Modern Operating Systems

Summary by V.D

April 8, 2021

1 Introduction

Operating System: Provides user programs with a cleaner model of the computer operations and handle the management of computer resources.

Shell: Text-based program that users interact with. Not part of the operating system, however utilises it.

GUI: Graphical User Interface, an icon-based program that users interact with to get access to the operating system.

Kernel Mode: Where the operating system runs. This mode has complete access to all the hardware and can execute any instruction that the machine is capable of executing.

User Mode: Where the rest of the software runs, and only has a subset of the machine instructions available.

User interface program: The shell or GUI is the lowest level of user-mode software and allows the user to start other programs which then use the OS.

Disk driver: Provides an interface to read and write disk blocks, so that the programmer doesn't have to deal with the complications.

Multiplexing: Sharing resources in time and space.

Time multiplexing: When a resource is time multiplexed, different programs or users take turns using it. Eg. Round robin rotation on a CPU. The OS is in charge of determining how to split the resources.

Space multiplexing: Rather than giving each user a turn, we can run the processes simultaneously and give each process a bit of memory. Eg. disk storage allows multiple users to store files at the same time. The OS is in charge of allocating disk space and keeping track of which user is using which disk blocks.

CPU: Considered the brain of the computer, it fetches instructions from memory and executes them. CPUs contain some registers inside them to hold variables and results, which can then be stored into memory.

Program Counter: A special register visible to the programmer. It contains the memory address of the next instruction to be fetched. It updates to the next instruction when the current instruction its holding has been fetched.

Stack Pointer: A special register that points to the top of the current stack in memory. It holds input parameters, local variables and temporary variables that are not kept in registers.

Program Status Word: A register that stores the condition code bits, CPU priority, mode (user or kernel) and other control bits.

System Call: Allows a user program to use the kernel mode and invoke the operating system. After finishing an instruction, returns control to the user program in user mode.

Multi-threading: Allows the CPU to hold multiple states of threads and switch between them. Processes aren't actually running in parallel, however whenever a process doesn't require the CPU, it can switch to a different one in a negligible amount of time.

1.1 Hardware

GPU: Graphics Processing Unit, a processor with thousands of tiny cores, best used for many small computations done in parallel (polygon rendering).

Memory: Ideally should be extremely fast, large amount of space and cheap. As there is no current technology that satisfies all of these requirements, a hierarchical memory system is used.

Registers: Internal to the CPU, made with the same material and are as fast as the CPU. Accessing the registers has no delay, but they have a very small capacity (1KB). Programs manage the registers themselves.

Cache: Mostly controlled by the hardware. Stores main memory that is accessed frequently, so the program checks if the cache has the required information (*cache hit*), and if not a memory request is sent over to the main memory. Also limited in size, but takes 2nsecs if if there is a cache hit.

Main memory: Also known as RAM (Random Access Memory). Handles all CPU requests that cannot be handled by the cache go to the main memory. Also has ROM (Read Only Memory) which cannot be changed once it is programmed at the factory. Generally used to start the computer and handle device controls.

Magnetic Disk: Cheaper than RAM and can be expanded. Only issue that it is much slower to access. Can have disks or SSDs, both which store data.

Virtual memory: Uses main memory as a cache, and stores the rest of the memory on the virtual memory. Allows the possibility of running programs larger than physical memory and gives the illusion that we have access to all the memory needed.

Memory Management Unit (MMU): Maps a program generated (virtual) address to the physical address in the RAM where the memory is located.

Busy Waiting: A method used to handle I/O operations. Done by continuously polling the device until all input is handled, then returning to the CPU. An issue with this method is that the process doesn't need the CPU while waiting for input but ties it up anyway.

Interrupts: Another method used to handle I/O operations. When we wait for input we can block the process and schedule another process on the CPU. After the blocked process is completed, it can interrupt the controller, and we can return to running the original process.

1.2 Operating System Concepts

Process: A program in execution. Associated with each process is its address space, a list of memory locations from 0 to some maximum, which the process can read and write. It also has a set of resources eg. registers, open files and other information needed to run the program. A container is essentially a container that holds all of the information needed to run a program.

Address space: Where a program is located. It has the executable program, the programs data and its stack.

Process table: Stores all the information about each process in a table. Contains an array of structures, one for each process currently in existence.

Process management system calls: Deals with the creation or termination of processes. A process can create a new process, and when that process is finished, terminates it.

Child process: When a process can create one or more other processes. Child processes can also create further children processes.

Interprocess communication: When two processes need to communicate with each other to complete a job (i.e waiting for a process to terminate before proceeding).

1.3 System Calls

System Call: A further abstraction to the interactions between the operating system and the user program. We can pass information and variables to the OS and then it can execute a TRAP instruction to switch from user to kernel mode, execute the required instructions and return the data.

TRAP instruction: Switches to kernel mode, the procedure call instruction does not change the mode.

System Call Services: Determines most of what the OS has to handle, as resource management tends to be minimal. Can involve creating and terminating processes, creating, deleting, reading and writing files, managing directories and performing input and output operations.

`fork():` The only method that can create a new process in POSIX. It creates an exact duplicate of the original process and the copy (parent and child) goes their own ways. The fork also returns a value which is 0 if the process is a child, and equal to the child's PID in the parent.

`waitpid():` The parent waits for the child, or any child process to terminate.

`execve():` Makes fork useful by replacing a child process with a new program, preventing two copies of the same program to be run. We need fork to create a new process, and execve to run a new program in the process.

2 Processes and Threads

Multiprogramming: Each process has its own virtual CPU, but in reality the CPU switches back and forth between multiple processes. After a long time interval each process has made progress, however only one is running on a CPU at a given time.

Daemons: Processes that run in the background eg. email.

Interrupt Vector: Contains the address of the interrupt service procedure. When a process is interrupted the computer can use this to jump to the address in the vector, and the rest is the software's responsibility.

Threads: Miniprocesses, ie. a process within a process. They can run in quasi-parallel and share address space, and data amongst themselves. Also easier to switch, create and destroy compared to processes. It is generally more convenient to have 3 threads with different responsibilities updating the same data, than 3 separate processes or 1 single process which handles everything.

`thread_create` A thread can create a new thread using this command.

`thread_exit:` A thread can exit by calling this procedure. It then vanishes and is no longer able to be scheduled.

`thread_join:` A thread can wait for another thread to exit by using `thread_join`. This blocks the calling thread from running until a specific thread has exited.

`thread_yield`: Allows a thread to voluntary give up the CPU so that another thread can run. As there are no clock interrupts in multiple threads, we need threads to give other threads a chance to run.

2.1 Interprocess Communication

Race Conditions: When the final result depends on what runs in which order.

Critical Regions: A solution to avoid race conditions. If a process is using a shared variable or file, then other processes will be excluded from accessing the same data until the first process is finished.

Disabling Interrupts: A way to achieve mutual exclusion for critical regions. Generally undesirable as we don't want to give that kind of power to processes and prevent things like hardware interrupts.

Lock Variable: Use a variable to check if a process is in a critical region or not. Still has the issue of a race condition, as two processes can read a lock at the same time before one modifies it.

Strict Alternation: Gives each process a turn in the critical region. A loop is used to check validity, and if it is the correct process it can enter the critical region. Once the process is finished it sets the turn to the next process, and the loop continues. While this does avoid race conditions, it can also delay the turn while not in a critical region.

TSL: Test, set and lock process. A separate process is used to control the shared lock variable and is responsible for setting it before and after entering the critical region. It also records the variable of lock and will only let it in when it is 0. It only works if all processes call `enter_region` and `exit_region` at the correct times.

2.2 Process Behaviour

CPU-bound: When a process has long CPU bursts and infrequent I/O waits.

I/O bound: When a process have short CPU bursts and frequent I/O waits. They don't normally compute much between I/O.

Preemptive: A scheduling algorithm that will pick a process and let it run for a maximum of some time. If it is still running at the end of some time, the scheduler will stop it and pick another process.

Non-preemptive: Picks a process to run until it blocks itself or voluntarily releases the CPU.

3 Memory Management

Memory Manager: Keep tracks of which parts of memory are in use, allocate memory to processes when needed and deallocate when it when done.

Address Space: A set of addresses that a process can use to address memory. Each process has its own address space.

3.1 Base and Limit Registers

Dynamic Relocation: Maps each process' address space onto a different part of physical memory.

Base register: The value of the lower limit of an address space (value that we start from).

Limit register: The value of the upper limit of an address space. Where the address space ends.

3.2 Swapping

Swapping: A simple strategy to dealing with memory overload. Brings an entire process, runs it for a while then puts it back on the disk. Idle processes are stored on disk so they don't take up memory while they aren't running, and we can just swap out processes as needed.

Memory Compaction: When swapping creates holes in memory, we can attempt to combine them into a big hole by moving all processes down as far as possible. This is not usually done as it requires a lot of CPU time.

Growing Processes: When a grows in memory from the initial assignment. Eg. Dynamically allocating memory. We can either allocate a little more memory when assigning it, swap it into another hole, or kill the process until some space has been freed up.

3.3 Memory Management Structures

Bitmap: A simple method to keep track of a fixed amount of memory. For each address, we can record `1` or `0` to denote if it is free or not. The downfall to this structure is that we have to search the bitmap for a specific length of free memory, which is a slow operation.

Linked List: Similar to a bitmap, however rather than recording each individual address, we can group consecutive addresses as a hole or process, record where it starts and the length of it.

3.4 Virtual memory

Virtual Memory: Allows a program to store pages onto physical memory, and the rest on virtual memory to be retrieved when needed.

Pages: A page is a contiguous range of addresses. Pages can be mapped onto physical memory, however not all pages need to be in the physical memory at the same time to run the program.

Virtual Address: Program generated addresses. Virtual addresses can be mapped to a physical memory address.

Page Frames: Where the virtual address has a page, the physical memory has page frames.

Memory Management Unit: Maps virtual addresses to physical addresses.

Page Fault: Occurs when a page is unmapped. The OS will stop the current instruction, picks a page frame that isn't frequently accessed and writes the contents back to disk, then replaces the frame with the page it requires.

Page Table: The virtual page number is used as an index for the page table to find the entry for that virtual page. From that entry we can find the physical page frame number (if it exists).

Translation Lookaside Buffer: Records the pages that are accessed frequently. Has a shorter access time as it stores the page frame directly, if the page is not in the TLB, then it will use the page table to compute the frame.

3.5 Page Replacement Algorithms

Optimal: Ideally we would know when the next page will be referenced, and just evict the page that will be referenced last. However its unlikely that we will know this in advance, so its only useful if we run the process multiple times.

Not Recently Used: Checks if a page has been referenced or modified recently, and removes a page at random from the lowest class (not referenced or modified).

FIFO: Gets rid of the least recent page arrival, when a new one is required. An issue that may occur is that the page that arrived first may be very useful and frequently accessed.

Second Chance Page Replacement: Similar to FIFO, however rather than just removing the last page, we check if it has been referenced or modified recently. If no, we can evict. If yes, we add it to the end of the queue so that it has another chance and reset the R bit. Once it gets to the front we can removed it if it hasn't been accessed recently. If all pages have been referenced then it becomes FIFO.

Clock page: Rather than moving pages around in a list, we can use a pointer (similar to a clock hand). If the page it is pointing to has been referenced, we can set the bit to 0 and move to the next page.

Working Set: Pages are loaded on demand rather than in advance. After pages are loaded, most of the pages that the instructions need should all be accessible if they exhibit a locality of reference. Generally most systems keep track of each process' working set and make sure it is in memory before letting the process run.

3.6 Security

Secure Communication: Has 3 main properties that we desire. Confidentiality, Integrity and Identification.

Confidentiality: Only the sender and the intended receiver should be able to understand the contents of a transmitted message. Generally we can encrypt messages so that an intercepted message cannot be understood by an interceptor.

Message Integrity: We want to ensure that the contents of a message is not altered (either maliciously or by accident) in transit.

End Point Authentication: Both the sender and receiver should be able to confirm the identity of the other party involved in the communication. Need to be able to verify that the person sending messages is the correct person.

Block Cipher: Splits plaintext into bits of a specified size (usually 64) then encrypts each block individually to prevent patterns.

Cipher Block Chaining: A random initialisation vector is used to generate the first block of cipher text. The output of this is then used as an input to the next block of plaintext to be encrypted and so on.

RSA: Example of public key cryptography.

Cryptographic Hash Functions: Should be computationally infeasible to have collisions, i.e different messages should have different outputs.

Message Authentication Code: Used to verify that the message hasn't been tampered with. Requires a shared secret between two parties, *the authentication key.* Alice can create a message concatenated with the auth key, then create a MAC by hashing that output. She then appends the MAC to the original message. Bob can calculate his own MAC with the sent message and their secret auth key, and verify that the MAC tag matches with the one sent.

Digital Signature: A cryptographic technique used to indicate that someone is the owner or the creator of an online document or message. This can be done by encrypting the message sent with a private key, and others can use the public key to verify the identity. I.e. Sign with private key, decrypt with public key. This also provides integrity, as the message itself is encrypted.

Public Key Certification: Certification Authorities can validate and issue certifications. There are various tiers of verification where the most basic is that you have control over a domain, and the highest is that a domain is attached to a physical entity.

SSL: Enhances TCP (Transmission Support Protocol) with confidentiality, data integrity, server authentication and client authentication.

1. **Handshake:** Establishes a connection between two parties, verifies identities, establishes the cryptographic method and SSL used, sends or generates a secret key.
2. **Data Exchange:** Uses a hash function to input data, and uses a session encryption key to encrypt. Also uses a MAC with a different key to add integrity.
3. **Connection Closure:** Uses a type field when sending over files through SSL to indicate if it will terminate the SSL session. If Alice receives a terminate before the type, then she would know if someone was interfering in the communication.