**OPERATING SYSTEMS LEC 01**

20th April – Lecture in tuition free week

**Operating System**

An OS is a program that acts as an interface between a user and the computer hardware. It provides a basis for running applications.

**Kernel**

Is a program that is running all the time on a computer and provides services to the system and its users. System programs are run at boot time. When running they are called system process or system daemons.

Once booted it waits for events to happen and probes the hardware buses to see what is present. These are called interrupts and are received from hardware through the system bus.

In Linux the first system process is called **init**. It runs other daemons.

**Purposes**

1. Provide an environment for a user to execute programs conveniently and efficiently
2. Simulate features not available on hardware
   1. Virtual machines, virtual memory, etc.
3. Control resources and provide a basis which applications can be written

**Computer**

* Has one or more CPU
  + Brain of system
* Memory system
* Input/output devices eg. Disks, printers

A system bus connects all of these together.

Each device has a controller in charge of a particular device type (eg. Disks) and each has a local buffer.

**Bootstrap**

An initial computer program that runs when powered up. It is stored in hardware called firmware such as ROM. Initialises all aspects of a system such as CPU registers. Knows how to locate the OS kernel and run it.

**Interrupts**

When an interrupt occurs, the CPU stops its current task and jumps to a specific location to execute the **interrupt service routine (ISR)**. It then continues its task once the interrupt is completed (which is saved before the ISR is carried out).

Lower number – higher priority.

During I/O various devices raise interruptions to indicate certain events (eg. that they have completed output). Also used for exceptions in programs.

**Speed**

Registers - > cache - > main memory (all these 3 are volatile)

Solid state drives - > magnetic drives - > optical drives - > magnetic tapes (non-volatile)

**CPU**

Executes instructions only from memory. Uses main memory (RAM) which is the only large storage media that it can access directly.

Execution cycle

1. Fetches an instruction from memory and stores it in a register.
2. Decodes the instruction
3. Fetch any operands (eg. X +y… get x)
4. Execute and store the results in memory
5. Repeat the steps

**I/O Concepts**

Any I/O device communicates with a system by sending signals over a cable or wireless via a port. A device controller is a collection of electronics that operate a port, a bus or a device. One or more devices may use a common set of wires called a bus.

**Host controller interactions**

Uses:

* Polling
* Interrupts

**Polling**

Used to determine the state of a device

* Host repeatedly reads the **busy** bit in status register until not busy
* Host sets **command ready** bit in status register and command bit (eg. Write) in control register when command is available for controller to execute and write **one byte** in data-out register
* Device controller sets **busy** bit in status register, read the command register, read the data-out register and does I/O to device
* Controller clears command ready bit, clear **error** bit in status register to indicate device I/O succeeded and clear busy bit

Polling is efficient if the device and its controller are fast

* Otherwise host should switch tasks (use interrupt)
  + Each poll only needs 3 CPU instructions (read register, logical AND, and jump)

A picture containing screenshot

Description automatically generated**Basic interrupt I/O Mechanism**

1. Device controller raises and interrupt by asserting a signal to the CPU
2. CPU catches interrupt and dispatches the interrupt to the ISR or interrupt handler
3. Interrupt handler clears the interrupt and resumes with interrupted task.

**Direct Memory access (DMA)**

Bypasses CPU to transfer data directly between I/O device and memory to improve the performance of large data transfer. Allows CPU to go on with other work concurrently.

CPU gives command to DMA controller, sending pointer with sender and receiver address and also the number of bytes to be transferred. Once finished the DMA interrupts the CPU.

**Multiprocessor Systems**

* Increased throughput – gets more work done in less time
  + Not by n times due to overhead
* Economy of scale – I/O devices, memory and power supplies can be shared.
* Increased reliability – failure of one processor does not make the system down.
  + Graceful degradation – system performs operations proportional to level of operations of remaining parts of the system
  + Fault tolerance – system continues in event of component failures

Asymmetric – One master processor which schedules and allocates work to slave processors. Each slave processor waits for instructions from its masters.

SMP (symmetric) – each processor runs an identical copy of the OS and has its own registers and cache allowing for many processes to un.

Multicore systems have multiple processor but on the same chip. These are more efficient because on-chip communication is faster and inter-chip communication.

A clustered system consists of multiple CPUs.

A Beowulf cluster consists of multiple computers interlinked by LAN.

**Multiprogramming**

Multiple tasks stores in main memory at the same time with the CPU multiplexed among them. Needs job scheduling, memory management, job scheduling and I/O allocation

**Time Sharing/Multitasking Systems**

Provide interactive use of computer system at reasonable cost (one computer – several users/jobs). Requires on-line communication between users and system and on-line file system for users to access data and code.

**Parallel vs Concurrent**

Use concurrent to explain when one core is performing multiple tasks while parallel is running multiple processes over multiple resources.

**Real-Time Systems**

The system is functional if it returns the correct result within the time constraint.

* Hard real time
  + Guarantees the critical tasks complete on time
  + Often used as a control device in a dedicated application.
  + Secondary storage is limited or absent
* Soft real time
  + Gets priority over others until completion
  + Limited utility in industrial control robotics
  + Useful in applications (multimedia) requiring advanced OS features

**Virtualisation**

Allows an OS to run as applications on another OS. It is a software that includes and emulation that is used when the type of source CPU is different from the target CPU.

**OS Operation**

In multiprogramming an OS must ensure that incorrect/malicious programs do not allow other programs to execute incorrectly. Most errors are detected by hardware and handled by the OS.

**Dual Mode Operation**

Hardware provides at least two modes of operations:

**User Mode** – in which user programs run

**Monitor Mode** – also called Supervisor, System or privileged mode

Provided with a monitor but by hardware. Monitor (0), user (1).

**Privileged Instructions**

Some instructions are given an instruction as a bit that allows it to only be executed in monitor mode helping to protect the hardware and OS from incorrect executions.

All I/O instructions are privileged instructions! Users utilise a system call, usually a trap, to verify and execute the request.

Must also provide memory protection by providing two registers that determine the range of legal addresses a program may access:

**Base Register** – holds smallest legal physical memory address

**Limit Register** – contains the size of the range.

**CPU**

Operating system uses a timer to ensure a user program doesn’t use the CPU all the time and ensure control is maintained especially in time sharing scenarios.

**System Calls**

It is a user interface to the OS services. Can use assembly-language instruction, high level languages for systems programming (eg. C) or Application Program Interface (API) to use system calls for the OS. These are expensive in terms for time.

Types of system calls:

* Process Control
* File Management
* Device Management
* Information Maintenance
* Communications
* Protection

**System Programs**

* File manipulation
* Status information
* File modification
* Programming-language support
* Program loading and execution
* Communication

**OS Structures**

* Simple Structure – no well-defined structures, not divided into modules
* Layered – Divided into layers each built on top of lower layers (bottom is hardware)
  + Only use functions of lower-level layers
  + Advantage – modularity, easy to debug
  + Disadvantage – less efficient, hard to define level functionality
* Microkernel – small-sized kernel providing minimal services such as process and memory management and communication
  + Advantages – easier to extend OS, more portable, better security and reliability
  + Disadvantages – Message passing increases system function overhead
    - Reduces performance
* Modules – current method, kernel has set of core components and adds additional modules (scheduling, device drivers, file systems) during boot/run time to kernel.
  + Like layered system but more flexible (module can call any other module)
  + Like microkernel but more efficient (no message passing)
  + Solaris, Linux, Mac OS X

**A screenshot of a social media post

Description automatically generated
**

**UNIX System Structure**

**Kernel** is everything below the system call interface but above the hardware controllers.

**Microkernels**

Smaller size and only provides minimal services such as process and memory management and communication. It’s easier to extend the OS, more portable and has better security and reliability however has reduced performance.

**Modular kernel**

Current method utilising object orientated programming techniques. It has a set of core components and adds additional modules during boot time to kernel. Like layered system but more flexible, like a microkernel but more efficient.