BITS Pilani
Pilani Campus

Network Programming

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

History of UNIX & C

A Brief History of Time (UNIX and C)

innovate achieve lead

- 1969 – First Unix Ken Thompson at AT&T Bell Labs
  - Ideas from Multics:
    - Tree structured file system
    - Program for interpreting commands (shell)
    - Files – unstructured streams of bytes
- 1970 Unix rewritten in assembly for DEC PDP-11
- C – Dennis Ritchie – a systems programming language
  - BCPL ➔ B (Thompson) ➔ C
- 1973 Kernel rewritten in C – eases porting to other machines.
- 1977 Bell Labs released Unix System III
- 1982 AT & T released Unix System V.

Berkeley Software Division (BSD)

- (1975) Thompson visiting Prof. at UC-Berkeley
- A student Bill Joy added new features
  - Vi editor
  - C shell
  - First paging virtual memory management (Unix) BSD 4.2
  - Sendmail, Pascal compiler
  - Later co-founded Sun Microsystems
- Berkely Software Distribution (BSD)
  - Released BSD 3 in 1979
  - Latest one in 1994 BSD 4.4
- Derivatives
  - Darwin, Free BSD, Net BSD, Open BSD

Unix after 1979

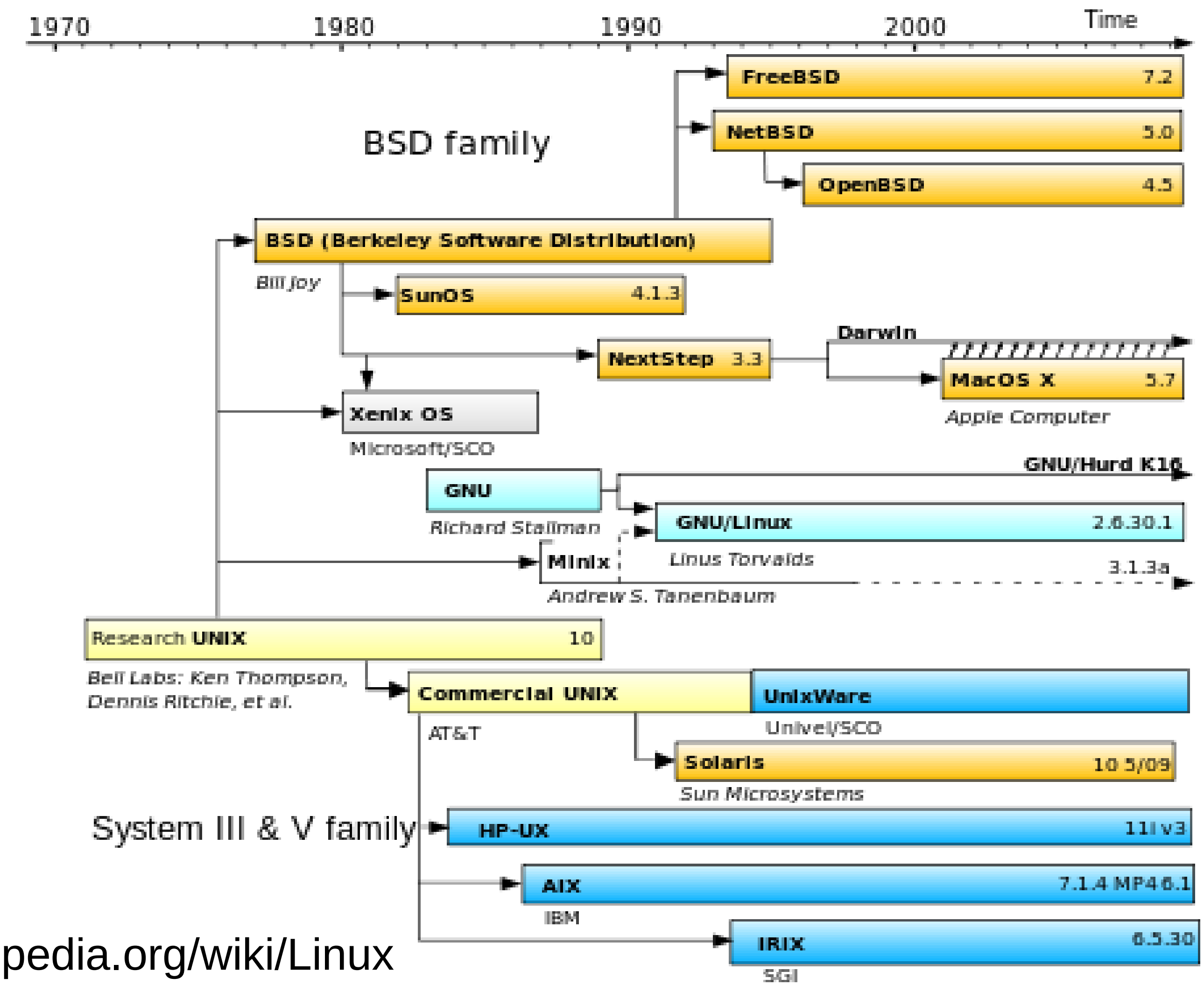
- BSD continued at UC-Berkeley
- Bell Labs System III ➔ Systems V
- POSIX standard (1988)
- Other Software
  - X windows
  - Free Software Foundation
  - GNU Public License
- Minix – (1988)
  - Unix like; MINI-uniX; for education; A. Tannenbaum

Unix Commercial Versions

- In mid 90s many companies supported high-end features and ported them to their own architecture.
  - Digital’s Tru64
  - HP-UX
  - IBM AIX
  - Sequent’s DYNIX
  - SGI’s IRIX
  - Sun Solaris
  - SunOS

Unix Timeline

- .



http://en.wikipedia.org/wiki/Linux

Unix Characteristics/Strengths

- Unix is Simple
  - Has only hundreds of systems calls where as other OSs have thousands.
  - Has clear design goals
- Everything is a file
  - Same system calls open(), read(), write() and close() can work with files, devices, networks sockets.
- Unix kernel and system utilities are written in C.
  - Easy portability to different architectures.
- Fast process creation time.
- Robust inter process communication (IPC).

GNU

- Richard Stallman (1983) Goal a free Unix
  - Known for Free Software movement, GNU, Emacs, gcc
  - Never really released GNU operating system

- Free Software Foundation
  - http://www.fsf.org/

http://en.wikipedia.org/wiki/GNU

Linux -

- Minix – (1988)
  - Unix like; MINI-uniX; for education; A. Tannenbaum
- (1991) Linus Torvalds
- For Intel x86 systems
- Moved to big Iron
- more than 90% of today's 500 fastest supercomputers run some variant of Linux
- Network routers
- Embedded systems
- Android

http://en.wikipedia.org/wiki/Linux

innovate achieve lead

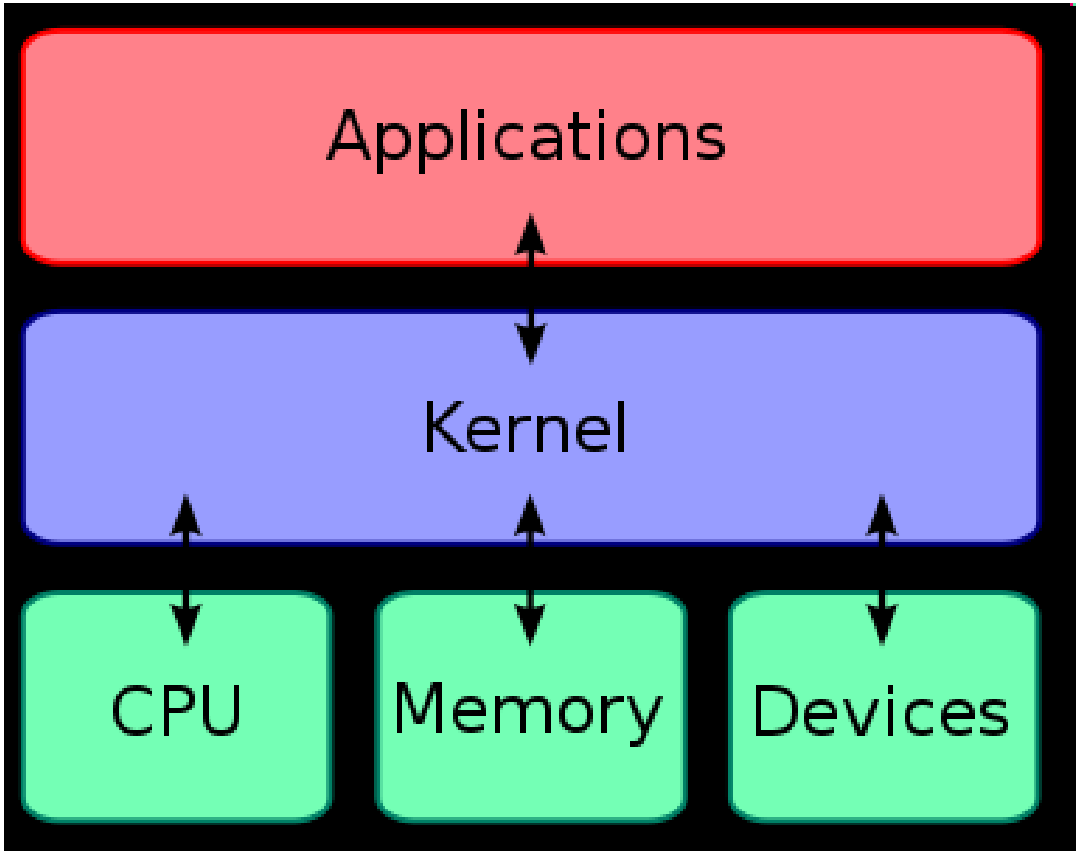
BITS Pilani
Pilani Campus

Overview of Linux OS Concepts

OS

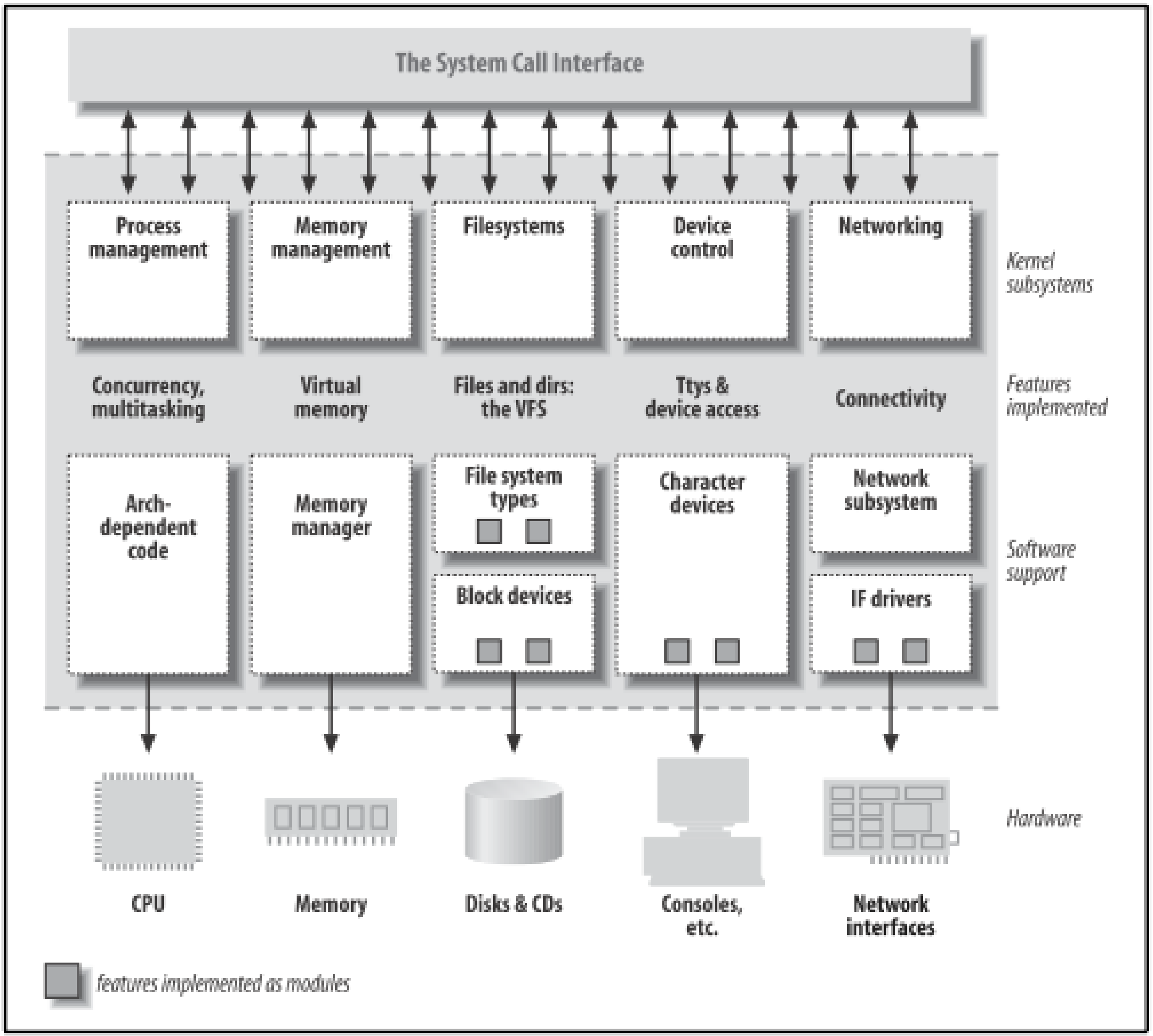
- Kernel
- C library
- Toolchain
- Basic system utilities
- Shell

About kernel



Source: Wikipedia

Kernel



Source: Linux Device Drivers

Kernel

- Central software that manages and allocates computer resources (CPU, RAM, devices etc).
- Kernel greatly simplifies writing programs and using other programs.
  - Processes
  - Virtual address space
  - Virtual CPU

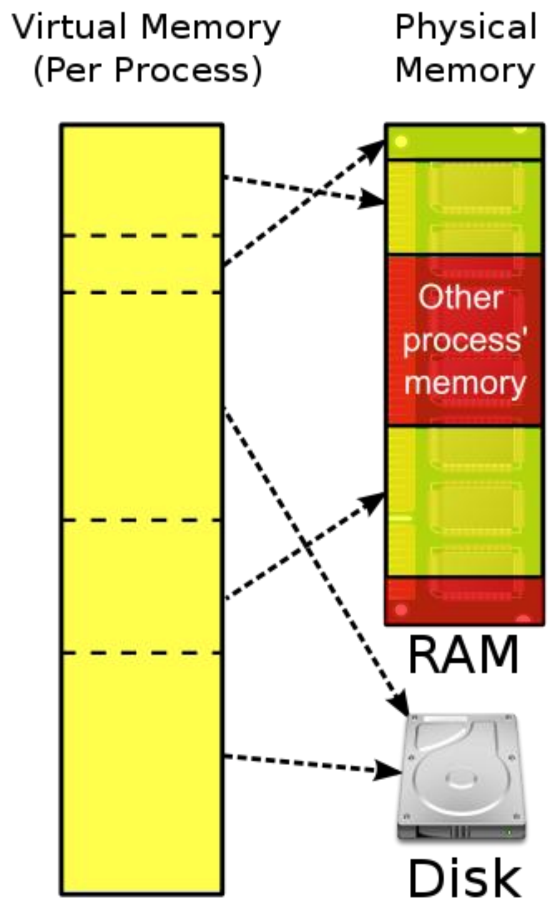
Kernel Tasks

- Process scheduling
- Memory management
- Provision of a file system
- Creation and termination of processes
- Access to devices
- Networking
- Provision of a system call application programming interface (API)

Memory Management

- Software size is enormous and RAM is a limited resource.
- Virtual Memory Management
  - Processes are isolated from one another and from the kernel.
  - Only a part of the process needs to be kept in memory. So multiple processes can be held in main memory simultaneously.
- Virtual Address Space
  - On 32 bit systems, there can be 4 GB address space.
  - Normally 2 GB is meant for Kernel and 2 GB for user process.
  - Current process has all the user address space with it.
  - Programmer flexibility.

Memory Management



Source: wikipeadia

Kernel Tasks

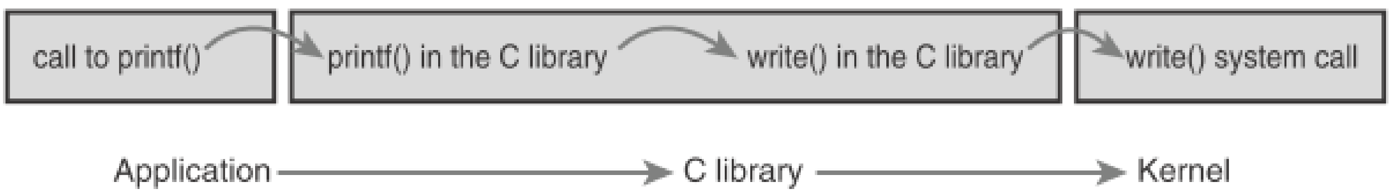
- Process Scheduling
  - Preemptive Multitasking
  - Multiple processes can simultaneously reside in memory
  - Preemptive means kernel decides which process and how long it gets CPU not the process itself.
- File System
  - File creation, retrieval, deletion etc.
  - More later
- Networking
  - TCP/IP stack
  - Sending/receiving messages on behalf of user processes.
  - More later

Kernel Tasks: Process Management

- Kernel can load new program into memory, provide it with the resources.
  - fork()
  - execve()
- Instance of a running program is termed as a process.
- Processes need to invoke system calls to access resources.
- Kernel executes the system calls on behalf of the process in kernel space.
- **Access to devices**:
  - Kernel provides an interface that standardizes and simplifies access to devices.
  - Keyboard, display etc.

Kernel Tasks: System Calls

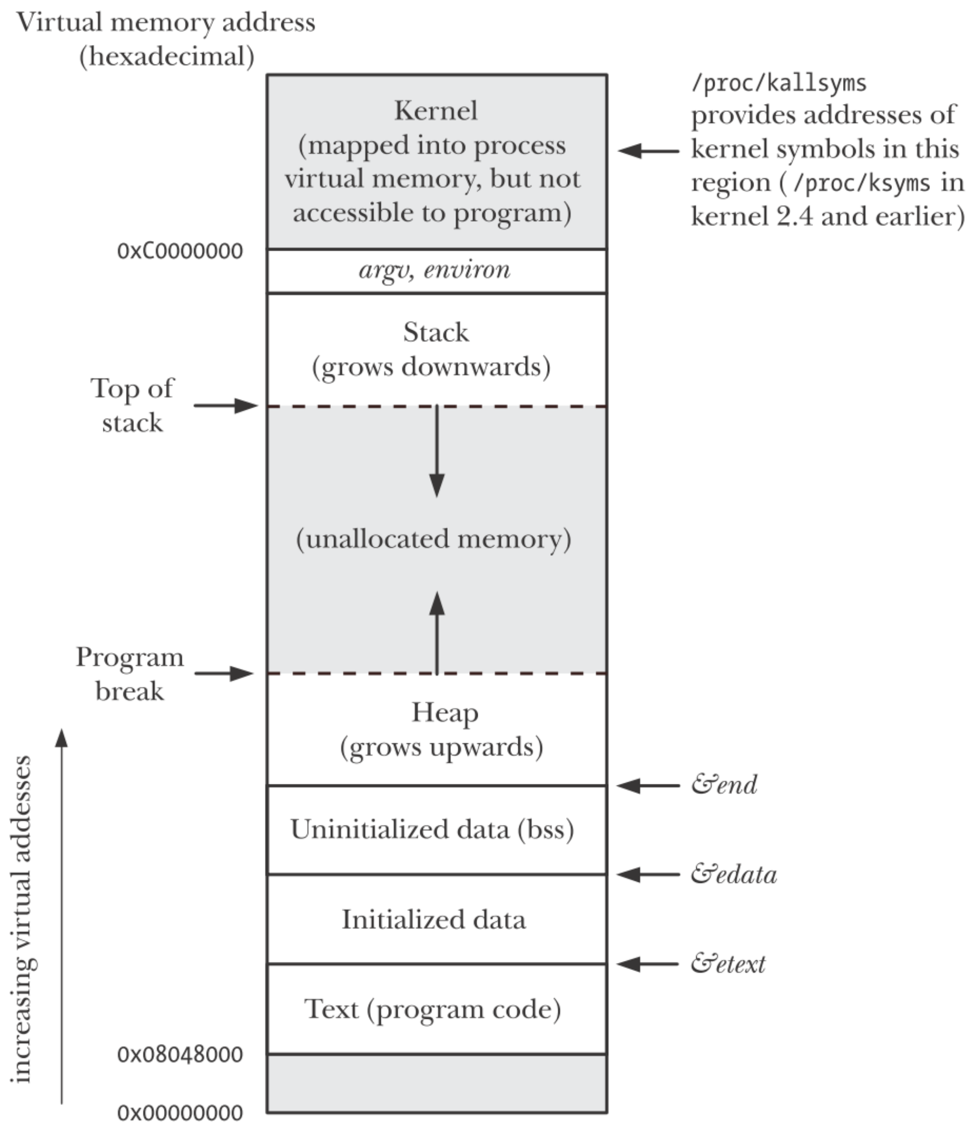
- Kernel entry points are known as system calls.
- System calls are software interrupts.



Kernel Mode vs User Mode

- Processor architecture allows the CPU to operate in at least two different modes:
  - User mode
  - Kernel mode
- Similarly virtual memory is marked as user space and kernel space.
  - When running user mode, the CPU can access only user space.
  - When running in kernel mode, the CPU can access both user and kernel space.
- By this arrangement, user processes can’t access instructions and data structures of kernel.

Kernel Space vs User Space



innovate achieve lead

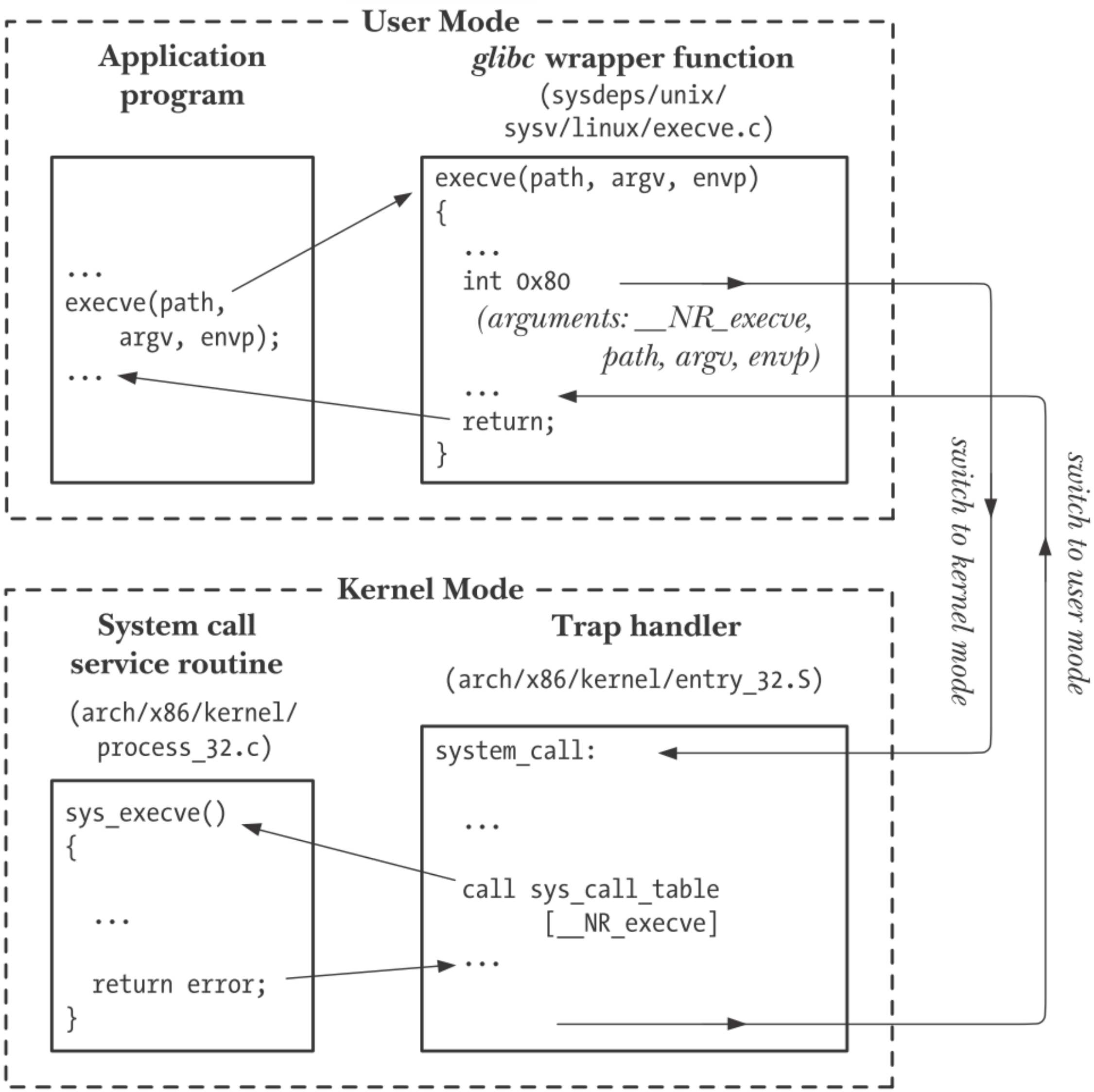
BITS Pilani
Pilani Campus

Sys Calls & C Library

Kernel Tasks: System Calls

- A system call changes the processor from user mode to kernel mode.
- The set of sys calls are fixed. Each system call is identified by a unique number.
- System calls arguments are copied from user space to kernel space (registers).
- Wrapper function executes a trap instruction (int 0x80) causing the CPU to enter kernel mode.
- Example: execve()
  - System call number is 11.
  - Sys_call_table contains the address of the routine sys_execve().
- Not all C library calls invoke system calls. E.g.: <string.h>

Kernel Tasks: System Calls



Sys Calls & C Library

- System call is a controlled entry point into the kernel.
- For every sys call there is a wrapper function in C library.
  - All library functions are not sys calls.
  - We use wrapper functions in programs
  - fork(), execve() …
- Wrapper function copies the arguments into specific registers. Also copies sys call number into a specific CPU register.
- Wrapper functions executes trap instruction int 0x80.
  - Causes CPU to switch from user mode to kernel mode.
- Kernel executes *system_call* routine at *arch/x86/kernel/entry.S*

Sys Calls & C Library

- On i386, the parameters of a system call are transported via registers.
  - The system call number goes into %eax
  - the first parameter in %ebx
  - the second in %ecx
  - the third in %edx
  - the fourth in %esi
  - the fifth in %edi
  - the sixth in %ebp.

Sys Calls & C Library

- *system_call()* routine:
  - Saves register values onto the stack
  - Checks the validity of the system call number.
  - Invokes the corresponding service routine.
  - 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 user process stack.
  - Returns to the wrapper function, simultaneously returning the process to enter user mode.
- Wrapper function checks the return value and if it is an error it sets the *errno* variable.
- System calls incur an appreciable overhead.
  - Calling a C library wrapper function is synonymous with invoking the corresponding system call routine.

C Library

- Many library functions do not make use of system calls.
- Often library functions provide a more caller-friendly interface than the underlying sys call.
  - fopen() uses open()
  - printf uses write()
  - malloc() uses brk()
- GNU C library (*glibc*)
  - libc.so.6
  - Where is it stored? List dynamic dependencies
    - `ldd a.out`
  - Finding the version
    - ./libc.so.6

BITS Pilani
Pilani Campus

Handling Errors

Errors from System Calls

innovate achieve lead

- A service routine in kernel returns a negative number in case of error. The negative number corresponds to standard error codes.

/usr/include/asm-generic/errno-base.h

```
#ifndef _ASM_GENERIC_ERRNO_BASE_H
#define _ASM_GENERIC_ERRNO_BASE_H

#define EPERM        1  /* Operation not permitted */
#define ENOENT       2  /* No such file or directory */
#define ESRCH        3  /* No such process */
#define EINTR        4  /* Interrupted system call */
#define EIO      5  /* I/O error */
#define ENXIO        6  /* No such device or address */
#define E2BIG        7  /* Argument list too long */
#define ENOEXEC      8  /* Exec format error */
#define EBADF        9  /* Bad file number */
#define ECHILD      10  /* No child processes */
#define EAGAIN      11  /* Try again */
#define ENOMEM      12  /* Out of memory */
#define EACCES      13  /* Permission denied */
#define EFAULT      14  /* Bad address */
#define ENOTBLK     15  /* Block device required */
```

Errors from System Calls

- Incase of error, wrapper function sores the positive value of the error code into errno variable and returns -1.

```c
cnt = read(fd, buf, numbytes);
if (cnt == -1) {
    if (errno == EINTR)
        fprintf(stderr, "read was interrupted by a signal\n");
    else {
        /* Some other error occurred */
    }
}

```

- perror() and strerror() can be used to print the error.

```c
fd = open(pathname, flags, mode);
if (fd == -1) {
    perror("open");
    exit(EXIT_FAILURE);
}

```

errno variable

- It is present one per each process.
- It is set in wrapper function after sys call returns error.
  - Every time it is over written.
  - So only after a sys call returns -1, we refer to errno.

Handling Errors from Library Functions

- Some library functions return error information exactly like system calls.
  - Return -1
  - errno is set
- Some return value other than -1 but set errno
  - fopen
- Others do not use errno at all
  - gethostbyname

Tracing System Calls

- *strace* command allows us to trace the system calls made by a program.

```
haribabuk@haribabuk-VirtualBox ~ $ strace ./a.out
execve("./a.out", ["./a.out"], [/* 48 vars */]) = 0
brk(0)                                  = 0x10e9000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd26b5e2000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=128950, ...}) = 0
mmap(NULL, 128950, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd26b5c2000
close(3)                                = 0
```

- Each system call is displayed with input and output arguments
- Arguments are printed in symbolic form.
- *ltrace* is used for tracing libray calls.

```
haribabuk@haribabuk-VirtualBox ~ $ ltrace ./a.out
__libc_start_main(0x4005f4, 1, 0x7fffd10e0ba8, 0x400700, 0x400790 <unfinished ...>
socket(2, 3, 1)                                                         = -1
recvfrom(0xffffffff, 0x7fffd10e04d0, 1500, 0, 0)                        = -1
+++ exited (status 0) +++
```

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You