OPERATING SYSTEMS FOUNDATIONS
MSIT 5170

University of Massachusetts Lowell
Department of Computer Science
Spring Semester 2020

Weeks 1 and 2 Lesson

1. The material in this lesson follows the reading assignment for ch 1.1 through ch 1.4 from the Tanenbaum text
2. Please read the text before you read through this lesson.
3. The material in chapter 1 provides a broad view of the topics that we will visit throughout the semester. We will work through chapter 1 during the first 3 weeks of this term, looking at ch 1.1 through ch 1.4 for this first 2 week lesson.
4. There are several examples of system level tools that are given here, and I encourage students to experiment with these tools. Each of you will be provided with web access to various systems to complete class assignments, many of which will require tools such as those shown here.

What is an Operating System?

The hardware environment provided by a typical digital computer is a complex collection of devices that require a substantial amount of programming before they can be generally useful to an application. An application that wishes to write data to a simple file, for example, will eventually have to accomplish the correct programming of the processor it's running on, the chipset that exposes the system bus to the processor, the device controller that provides access to the persistent storage used by the file system, and even the target device itself.

These types of operations have to be done over and over again for any applications that have a need to store some data. Rather than require every application to take full control of the entire system to complete its mission, operating systems have been developed to manage the details of the system and provide a set of services that applications can share, thus avoiding the need for each application to provide such features on its own.

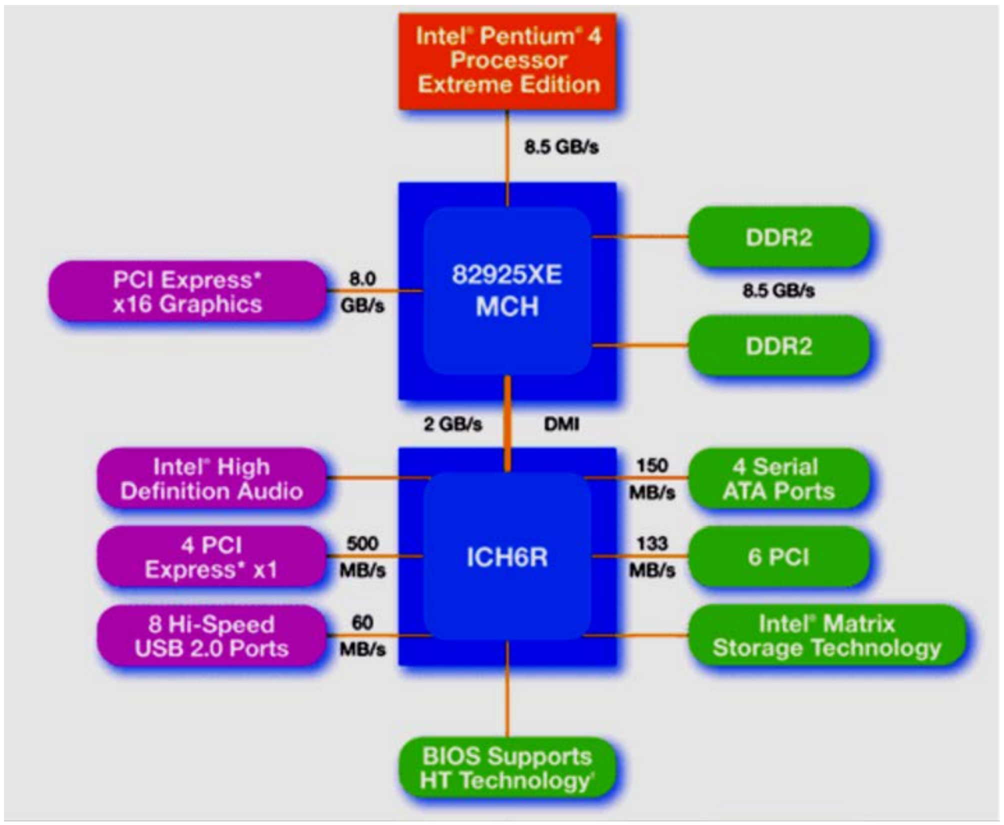
The first operating systems were little more than a collection of run-time routines that one application at a time could leverage to control the computing environment. As systems became more powerful, however, it became apparent that the concurrent execution of multiple applications would provide a much more cost-effective use of the expensive system resources.

Operating systems have evolved into resource managers, taking control of the hardware environment they are booted into, and providing support to applications through a set of procedures that we commonly refer to as the **system call Application Programming Interface** (API). Managing resources is a complex business, and thus operating systems are among the most complex entities created by human beings. Enterprise grade operating systems like Solaris, Linux, Windows and others represent millions of man hours of development and testing, and yet for all their complexity, they are well organized and systematically deployed. Our job in this course is to tackle this systematic organization, and expose the architecture of contemporary operating systems.

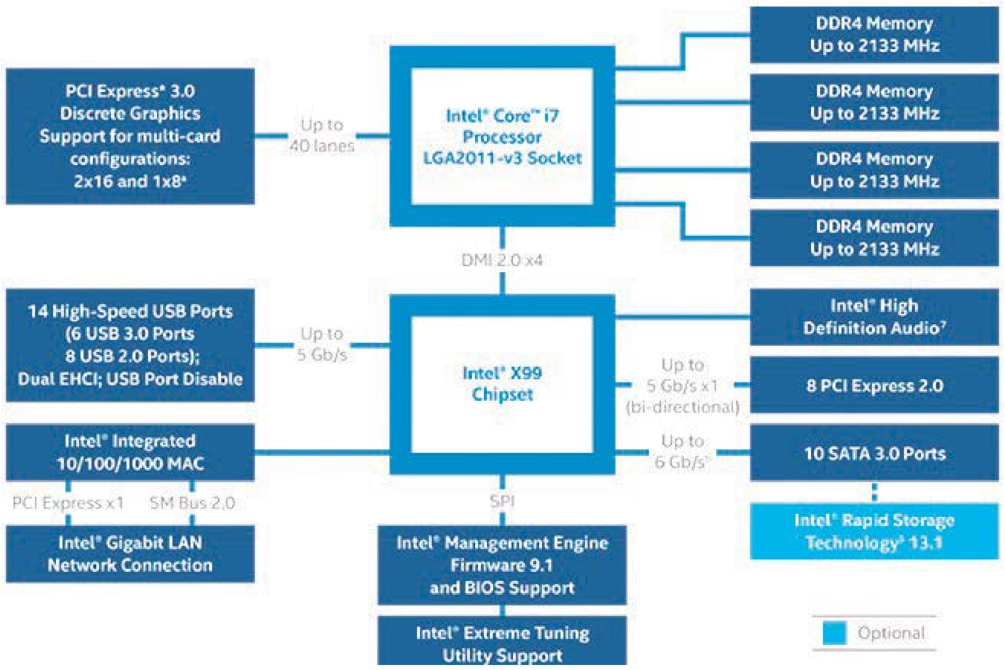
Computer hardware review:

A quick review of the hardware environments of typical systems will help us build a foundation for our operating systems studies. While hardware components are fast moving targets, the generic architecture of low and mid-range systems has been fairly stable for the past two decades. We generally regard the components of these systems as:

- One or more processing elements which we refer to as Central Processing Units (CPUs), although the term "core" has come into recent use as well
- A memory channel or north bridge chip (shown below as an Intel MCH 82925XE) that exposes a system bus to the CPUs
    - This device typically provides one or more memory controllers (if the CPU chip does not have this functionality built in)
    - It also provides an IO bridge (usually some form of Peripheral Component Interconnect (PCI)) or related interconnect technology
- The north bridge chip typically interfaces to an IO hub or south bridge chip (shown below as the Intel ICH6R) that provides embedded controllers for common peripheral interfaces like USB devices, IDE devices, and legacy parallel and serial devices.


Intel® Pentium® 4 Processor Extreme Edition
8.5 GB/s
82925XE MCH
PCI Express* x16 Graphics
8.0 GB/s
DDR2
8.5 GB/s
DDR2
2 GB/s
DMI
Intel® High Definition Audio
150 MB/s
4 Serial ATA Ports
4 PCI Express* x1
500 MB/s
ICH6R
133 MB/s
6 PCI
8 Hi-Speed USB 2.0 Ports
60 MB/s
Intel® Matrix Storage Technology
BIOS Supports HT Technology†

The picture above shows an Intel system from about 2009 that uses a memory channel hub (MCH) to provide access to DDR2 RAM, a 16x PCI-express interface for a graphics controller, and a DMI bridge (also a form of PCI-express) to connect to an IO channel hub (ICH). The ICH provides integrated controllers for IDE devices, additional PCI-express capability to support USB controllers, an audio controller and PCIe slots, as well as a conventional PCI bridge to provide legacy parallel PCI support. Later versions of Intel processors known as the i-series (i3, i5, and i7), modify the above picture by relocating the memory controller (and possibly graphics) functionality directly into the CPU complex, leaving a Southbridge type chipset to provide the peripheral functionality used to connect the rest of the system.

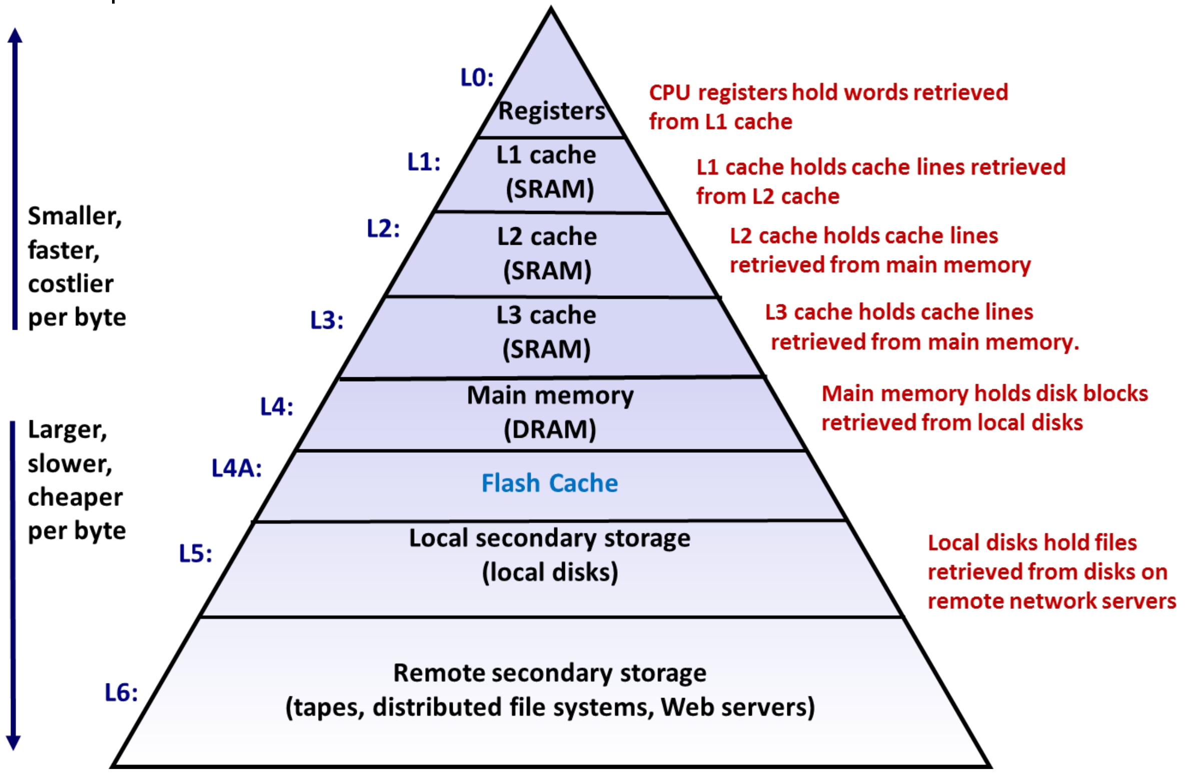

PCI Express* 3.0 Discrete Graphics Support for multi-card configurations: 2x16 and 1x8*
Up to 40 lanes
Intel® Core™ i7 Processor LGA2011-v3 Socket
DDR4 Memory Up to 2133 MHz
DDR4 Memory Up to 2133 MHz
DDR4 Memory Up to 2133 MHz
DDR4 Memory Up to 2133 MHz
DMI 2.0 x4
14 High-Speed USB Ports (6 USB 3.0 Ports 8 USB 2.0 Ports); Dual EHCI; USB Port Disable
Up to 5 Gb/s
Intel® X99 Chipset
Intel® High Definition Audio
Up to 5 Gb/s x1 (bi-directional)
8 PCI Express 2.0
Intel® Integrated 10/100/1000 MAC
Up to 6 Gb/s
10 SATA 3.0 Ports
PCI Express x1
SM Bus 2.0
SPI
Intel® Rapid Storage Technology 13.1
Intel® Gigabit LAN Network Connection
Intel® Management Engine Firmware 9.1 and BIOS Support
Intel® Extreme Tuning Utility Support
Optional

The picture above shows a newer Intel system from about 2016 that provides both memory and graphics support integrated into the CPU package. The Core i7 package includes 4 cores, 4 DDR4 memory controller channels and up to 40 Lanes of PCI Express 3.0 peripheral support (specifically multiple graphics adaptor cards). The X99 chipset supports additional devices such as USB (3.0 and 2.0 ports), SATA 3.0 ports, additional PCI Express 2.0 lanes, Ethernet, an audio controller and system management support.

Processors typically support a local L1 (level 1) cache per core, and either a local or shared L2 cache per core, with the Intel i-series providing a shared L3 cache among all the cores on a single processor package (chip). Processor caches provide an efficient interface to system RAM by leveraging the spatial and temporal locality that is typically found in virtually all software implementations. When a processor is looking for an instruction or a piece of data, the odds are very much in favor of the fact that an adjacent instruction or piece of data will be needed in the near future. This statistical fact motivates the cache strategy that basically says, “if I need something at location x, I will soon need something at location x +/- some local offset”. The processor’s memory interface responds to this logic by implementing a cache line load policy (cache lines are typically 64 bytes or greater), with the premise that, even though a current instruction requires only 4 bytes of data, reading into cache a full cache line of data is efficient, since the current executing thread will, with high probability, require something else from this cache line in the near future and reading 64 bytes in one 8 beat burst transaction is much more efficient than reading the same 64 bytes in a series of 8 separate transactions (8 because the Intel front side bus is 8 bytes wide and 8X8 = 64).

The general organization of caches is based on the memory hierarchy shown below. The fastest memory components are registers, located in the processor, and providing source operands to the Arithmetic and Logic Unit (ALU) of the CPU. Registers, in turn, are supported by the LI cache components, which are fast and near to the processor's registers. The LI cache depends on the services of higher order caches, such as dedicated or shared L2, L3, etc. cache components.

When operands are found in the L1 cache, a processor (core) can complete an operation in a single clock cycle. Cores running at a typical 3 GHz clock rate can therefore complete ~3 billion machine operations per second with 100% L1 hit rates. An L1 miss extends the execution time to 5-10 clock cycles if the required operand is in L2 cache and 10-50 clock cycles if in L3 cache. The picture drops off dramatically if operands must be fetched from RAM and worse still if from secondary storage (SSD or HDD). High L1 hit rates are necessary to get the best performance from a processor, so cache sizes have to be sized in ways that provide the best performance for the least cost.



L0:
Registers
CPU registers hold words retrieved from L1 cache
L1:
L1 cache (SRAM)
L1 cache holds cache lines retrieved from L2 cache
L2:
L2 cache (SRAM)
L2 cache holds cache lines retrieved from main memory
L3:
L3 cache (SRAM)
L3 cache holds cache lines retrieved from main memory.
L4:
Main memory (DRAM)
Main memory holds disk blocks retrieved from local disks
L4A:
Flash Cache
L5:
Local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers
L6:
Remote secondary storage (tapes, distributed file systems, Web servers)
Smaller, faster, costlier per byte
Larger, slower, cheaper per byte

As mutli-core architectures begin to dominate the landscape, a move towards higher level cache tiers is inevitable. The latest Intel processors (i3, i5, i7) are using an L3 cache to share system bus components among the growing number of cores in a single chip (now at 10 cores for the high end i7 chips). Each core in the 10 core implementation (the Broadwell-E processor), has its own L1 and L2 local caches, and shares a large (up to 25 MB) L3 cache among all the cores.

Allowing multiple cores to share a common cache (L3) can minimize cache invalidation issues, and keep the collected cores away from the front side system bus as much as possible.

Buses

From the processor complex, the outside world is exposed by the memory channel hub (north bridge chip). As we've seen, this chip provides access to RAM across a synchronous memory channel, and access to peripheral devices across another synchronous bus called the PCI bus. These busses are synchronous in the sense that data transfer across them occurs at a fixed rate, using some clock generated pulses to enable each transfer.

Transfers across a memory channel often occur on both the rising and falling edges of the clock signal, and are thus referred to as double data rate (DDR) interfaces. PCI transfers, however, occur only on the top of the clock, and are thus considerably slower than memory channel transfers (beginning with a much slower clock, and performing only one transfer per clock cycle).

A variety of different controller devices are likely to be found on a PCI bus, some, such as a Small Computer Systems Interface (SCSI), can operate in both a synchronous or an asynchronous mode, and others such as a Fibre Channel (FC) interface or an Ethernet controller, operate only in a synchronous mode.

In essence, each of these controllers that can be found on a PCI bus, makes their own bus (synchronous or asynchronous) available to target devices like disks, network transceivers, printers or keyboards. The PCI bus they are attached to, concentrates their data flow, and provides a path into and out of the system RAM.

We view these collections of end point target devices as capillaries, flowing into larger vessels in the form of intermediate busses such as fibre channel or SCSI, which, in turn, are collected into larger conduits like a PCI bus that interfaces with the processor and the memory system through the memory channel hub (aka North Bridge).

Since the processor(s) and the PCI bridge are constantly contending for access to system RAM, and other parts of the physical address space, and since the system bus can only be accessed on discrete clock edges, the memory channel hub must provide arbitration among the contending parties to ensure efficient use of the physical address space. If a processor, for example, is trying to fetch an instruction from some address in RAM, and the PCI bridge has some buffered data from one of its attached devices that must be placed to some address in RAM, both the processor and the bridge must arbitrate for access to the bus. The winner will place its target RAM address on the bus and begin a data transfer on the next clock edge. The data component of the Intel x86 system bus, for example, is 64 bits (8 bytes) wide, so this is the extent of a single bus transaction. If more data has to be read or written by a bus master like the CPU or the PCI bridge, then additional bus cycles will have to be won in arbitration.

Accessing the system bus in random 8 byte chunks, requires an address component and a data component, and thus retrieving 32 bytes of data in a set of random 8 byte chunks would require 4 address and 4 data accesses to the system bus. The arbiter, however, recognizes a form of request called a burst transfer, in which a single address is provided, followed by multiple 8 byte data loads that will be transferred to/from contiguous locations along the system bus. This is the key to caching, providing a way to load or store a cache line with minimal overhead, and dealing with the entire line in only slightly more time than it would take to deal with a single arbitrary location along the system bus. If other parts of the line will eventually be used (locality tells us that this is likely), then caching is a winning strategy.

Collecting Hardware Information

Most installed operating systems provide tools to examine, to one degree or another, the platform upon which the operating system is running. Throughout this course we will look at specific platform and third party tools for the Windows and Linux environments. In the case of the Windows world, for example, we are able to collect a significant amount of information about the system we're running on using the Device Manager applet from the Computer Management option of "My Computer" ("This PC" in Win10):

- From the Start menu, right click on the "My Computer" list element
- From this drop-down menu, select the "Manage" list element
- From the Computer Management window select the "Device Manager"
- From the "View" drop down menu, select "Devices by Connection"

A quick look at the resulting details shows that my system (an older core-2 duo system) has a dual core processor, and a Memory Channel Hub (Q35, north bridge) chip that provides PCI and PCI-express interfaces that consolidate most of the remaining components, several of which are implemented by the ICH9 (south bridge) chip. Of course there are many other tools available from both Microsoft and third parties that provide a much richer collection of hardware details than the device manager. One of the most useful freeware tools I've found for the Windows world is called HWiNFO Diagnostic Software, which can be downloaded from:

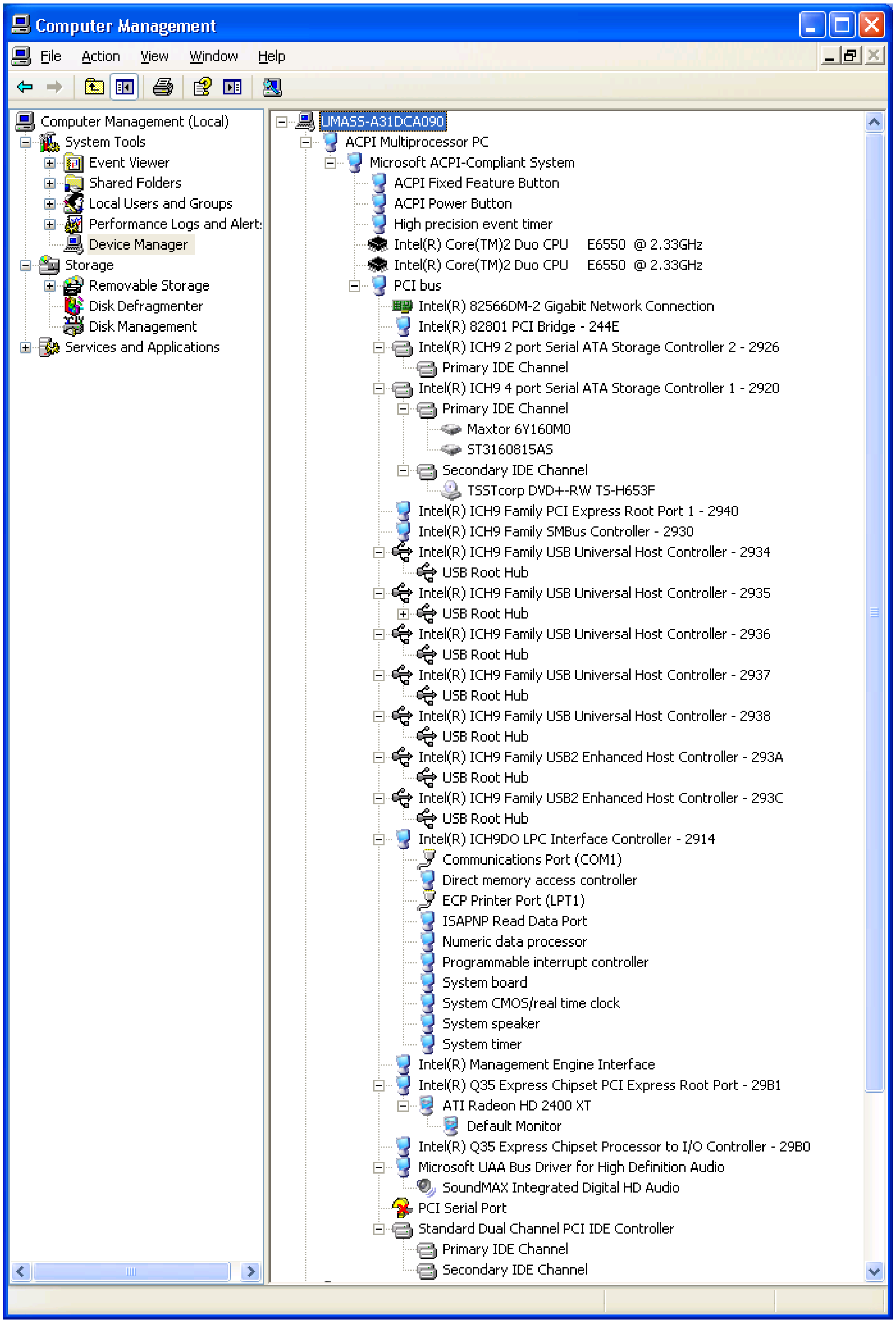
https://www.hwinfo.com/

HWiNFO provides detailed information about most system components, including the system processor(s), chipsets, memory, IO busses and attached controllers and target devices.

Another useful tool for mapping and analyzing your systems PCI connectivity can be found from PCI-Tree. PCI-Tree is another freeware application that can provide very low-level details about the busses, bridges and controllers comprising your system. You can find PCI-Tree at:

http://www.pcitree.de/index.html

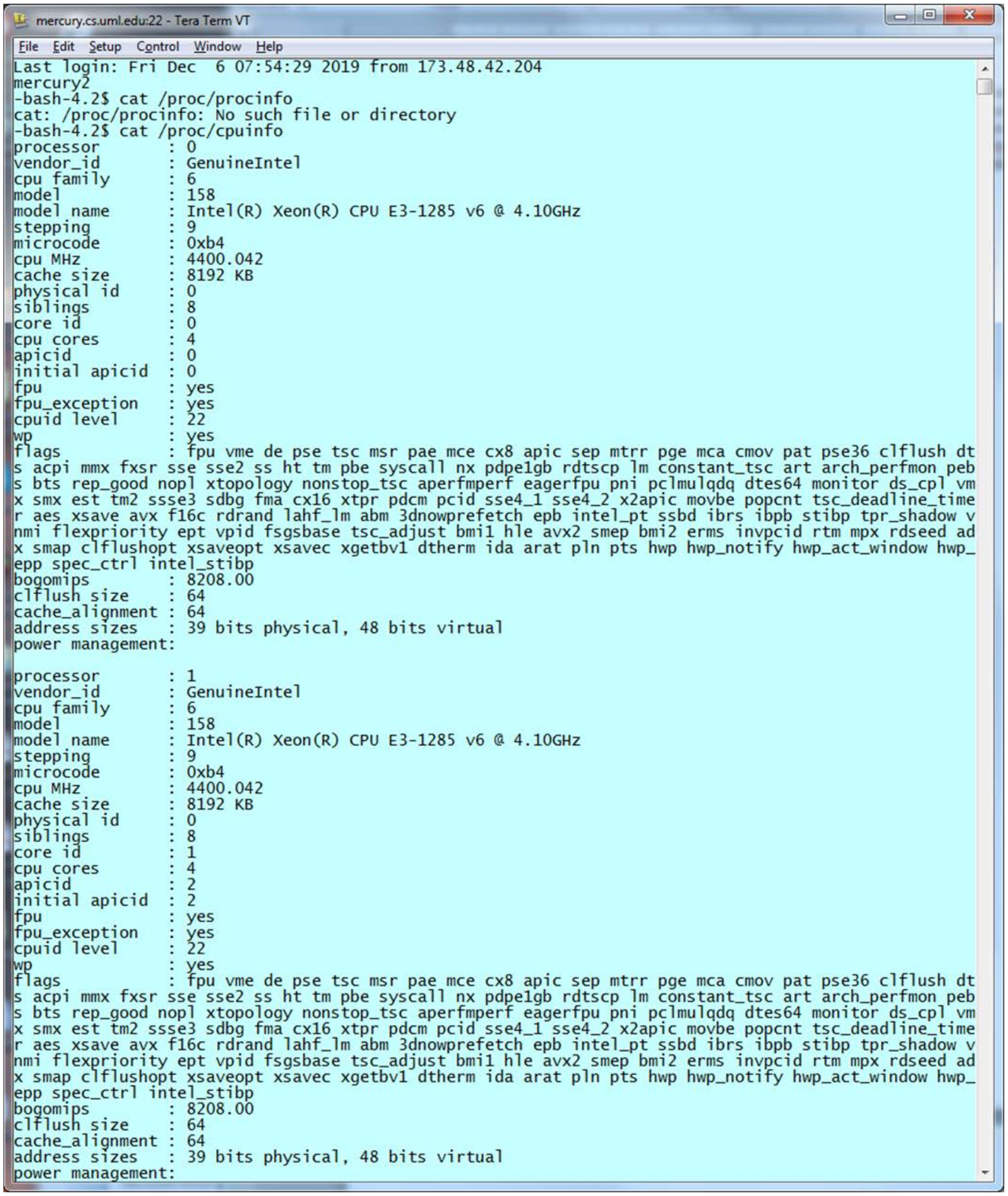
The "Manage" option discussed above is easiest accessed from a Windows 10 system by pressing the start button and selecting the "File Explorer" button (above the "Settings" button). Within "File Explorer", find the "This PC" icon and right click it to get the same menu described above.



```
Computer Management (Local)
  System Tools
    Event Viewer
    Shared Folders
    Local Users and Groups
    Performance Logs and Alert:
    Device Manager
  Storage
    Removable Storage
    Disk Defragmenter
    Disk Management
  Services and Applications
```

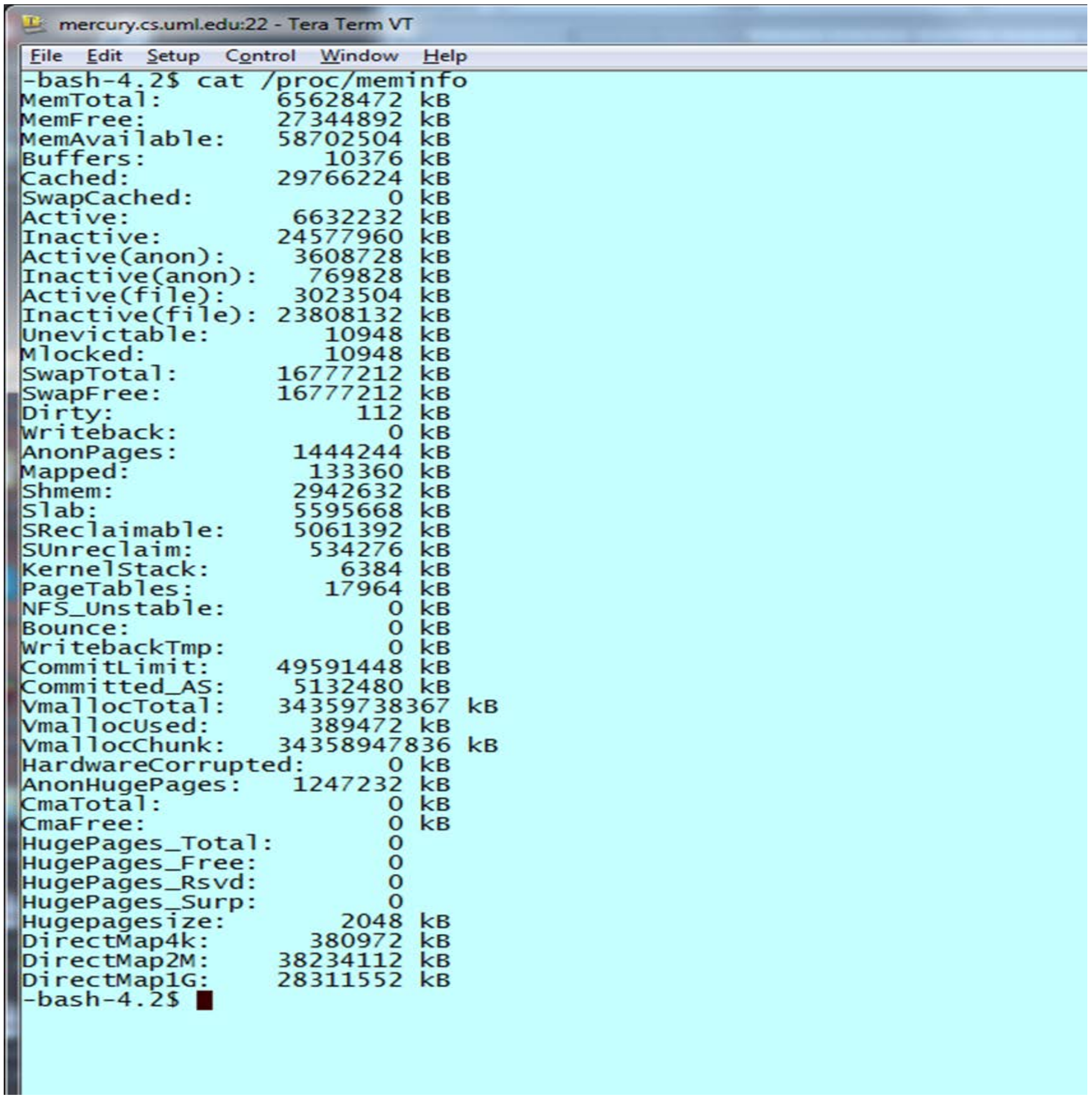
```
UMASS-A31DCA090
  ACPI Multiprocessor PC
    Microsoft ACPI-Compliant System
      ACPI Fixed Feature Button
      ACPI Power Button
      High precision event timer
      Intel(R) Core(TM)2 Duo CPU     E6550  @ 2.33GHz
      Intel(R) Core(TM)2 Duo CPU     E6550  @ 2.33GHz
      PCI bus
        Intel(R) 82566DM-2 Gigabit Network Connection
        Intel(R) 82801 PCI Bridge - 244E
        Intel(R) ICH9 2 port Serial ATA Storage Controller 2 - 2926
          Primary IDE Channel
        Intel(R) ICH9 4 port Serial ATA Storage Controller 1 - 2920
          Primary IDE Channel
            Maxtor 6Y160M0
            ST3160815AS
          Secondary IDE Channel
            TSSTcorp DVD+-RW TS-H653F
        Intel(R) ICH9 Family PCI Express Root Port 1 - 2940
        Intel(R) ICH9 Family SMBus Controller - 2930
        Intel(R) ICH9 Family USB Universal Host Controller - 2934
          USB Root Hub
        Intel(R) ICH9 Family USB Universal Host Controller - 2935
          USB Root Hub
        Intel(R) ICH9 Family USB Universal Host Controller - 2936
          USB Root Hub
        Intel(R) ICH9 Family USB Universal Host Controller - 2937
          USB Root Hub
        Intel(R) ICH9 Family USB Universal Host Controller - 2938
          USB Root Hub
        Intel(R) ICH9 Family USB2 Enhanced Host Controller - 293A
          USB Root Hub
        Intel(R) ICH9 Family USB2 Enhanced Host Controller - 293C
          USB Root Hub
        Intel(R) ICH9DO LPC Interface Controller - 2914
          Communications Port (COM1)
          Direct memory access controller
          ECP Printer Port (LPT1)
          ISAPNP Read Data Port
          Numeric data processor
          Programmable interrupt controller
          System board
          System CMOS/real time clock
          System speaker
          System timer
        Intel(R) Management Engine Interface
        Intel(R) Q35 Express Chipset PCI Express Root Port - 29B1
          ATI Radeon HD 2400 XT
            Default Monitor
        Intel(R) Q35 Express Chipset Processor to I/O Controller - 29B0
        Microsoft UAA Bus Driver for High Definition Audio
          SoundMAX Integrated Digital HD Audio
        PCI Serial Port
        Standard Dual Channel PCI IDE Controller
          Primary IDE Channel
          Secondary IDE Channel
```

Collecting configuration information on a Linux system is generally a matter of visiting the /proc local file system. System configuration routines as well as driver and module components typically create and populate simple text files in the /proc RAM based file system during system initialization or module loading. For example, the files /proc/cpuinfo and /proc/meminfo provide considerable detail about a system's processor(s) and RAM, as can be seen below (the cat command in Linux/Unix systems will print the content of a file to the screen):


mercury.cs.uml.edu:22 - Tera Term VT
File Edit Setup Control Window Help

```
Last login: Fri Dec  6 07:54:29 2019 from 173.48.42.204
mercury2
-bash-4.2$ cat /proc/procinfo
cat: /proc/procinfo: No such file or directory
-bash-4.2$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Xeon(R) CPU E3-1285 v6 @ 4.10GHz
stepping        : 9
microcode       : 0xb4
cpu MHz         : 4400.042
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dt
s acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon peb
s bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vm
x smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_time
r aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb intel_pt ssbd ibrs ibpb stibp tpr_shadow v
nmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed ad
x smap clflushopt xsaveopt xsavec xgetbv1 dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_
epp spec_ctrl intel_stibp
bogomips        : 8208.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Xeon(R) CPU E3-1285 v6 @ 4.10GHz
stepping        : 9
microcode       : 0xb4
cpu MHz         : 4400.042
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 1
cpu cores       : 4
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dt
s acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon peb
s bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vm
x smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_time
r aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb intel_pt ssbd ibrs ibpb stibp tpr_shadow v
nmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed ad
x smap clflushopt xsaveopt xsavec xgetbv1 dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_
epp spec_ctrl intel_stibp
bogomips        : 8208.00
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```


mercury.cs.uml.edu:22 - Tera Term VT
File Edit Setup Control Window Help

```
-bash-4.2$ cat /proc/meminfo
MemTotal:         65628472 kB
MemFree:          27344892 kB
MemAvailable:     58702504 kB
Buffers:             10376 kB
Cached:           29766224 kB
SwapCached:              0 kB
Active:            6632232 kB
Inactive:         24577960 kB
Active(anon):      3608728 kB
Inactive(anon):     769828 kB
Active(file):      3023504 kB
Inactive(file):   23808132 kB
Unevictable:         10948 kB
Mlocked:             10948 kB
SwapTotal:        16777212 kB
SwapFree:         16777212 kB
Dirty:                 112 kB
Writeback:               0 kB
AnonPages:         1444244 kB
Mapped:             133360 kB
Shmem:             2942632 kB
Slab:              5595668 kB
SReclaimable:      5061392 kB
SUnreclaim:         534276 kB
KernelStack:          6384 kB
PageTables:          17964 kB
NFS_Unstable:            0 kB
Bounce:                  0 kB
WritebackTmp:            0 kB
CommitLimit:      49591448 kB
Committed_AS:      5132480 kB
VmallocTotal:     34359738367 kB
VmallocUsed:        389472 kB
VmallocChunk:     34358947836 kB
HardwareCorrupted:       0 kB
AnonHugePages:     1247232 kB
CmaTotal:                0 kB
CmaFree:                 0 kB
HugePages_Total:         0
HugePages_Free:          0
HugePages_Rsvd:          0
HugePages_Surp:          0
Hugepagesize:         2048 kB
DirectMap4k:        380972 kB
DirectMap2M:      38234112 kB
DirectMap1G:      28311552 kB
-bash-4.2$
```

There are many sources of system information, and we will continue to consider them as we work our way through the course.

Operating System Flavors

At this point, hopefully we have some perspective on the basic hardware environment that our PCs deploy, and the need for some sort of operating system to manage this collection of iron. In fact, there are many different varieties of operating systems out there, each of which is typically designed to meet slightly different hardware and end user requirements. These software systems range from very hardware specific implementations such as mainframe operating systems, to the more general purpose multiple platform systems (Linux, Window, etc.) to the specialized light-weight systems (embedded systems) that we associate with everything from smart phones to smart cards.

In each case, we find a collection of software that is designed to make the underlying hardware platform accessible with a minimal amount of application effort. The APIs of these various systems provide the specific services that make sense for the platforms they are deployed on. These APIs allow applications to focus on a more logical view of the hardware topology, and not have to concern themselves with platform details. To leverage the services that a given system offers, however, one has to have an overview of how these services are organized, and that's really the essence of this course. We need to expose the basic architecture that a contemporary operating system embraces, beginning with the concept of a process, and exploring the components, construction and lifetime of these system management entities.

First Lesson (weeks 1 and 2) Summary:

Operating Systems have evolved from the simple collection of run-time routines that were shared by various applications on early machines, to the complex resource managers and service providers that we find in systems today. In a sense, a contemporary operating system provides a user accessible abstraction of an underlying collection of complex and user hostile devices.

Computing platforms come in many different shapes and sizes, and so too do the various operating systems that accompany them. We will see, however, that the principals of operating systems implementation are well understood, and the architecture of systems like Windows and Linux , which may seem to have little in common on the surface, are remarkably similar in their deployment.

An understanding of operating systems goes hand-in-hand with an understanding of the hardware environment that these systems run in. In particular, it is critically important to build an understanding of how information moves along various types of busses, how these busses behave, and how their use must be carefully coordinated to preserve the integrity of the data that they carry. There is nothing more fundamental to an operating system than to maintain the consistency of the resources such a system must manage. An operating system is a vast repository of the shared data that defines the state of a computing system at any given time. These data are constantly changing in response to events that occur at nanosecond frequencies, but the operating system must ensure that each new state has been coherently mapped from its predecessor, and, regardless of the degree of contention, all updates to shared data are atomic and consistent.