Operating Systems

1 Introduction to Operating Systems

Operating Systems - A program that acts as an intermediary between a user and the hardware

Goals of an operating system:

- Execute user programs
- Make solving user problems easier
- Make the computer system convenient to use
- Use the resources of the system fairly and efficiently

An operating system is a:

- Resource allocator - Responsible for the management of the computer system resources
- Control program - Controls the execution of user programs and operation of I/O resources
- Kernel - The one program that runs all the time

1.1 Processes

Process - A unit of execution

The operating system is responsible for process management including:

- Process creation and deletion
- Process holding and resuming
- Mechanisms for process synchronization

A process includes:

- Code: text selection
- Current activity, represented by the program counter and the contents of the CPU's registers
- Data stack: temporary data such as local variables
- Data section: Global variables
- Heap: Memory allocated while the process is running

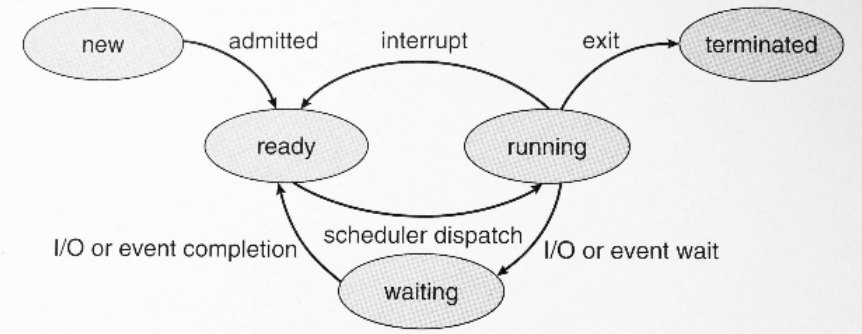
Information about the process is represented by a PCB including:

- Unique identifier
- State
- CPU utilization
- CPU scheduling
- Memory usage
- Other information

Process states:

- New - The process being created
- Running - Instructions being executed

- Waiting - Process waiting for some event to occur
- Ready - Process ready to be dispatched
- Terminated - Process completed execution



1.2 Process Creation

A new process, as a parent process, can create a number of child processes, which, in turn create other processes, forming a tree of processes

Resource sharing: three possible cases

- The parent and child processes share all resources
- The child process shares a subset of parent's resources
- The parent and child processes share no resources

Execution: two possible cases:

- The parent and child execute concurrently
- The parent waits until the child terminates

1.3 Process Termination

The process executes its last statement and asks the operating system to delete it:

- Outputs data from the child's process to parent
- The child process's resources are de-allocated by operating system

The parent process may terminate execution of child processes if:

- The child process has exceeded its allocated resources
- The task assigned to child is no longer required
- The parent is itself terminating (cascade termination)

1.4 The Kernel

Aims to provide an environment in which processes can exist.

Four essential components:

- Privileged instruction set
- Interrupt mechanism
- Memory protection

- Real-time clock

The kernel consists of:

- The first-level interrupt handler: to manage interrupts
- The dispatcher: to switch the CPU between processes
- Intra operating system communications

1.5 Interrupts

Interrupt - A signal from either hardware or software of an event that will cause a change of process, for example

- Hardware - Triggers an interrupt by sending a signal to the CPU
- Software - Triggers an interrupt by executing a system call for some action by the operating system

Interrupt Routines - OS routines that execute whenever an interrupt occurs

1.6 First Level Interrupt handler

The function of the FLIH is to:

- Determine the source of the interrupt (prioritise)
- Initiate servicing of the interrupt (selection of suitable process of the dispatcher)

1.7 Privileged instructions

Some instructions must be accessible only to the operating system: privileged instruction set

Privileged instructions include functions as:

- Managing interrupts
- Performing I/O
- Halting a process

1.8 Dual Mode

Aim: To distinguish between execution of an operating system code and user-defined code. We do not let the user execute instructions that would cause harm.
Two modes:

- User mode
- Kernel mode

Switching from user mode to kernel mode occurs when:

- A user process calls on the operating system to execute a function needing a privileged instruction (system call)
- An interrupt occurs (hardware)
- An error condition occurs in a user process (software)
- An attempt is made to execute a privileged instruction while in user mode

The dispatcher

- Assigns processing resource for processes
- Is later initiated when
  - A current process cannot continue
  - The CPU may be better used elsewhere, for instance:
    * After an interrupt changes a process state
    * After a system call which results in the current process not being able to continue
    * After an error which causes a process to suspend

2 Process Management

Independent process - Cannot affect or be affected by the execution of other processes
Co-operating processes - Can affect or be affected by the execution of another process

2.1 Multiprogramming

Advantages of multiprogramming:

- Computation speed-up - if there are multiple CPUs and/or cores
- Convenience - Single user wants to run many tasks
- Information sharing - For example shared files
- Modularity - Programming

The aim of multiprogramming is to maximise CPU utilisation
With multiprogramming several memory are kept in memory concurrently
Use CPU scheduling to fit them so the CPU is always in use

2.2 Schedulers

Long term scheduler - Selects processes to be brought into the ready queue
Medium term scheduler - Removes process from active contention for the CPU by swapping processes in and out of the ready queue
Short term scheduler - Selects the process to be executed next and allocated to the CPU

2.3 Scheduling Algorithms

There are a number of different algorithms including:

- First Come, First Served
  - Just put them in a queue as they turn up
- Shortest Job First
  - A priority queue with the length of the job as the priority
- Shortest Remaining Time First
  - A pre-emptive version of SJF, if a new process arrives with time less than the remaining time of the process currently running, then pre-empt it
- Priority (with or without pre-empting)
- Priority (with or without ageing)
- Round-Robin
  - Give each a time slice unless they finish early, rotate FCFS
  - If using priority, if two processes compete for the back of the queue, the higher priority goes first
- Multilevel queue/feedback queue

2.4 Priority Scheduling

Some algorithms use priority for scheduling

The problem with this is starvation - The low priority process may never execute
The solution to this is ageing - As time progresses increase the priority of the process

Multilevel queue scheduling - Processes are partitioned into different queues, which have different (performance) requirements

This has a problem that a process stays in the same queue regardless of adapting requirements
Feedback queues are used to solve this problem, allowing processes to proceed between queues, this is difficult to implement.

- The ready queue is partitioned into separate queues (for example foreground and background)
- Each queue has its own scheduling algorithm
- Scheduling must then be undertaken between queues:
  - Fixed priority scheduling
  - Time slice - each queue gets a certain amount of CPU time which it can schedule amongst its processes

2.5 Threads

Thread - A basic unit of CPU utilization
A thread shares some attributes with its peer threads (the same process) but may execute different code

The benefit of using threading includes the following:

- Responsiveness: The process continues running even if part of it is blocked or performing a lengthy operation
- Resource sharing: threads share the resources of their common process
- Economy: Allocating memory and resources for process creation is costly
- Multiprocessor architectures: a single threaded process can only run on one processor
- Performance: Cooperation of multiple threads in same job delivers higher throughput and improved performance

2.5.1 Multithreading models

Many to one:

- Many user application threads to one kernel thread
- Programmer controls the number of application threads

One to one:

- One user application to one kernel thread

Many-to-many

- N user application threads to $\leqslant N$ kernel threads
- Defined by the capabilities of the operating system

3 Memory Management: Main Memory and Virtual Memory

3.1 Types of Memory

Memory comes in many types:

- Cache memory/ CPU cache
- Main memory (e.g. RAM)
- Storage memory
- Virtual memory

3.2 Logical/Physical Addressing

Logical address - An address created by the CPU
Physical address - An address on the physical memory
Memory Management Unit (MMU) - Automatically translates virtual addresses to physical addresses A user (application) program deals with logical addresses and never knows the real physical addresses

3.3 Memory Partitioning

Main memory is usually split into two partitions:

- Kernel processes
- User processes

Each partition is contained in a single contiguous section of memory

3.4 Contiguous Allocation

Hole - Block of available memory
When a process arrives, it is allocated memory from a hole large enough to accommodate it

3.5 Memory Hole Allocation

There are a variety of strategies to satisfy a request from a list of free holes

- First fit - first hole big enough
- Best fit - smallest hole big enough
- Worst fit - largest hole

First fit is fast and best fit is efficient

3.6 Fragmentation

External fragmentation - Memory space able to satisfy a request but not contiguous
Compaction - Shuffle memory contents to place all free memory in one block
Internal fragmentation - Memory allocated successfully but may be slightly larger than the requested amount of memory (due to blocks being powers of 2)

3.7 Paging

Frames - Fixed size blocks of physical memory
Pages - Blocks of logical memory
Paging - Finding the number of free frames needed to load the program
Page table - Maps logical and physical addresses

3.8 Address translation scheme

The logical address generated by the CPU is divided into:

- Page number (p): used as an index into a page table which contains the base address of each page in physical memory
- Page offset (d): combined with base address to define the physical memory address that is sent to the memory unit

3.9 Protection

Memory protection implemented by associating protection bit with each frame

A valid-invalid bit is attached to each entry in the page table:

- Valid indicates that the associated page is in the process' logical address space, and is thus a legal page

3.10 Virtual memory

Virtual memory - The capability of the operating system that enable programs to address more memory locations than are actually provided in main memory

3.11 Demand paging

Bring a page into memory only when it is required by the process during its execution

3.12 A Process's Page Table

- When reference is made to a page's address
  - Invalid reference → abort
  - Not in memory → bring into memory
- A valid/invalid bit is associated with each process's page table entry to indicate if the page is already in memory
  - 1 → in memory
  - 0 → not in memory
- Address translation
  - When valid/invalid bit in page table entry is 0 → page fault trap
- Initially the valid/invalid bit is set to 0 on all entries

3.13 Handling Page Fault Traps

If there is a page fault trap

- Identify a free frame from the free frame list
- Read the page into the identified frame
- Update the process's page table
- Restart the instruction interrupted by the page fault trap

If there is no free frame then page replacement needs to be used
Page replacement - Find some page in memory, but not really in use, and swap it out

3.14 Page replacement algorithms

3.14.1 Belady's Anomaly

This is that for some page fault algorithms, the page fault rate my increase as the number of allocated frames increases

3.14.2 Optimal Algorithm

Replace the page that will not be needed for the longest time
This is impossible to implement as it would need you to know the future

3.14.3 Least Recently used

- Associate with each page the time of its last use
- Page to replace: Page that has not been used for the longest period of time
- This will never exhibit Belady's Anomaly as the pages of smaller frames are always a subset of pages for larger frames

This has an approximation with an additional reference bit:

- When a page is referenced set to 1
- Replace one of the pages with the bit set to 0
- Replace a random page when none are set to 0

Another approximation is the second chance algorithm:

- A circular FIFO queue
- When a page enters main memory it joins the tail of the queue and has reference bit set to 1
- When a victim page needs selecting
  - Start at the head of the queue
  - If the reference page is 0 select it as the victim page
  - If the page reference is 1, change it to 0 and move on to the next page

3.15 Allocation of frames

Global replacement - Select a replacement frame from the set of all frames, one process can take a frame from another
Local replacement - Select from only the process' own set of allocated frames
Fixed allocation - Give a set of frames at the start of the process
Priority allocation - Get the replacement frame from a process with a lower priority

3.16 Thrashing

This occurs when a process spends more time paging than doing actual work

3.17 Prepaging

Prepaging - Page into memory at one time all the pages that will be needed
This should prevent thrashing, but we need to evaluate the cost of prepaging

3.18 Page-Fault Frequency Scheme

- Establish an acceptable page fault rate
- If too low then take frames away
- If too high give frames

4 Mass-Storage Systems

Transfer rate - Rate of data flow between drive and computer
Positioning time - Time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
Head crash - Head makes contact with disk surface

Access Latency=Average access time=Average seek time + average latency
Average I/O time=average access time +(amount to transfer/transfer rate)+controller overhead

4.1 Disk scheduling algorithms

4.1.1 First come first served

Just move through the list

4.1.2 Shortest Seek Time First

Selects the request with the minimum seek time from the current head position May cause starvation of some requests

4.1.3 SCAN

Start at one end of the disk and move to the end of the disk servicing requests, then reverse and continue servicing

4.1.4 C-SCAN

Like SCAN, but doesn't service requests on the way back

4.1.5 C-LOOK

Arm only does as far as the last request in each direction

4.2 Disk management

Low level formatting - Dividing a disk into sectors that the disk controller can read and write
Partition - Dividing a disk into one or more groups of cylinders, each treated as a logical disk
Logical Formatting - Making a file system
Cluster - A group of blocks
Bootstrap loader - A program stored in boot blocks of boot partition

4.3 Swap-space management

Swap-space - Virtual memory uses disk space as an extension of main memory
Swap space can be carved out of the normal file system, but is usually in a separate partition

4.4 RAID

RAID - Using multiple disk drives to provide reliability via redundancy
Increases mean time to failure
Striping - Using a group of disks as one storage unit

5 File-System Interface

File - Contiguous logical address space

5.1 File locking

File locks can exist either shared for many processes (used when reading) or exclusive lock (used when writing) File locks can be:

- Mandatory - Access denied depending on locks held
- Advisory - Process find status of locks and decide what to do

6 File System Implementation

File control block - Storage structure consisting of information about a file
Device driver - Controls the physical device
Boot control block - Contains info needed by system to boot OS from that volume
Virtual File system - A way to provide an object oriented way of implementing file systems
Linear List - List of file names with pointer to the data blocks
Hash Table - Linear list with file data structure
Collisions - Situations where two file names hash to the same location
Contiguous Allocation - Each file occupies a set of contiguous blocks
Extent - Contiguous chunk of blocks
Indexed allocation - Each file has its own index block(s) of pointers to its data blocks