Assignment 1

By: Ian Edington
Student ID: 3236986
Date: July 17th, 2017

Part 1: Concepts and Principles

1. Define the concepts interrupt and trap, and explain the purpose of an interrupt vector.

An interrupt is a signal sent by a hardware device, or software to the CPU when an action is required. In general the interrupt triggers a context switch from the running process to an interrupt service routine (interrupt handler) corresponding to that interrupt. The interrupt handler performs an action based on the type of the interrupt and passes control to the dispatcher or the original program, depending on whether the system in preemptive or not. In certain systems, interrupts can be turned off, in which case the system will wait until interrupts are turned back on to handled the interrupt. This can result in lost or overwritten interrupts. A trap is a type of interrupt that is triggered by an error, for example a divide by zero error in the ALU. The interrupt vector is an array of addresses pointing to interrupt service routines. This array is indexed using the interrupt to provide a fast method of calling the corresponding interrupt service routine.

2. How does a computer system with von Neumann architecture execute an instruction?

A computer system using Von Neumann architecture executes each instruction in three steps: fetch, decode, execute. First the CPU fetches the instruction from main memory for the address currently held in the Program Counter (PC). The instruction is then decoded. Finally the instruction is executed. Using these three steps, data is copied into registers, mathematic and logic operations are performed on the data, and the registers content are saved back to memory.

3. What role do device controllers and device drivers play in a computer system?

Device controllers and drivers provide a layer of abstraction between the OS and an IO device. A device controller allows a CPU to connect with multiple IO devices using a single hardware interface. The device driver has detailed knowledge of the device controller hardware and provides a standard interface to access all the devices from the OS. Specialized driver may also exist for specific IO devices. Using this paradigm allows the most implementation flexibility for hardware manufacturers, while maintaining simple API's for OS's to interact with hardware devices. If a generic driver does not meet the needs of a manufacturer, it can create it's own driver to be distributed with the device without the OS needing to do the work of integrating it into the kernel.

4. Why do clustered systems provide what is considered high-availability service?

A high-availability service is one that continues to be available for use despite multiple failures. There are many causes of failures and in HA systems the risk of each of these causes needs to be assessed an if it posses a large enough risk it needs to be mitigated. Aside from designing fault tolerant software the best tool for this type of risk mitigation is redundancy. In a clustered system high-availability is accomplished by redundancy in order to eliminate single points of failure. The advent of cloud computing has brought with it redundancy in many forms. Some examples include redundant DNS name-servers, failover IP's, load-balancers, geographically displaced servers, advanced SAN and database systems like Google's Cloud Spanner. Clustered systems can achieve high-availabilities when these risk factors are mitigated and availability rates as high as 99.9% are common for enterprise level services.

5. Describe an operating system's two modes of operation.

Many modern CPUs provide hardware support for two or more operating modes using a private CPU register. The most common use for this is to define a kernel mode and user mode. When the CPU is in kernel mode, certain "privileged" instructions are possible that won't execute in user mode. Using this hardware feature it is possible to build protection and security into modern OS's that weren't possible before. Before this advance in CPU architectures it wasn't possible for an OS to force a process not to overwrite it's memory and take over the system since all of the CPU commands were available to any process running on the system.

6. Define cache, and explain cache coherency.

Cache is the process of moving data from a larger slower forms of memory into a smaller faster form of memory in order to increase the speed of accessing that piece of data. Cache coherency is the processes of ensuring that modifications to a piece of cached data are available to other clients using that data. This is accomplished through write propagation and transaction serialization. Write propagation it the processes of writing cache changes to all the places that data is stored. Transaction serialization means that any reading and writing to an individual data block has to be seen by all Clients as having happened in the same order. Cache coherency is largely a hardware issue, however, it impacts how be build SMP systems in that at a certain point message passing becomes preferable to shared memory because of the overhead of cache coherency in multi-processor systems.

7. Describe why direct memory access (DMA) is considered an efficient mechanism for performing I/O.

In a traditional IO process the CPU manages the process of transferring data from the device to memory. Using DMA the IO device has a separate path to main memory over the bus. This enables the CPU to instruct the controller where in main memory to place the data along with how much, and the controller will move the data directly to memory. Once the controller has copied the data into memory it sends an interrupt to the CPU informing it that it was successful. By cutting out an interrupt every time a piece of data is available this greatly reduces the load on the CPU when large amounts of data need to be copied.

8. Describe why multi-core processing is more efficient than placing each processor on its own chip.

All of the reasons multi-core processors are more efficient than multi-processor systems stem from the reduction in distance electrons need to travel. Communication between the processors on a single chip is faster than for processors on separate chips. Interprocess communication has less distance to travel. A shared cache is possible since they are close enough to share a cache. Shared cache decreases writing to main memory while maintaining cache coherency. Moving processes from one core to another incurs less of a penalty. Generally produce less heat. Buying two processors vs 1 dual-core processor is less expensive for the same processing power,

9. Describe the relationship between an API, the system-call interface, and the operating system.

Modern CPUs have separate kernel and user mode in order for operating systems to maintain control of the system. However, programs need access to the system utilities that are only available in kernel mode, such as memory allocation. For this reason operating systems open up a system-call interface for programs, in order to safely expose these instructions to a user-space program. The API is user-space program that provides a way to interact with system-calls. The goal is usually to provide cross platform support and an easier more standard way of interacting with system-calls. This is especially useful in order to write a program that works on different systems since the API takes care of the changes in system-calls between systems.

10. Describe some requirements and goals to consider when designing an operating system.

Because the OS determines large parts of how a system will operate, there are as many requirements and goal for OS's as there are for systems. What type of hardware will it be running on? Mobile, cluster, workstation, server, embedded. What type of jobs will be run? Batch, time-sharing, single user, multiuser, distributed, real time? What are the users goals for the system? Fault tolerant, secure, resource efficient, robust, responsive, easy to learn, easy to use. What are the OS programmers goals for the system? Easy to design, implement, and maintain. What are the User programmers goals for the system? Easy to program, well documented, reliable, error free. Each system will prioritize some of these requirements and others will be completely ignored. However, since many of these requirements are mutually exclusive it isn't really possible to make an OS that meets all these needs.

11. Explain why a modular kernel may be the best of the current operating system design techniques.

As per question 10, operating systems have such a large range in their requirements that it's difficult for one operating system to meet every systems requirements. Monolithic kernels allow for performant systems, but are difficult to maintain since they are extremely complex and require compiling the entire system every time a change is made. Layered systems are simple to design but have greatly reduced flexibility, since the data flow through the layer hierarchy is rigid. Micro-kernels tried to make the base OS as small as possible, allowing maximum flexibility with most OS services in user-space. This made for flexible solutions, however, because of the added route complexity, it decreased performance. The modular kernel strikes a balance between kernel efficiency, extensibility, and programmers cognitive load by splitting the kernel into smaller distinct pieces that work together in kernel space. This results in a system that has a huge number of flexible pieces but only loads them when they are needed. The core kernel isn't as big because lots of pieces are off loaded to kernel modules. You don't need to recompile the entire system in order to update one module. Modules are kernel space programs so they don't suffer from as much of the added runtime paths of layers and micro-kernels. Modules are distinct programs with their own area of control so they don't suffer from they massive complexity of monolithic kernels. Because of all these factors, modular kernels are most likely the way forward in kernel design.

12. Distinguish between virtualization and simulation.

Virtualization and simulation are similar in that they both run one system (guest) inside/on-top of another system (host). Where they differ is in where the host system and the guest system diverge. In a simulated system the guest and host systems run on different hardware. This means that the host system needs to emulate the hardware of the guest system. This comes at a very high performance price since for each instruction, register, ect. the host system has to translate them to use the current systems architecture. In virtualization the host and guest hardware is the same. The host system provides a virtual processor, memory, and IO devices that are scaled down versions of the host systems environment. However, when the guest system writes to memory, accesses the CPU or writes to disk, all these actions are performed on the actual hardware with minimal interference from the host OS. This results in much better performance, while maintaining separate systems.

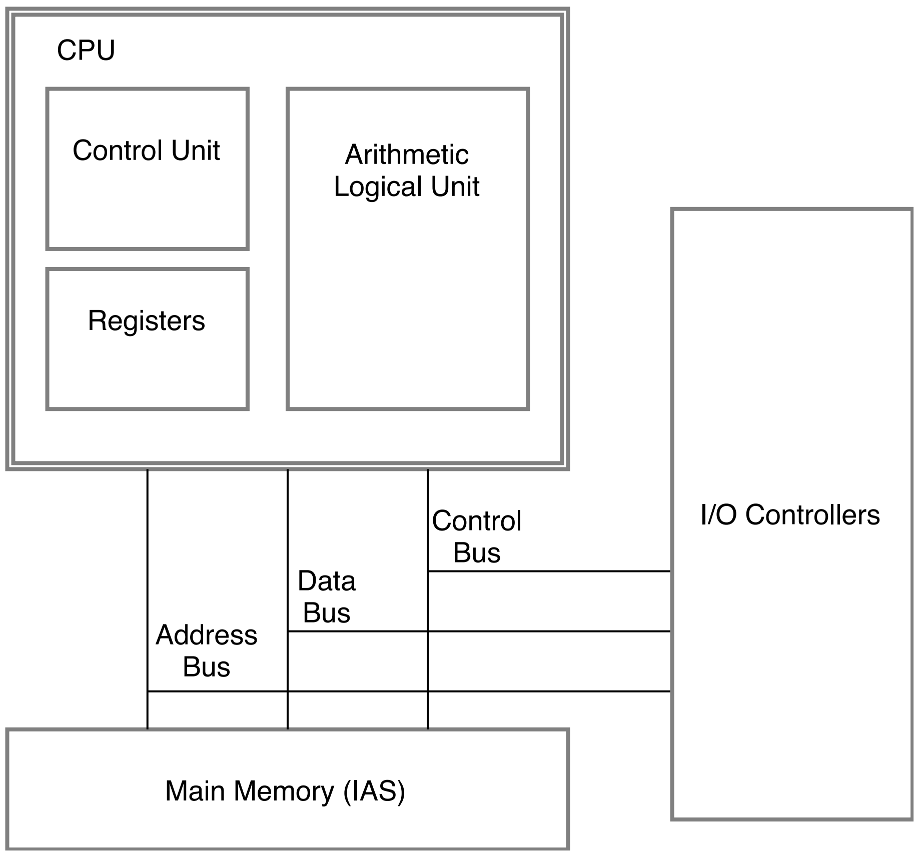
Part 2: Design Considerations

1. Draw a typical computer organization figure that includes the main components of Von Neumann architecture. Identify each component, and explain its function and interaction relative to other components.

Von Neumann architecture is the basis of all modern computers. When Von Neumann proposed this architecture it was a big jump to realize there was no difference between program instructions and data. The simplification brought by treating data and instructions the same way greatly decreased the complexity and implementation cost of computers. It also lead to many improvements, including interpreted languages, JIT compilers, and dynamic programming. The simplicity of this architecture is the main reason it has become so prevalent.

However, there is one downside to the Von Neumann architecture when compared with other models, the memory bottleneck. Since the instruction set and the application data are both stored in the memory and accessed over the same bus, the amount of useful work is limited by how quickly data can pass though the BUS. This bottleneck results in large amounts of CPU and human resources being focused on moving data back and fourth from main memory as quickly as possible, rather than other areas of computer science. The study of caching and cache coherency is a direct result of the Von Neumann bottleneck.

Figure 1: Von Neumann Architecture



Components

CPU: Retrieve instructions from memory and execute them

- Control Unit: ensures other elements are executing correctly and at the correct time.
- Registers: Holds the current instruction being executed, information used by the instruction, and a pointer to the next instruction.
- Arithmetic/Logical Unit: performs operations provided by Control Unit on registers.

Main memory: Holds both program instructions and application data.

BUS: A single set of wires connects the memory, IO Controllers, and the CPU. These wires relay instructions, data, and addresses between them.

- Control BUS: The control BUS is a unidirectional bus from the CPU to the Memory and IO Controllers, which determines which device will perform what operation.
- Address BUS: The address where the memory should be read from or written too. Unidirectional from the CPU to memory or IO Controller.
- Data BUS: Bidirectional bus allowing data to travel between the CPU, Memory, and IO Controller. The from where and to where is determined by the Control and Address BUS.

IO device controller: Provides an interface to access external devices and return data to the CPU as requested.

2. Define system call, and list the main types of system calls. Elaborate on how a system call interacts with a standard C library and hardware under a dual-mode operating system environment.

Define system call.

A system call is a entry point to an operating system from a user-space program. A user-space program uses the system-call to request a service from the operating system. In this way the available system-calls act as an interface between user-space programs and the operating system. System calls are a useful tool because the allow for separation of concerns between the OS and the user program, they help the operating system maintain control of the system, and they provide hardware optimized services to user programs that otherwise would need to be implemented by each program.

List the main types of system calls.

These services can be broken into 6 main types: process control, file management, device management, information maintenance, communications, and protection.

The process control system calls allow a process to control it's own processing and control other processes. They involve managing the PCB, memory, and execution of processes. Some potential system calls associated with this group are end, abort, fork, load, execute, terminate, get and set process attributes, wait for time or event, signal event, acquire and release lock, and allocate and free memory.

The file management system calls provide a systems view of a file system. This can vary greatly between systems since files are a conceptual framework and not a hardware implementation detail. However, most file systems have the concept of a file and a directory, and user programs need to be able to create, edit, and delete both files and directories. Some system calls associated with this group are create, delete, open, close, read, write, reposition, get and set file attributes.

The device management system calls provide access to the I/O devices attached to the system. Depending on how the file management system calls are structured, sometimes the device and file management system calls will overlap. Having them overlap makes it easier to reason about them and easier to implement new drivers. However, limiting the device management system calls scope in this way might not allow enough flexibility to access special device functionality. Some system calls associated with this group are request device, release device, read, write, reposition, get and set device attributes, logically attach and detach devices.

The information maintenance system calls pass info between kernel and user program. These are useful for debugging, reporting, profiling, and managing system information. Some system calls associated with this group are get or set time or date, get or set system data, and get or set process, file, or device attributes.

Interprocess Communication system calls allow access to any IPC functionality the OS provides. These services are generally split into communication between processes and between systems, and may have overlap with both the file and device management system-calls. Depending on the system both shared memory and message passing protocols are available. Some system calls associated with this group are create and delete communication connection, send and receive messages, transfer status information, attach and detach from remote devices.

Protection system-calls are designed to make sure only the right users are allowed to access the right services. Many of the system calls above will have some type of permissions control in order to decide which users can make which system calls. What can or can't have permissions associated with it is largely up to the OS, however, there is usually the concept of users and groups and different privilege based on an individuals user or group. The system calls associated with this group are set or get permission, allow or deny user.

Elaborate on how a system call interacts with a standard C library.

Each operating system defines it's own system-calls based on the services available, which results in different operating systems having different interfaces. Many API's exist in order to mitigate the differences between OS's and provide simpler tools for a programmer to use. The standard C library is one such API.

The standard C library provides a standard set of functions that use system calls under the hood to provide their functionality. However, on different operating systems the standard C library will use different system calls. The benefits are that the programmer only needs to learn one interface to program across multiple environments, and programs tend to be more portable. For example the `malloc()` function in the standard C library will look at the programs available memory and allocate a chunk to an object. However, if the program needs more memory from the system, `malloc()` will perform those system calls on the programmers behalf. Better still, `malloc()` knows which system calls to invoke based on the current system and will chose between `brk()` , `sbrk()` , `mmap()` , and `VirtualAlloc()` based on the current needs of the program and the system it is executing on. Since `malloc()` is independent from the OS a call to `malloc()` will work the same on both Windows and Unix systems.

The goal of this API and many other user level API's is to simplify the management of the underlying system. This simplification makes it easier for programmers to design systems since they don't have to rethink the best strategy to make requests from the system each time.

Elaborate on how a system call interacts with hardware under a dual-mode operating system environment.

Many new CPU's have a dual-mode architecture, where one mode is only allowed to execute a reduced instruction set. Examples of privileged instructions are timer management, turning off or on interrupts, memory management, and I/O access or management. If a user program had access to any of these CPU instructions it would be able to take over the system. For this reason the mode with a reduced instruction set is called user mode and the mode with full instruction set is called kernel mode.

However, restricting a user program to instructions from user mode would result in mostly useless programs, without access to memory, or devices. Instead access to these services is provided through system-calls. This allows the operating system to monitor access to restricted areas and deny access if a process doesn't have the right privileges.

When a system call is initiated by a user program, the CPU is initially still in user mode. The system call places values in unprivileged registers to indicate what is being requested of the OS. The system call then issues a software interrupt and control is handed over to the kernel. Since a CPU interrupt is handled by a known kernel level interrupt service routine, the CPU switches into kernel mode. Now in kernel mode, the kernel determines if the request can be completed based on the registers passed. Just before returns the results to the user program, the kernel switches from kernel mode to user mode. The user program continues with the results of it's system call without ever having access to kernel mode.

This hardware feature is a necessary building block for OS protection and security in modern OS's that weren't possible prior to these advances. Before this change in CPU architectures it wasn't possible for an OS to force a process not to overwrite it's memory and take over the system since all of the CPU commands were available to any process running on the system.

3. Describe the overall structure of virtual machines, and compare VMware and JVM.

Describe the overall structure of virtual machines.

Although not a very precise definition, a virtual machine (VM) is a system that runs within another system. The underlying system that runs these virtual systems is called a Virtual Machine Manager (VMM) or Hypervisor. Virtual environments have been implemented in many different ways, and four of these virtual systems are not considered virtual machines: paravirtualization, emulation, virtual programming environments, and application containment. However, over the last few years a certain amount of consensus has been reached on what exactly qualifies as a virtual machine environment. A virtual machine environment can be summarized in three points:

1. The environment provided is indistinguishable from that of a hardware system.
2. The resulting virtual system runs with only minor efficiency lose when compared to a hardware system.
3. The VMM maintains complete control of the underlying hardware.

These requirements along with the active management of VMMs provide certain benefits over hardware systems. OS's are isolated from each-other so that one bad program doesn't cause the entire mainframe to crash. Live migration allow a VM to be migrated from one server to another without downtime, resulting in lower service downtime caused my hardware maintenance. VM snapshots save the state of a system as a backup or in order to create another instance of the VM. Running one large server with multiple VMs is more resource efficient than running multiple small servers. VMMs are able to perform management actions across multiple VMs at the same time including backups, installation of patches, and other management task. All these benefits result in data-center management that is easier and more robust than the equivalent hardware management.

Three main VMM architectural strategies exist, which are denoted as type 0, 1, and 2 hypervisors. Type 0 hypervisors are hardware only solutions that provide dedicated hardware to each VM. These were some of the first hypervisors and were designed to allow mainframes to run more flexible environments. Compared to earlier versions of the type 1 hypervisor, these hardware solutions provided better performance. However, with advances in hardware supported type 1 hypervisors, the restriction of having dedicated hardware for each VM has lead to comparatively less flexibility.

Type 1 hypervisors are software VMMs that either run on bare metal (hostless) or as part of the kernel in a general purpose OS, and provide almost direct CPU and Memory access to their guest system. This is by far the most widely used implementation of VMMs in the world today, and is the basis of cloud computing. Since a host and guest OS run on the same CPU instruction set any non-privileged instructions from the guest OS can be executed directly on the CPU. However, when the kernel of the guest system tries to execute privileged instructions an error occurs, since the guest systems kernel is still executing in usermode on the physical CPU. There are a number of solutions to this and other issues software hypervisors face, for example trap-and-emulate and binary translation. As hypervisors become more popular, ever increasing hardware support has become available. Hardware support in the form of vCPU management, IO controllers, and nested page tables has had a drastic impact on overall performance of type 1 hypervisors. This has lead to type 1 hypervisors outperforming type 0 hypervisors, and is the main reason it is the solution of choice for data-centres.

Type 2 hypervisors are also a software only implementation, however, instead of running in a kernel or on bare metal, they run within an existing operating system (hosted). Because of being a hosted solution, type 2 hypervisors do not have access to hardware support for VMs, which limits their performance. However, one benefit is that without privileged access these hypervisors can run in a user context without access to the main system. This makes it an idea solution for workstations where a user needs access to another system but can't have administrative control of the entire system.

Compare VMware and JVM.

VMware has a ride range of products including hosted type 1, hostless type 1, and type 2 hypervisors. However, their initial product offering was a type 2 hypervisor called VMware Workstation. This was the first VMM to garner large scale adoption and is still used today in order to provide multiple OS support on workstations. Since VMware runs in user-space on consumer machines it employs binary translation to manage a vCPU and nested page tables. This has all the benefits and downsides of any other type 2 hypervisor.

In contrast, the JVM is not a VMM. The JVM is a Virtual programming environment, designed to work across many processor environments. Instead of running the same instruction set as the host machine, the JVM runs a set of instructions that are optimized for portability across all machines. The JVM then translates these instructions into native machine code at the time of execution, called just in time (JIT) compilation. Although not as efficient as native code the JVM has seen huge adoption because of it's portability, most notably the android mobile platform.