Linux Internals & Networking

System programming using Kernel interfaces

Team Emertxe

ΣMERTXE

Contents

Linux Internals & Networking

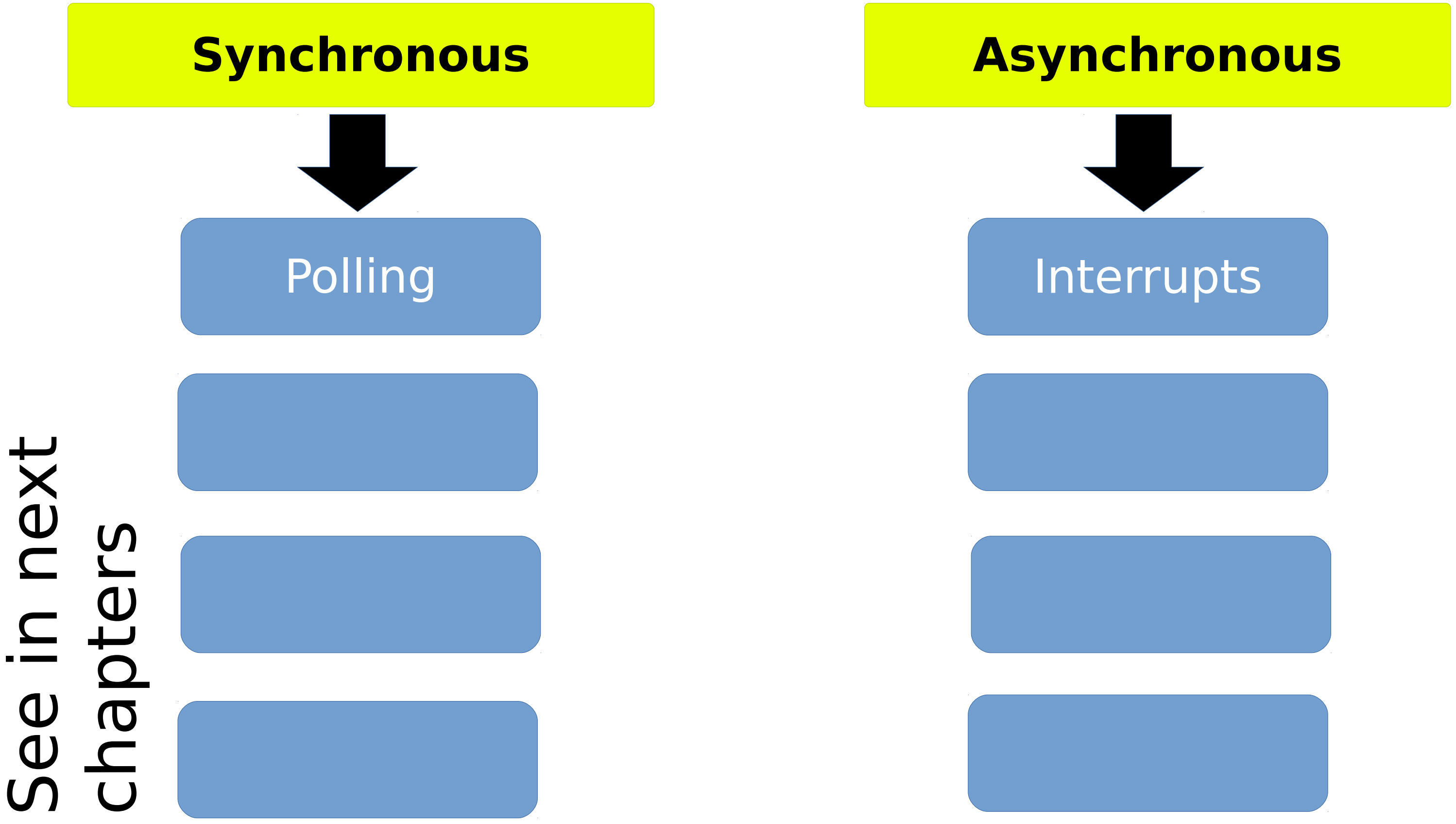
Contents

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

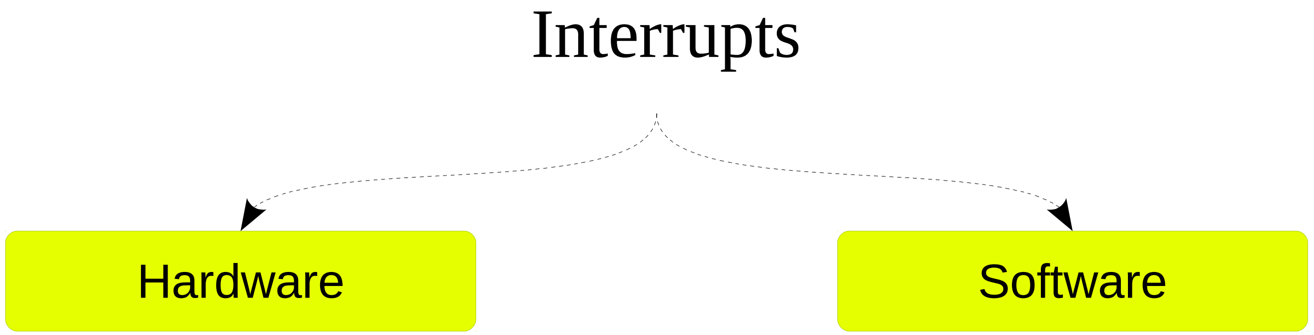
System Calls

Synchronous & Asynchronous

- Communications are two types



Interrupts


Interrupts

Hardware

Software

Generates whenever a
Hardware changes happens

Generates by instruction from
code (eg: INT 0x80)

- Interrupt controller signals CPU that interrupt has occurred, passes interrupt number
- Basic program state saved
- Uses interrupt number to determine which handler to start
- CPU jumps to interrupt handler
- When interrupt done, program state reloaded and program resumes

System calls

- What?
  - Is a controlled entry point into the kernel, allowing a process to request that the kernel perform some action on the process’s behalf.
  - The kernel makes a range of services accessible to programs via the system call application programming interface (API).
  - A set of interfaces to interact with hardware devices such as the CPU, disks, and printers.
  - **Example:**
    - Creating new process
    - Performing I/O
    - Creating PIPE for IPC

System calls

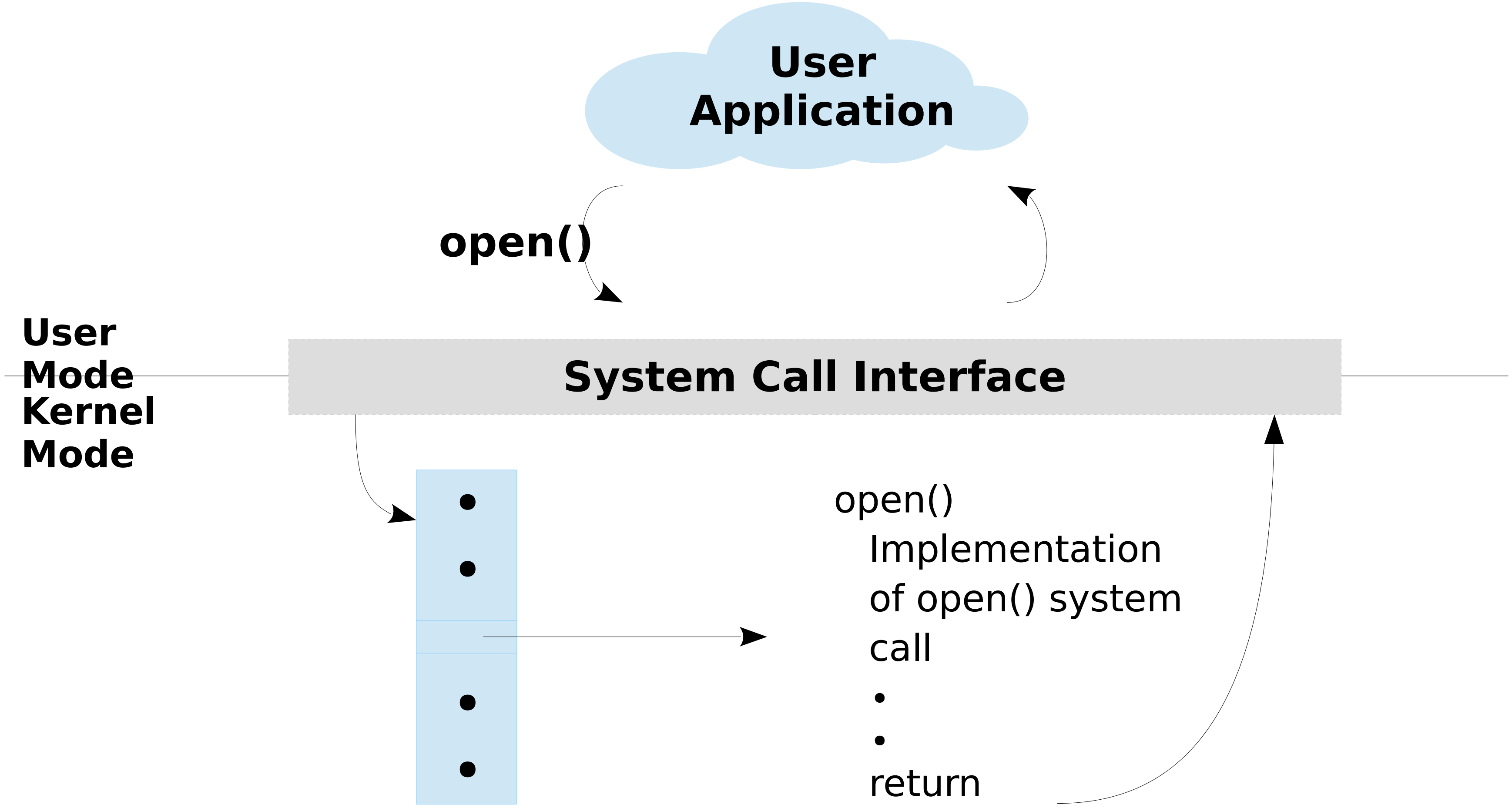
- General points to be noted
  - A system call changes the processor state from user mode to kernel mode, so that the CPU can access protected kernel memory.
  - Each system call is identified by a unique number.
  - Each system call may have a set of arguments that specify information to be transferred from user space to kernel space and vice versa.

System calls

- Advantages:
  - Freeing users from studying low-level programming
  - It greatly increases system security
  - These interfaces make programs more portable

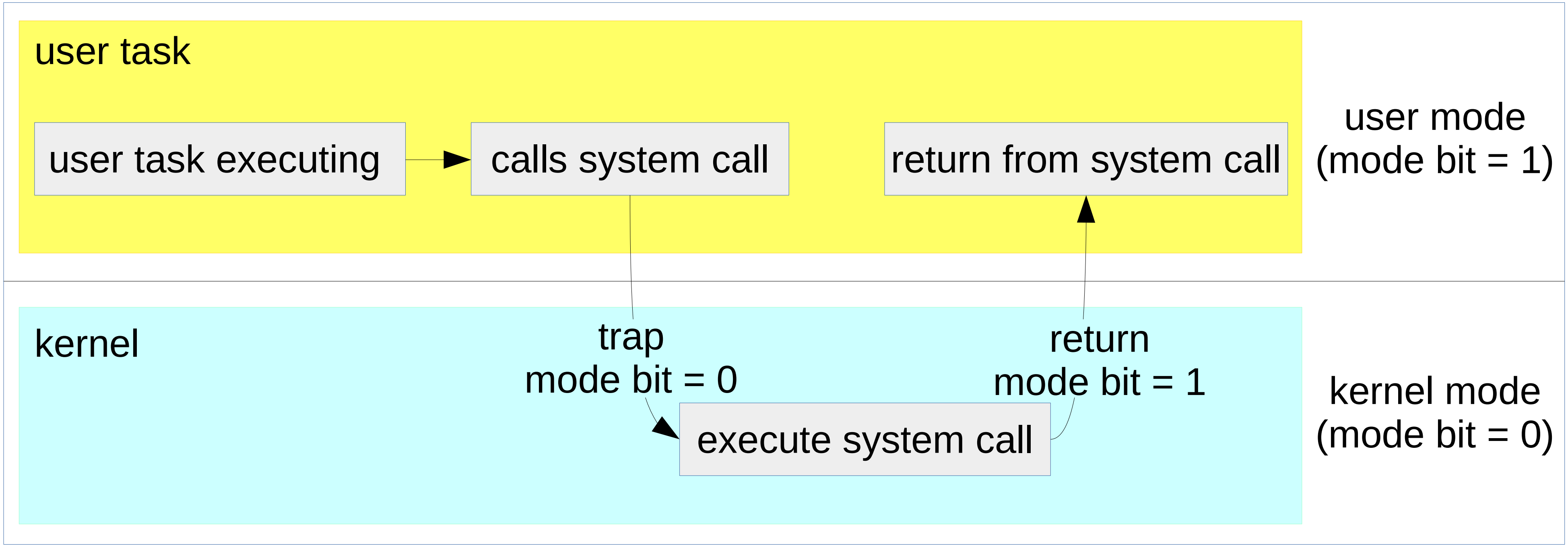
For a OS programmer, calling a system call is no different from a normal function call. But the way system call is executed is way different.

System calls



System Call

Calling Sequence



Logically the system call and regular interrupt follow the same flow of steps. The source (I/O device v/s user program) is very different for both of them. Since system call is generated by user program they are called as ‘Soft interrupts’ or ‘Traps’

System Call
vs Library Function

- A library function is an ordinary function that resides in a library external to your program. A call to a library function is just like any other function call

- A system call is implemented in the Linux kernel and a special procedure is required in to transfer the control to the kernel

- Usually, each system call has a corresponding wrapper routine, which defines the API that application programs should employ

✓ Understand the differences between:
  - Functions
  - Library functions
  - System calls

✓ From the programming perspective they all are nothing but simple C functions

System Call

Steps

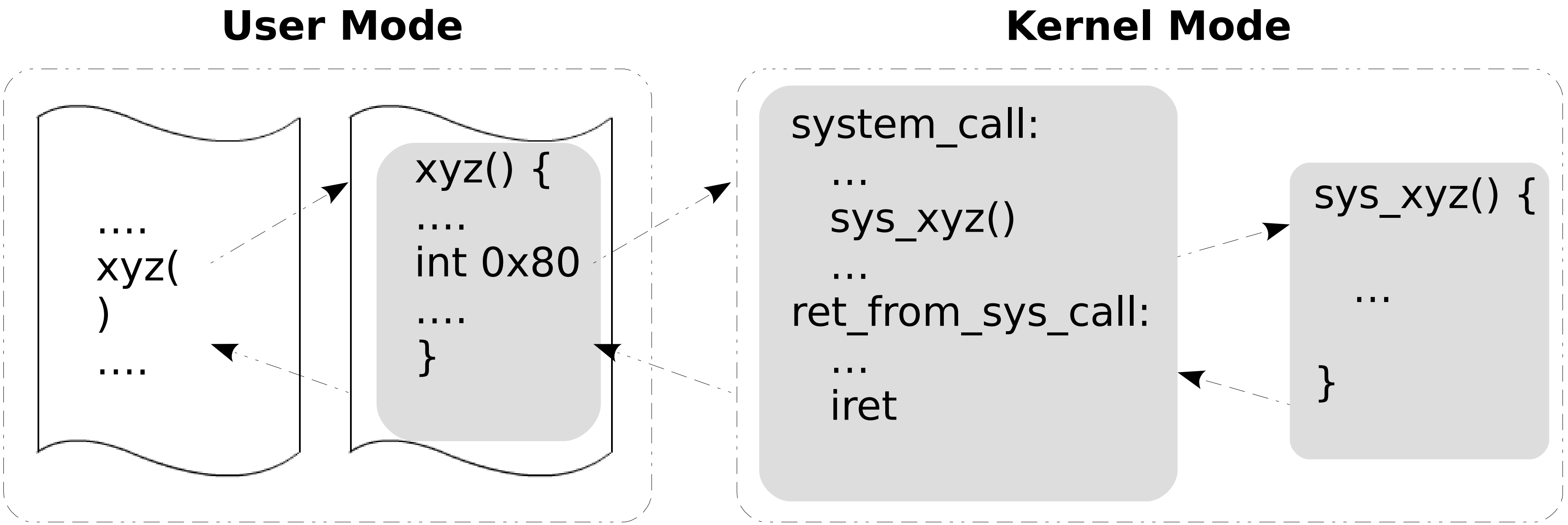
- The application program makes a system call by invoking a wrapper function in the C library.
- The wrapper function must make all of the system call arguments available to the system call trap-handling routine.
  - These arguments are passed to kernel via specific registers and the stack.
- All system calls are identified by the unique numbers by the kernel.
  - The wrapper function copies the system call number into a specific CPU register (%eax).
- The wrapper function executes a trap machine instruction (int 0x80).
  - This causes the processor to switch from user mode to kernel mode and execute code pointed to by location 0x80 of the system's trap vector.

System Call

Steps

- The kernel invokes system_call() routine to handle the trap. This handler:
  - Saves register values onto the kernel stack
  - Checks the validity of the system call number.
  - Invokes the appropriate system call service routine, from the system call table.
  - The service routine returns a result status to the system_call() routine.
- Restores register values from the kernel stack and places the system call return value on the stack.
- Returns to the wrapper function, simultaneously returning the processor to user mode.
- Note:
  - If the return value of the system call service routine indicated an error, the wrapper function sets the global variable errno.
  - The wrapper function then returns to the caller, providing an integer return value indicating the success or failure of the system call.

System Call
Implementation



```
....
xyz(
)
....
```

```
xyz() {
....
int 0x80
....
}
```

```
system_call:
   ...
   sys_xyz()
   ...
ret_from_sys_call:
   ...
   iret
```

```
sys_xyz() {

   ...

}
```

System Call Invocation in application program

Wrapper routine in libc standard library

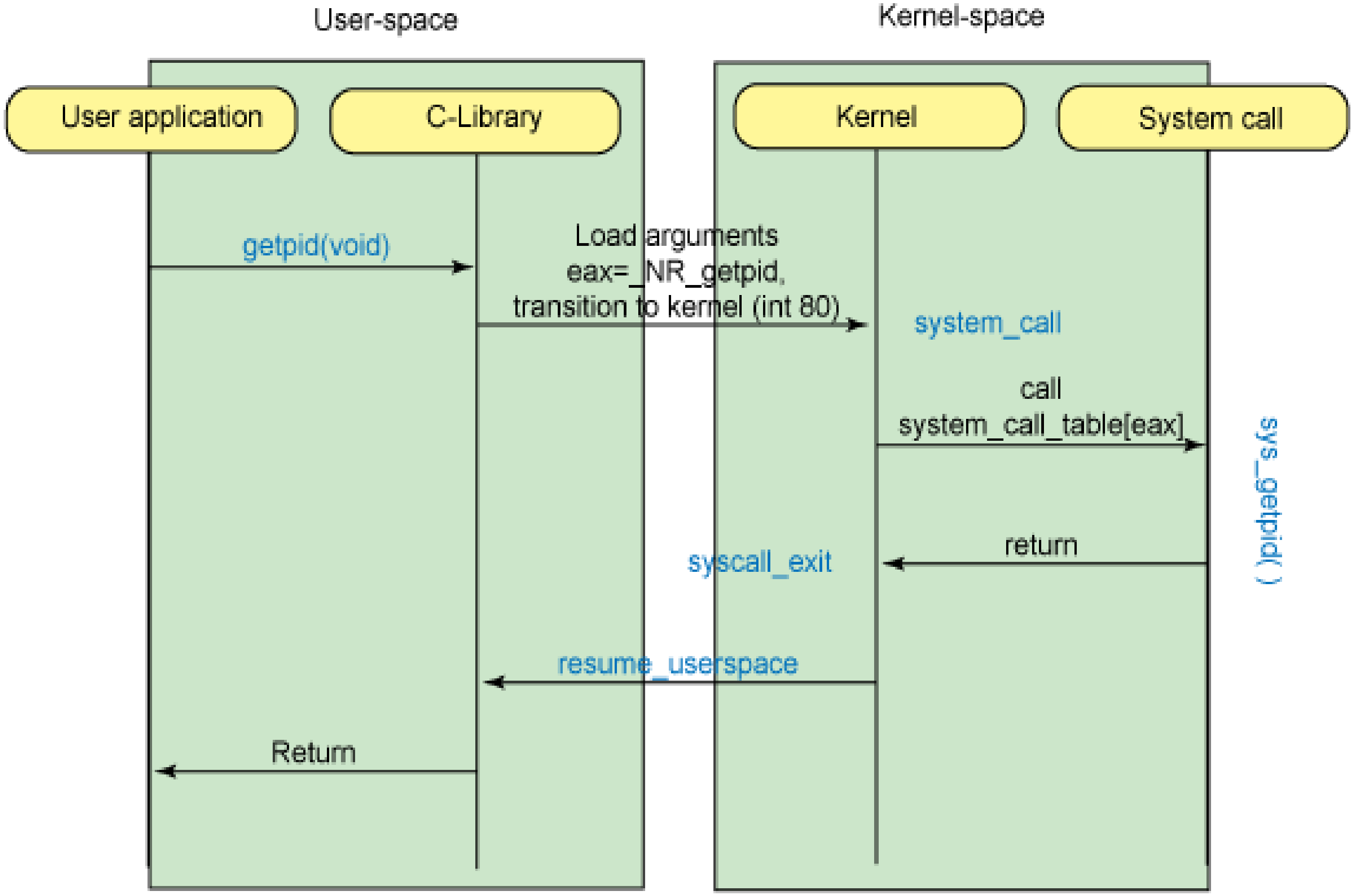
System call handler

System call service routine

- ✓ Use strace command, to trace the system calls made by a program, either for debugging purposes or simply to investigate what a program is doing.

System Call

Simple Flow



System Call

Example: gettimeofday()

- Gets the system’s wall-clock time.
- It takes a pointer to a struct timeval variable. This structure represents a time, in seconds, split into two fields.
  - tv_sec field - integral number of seconds
  - tv_usec field - additional number of usecs

System Call

Example: nanosleep()

- A high-precision version of the standard UNIX sleep call
- Instead of sleeping an integral number of seconds, ***nanosleep*** takes as its argument a pointer to a ***struct timespec*** object, which can express time to nanosecond precision.
  - tv_sec field - integral number of seconds
  - tv_nsec field - additional number of nsecs

System Call

Example: Others

- open
- read
- write
- exit
- close
- wait
- waitpid
- getpid
- sync
- nice
- kill etc..

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

Thank You