Linux Internals & Networking

System programming using Kernel interfaces

Team Emertxe

Contents

Linux Internals & Networking
Contents

EMERTXE

Signals

Signals
Introduction

- 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
Life Cycle

- 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

Signals
Signal Identifiers

- 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

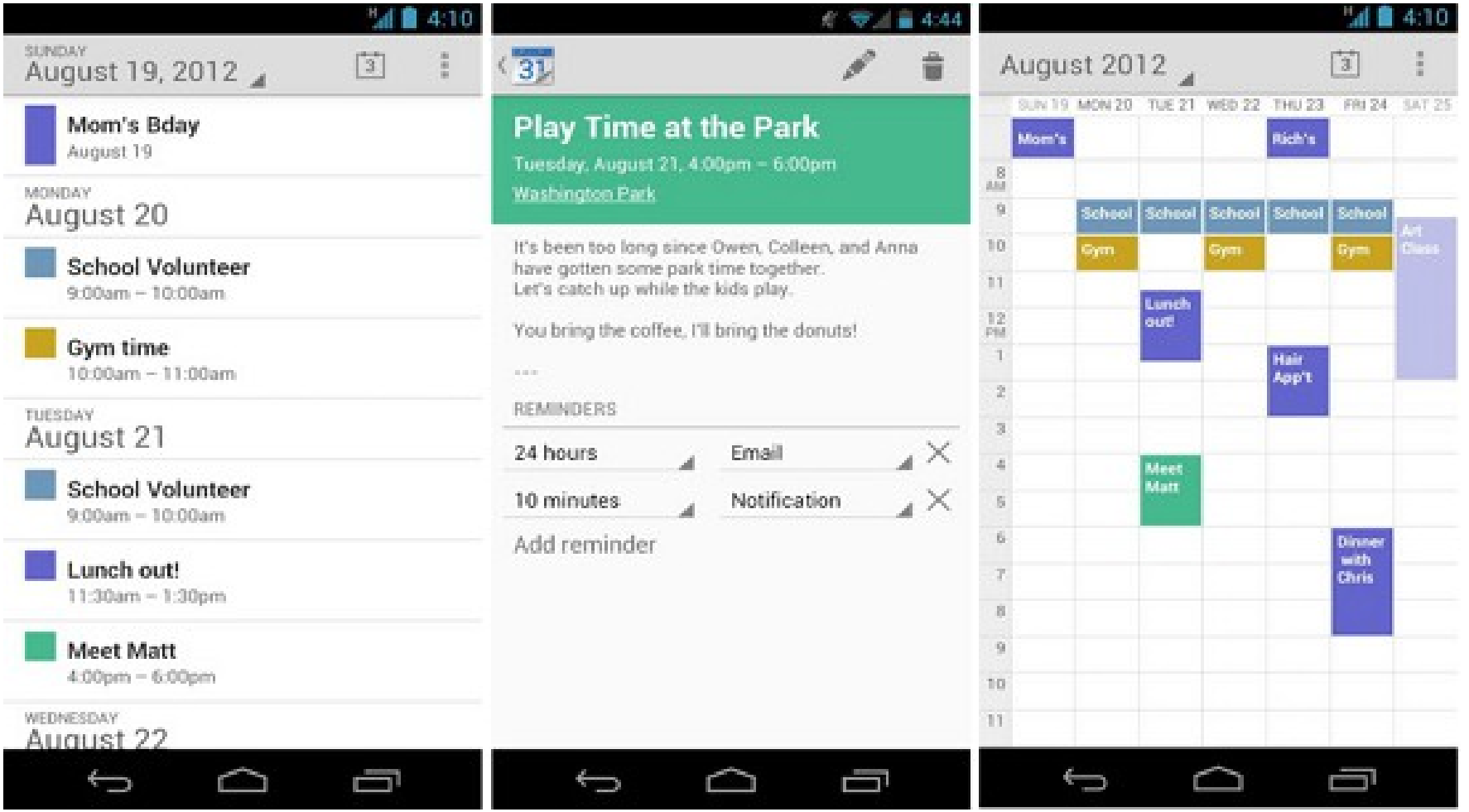
Get Basics Right
Function pointers

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

Function Pointer	Data Pointer
• Holds address of function	• Holds address of an object
• Pointing to a address from code segment.	• Pointing to a address from stack/heap/data
• Dereference to execute the function	• Dereference to get value from address
• Pointer arithmetic not valid	• 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.

Signals
Names

- Signals are standard, which are pre-defined
- Each one of them have a name and number
- Examples are follows:

Signal name	Number	Description
SIGINT	2	Interrupt character typed
SIGQUIT	3	Quit character typed (^\)
SIGKILL	9	Kill -9 was executed
SIGSEGV	11	Invalid memory reference
SIGUSR1	10	User defined signal
SIGUSR2	12	User defined signal

To get complete signals list, open /usr/include/bits/signum.h in your system.

Signals
Origins

- 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

Signals
Handling

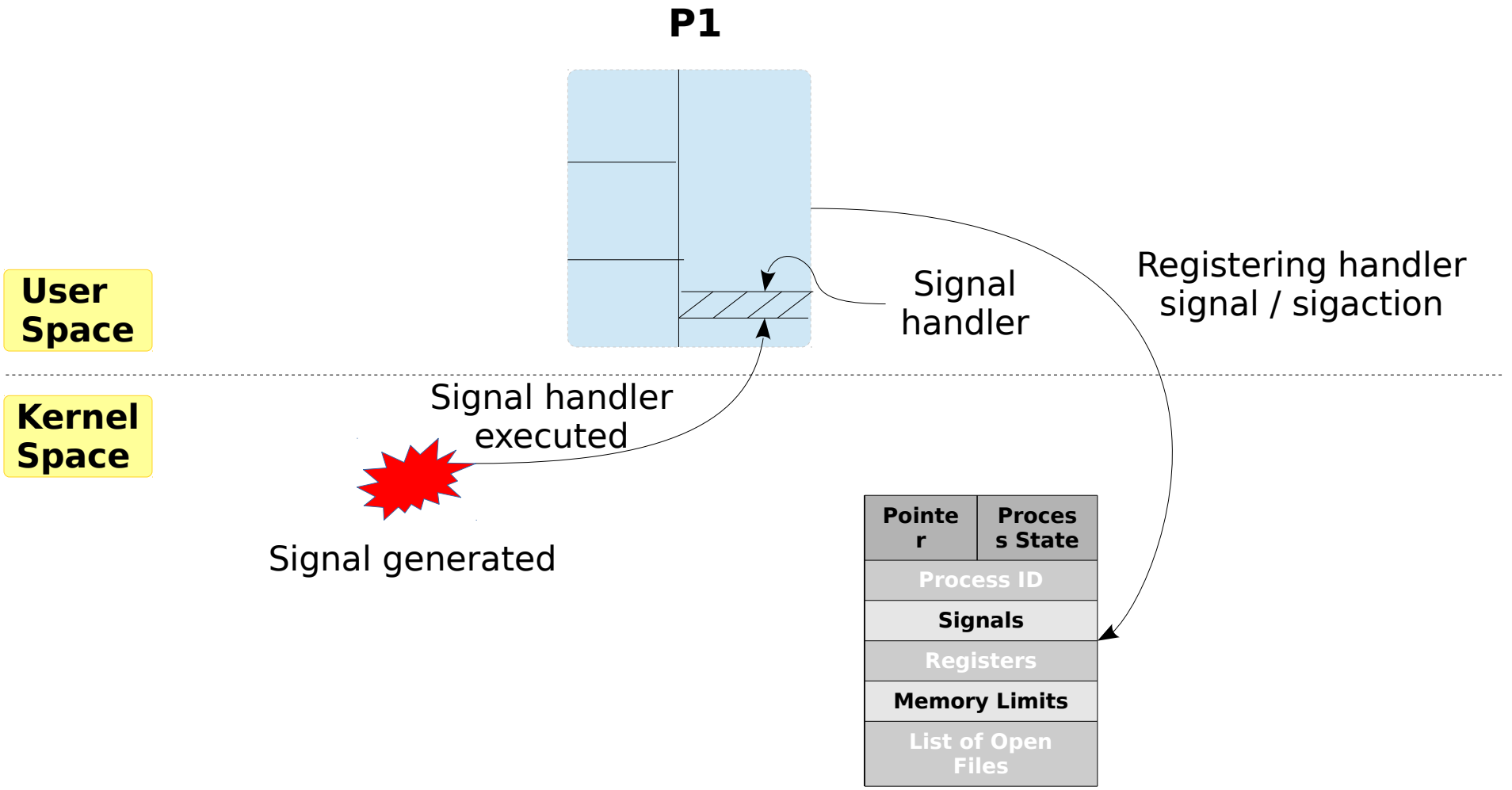
- When a process receives a signal, it processes by handling immediately.
- For all possible signals, the system defines a default disposition or action to take when a signal occurs
- There are four possible default dispositions:
  - Exit: Forces process to exit
  - Core: Forces process to exit and create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

Signals
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

Signals
Handling



Signals
Handler

- 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**

Signals
vs Interrupt

- Signals can be described as soft-interrupts
- The concept of 'signals' and 'signals handling' is analogous to that of the 'interrupt' handling done by a microprocessor
- When a signal is sent to a process or thread, a signal handler may be entered
- This is similar to the system entering an interrupt handler

- System calls are also soft-interrupts. They are initiated by applications.
- Signals are also soft-interrupts. Primarily initiated by the Kernel itself.

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**
sigaction(int signum, const struct sigaction *act, struct sigaction *oldact)	signum : Signal number that needs to be handled act: Action on signal oldact: Older action on signal

EMERTXE

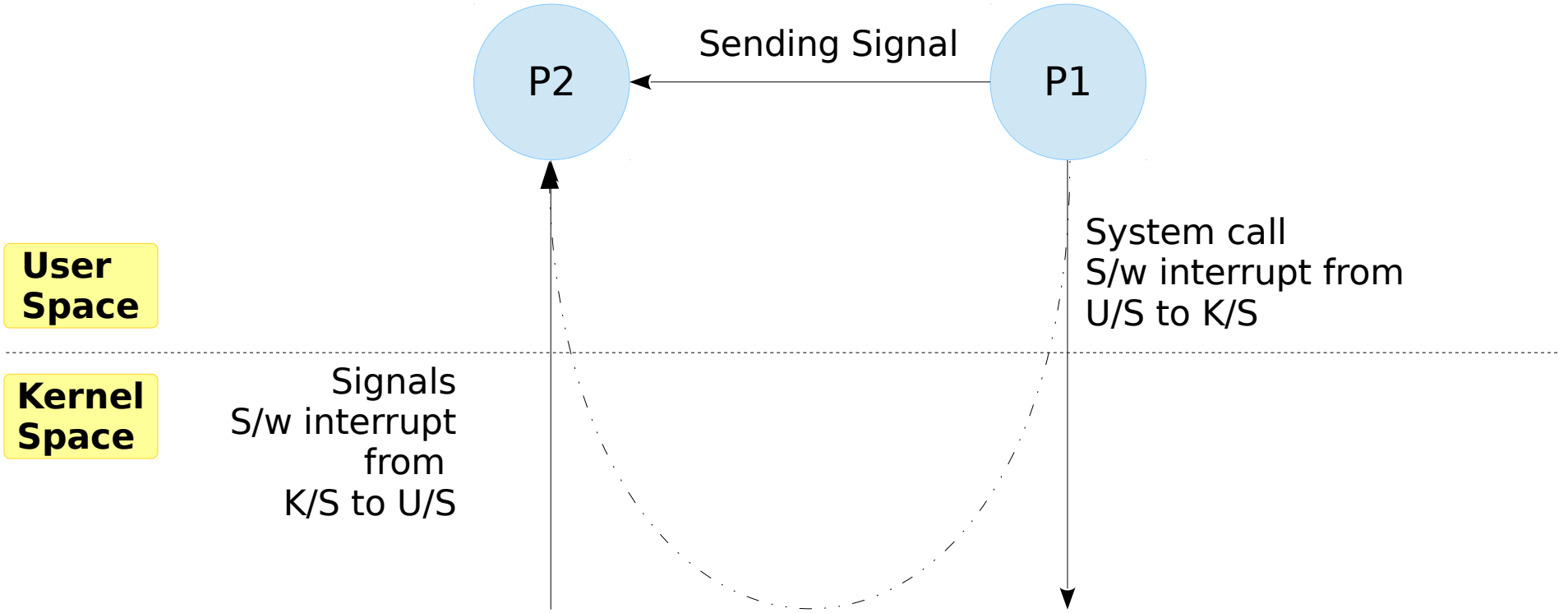
Signals
Advanced Handling – sigaction structure

```
struct sigaction
{
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_restorer)(void);
}
```

- sa_handler: SIG_DFL (default handling) or SIG_IGN (Ignore) or Signal handler function for handling
- Masking and flags are slightly advanced fields
- Try out sa_sigaction during assignments/hands-on session along with Masking & Flags

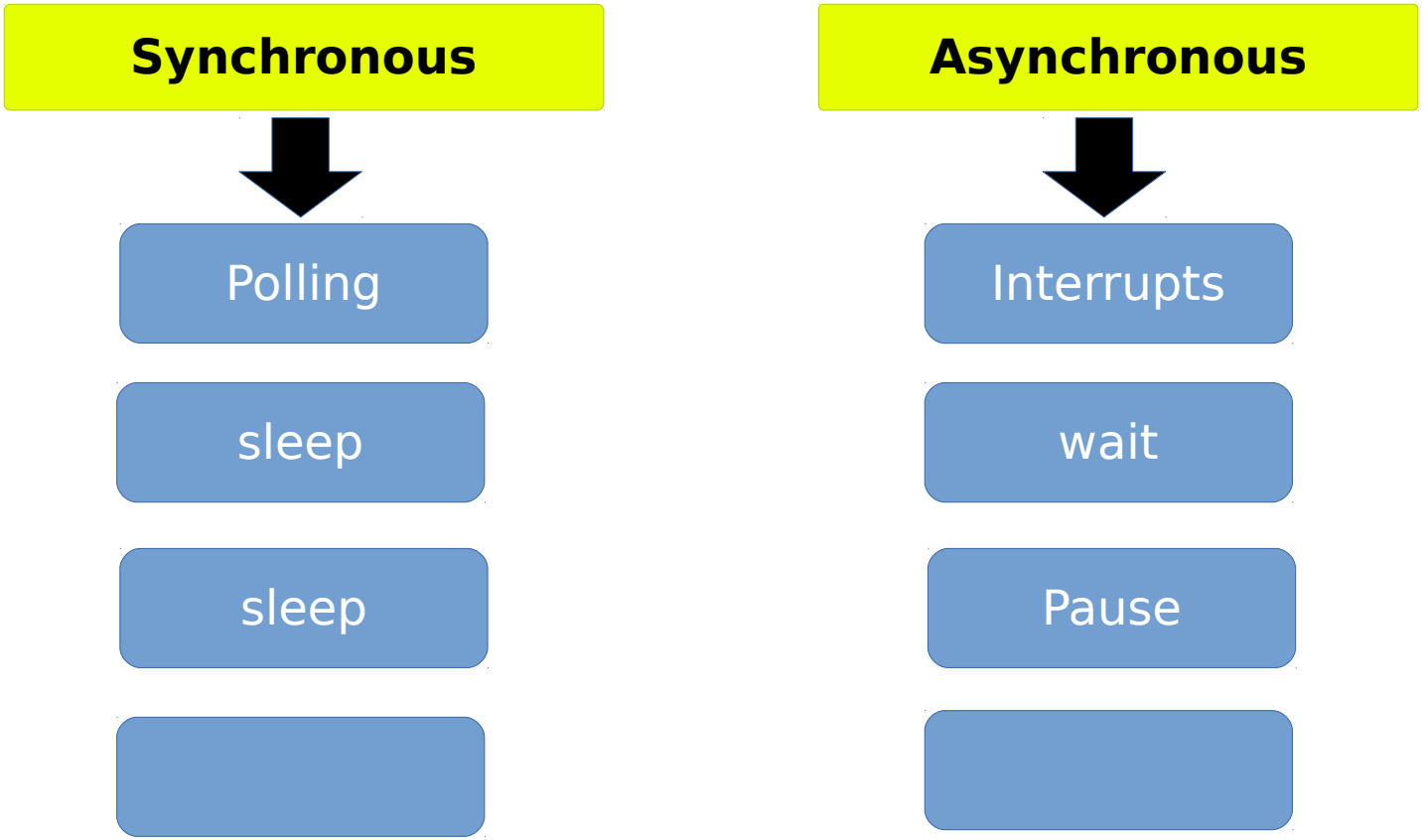
EMERTXE

Signals
vs system calls



Synchronous & Asynchronous

- Wait for child to finish



Signals
Self Signaling

- 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

Stay Connected

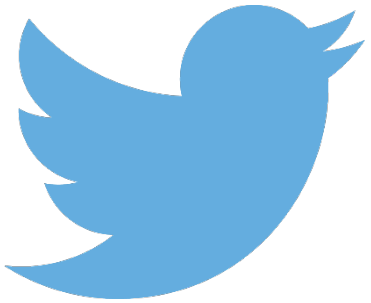
About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,

No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046

T: +91 80 6562 9666
E: training@emertxe.com

https://www.facebook.com/Emertxe



https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides

EMERTXE

Thank You