File Management

Application Programming vs System Programming

- Application Programming
- System Programming

File Management

File System

- The file system
  - manages files,
  - allocating file space,
  - administrating free space,
  - controlling access to files
  - and retrieving data for user.
- The internal representation of file is given an ***inode table***.

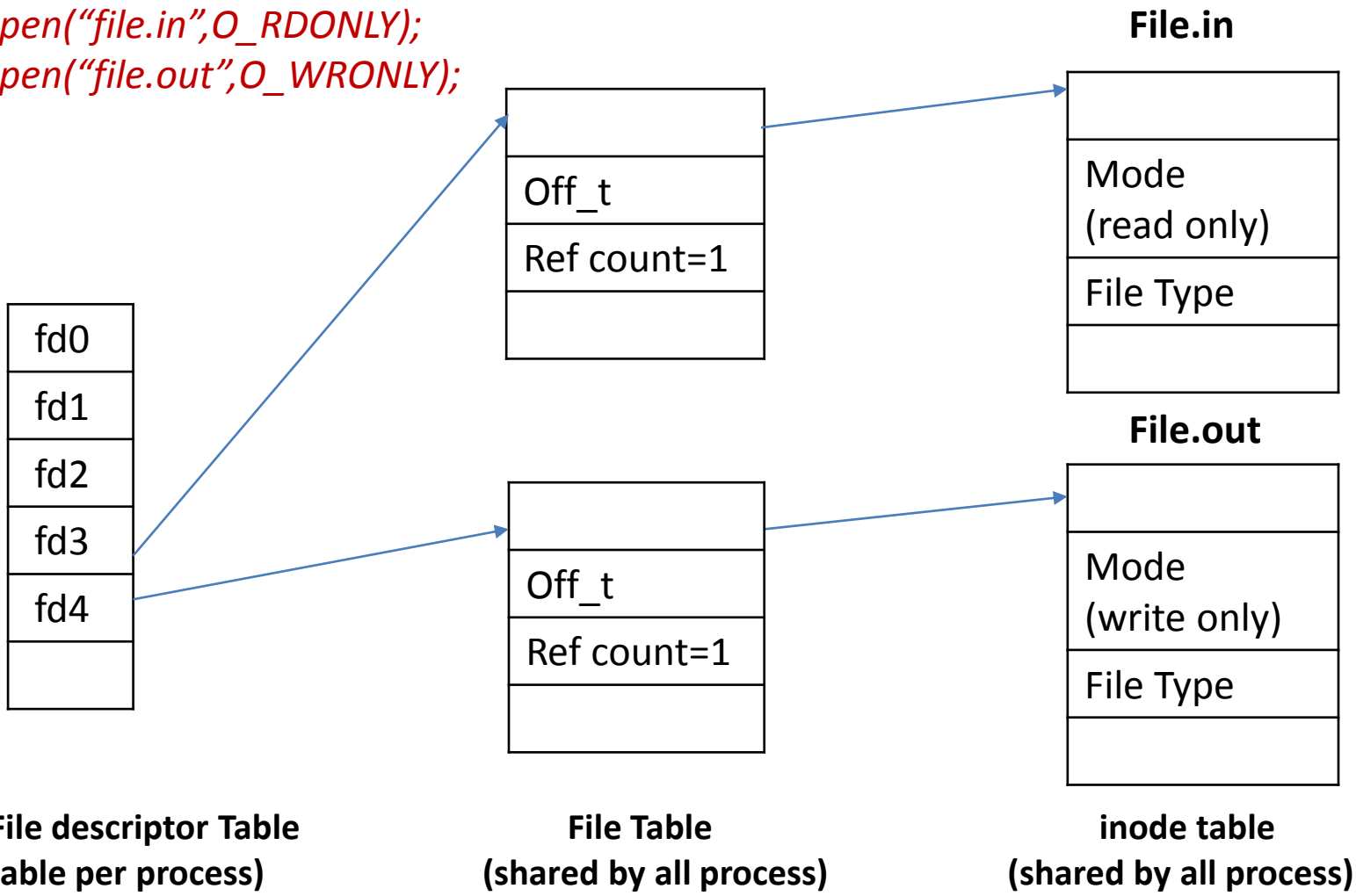
File System

- ***User File Descriptor table:*** Each process has its own separate *descriptor table* whose entries are indexed by the process's open file descriptors. Each open descriptor entry points to an entry in the *file table.*
- ***File table:*** Each file table entry consists of (for our purposes) the current file position, a *reference count* of the number of descriptor entries that currently point to it, and a pointer to an entry in the *inode table*.
- ***Inode table:*** Each entry contains most of the information in the stat structure, including the st_mode and st_size members.
- ***File table*** is global kernel structure where as ***user file descriptor table*** is allocated per process.

File System - Kernel Data Structures

In this example, two descriptors reference distinct files. There is no sharing.

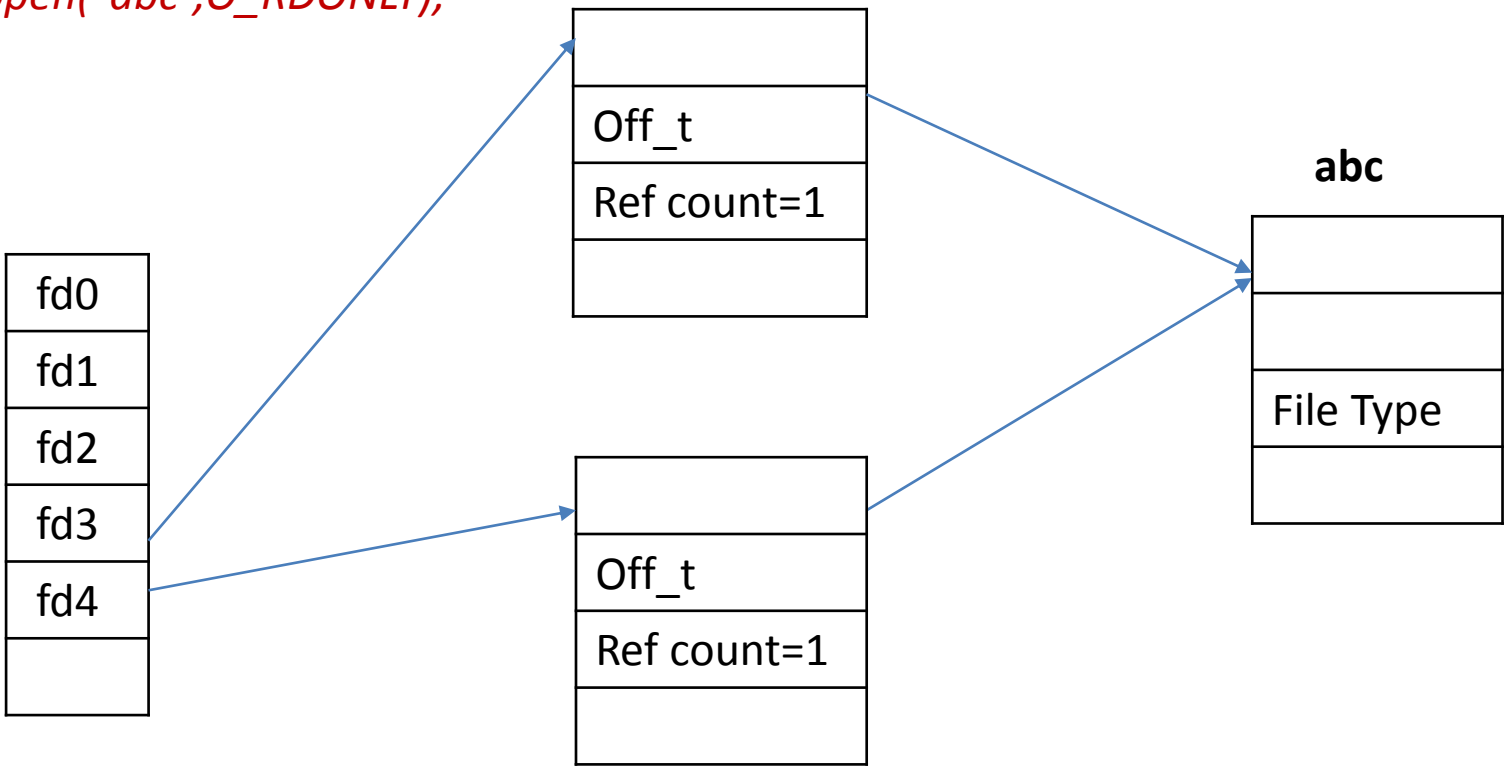
fd3=open("file.in",O_RDONLY);
fd4=open("file.out",O_WRONLY);



File System - Kernel Data Structures

This example shows two descriptors sharing the same disk file through two open file table entries.

fd3=open("abc",O_RDONLY);
fd4=open("abc",O_RDONLY);



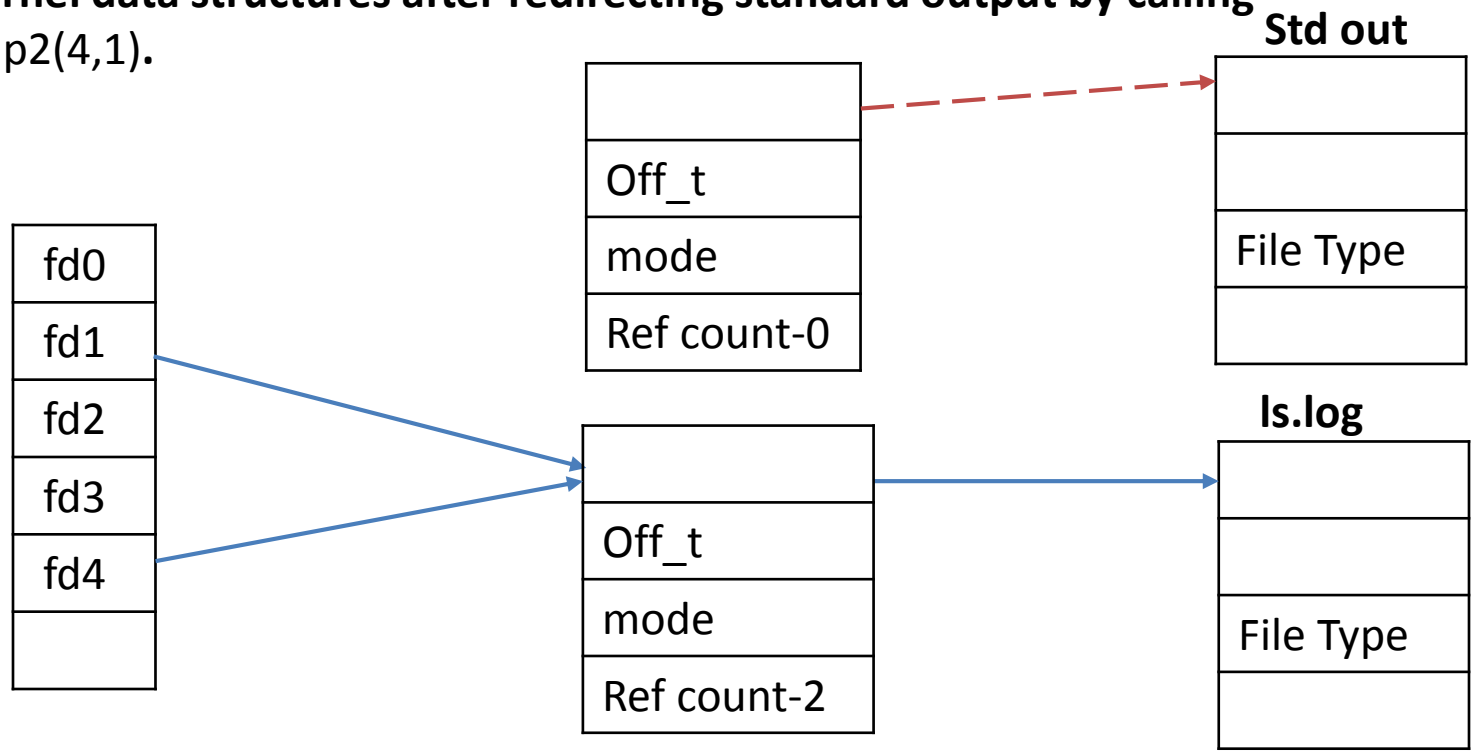
User File descriptor Table (one table per process)

File Table (shared by all process)

inode table (shared by all process)

I/O Redirection – ls > ls.log ?

Kernel data structures after redirecting standard output by calling dup2(4,1)**.**



User File descriptor Table (one table per process)

File Table (shared by all process)

inode table (shared by all process)

File System Layout

Boot Block: contain bootstrap code that is read into machine to boot, or initialize, the operating system.

Super Block: Describes the state of a file system – How large it is, how many files it can store, where to find free space on the file system.

Inode (index node)list: inode represents the type of the file.

Data Block: contain file data and administrative data.

Monitoring File System Events

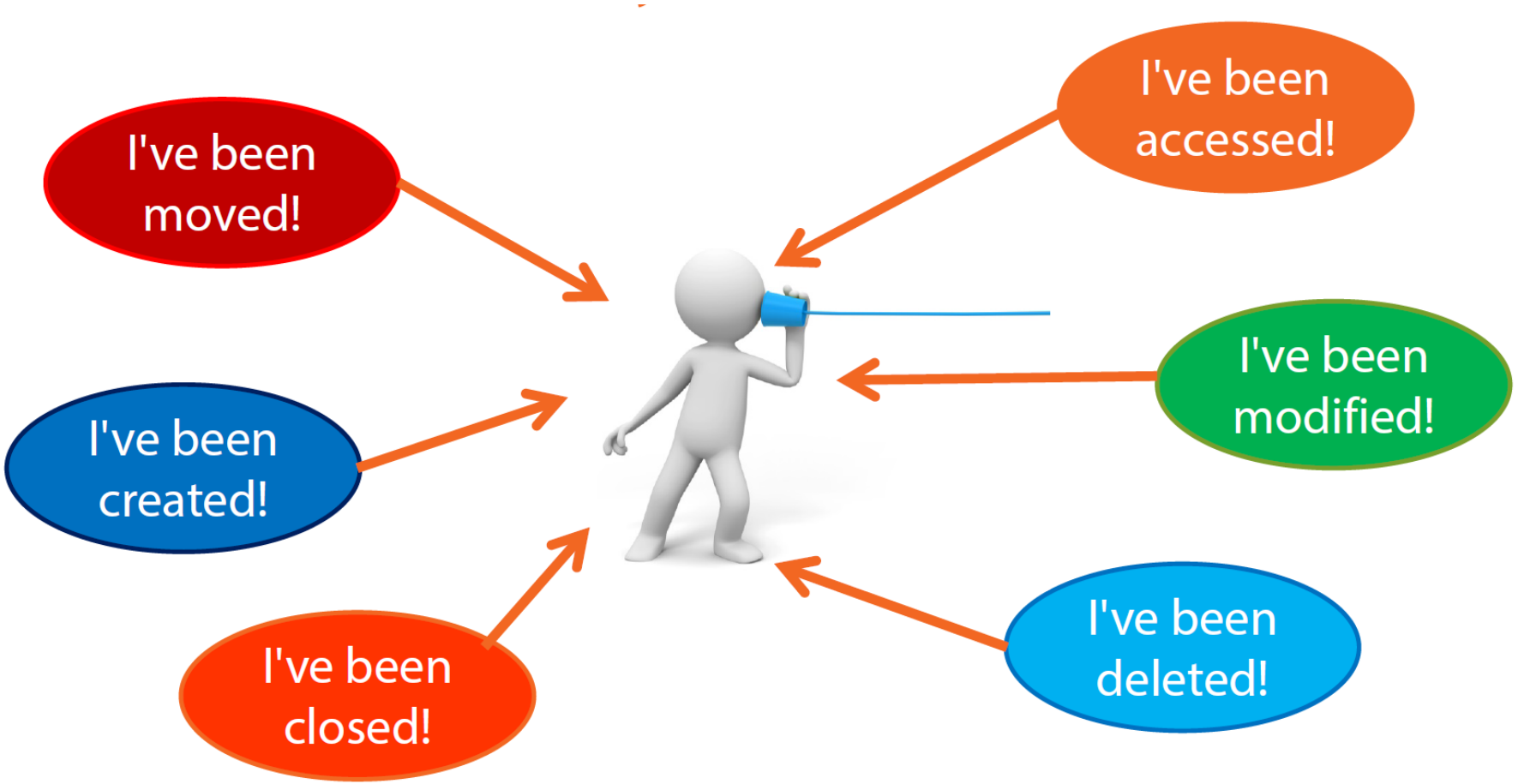
$ tail -f <file_name>

-follow: output appended data as the file grows.

Monitoring File System Events

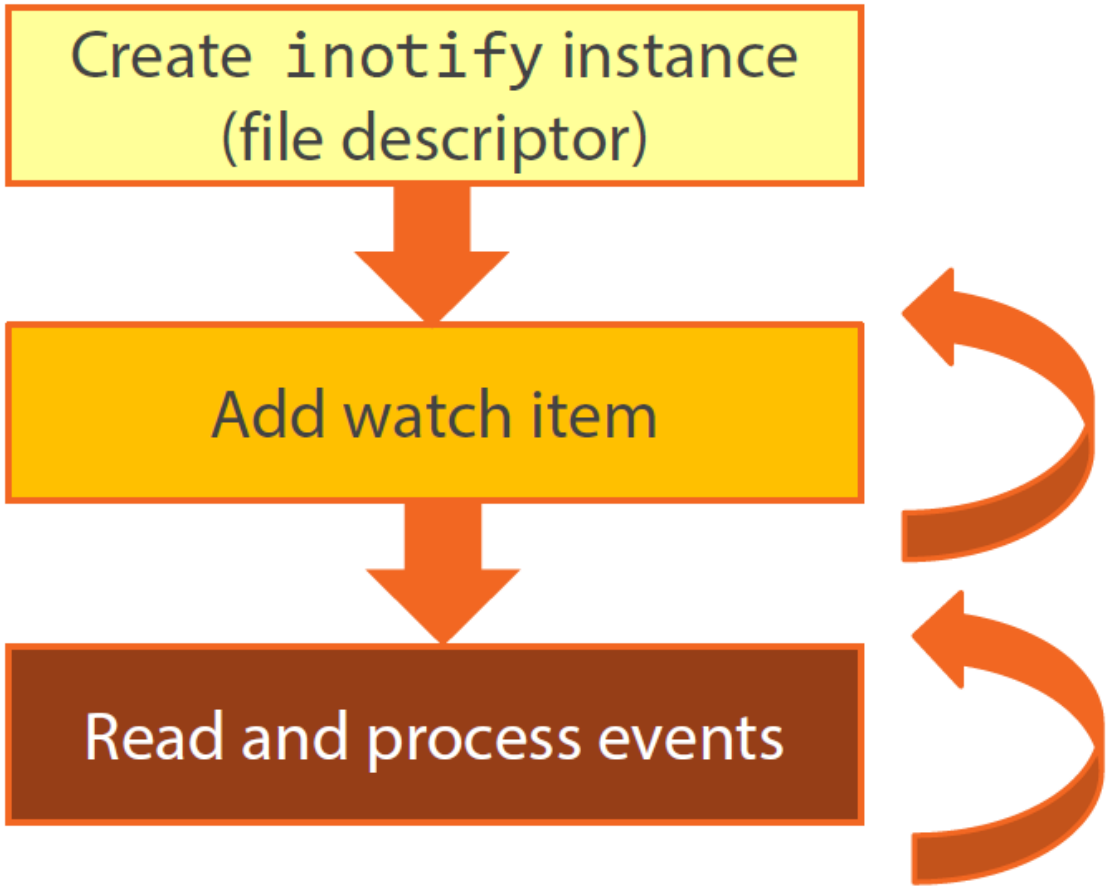
- The inotifyAPI
- Creating an inotifyinstance
- Adding to the watch list
- Reading events

File System Events



Individual files or whole directories can be watched

Three Steps

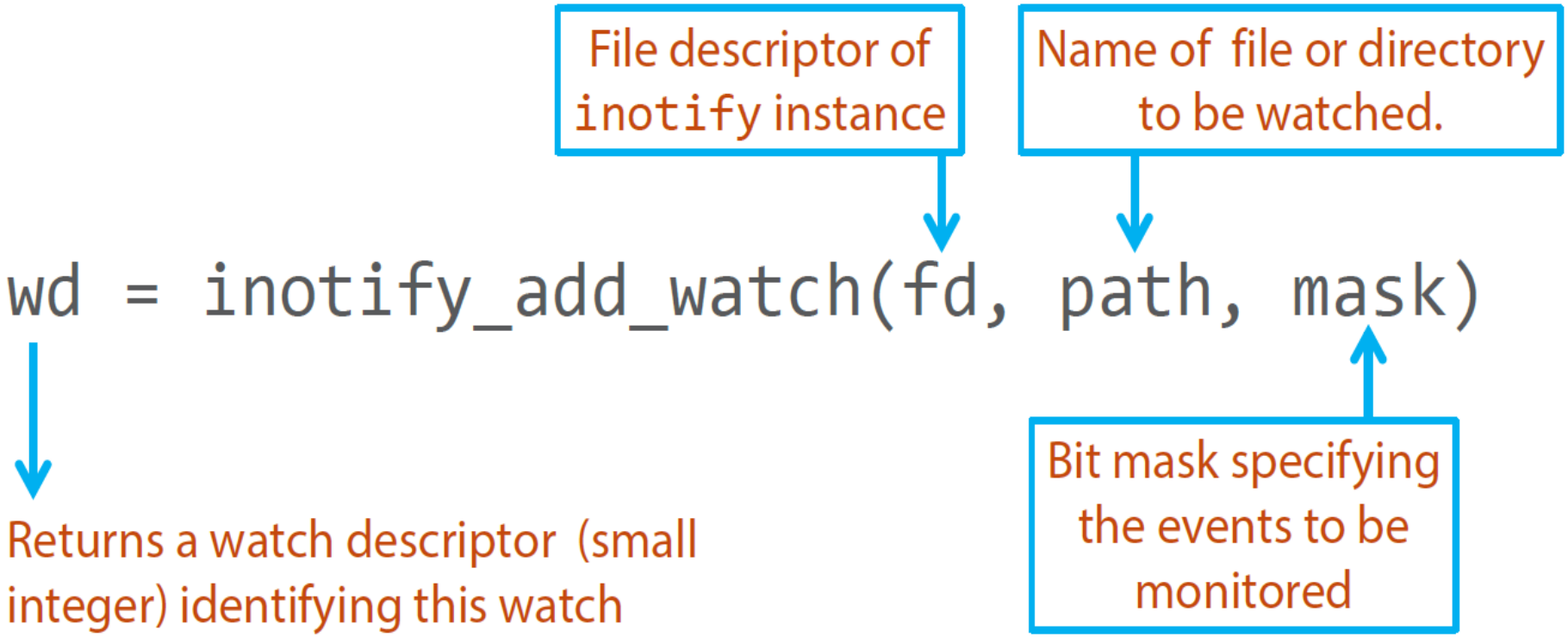

Create inotify instance
(file descriptor)
Add watch item
Read and process events

Creating an inotify Instance

```
fd = inotify_init()
```

Returns a file descriptor on whichwe can later read() the events.

Adding a watch Item

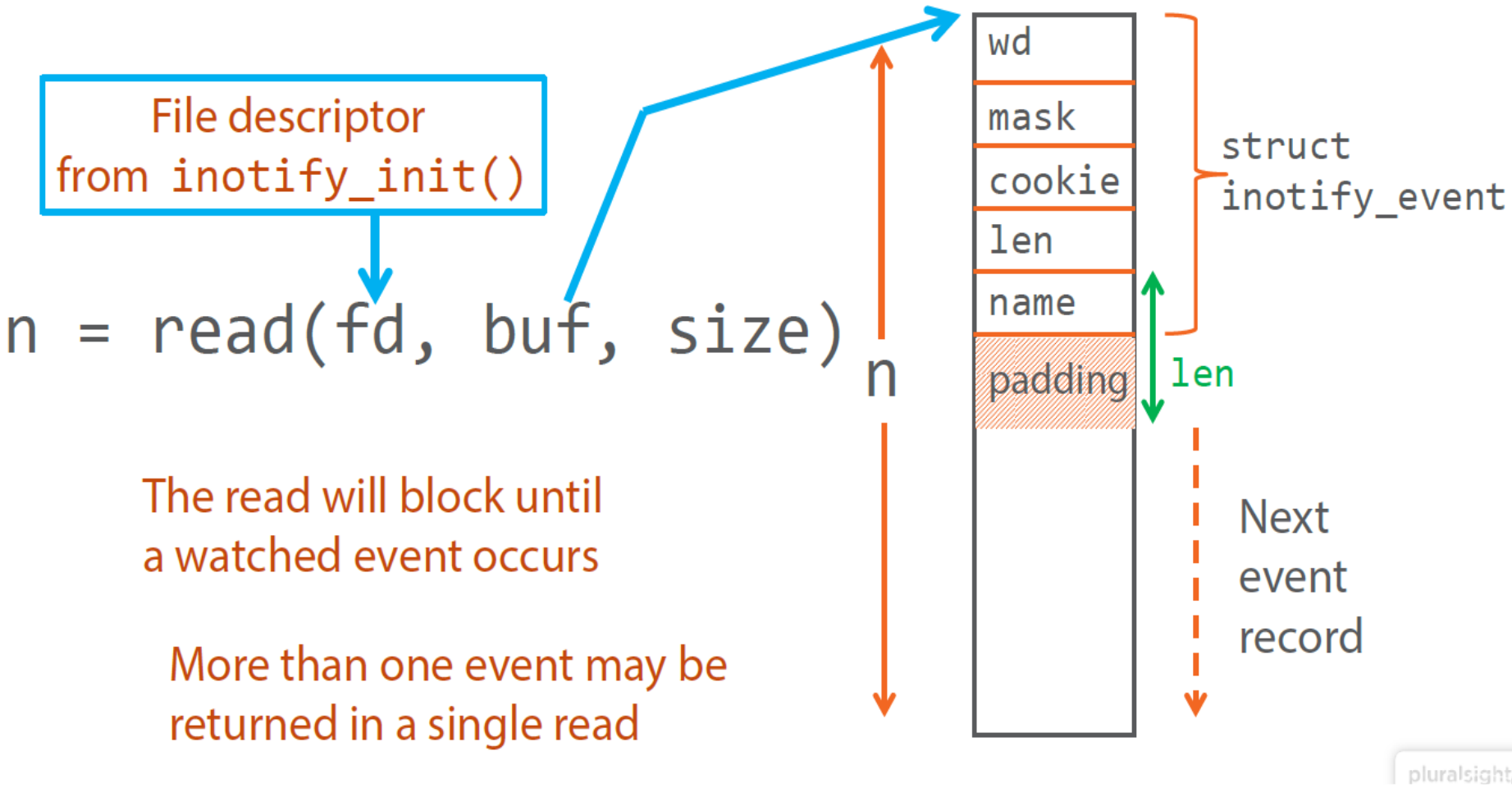


Watches and Events in Detail

Each event is specified by a single-bit constant
-- bitwise OR them together

Bit value	Meaning
IN_ACCESS	File was accessed
IN_ATTRIB	File attributes changed (ownership, permissions etc.)
IN_CREAT	File created inside watched directory
IN_DELETE	File deleted inside watched directory
IN_DELETE_SELF	Watched file deleted
IN_MODIFY	File was modified
IN_MOVE_SELF	File was moved

Reading Events



Demonstration

- Read a list of files (*not*directories)from a configuration file
- Watch them for modification or deletion
- Report changes to a log file

Process Management

Process Termination

```
exit(n);
```

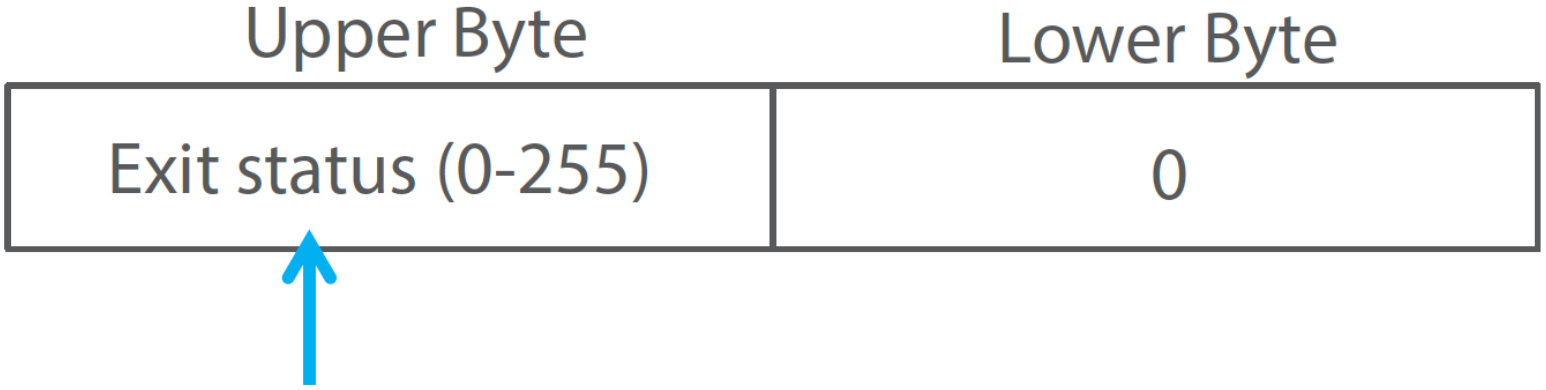
Exit status
Passed back to parent
0 means success
1-255 means failure

```
int status;
wait(&status);
```

Call waits until a child process terminates.
Returns PID of the child

The child's exit status is returned here.
Pass 0 (NULL) if not interested

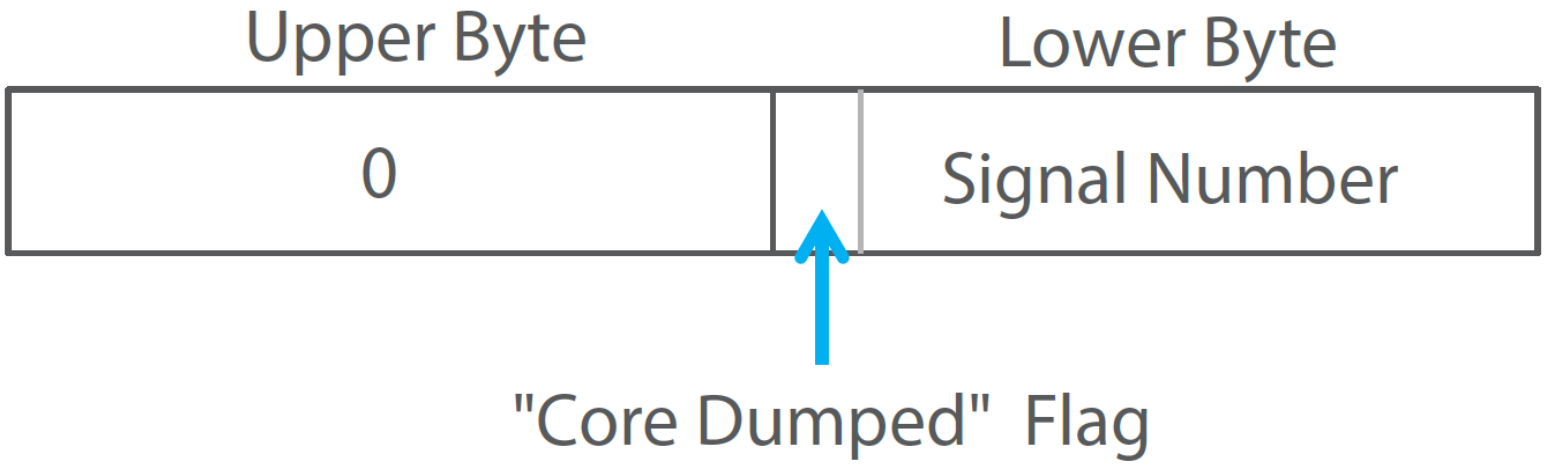
Exit Status – Normal Termination



Conventionally: zero = success, nonzero = "failure"

MACRO	Meaning
`WIFEXITED(status)`	True if child exited normally
`WEXITSTATUS(status)`	The exit status

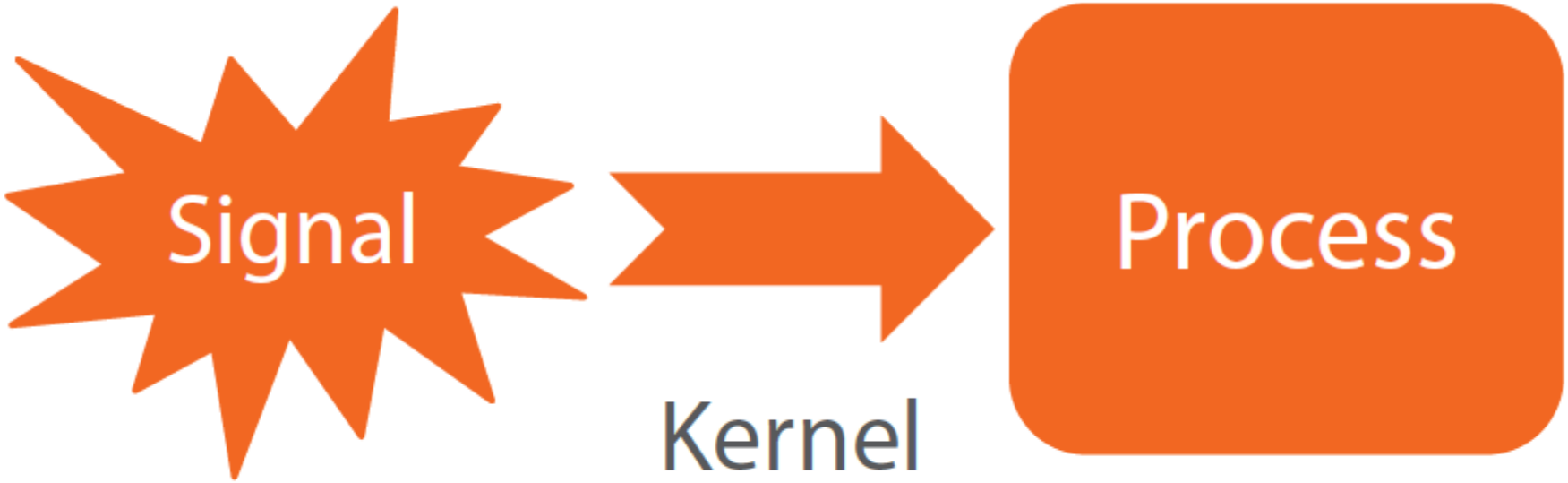
Exit Status – Killed by Signal



MACRO	Meaning
`WIFSIGNALED(status)`	True if child terminated by signal
`WTERMSIG(status)`	The signal number

Signals

What Is a Signal?



Signals

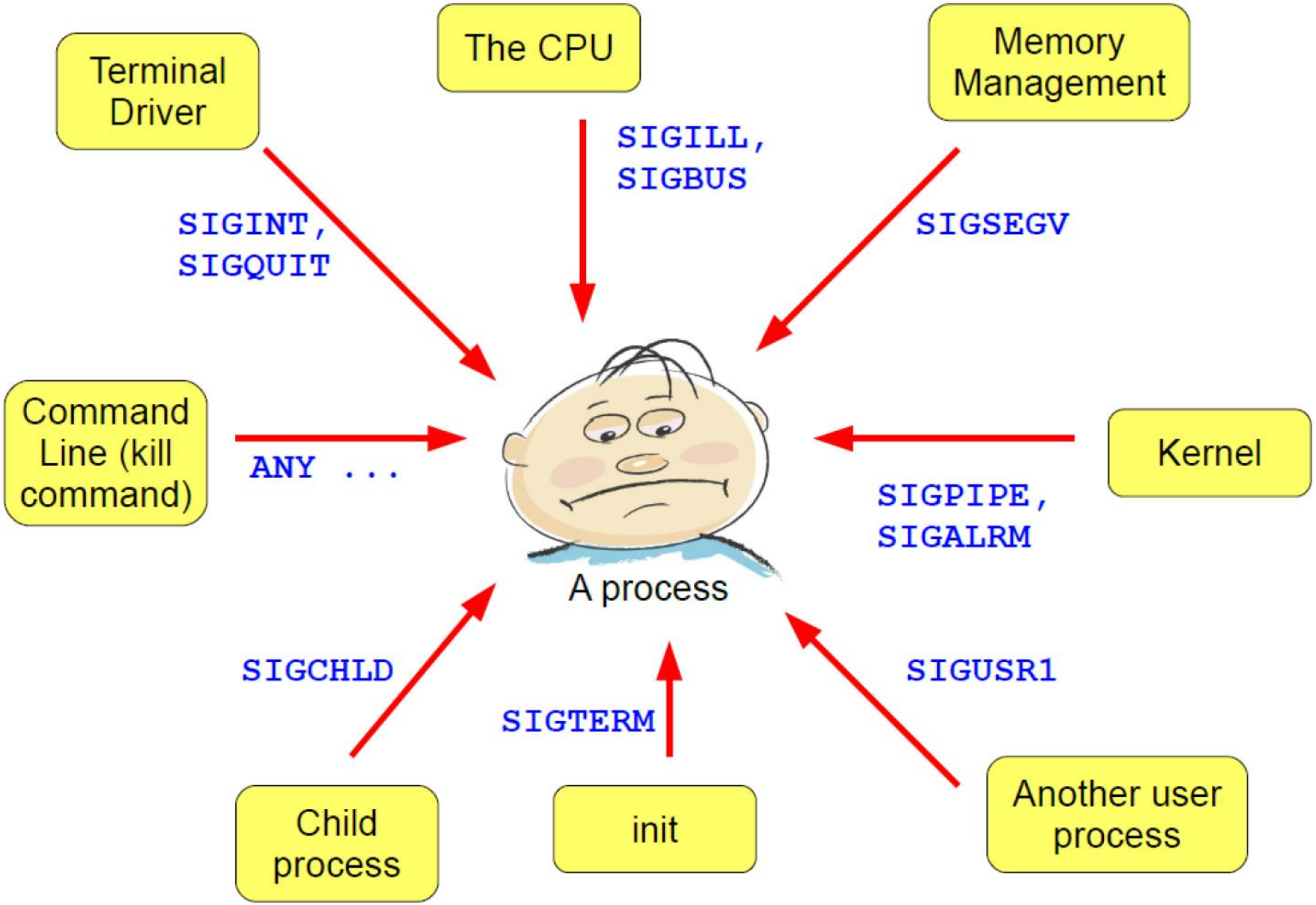
- *Signals* are software interrupts for handling asynchronous events
  - External – eg. the interrupt character (Ctrl-C)
  - Internal – as when the process divides by zero
  - A process can also send a signal ("raise") to another process.

Action	Description
Term	Default action is to terminate the process.
Ign	Default action is to ignore the signal.
Core	Default action is to terminate the process and dump core
Stop	Default action is to stop the process.
Cont	Default action is to continue the process if it is currently stopped.

Signal Types

Signal Name	Number	Default Action	Description
SIGHUP	1	Term	Some daemons interpret this to mean "re-read your configuration file"
SIGINT	2	Term	The signal sent by ^C on terminal
SIGTRAP	5	Core	Trace/breakpoint trap
SIGFPE	8	Core	Arithmetic error, e.g. divide by zero
SIGKILL	9	Term	Lethal signal, cannot be caught or ignored
SIGSEGV	11	Core	Invalid memory reference
SIGALRM	14	Term	Expiry of alarm clock timer
SIGTERM	15	Term	Polite "please terminate" signal
SIGCHLD	17	Ignore	Child process has terminated

Signals


Terminal Driver
The CPU
Memory Management
SIGILL, SIGBUS
SIGINT, SIGQUIT
SIGSEGV
Command Line (kill command)
ANY ...
Kernel
SIGPIPE, SIGALRM
A process
SIGCHLD
SIGTERM
SIGUSR1
Child process
init
Another user process

Signals

- **Signal life cycle**
  - A signal is “raised”
  - Kernel stores and delivers the signal
  - The process handles the signal

- **Signal handling**
  - **SIGKILL & SIGSTOP** cannot be ignored.
  - Catch and handle the signal by registered functions (signal handlers)
    - SIGINT and SIGTERM are two commonly caught signals.
  - Default action – terminate the process (result in core dump)

Signals

The following system calls and library functions allow the caller to send a signal:

- **Sending a signal:**
  - raise(3) Sends a signal to the calling thread.
  - kill(2) Sends a signal to a specified process, to all members of a specified process group, or to all processes on the system.
- **Catching a signal:**
  - sigaction(2) or signal(2) process can change user defined signal.
- **Waiting for a signal**
  - pause(2) Suspends execution until any signal is caught.

Process Priority

Threads

Process vs Thread

Process

- Process is an execution of a program. Program by itself is not a process.
- Process is a parallel execution.
- Whenever you are creating a process text segment, data segment, stack segments are created.

Thread

- Thread is a smallest unit of execution (Light weight process).
- Thread is a sequence of control with in a process.
- Whenever you are creating a thread it will create only stack segment and share text and data segment. i.e., why we are saying thread is smallest unit of execution.

Pthreads

POSIX 1003.1c

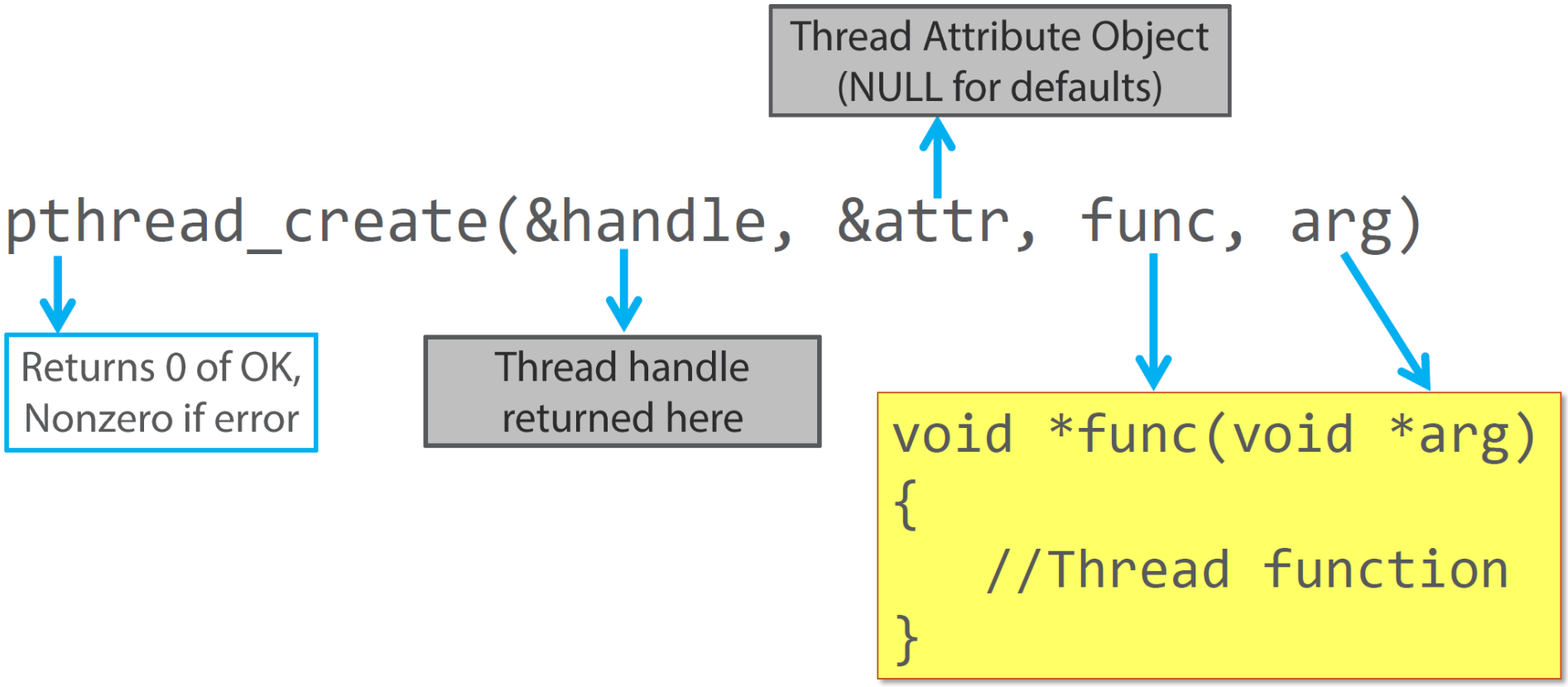
A standardised set of C library routines for thread creation and management

— Upwards of 60 functions

Threads

- Implementation thread is almost same as a function like function declaration, function definition except that function invocation.
- **How to invoke a Thread function:**
  - Whenever one thread blocks than other thread runs.

Thread Creation



Thread Termination

- A thread can terminate in several ways:
  1. By calling `pthread_exit(exit_status)`
  2. By returning from its top-level function
  3. By some other thread sending a cancellation request:

     `pthread_cancel(handle);`
- Parent can wait for thread to finish and get exit status:
  - `pthread_join(handle, &exit_status);`
- Parent should detach thread if they don't need to join on it:

  `pthread_detach(handle);`

Thread Life Cycle