signal
- Handling can be done, called 'signal handling'



- .The signal() function can be called by the user for capturing signals and handling them accordingly
- .First the program should register for interested signal(s)
- .Upon catching signals corresponding handling can be done

Function	Meaning
signal (int signal_number, void *(fptr) (int))	signal_number : Interested signal fptr: Function to call when signal handles



P1

User Space

Kernel Space

Registering handler
signal / sigaction

Signal handler

Signal handler
executed

Signal generated

Pointer	Process State
Signals	
Memory Limits	



- A signal handler should perform the minimum work necessary to respond to the signal
- The control will return to the main program (or terminate the program)
- In most cases, this consists simply of recording the fact that a signal occurred or some minimal handling
- The main program then checks periodically whether a signal has occurred and reacts accordingly
- Its called as **asynchronous handling**

Team Emertxe



Signals

Advanced Handling



- The `signal()` function can be called by the user for capturing signals and handling them accordingly
- It mainly handles user generated signals (ex: `SIGUSR1`), will not alter default behavior of other signals (ex: `SIGINT`)
- In order to alter/change actions, `sigaction()` function to be used
- Any signal except `SIGKILL` and `SIGSTOP` can be handled using this

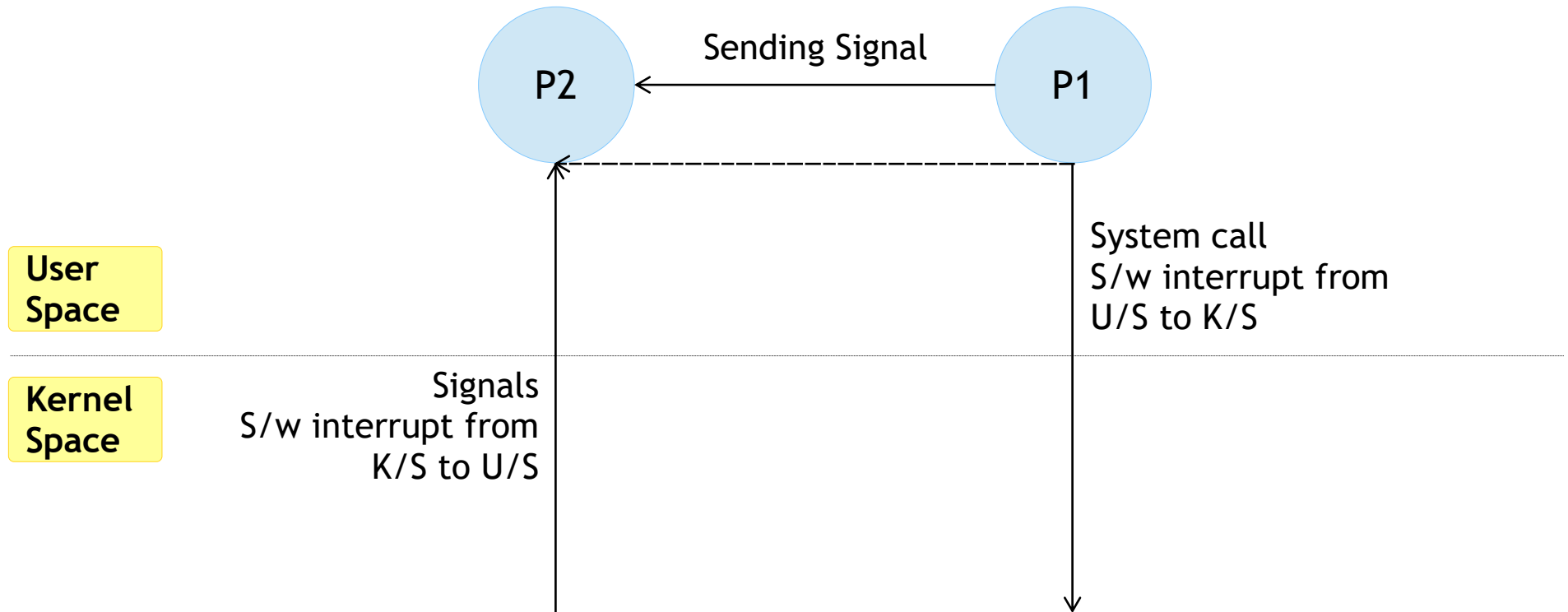
Function	Meaning
<code>sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)</code>	<p>signum : Signal number that needs to be handled</p> <p>act: Action on signal</p> <p>oldact: Older action on signal</p>

Team Emertxe



Signals

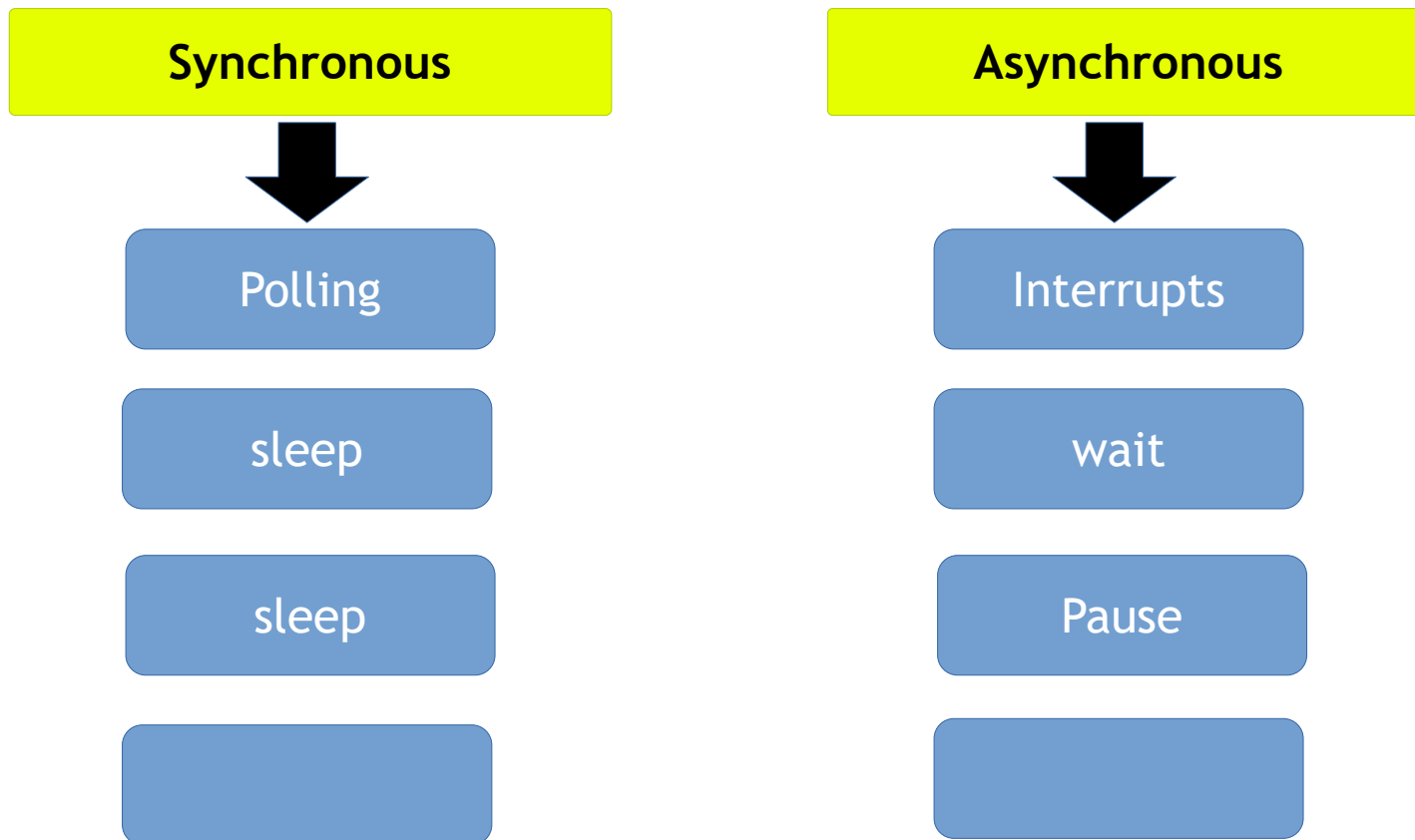
vs system calls



Synchronous & Asynchronous



.Wait for child to finish





- .A process can send or detect signals to itself
- .This is another method of sending signals
- .There are three functions available for this purpose
- .This is another method, apart from 'kill'

Function	Meaning
raise (int sig)	Raise a signal to currently executing process. Takes signal number as input
alarm (int sec)	Sends an alarm signal (SIGALRM) to currently executing process after specified number of seconds
pause()	Suspends the current process until expected signal is received. This is much better way to handle signals than sleep, which is a crude approach

Inter Process Communications

Summary



.We have covered

Data exchange

Communication

- .Pipes
- .FIFO
- .Shared memory
- .Signals
- .Sockets

Resource usage/access/control

Synchronization

- .Semaphores

Team Emertxe



Thank You