Goals for an operating system:

From the system point of view: allocate resources and manage processes

From the user point of view : allow user to get the maximum amount of work done, optimize performance

Definition of An Operating System: one program that runs all other programs, generally reffered to as the kernel

Types of Operating systems:

Batch Processing: is the execution of a series of jobs in a program on a computer without manual intervention (non-interactive)

Multiprogramming: multiprogramming keeps cpu utilization maximized by organizing jobs(code and data) so that the cpu always has one to execute

Time Sharing systems: cou constantly switches between different task, but the switches are so frequent that the user can interact with each program while its running(requires an interactive computer system)

Hardware Protection

Interrupts (Hardware and Software, Handling) : The occurrence of an event is usually signaled by an Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a system call

(also called a monitor call). Handler method is called and then requested interrupt is called, using a table of interrupt pointers.

Dual-mode operation At the very least, we need two separate *modes* of operation: user mode

and kernel mode (also called supervisor mode, system mode, or privileged

mode). A bit, called the mode bit, is added to the hardware of the computer

to indicate the current mode: kernel (0) or user (1). Privileged instructions(kernel instr) include io control timer management and interrupt

System Calls user specified interrupts that switch cpu to kernel mode to execute a specific instruction that can only be ran in kernel mode

I/O protection: to protect against the user making an io call that could be harmful to the system, all io calls are defined as privileged instructions (io calls can only be made through the os through syscall) device control registers are not accessible to the user either.

Memory protection: memory-addressing hardware ensures that a process can execute only within its own address space

Cpu protection: timer ensures that no process can gain control of the CPU without eventually

relinquishing control.

Operating System Structures

Operating System Components and Services : Process Management, Memory Management, I/O Device Management, File System ,Protection, Network Management, Network Services, User Interface

System Calls and System programs System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). In some sense, they are bundles of useful system calls.

System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process.

System Structure

Simple structure (MS-DOS): application programs are able to access the basic I/O routines

to write directly to the display and disk drives. Application program->resident system program -> ms dos device driver -> rom bios device drivers

Layered Structure: operating system is broken into a number of layers(levels). The bottom layer (layer 0) is the hardware; the highest (layer *N*) is the user interface. main advantage of the layered approach is simplicity of construction and debugging. layers are selected so that each uses functions (operations)and services of only lower-level layers Each layer adds overhead to the system call

Microkernels: removes all nonessential components form the kernel and implements them as system and user-level programs. Communication is provided through message passing

Module-based. Kernel has a set of core components and link in additional services via modules either at boot time or during run time. we might build CPU scheduling and memory management algorithms directly into the kernel and then add support for different file systems by way of loadable modules. any module can call any other module

Virtual Machines: dalvik virtual machine

Processes

Process definition: a program in execution needs certain resources—such as CPU time, memory, files, and I/O devices—to accomplish its task.

Process states and state diagrams: state of a process is defined in partby the current activity of that process(new running waiting ready terminated)

Process Control Block (PCB) pcb contains info about a process like process state, program counter, cpu registers, cpu scheduling info, memory management info

Context switch: the system

needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done context is represented by pcb we perform a state save of the current state of the CPU, be it in kernel or user mode, and then a state restore to resume operations.

Scheduling Queues short term scheduler picks processes in ready state and allocates the cpu to one of them. Long term scheduler stores proesses on mass storage disk and then slects from this pool and loads them into memory for execution. Medium term scheduler may remove processes from memory.

Process creation and Termination processes can create new processes which are called their children. These children also need resources like cpu time and memory which they can either get from the os or from their parent. Restricting the child to only using the parents resources can remove the issue of a parent forking to many child processes. Processes are terminated when they reach the last statement, then they make the syscall exit() and can return a status value to the OS. Generally a parent cant terminate without terminating the children. So cascading termination usually occurs, where if a process is terminated, its children are as well.

Cooperating processes : can affect or be affected by the other processes executing in the system.

Inter-process communication

Shared memory and message passing: two fundamental models of interprocess communication.

Shared memory: an area of shared memory shared by two cooperating programs is established. The programs then share info by reading and writing to the shared region of memory Message passing:

Is where the two cooperating process share data by sending messages to each other. Message passing is good since there will be no conflicting data since they don’t share a memory space, message passing also requires less space. Shared memory is good because it is fast since they aren’t implemented with syscalls like message passing. Message passing is easier to implement.

Producer/Consumer problem : A producer process produces information that is consumed by a consumer process. For example, a compiler may produce assembly code that is consumed by an assembler. Abuffer of shared memory is created to solve the produce consumer problem. The produces pushes data on to the buffer while the consumer pulls data from the buffer. Unbounded buffer: no size limit on the buffer, producer can always produce, but the consumer will have to wait to consume sometimes.

Bounded buffer: fixed size buffer, consumer waits if buffer is empty , producer waits if its full

Message passing, Sockets, RPC, RMI allow processes to communicate and to synchronize their actions without sharing the same address space. A socket is a type of message passing system where a server will listen in on a port and waits for incoming client requests

UNIX Processes

fork( ), exec( ), and wait( )

fork() creates a child process and starts running the code from the point where the fork is called, the pid of the child process is returned to the parent while the child sees itself of having a pid of 0. Exec() an [executable file](https://en.wikipedia.org/wiki/Executable_file) in the context of an already existing [process](https://en.wikipedia.org/wiki/Process_(computing)). Pid does not change but the machine code data heap and stack of the process are replaced by those of the new program. Wait() puts the process in the wait state so it is removed from the ready queue, can be placed back on the ready queue later.

UNIX pipes: acts as a conduit allowing two processes to communicate. Pipes wereone of the first IPC mechanisms in early UNIX systems 4 issues : bidirectional? Half(travel only one direction at a time) or full duplex(data can travel both directions same time)?does a relationship exist(parent- child)? Local or network communication? Ordinary pipes are unidirectional ie producer consumer type communication, force parent child relationship, terminates after processes terminate. Named pipes: doesn’t force a parent child relationship, bidirectional, don’t terminate upon process termination.

Threads

Definition. Processes vs. Threads: a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. code section, data section, and other operating-system resources, such as open files and signals A traditional (or *heavyweight*) process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.

Benefits: responsiveness(program can continue to run even if part is blocked or performing a lengthy operation. Resource sharing. Threads share memory all ready so no need for message passing like with processes. Economy: threads share resources with their parent process so they are a lot cheaper on time and resources compared to forking a new process. Scalability:

Threads can run on different cores, a non threaded process can only run on one core no matter what.

User threads: upported above the kernel andare managed without kernel support

Kernel threads: supported and managed directly by the operating system

Mapping: Many-to-One Model maps many user-level threads to one kernel thread process will block if a thread makes a blocking system call multiple threads are unable to run in parallel on multicore systems. One-to-One Model: maps each user thread to a kernel thread allowing another thread to run when a thread makes a blocking system call. also allows multiple threads to run in parallel on multiprocessors creating a user thread requires creating the corresponding kernel thread. Many-to-Many Model multiplexes many user-level threads to a smaller or equal number of kernel threads. two-level model similar to many to many model except it also also a thread to be bound to a kernel thread.

Implicit threading: Thread pools, OpenMP: transfer the creation and management of threading from application developers to compilers and run-time libraries. a thread pool create a number of threads at process startup and place them into a pool, where they sit and wait for work. Advantages are thread creation is limited, don’t have to waste time creating a new thread as they are all created at the beginning openmp parallel regions as blocks of code that may run in parallel. Application developers insert compiler directives into their code at parallel regions, and these directives instruct the OpenMP run-time library to execute the region in parallel. #pragma omp parallel it creates as many threads are there are processing cores in the system

Pthreads the POSIX standard defining an API for thread creation and synchronization. This is a ***specification*** for thread behavior, not an ***implementation***.

Thread cancellation involves terminating a thread before it has completed if multiple threads are concurrently searching through a database and one thread returns the result, the remaining threads might be canceled **Asynchronous cancellation**. One thread immediately terminates the target thread. **Deferred cancellation**. The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion.

Thread-Local Storage data that is unique to a thread, similar to local variables except theyre alive an accessible throughout the entire invocation of the thread rather than just a single function.

Process Synchronization:

race conditions: A situation like this, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access

takes place

Atomic operations: therefore provide special hardware instructions that allow us either to test and modify the content of a word or to swap the contents of two words **atomically**—that is, as one uninterruptible

Unit

Critical section problem and its solutions a portion of code in a program that modifies shared data or accesses a file that another cooperating process may try to access

Three requirements of a correct solution: the solution must be mutually exclusive, so the solution should only allow for one process to access their critical section at a time, progress: only processes not executing in their remainder section can participate deciding which will enter its critical section next, bounded waiting: processes are limited by the number of times they are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Peterson’s Algorithm no guarantee of functioning on modern architecture because of the way load and store functions work

synchronization hardware test and set() this instruction is that it is executed atomically Thus, if two test and set() instructions are executed simultaneously (each on a different CPU), they will be executed sequentially in some arbitrary order compare and swap() operates on three operands lways returns the original value of the

variable value

Mutex lock and busy wating: That is, a process must acquire the lock before entering a critical section; it releases the lock when it exits the critical section. **busy waiting**.While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the call to acquire().