CSE 3320

Chapter 2: Operating System Concepts, Components and Architectures

Trevor Bakker

The University of Texas at Arlington

Agenda

- What does the OS do?
- Managed resources
- Classes of operating systems
- Architectural approaches
- OS philosophy

What does the OS do?

- When I sit down at my desk with my morning coffee and open a file to write code what does the OS do?
  - Coffee is <u>not</u> a service of the OS. I will fund your kickstarter if you figure out how to make this happen
- We'll take a high level view in the next few slides

Edit the file with vi

- I type `vi proc.c` and press return
- What occurs?
    - Each keystroke causes the keyboard controller (the USB controller in the case of my laptop) stores the keystroke information in a buffer and issues an interrupt to the CPU
    - The interrupt causes the CPU to stop its processing and call the OS’s interrupt service routine

Pardon the interruption

- The keyboard interrupt service routine is part of the interrupt handling and device handling of the OS
- The interrupt and ISR are repeated for each character typed

```
/* This function services keyboard interrupts. It reads the relevant
 * information from the keyboard and then scheduales the bottom half
 * to run when the kernel considers it safe.
 */
void irq_handler(int irq, void *dev_id, struct pt_regs *regs)
{
   /* This variables are static because they need to be
    * accessible (through pointers) to the bottom half routine.
    */
   static unsigned char scancode;
   static struct tq_struct task = {NULL, 0, got_char, &scancode};
   unsigned char status;

   /* Read keyboard status */
   status = inb(0x64);
   scancode = inb(0x60);

   /* Scheduale bottom half to run */
#if LINUX_VERSION_CODE > KERNEL_VERSION(2,2,0)
   queue_task(&task, &tq_immediate);
#else
   queue_task_irq(&task, &tq_immediate);
#endif
   mark_bh(IMMEDIATE_BH);
}
```

Process Management

- When I type `vi proc.c` and press enter the OS creates a process
  - A process is a program in execution
  - Processes are also called task or job
    - In particular, you will see the term task used with Linux.

Memory Management

- Before the process can start the binary executable must be brought into main memory .

    1. Memory to hold the programs executable code must be allocated.

    2. Memory for the program's data and temporary storage.

- Multiple processes can be resident in memory at the same time.

```
REGION TYPE                      VIRTUAL RESIDENT
===========                      ======= ========
Kernel Alloc Once                     4K       4K
MALLOC                             36.2M     412K
MALLOC (admin)                       24K       8K
MALLOC_LARGE (reserved)             128K       0K
STACK GUARD                        56.0M       0K
Stack                              8192K      20K
VM_ALLOCATE                           8K       8K
__DATA                              784K     536K
__LINKEDIT                         66.2M    10.9M
__TEXT                             7556K    5544K
shared memory                         4K       4K
===========                      ======= ========
TOTAL                             174.6M    17.3M
TOTAL, minus reserved VM space    174.5M    17.3M
```

What about the GUI?

- Is the windowing system a part of the operating system?
  - Some argue that it is a systems program and not part of the operating system.
  - Others argue that the windowing system is part of the operating system
  - Not everyone agrees on what parts constitute an operating system
    - In the court case United States v. Microsoft Corporation (2001), Microsoft argued that even the web browser Internet Explorer was a part of the Windows operating system
-

File Management

- When saving the newly created file the file management portion of the OS executes several tasks:
  - Make sure no file name duplication
  - Find free disk space for the file
  - Create the file entry
  - etc

```c
if ( rip == NIL_INODE && err_code == ENOENT) {
        /* Last path component does not exist.  Make new directory entry. */
        if ( (rip = alloc_inode((ldirp)->i_dev, bits)) == NIL_INODE) {
                /* Can't creat new inode: out of inodes. */
                return(NIL_INODE);
        }

        /* Force inode to the disk before making directory entry to make
         * the system more robust in the face of a crash: an inode with
         * no directory entry is much better than the opposite.
         */
        rip->i_nlinks++;
        rip->i_zone[0] = z0;            /* major/minor device numbers */
        rw_inode(rip, WRITING);         /* force inode to disk now */

        /* New inode acquired.  Try to make directory entry. */
        if((r=search_dir(ldirp, string, &rip->i_num, ENTER, IGN_PERM)) != OK) {
                rip->i_nlinks--;        /* pity, have to free disk inode */
                rip->i_dirt = DIRTY;    /* dirty inodes are written out */
                put_inode(rip); /* this call frees the inode */
                err_code = r;
                return(NIL_INODE);
        }

} else if (err_code == EENTERMOUNT || err_code == ELEAVEMOUNT) {
        r = EEXIST;
} else {
        /* Either last component exists, or there is some problem. */
        if (rip != NIL_INODE)
                r = EEXIST;
        else
                r = err_code;
}
```

Resource Management

- Even for a simple task like opening a file there is a lot going on behind the scenes.
- For the next 15 weeks we will delve deeper into each system.
- But first, let's discuss the resources managed by the operating system.

Types of Resources

- CPU - The OS needs to schedule which processes to run
  - Older single-process systems had it simple
    - Start the process and give it control of the CPU
    - Also had to setup some memory protection registers and set user execution mode

Types of Resources

- Multitasking systems are more complex.
  - Multiple processes may be resident in memory
  - Multiple CPUs
  - Ready Queue - scheduling queue containing all processes ready to run
    - May have separate queues for each priority level.
  - Time Quantum - maximum period of time a process is given to run on the CPU before having to yield
  - Context Switching - switching control of the CPU to another process

Types of Resources

- Main Memory
    - Executable code is on disk and needs to be transferred to main memory to execute.
    - May only partially load large programs
    - What if memory is already full with other processes?
        - Swap it out
- Secondary Storag
- I/O Devices

Types of Resources

- Secondary Storage
  - Hard disks
  - OS must schedule requests for data or code not resident in memory
  - Multiple requests to read or write may be made at a time
    - OS must prioritize using various scheduling algorithms (Chapter 14)

Types of Resources

- I/O Devices
  - O/S includes device drivers that control acces to the devices
  - I/O Management (Chapter 12)

Types of Resources

- File systems - O/S module that provides a high level interface to allow the user and programs to create, delete, modify, and apply other operations to various types of files
- Chapter 12 (This stuff is FUN)

Types of Resources

- User interfaces
  - Handle user interactions
- Network access
  - Allows users and programs on one computer to access other services and devices on a network.
- Security
  - Access authorizations

Interrupts

- Mechanism used by the OS to signal the system that a high-priority event has occurred that requires immediate attention.
    - I/O drives a lot of interrupts. Mouse movements, disk reads, etc
- The controller causing the interrupt places the interrupt number in an interrupt register. The OS must then take action

Interrupt Vector

- Normal technique for handling interrupts is a data structure called the interrupt vector.
- One entry for each interrupt.
- Each entry contains the address for the interrupt service routine.
- Some small hardware devices don’t provide an interrupt system and instead uses an event loop. This is known as a status-driven system.

System Calls

- How do our programs utilize the resources controlled by the OS or communicate with other process?
- Because user mode software can not access hardware devices directly, they must notify the operating system in order to complete system tasks. This includes displaying text, obtaining input from user, printing a document, etc.

System Calls

- System calls - Function call provided by the operating system
    - Different than a normal function call

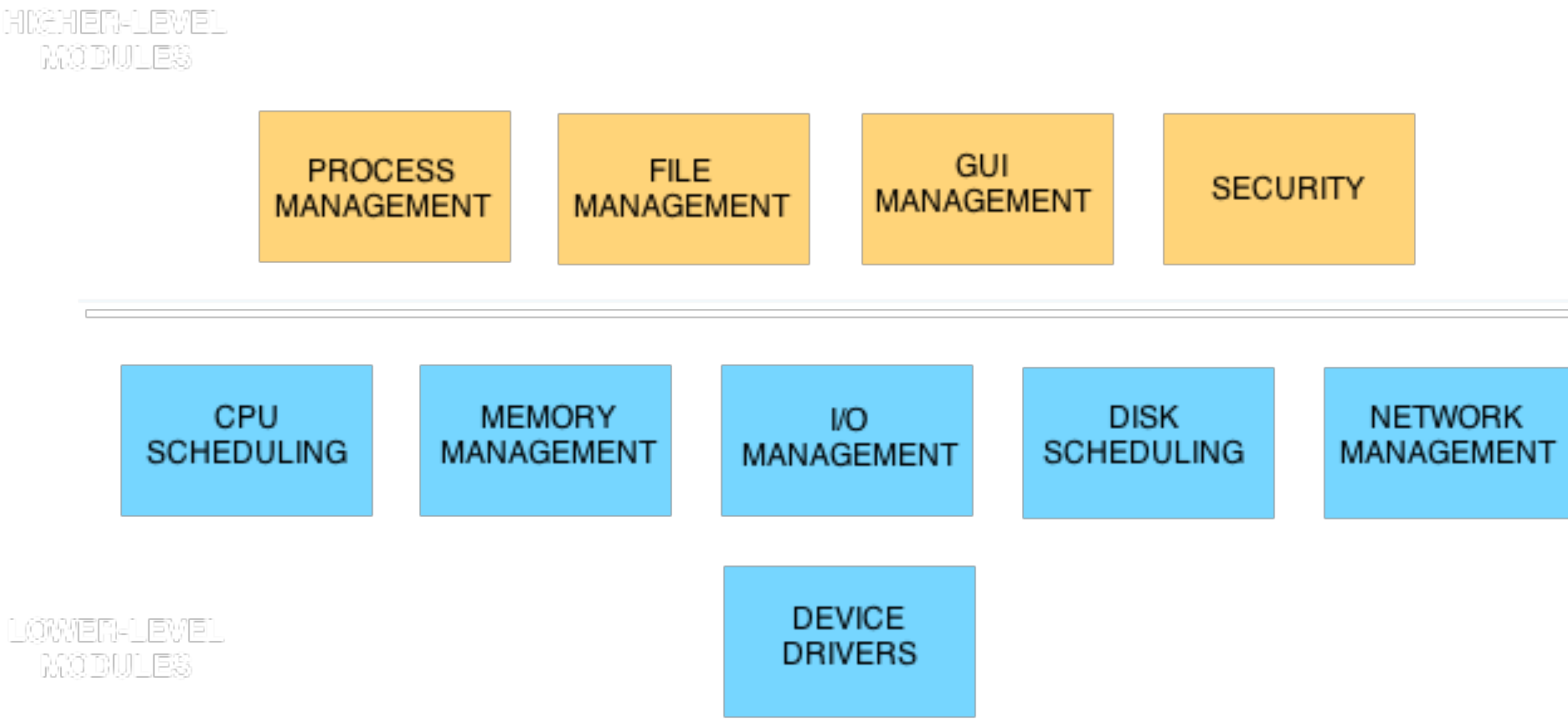
System Calls

- Instead of directly calling a section of code the system call instruction issues an interrupt.

- By not allowing the application to execute code freely the operating system can verify that the application has appropriate privileges to call the function.

    - glibc library

Major Modules of OS



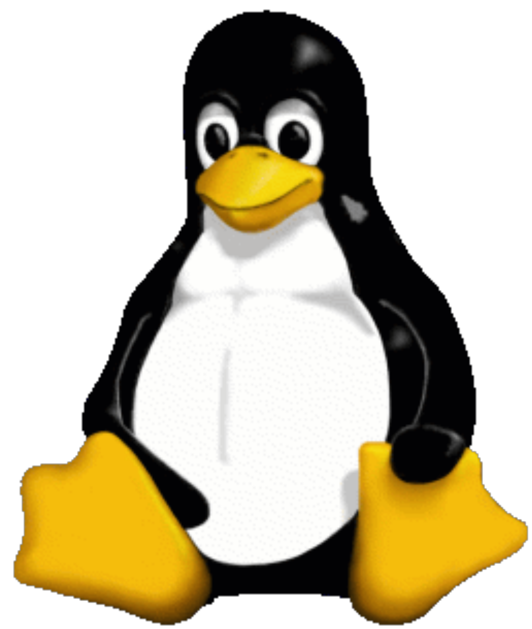
Not how the modules interact

Monolithic Kernel

- The first operating systems were written as a single program.
- Core functionality such as memory allocation and scheduling, as well as services such as device drivers and the file system exist in the same space.
- Problems with bloat. OS occupies more and more memory
- A bug in a device driver can bring down the entire system

Linux

- Linux is a monolithic kernel
- Monolithic does not mean it isn't modular
  - Linux supports dynamic loadable modules
- Chapter 6 we will discuss Linux

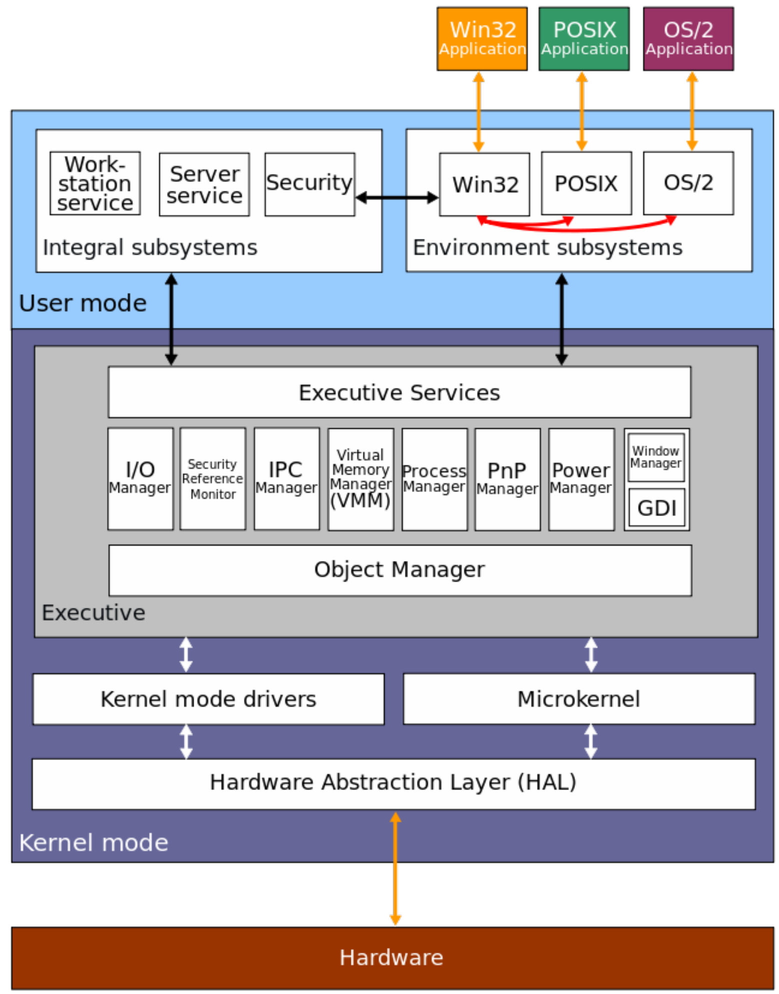


Tux image © 1995 by lewing@isc.tamu.edu

Layered Architecture

- Modules at one level call functions provided by modules at the same or lower level.

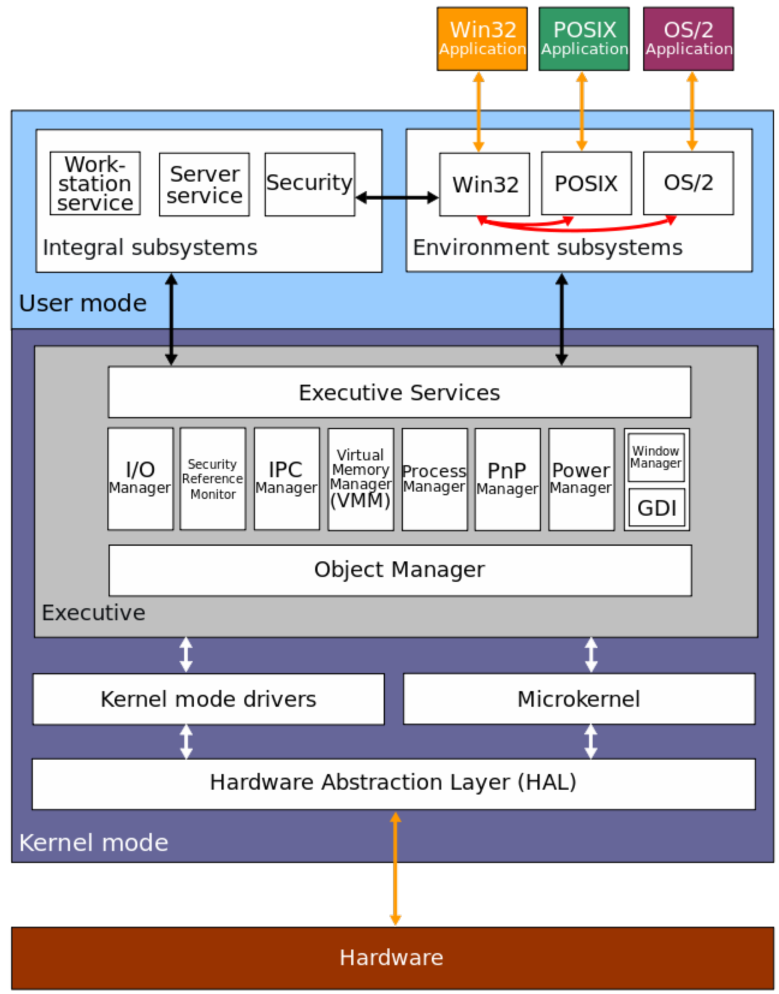
- Example: Windows NT Architecture (Windows 2000, Windows XP, Vista, Windows 7, Windows 8)



Windows NT Architecture © 2005 Grm Wnr licensed under GNU Free Documentation License.

Layered Architecture

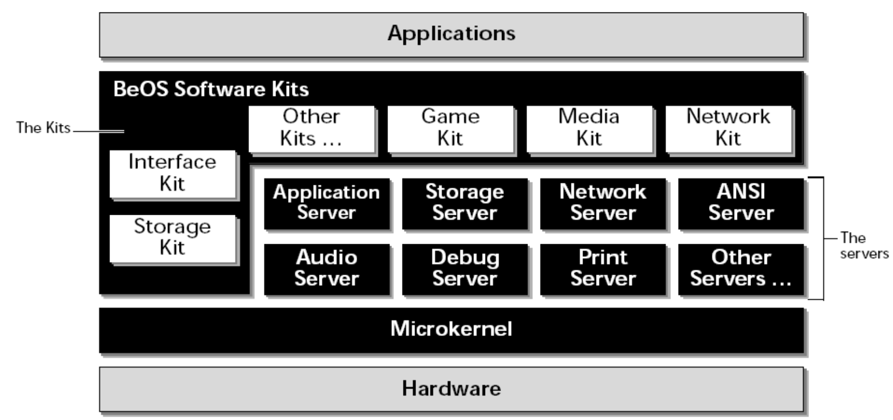
- Each layer provides a more abstract view than the layer below
- Usually only 2 or 3 layers are used because it's difficult to separate complex functionality into multiple clean layers



Windows NT Architecture © 2005 Grm Wnr licensed under GNU Free Documentation License.

Object Oriented Architecture

- Each O/S module is implemented as an object and provides services
- Any object can invoke the services of another
- Example: BeOS



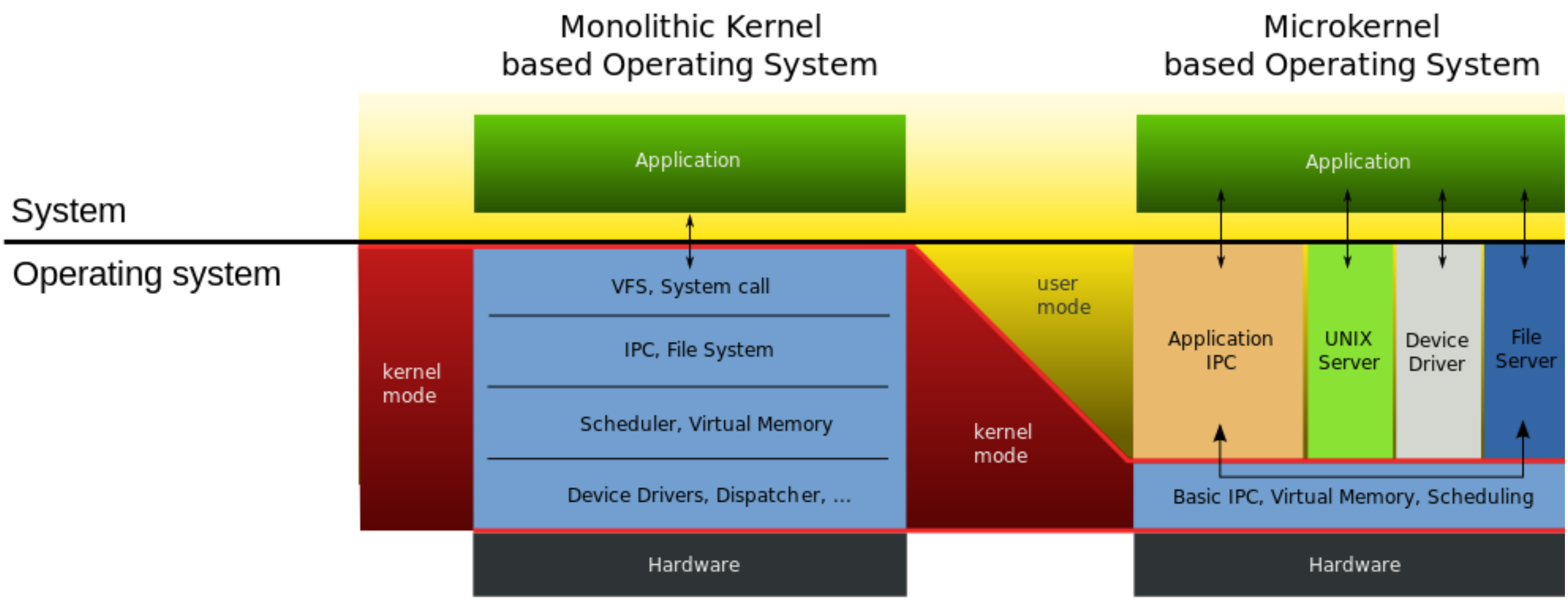
Microkernel

- Only basic functionality is included in the kernel
    - What is basic? Only code that must run in supervisor mode because it must use privileged resources such as protected instructions
- Everything else runs in user space.

Microkernel

- Theoretically more robust since limiting the amount of code that runs in protected mode limits the number of catastrophic crashes
- Easier to inspect for flaws since a smaller portion of code exists
- May run slower since there are more interrupts from user space to kernel
- Example: Minix

Micro v. Monolithic



"OS-structure2" by Golftheman - http://en.wikipedia.org/wiki/Image:OS-structure.svg. Licensed under Public domain via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:OS-structure2.svg#mediaviewer/File:OS-structure2.svg

Min/Max

- What goes in the kernel?
- Minimalist - Only things that absolutely must go into the kernel go in. Everything else is a library or a user space program.
- Users can pick what they want to run
- Easier to debug and design
- Claim it's "cleaner" and more "elegant"

Min / Max

- Maximalist - All commonly used services go into the kernel.
- Creates consistency across applications
- Common features used by almost every program, for example mouse, screen drawing, and menus, should be placed in the most efficient place.
- Claim security must be done in the kernel

Min / Max Reality

- Few, if any, operating systems are minimalist or maximalist.

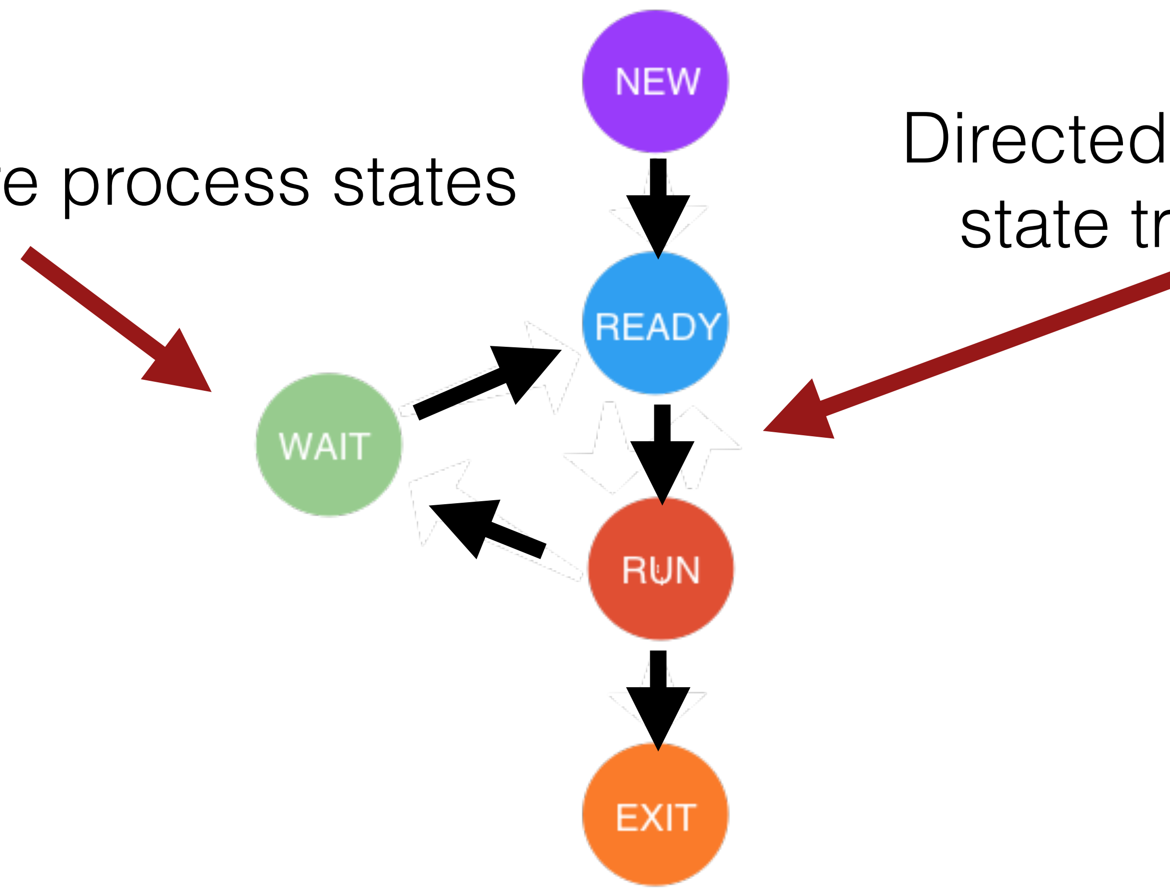
Programs v. Processes

- Program - a sequence of instructions written to perform a specified task.
- Process - an instance of a program in execution.
- A computer program is a passive collection of instructions; a process is the actual execution of those instructions.

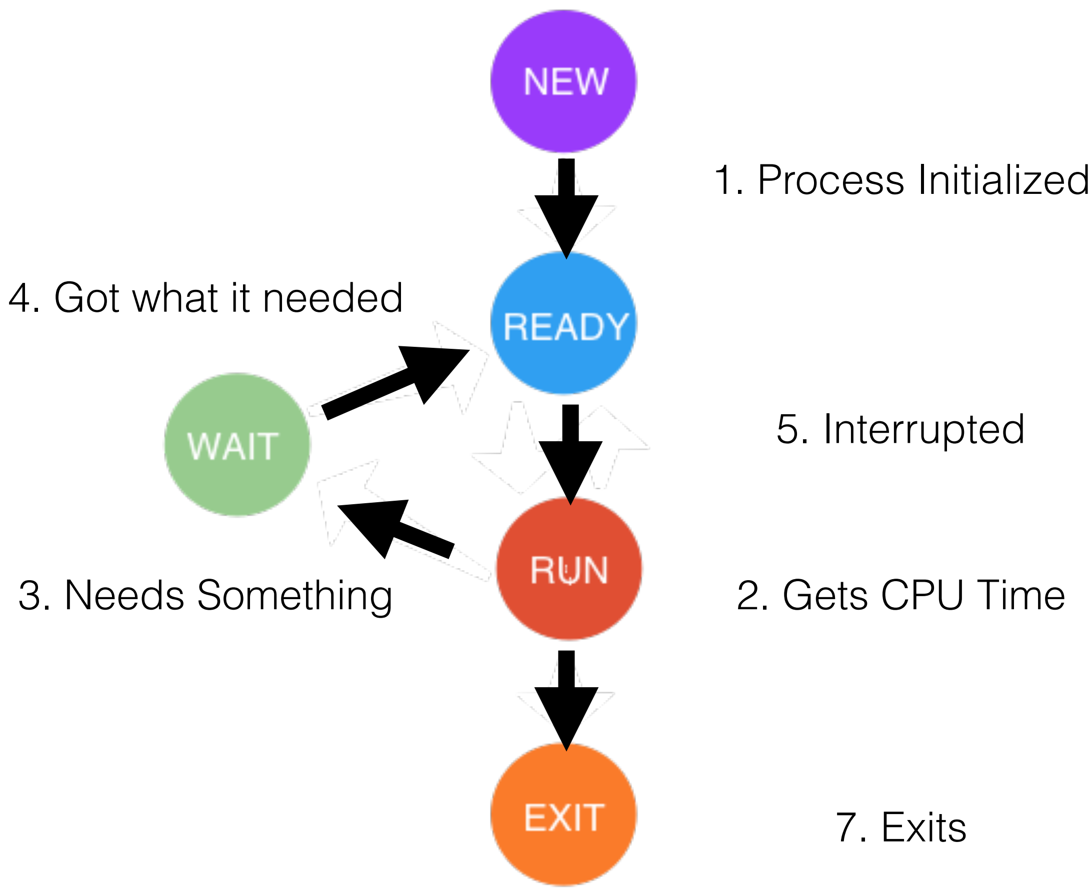
Process State Diagram

Nodes are process states

Directed edges are state transitions



Process Lifecycle



Types of Processes

- User Processes - Applications executing on behalf or a user.
  - Example: World of Warcraft
- System Program Processes - Programs that perform a common system service instead of a specific end-user service.
  - Example: gcc
- OS Processes - Also known as daemons. These are processes that execute OS functions.
  - Example: network services

Process Execution Modes

- Privileged - OS kernel processes which can execute all types of hardware operations and access all memory
- User Mode - Can not execute low-level I/O. Memory protection keeps these processes from trashing memory owned by the OS or other processes.

Kernel Space v. User Space

- A process is executing either in user space, or in kernel space. Depending on which privileges, address space a process is executing in, we say that it is either in user space, or kernel space.

- When executing in user space, a process has normal privileges and can and can't do certain things. When executing in kernel space, a process has every privilege, and can do anything.

- Processes switch between user space and kernel space using system calls.

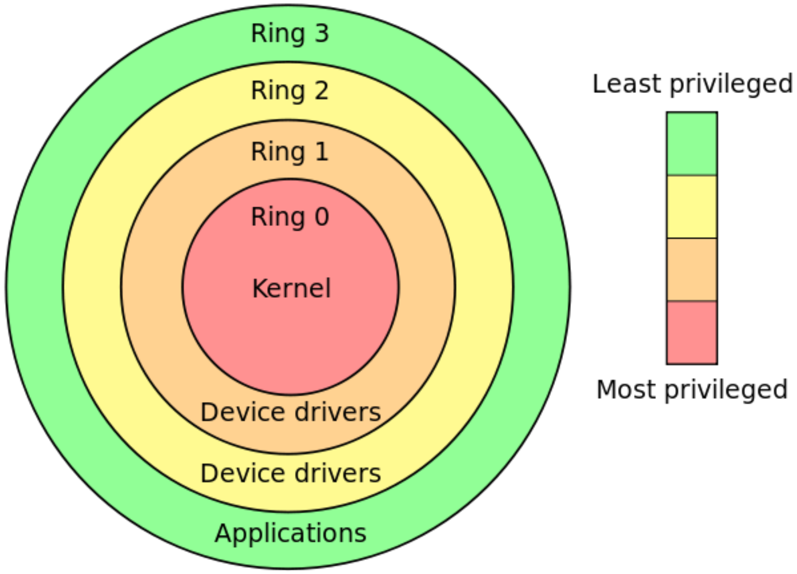
Kernel Space v. User Space

- These two modes aren't just labels; they're enforced by the CPU hardware.
- If code executing in User mode attempts to do something outside its purview such as accessing a privileged CPU instruction or modifying memory that it has no access to:
  - Trappable exception is thrown. Instead of your entire system crashing, only that particular application crashes.

x86 Protection Rings

- Four privilege levels or rings, numbered from 0 to 3, with ring 0 being the most privileged and 3 being the least.

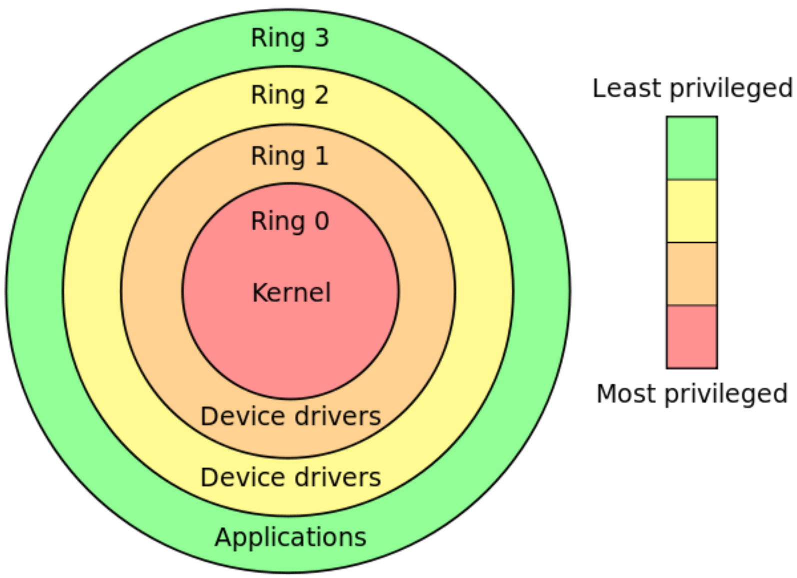
- Rings 1 and 2 aren't used in practice.



x86 Protection Rings

- Programs that run in Ring 0 can do anything with the system.

- Code that runs in Ring 3 should be able to fail at any time without impact to the rest of the computer system.



CPU Rings and Privilege

- CPU privilege level has nothing to do with operating system users.
- Whether you’re root, Administrator, guest, or a regular user, it does not matter.
- All user code runs in ring 3 and all kernel code runs in ring 0, regardless of the OS user on whose behalf the code operates.

CPU Rings and Privilege

- Due to restricted access to memory and I/O ports, user mode can do almost nothing to the outside world without calling on the kernel.
  - It can't open files, send network packets, print to the screen, or allocate memory.
  - User processes run in a severely limited sandbox set up by ring zero.

CPU Rings and Privilege

- That's why it's impossible, by design, for a process to leak memory beyond its existence or leave open files after it exits. All of the data structures that control such things – memory, open files, etc – cannot be touched directly by user code; once a process finishes, the sandbox is torn down by the kernel.

How?

- The switch consists of three steps:

    1. Change the processor to kernel mode;

    2. Save and reload the MMU to switch to the kernel address space;

    3. Save the program counter and reload it with the kernel entry point.

How, in more details

- Accomplished via gate descriptors and via the sysenter instruction.

- A gate descriptor is a segment descriptor and comes in four sub-types:

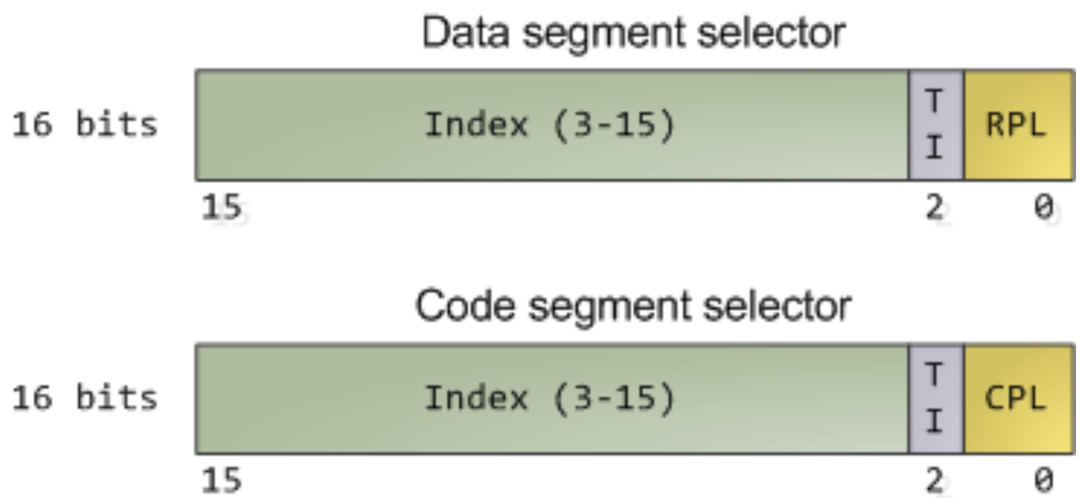
  1. call-gate descriptor.

  2. interrupt-gate descriptor.

  3. trap-gate descriptor.

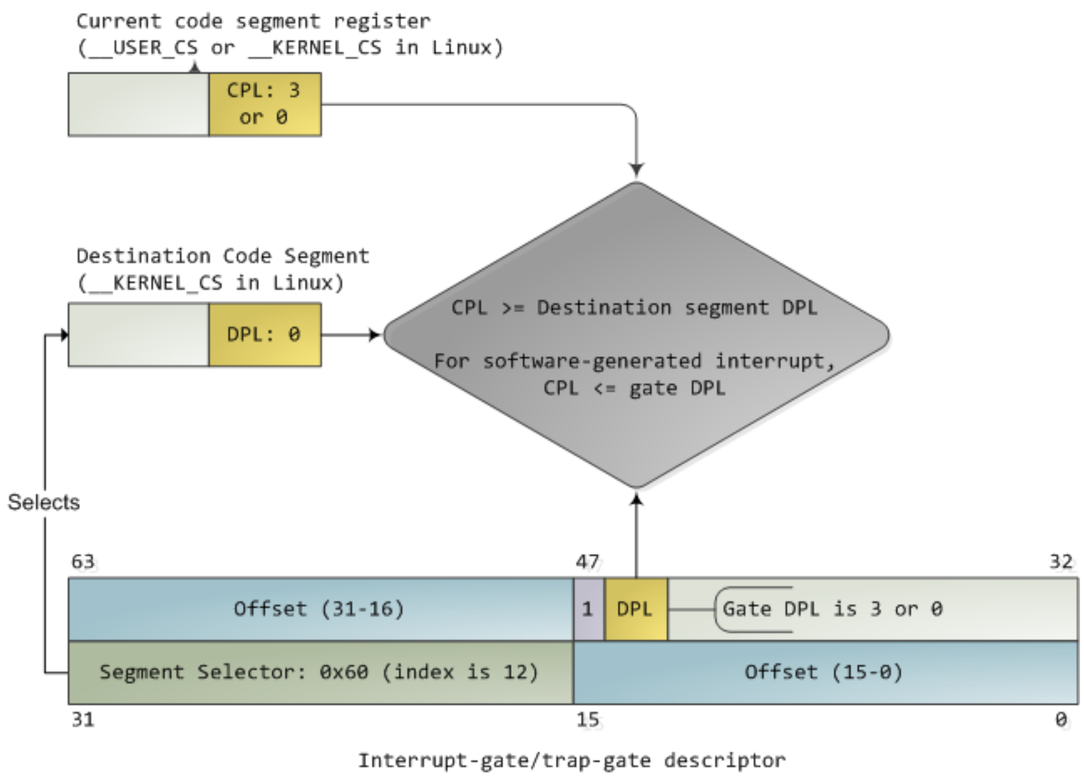
  4. task-gate descriptor.

How does it know?



- The CPU keeps track of the current and requested privilege levels via the data segment selector and the code segment selector.

How, in more details



- DPL - Desired privilege level
- CPL - Current privilege level

CPU States

- Running a user process (ring 3, your code)
- Running a syscall (ring 0, kernel code)
- Running a interrupt handler (ring 0, kernel code)
- Running a kernel thread (ring 0, kernel code)

x86 Privileged Instructions

Privileged Level (Ring 0) Instructions

Instruction	Description
LGDT	Loads an address of a GDT into GDTR
LLDT	Loads an address of a LDT into LDTR
LTR	Loads a Task Register into TR
MOV *Control Register*	Copy data and store in Control Registers
LMSW	Load a new Machine Status WORD
CLTS	Clear Task Switch Flag in Control Register CR0
MOV *Debug Register*	Copy data and store in debug registers
INVD	Invalidate Cache without writeback
INVLPG	Invalidate TLB Entry
WBINVD	Invalidate Cache with writeback
HLT	Halt Processor
RDMSR	Read Model Specific Registers (MSR)
WRMSR	Write Model Specific Registers (MSR)
RDPMC	Read Performance Monitoring Counter
RDTSC	Read time Stamp Counter

Returning back to user space

- Finally, when it’s time to return to ring 3, the kernel issues an iret or sysexit instruction to return from interrupts and system calls, respectively, thus leaving ring 0 and resuming execution of user code with a CPL of 3.

Cost of Promotion

- Promoting from user to kernel space is expensive.
  - 1000 - 1500 cycles.
- This mechanism consists of three steps:
  1. Change the processor to kernel mode.
  2. Save and reload the MMU to switch to the kernel address space.
  3. Save the program counter and reload it with the kernel entry point.

Cost is worth it

- The CPU's strict segregation of code between User and Kernel mode is completely transparent to users, but it is the difference between a computer that crashes all the time (applications) and a computer that crashes catastrophically all the time (entire OS, think blue screen of death).

How does the OS track a process?

- Each process has a unique process identifier, or PID
  - The POSIX standard guarantees a PID as a signed integral datatype.
  - The datatype is an opaque type called pid_t

Process Control Block

- The kernel maintains a data structure to keep track of all the process information called the process control block or (PCB).

- The PCB also includes pointers to other data structures describing resources used by the process such as files (open files table) and memory (page tables).

- Maintains the state of the process

  - Over 170+ fields

Some Process Control Block Fields

Memory

Open streams/files

Devices, including abstract ones like windows

Links to condition handlers (signals)

Processor registers (single thread)

Process identification

Process state

Priority

Owner

Which processor

Links to other processes (parent, children)

Process group

Resource limits/usage

Access rights

Process Control Block

- Every task also needs its own stack
- So every task, in addition to having its own code and data, will also have a stack-area that is located in user-space, plus another stack-area that is located in kernel-space
- Each task also has a process-descriptor which is accessible only in kernel-space

Process Control Block

- Different information is required at different times
- UNIX for example has two separate places in memory with the process control block and process stack. One of them is in the kernel the other is in user space.
- Why? User land data is only required when the process is running.

Why a kernel stack?

- Kernels can't trust addresses provided by user
- Address may point to kernel memory that is not accessible to user processes
- Address may not be mapped
- Memory region may be swapped out from physical RAM
- Leftover data from kernel ops could be read by process
    - Kernel-level heartbleed bug

Process Tables

- The OS holds the process control blocks in the process table
- Usually implemented as an array of pointers to process control block structures
  - Linux calls the PCB task_struct

Linux PCB Data Structure

- /include/linux/sched.h

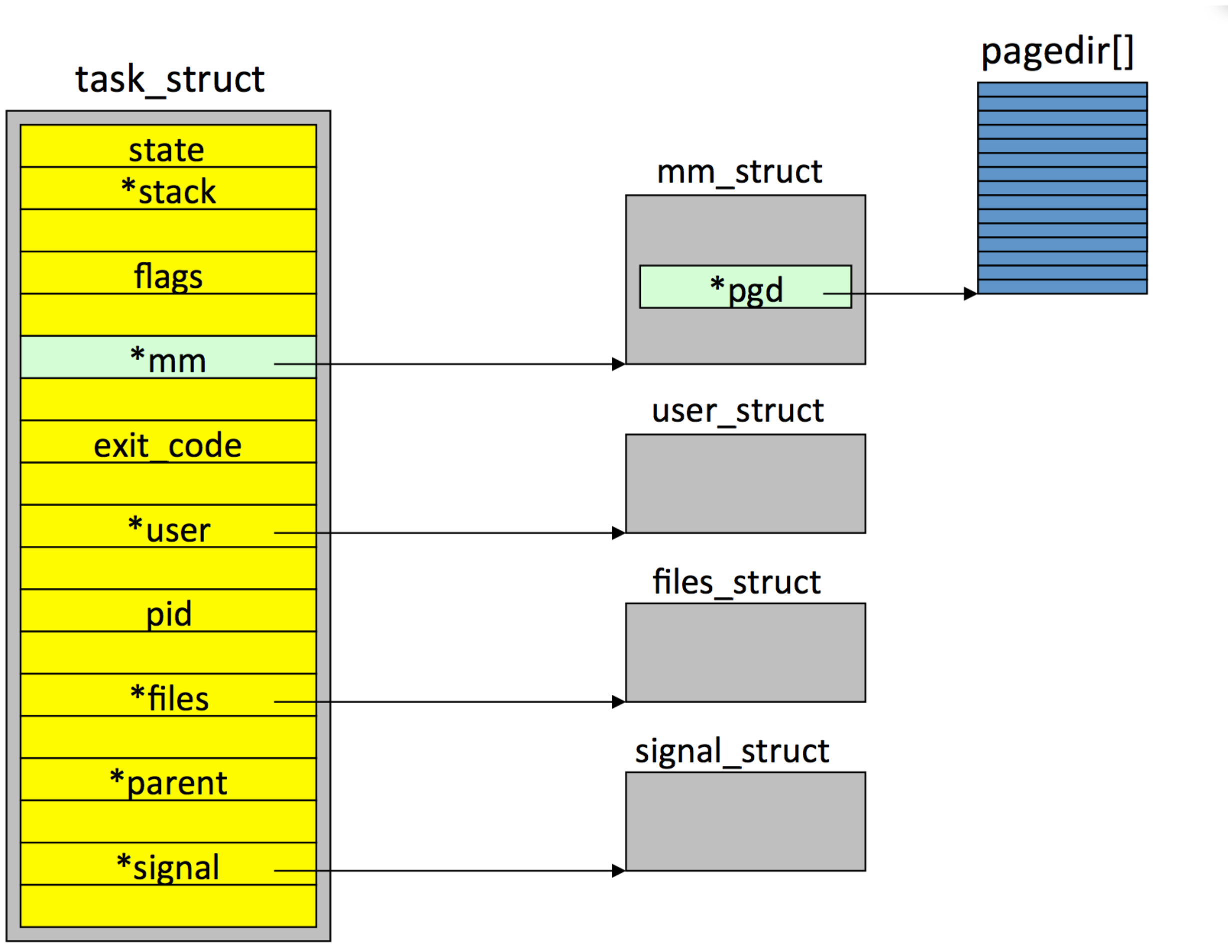
```
struct task_struct {
        volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process flags, defined below */
        unsigned int ptrace;

        int lock_depth;         /* BKL lock depth */

#ifdef CONFIG_SMP
#ifdef __ARCH_WANT_UNLOCKED_CTXSW
        int oncpu;
#endif
#endif

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;

#ifdef CONFIG_PREEMPT_NOTIFIERS
        /* list of struct preempt_notifier: */
        struct hlist_head preempt_notifiers;
#endif

        /*
         * fpu_counter contains the number of consecutive context switches
         * that the FPU is used. If this is over a threshold, the lazy fpu
         * saving becomes unlazy to save the trap. This is an unsigned char
         * so that after 256 times the counter wraps and the behavior turns
         * lazy again; this to deal with bursty apps that only use FPU for
         * a short time
         */
        unsigned char fpu_counter;
        s8 oomkilladj; /* OOM kill score adjustment (bit shift). */
```

Linux Task_Struct

Each process descriptor contains many fields

and some are pointers to other kernel structures

which may themselves include fields that point to structures

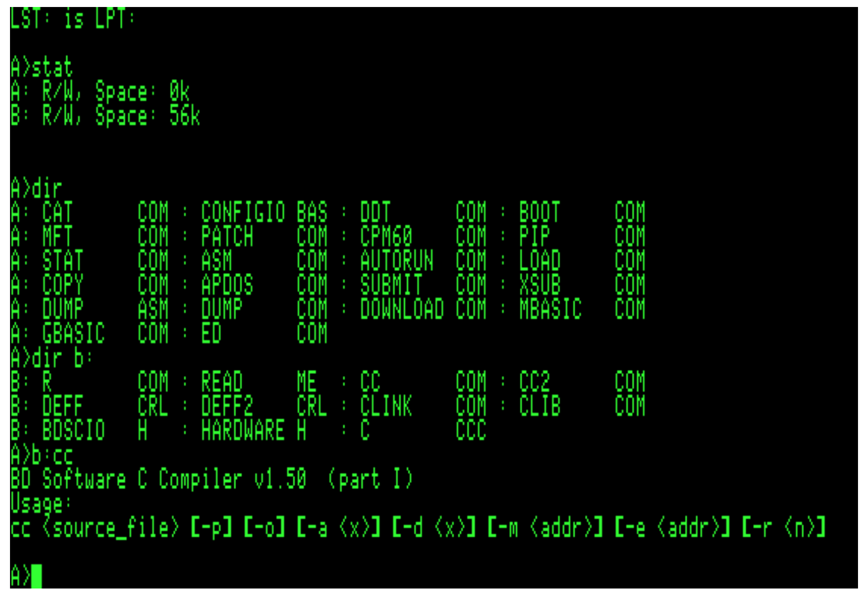


Why Linux uses the term "task"

- Tasks represent both processes and threads
- Linux viewpoint
  - Threads are processes that share address space
  - Linux tasks are "kernel threads“
  - Lighter-weight than traditional processes
    - Copy-on-write

Classes of Operating Systems

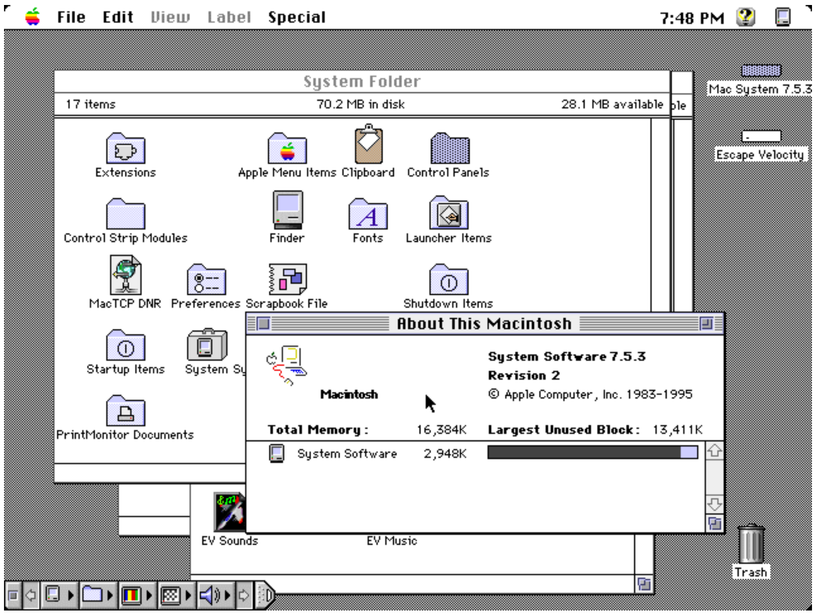
- Single User Single Tasking
  - Single process run at a time.
  - Fairly simple memory management.
  - No need for CPU scheduling.
  - Examples: CP/M, MS-DOS



CPM image released into the public domain by Vadim Rumyantsev. http://commons.wikimedia.org/wiki/File:CPM.png

Classes of Operating Systems

- Multitasking or Multiprogramming
  - Multiple processes run concurrently so the CPU must schedule tasks
  - Originally developed to give the CPU something to do while a process waited for I/O.
  - Context switching is expensive. Entire process state including all registers must be saved so the process can be restored later.



http://en.wikipedia.org/wiki/File:Macintosh_System_7.5.3_screenshot.png

Classes of Operating Systems

- Multiuser Multitasking
  - Also called time-sharing since the computer time was shared by many users
  - Can either run interactive jobs
    - Rapid response to user input but a lot of context switching causes a lot of wasted overhead
  - or batch jobs
    - No rapid response but less context switching so less overhead

Classes of Operating Systems

- Network allows information and resource sharing among multiple machines.
- Distributed OS can allow a user to transparently access resources across machines

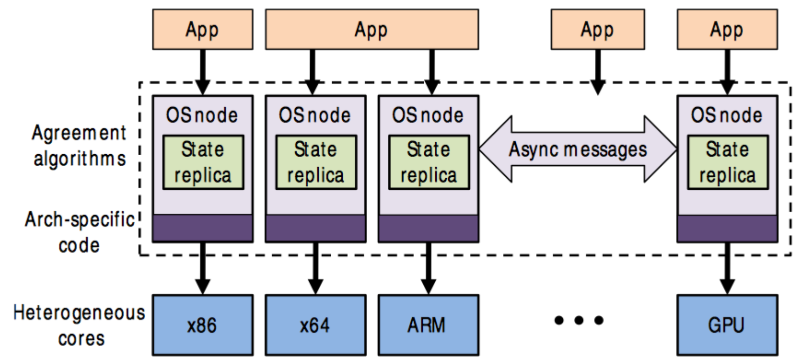


Figure 2: The multikernel architecture

"Your computer is already a distributed system. Why isn't your OS?"
Andrew Baumann, Simon Peter, Adrian Schüpbach, Akhilesh Singhania, Timothy Roscoe∗, Paul Barham†, Rebecca Isaacs

Virtual Machines

- Virtualizing allows us to abstract a total system
  - Virtual machines allow different emulation environments to be protected form one another.
  - System being abstracted can be an actual hardware design or an idealizes application virtual machine.

Virtual Machines

- Host OS - OS providing the emulation
- Guest OS - OS kernel being hosted and emulated
- Prime difficulty of the virtual machine model is creating a VM that accurately models the hardware.

Application Virtual Machine

- If the system is an idealized machine specification designed to support a language it's known as an application virtual machine.

- p-code - early design by UC Sand Diego to support their Pascal system

- JVM - Java Virtual Machine

- CLR - Common Language Runtime from Microsoft