4. System Software

16.1 Purpose of an Operating System

Maximising Use of Resources

- *implements process scheduling – increases efficiency of CPU*
- *manages main memory*
- *optimises input/output operations – direct memory access (DMA) allows hardware to access main memory independently of the CPU, meaning the CPU is not utilised and free to carry out other tasks during data transfer*

User Interface

- hides the complexities of the computer/hardware/operating system from the user
- provides appropriate access systems for users with differing needs
- avoids complex commands involving memory locations or computer hardware
- e.g. a graphical user interface uses icons for navigation

Process Management

Multi-tasking

- Allows computers to (appear to) carry out more than one process at a time

  Implementation

- Processor time, hardware and resources are shared between tasks
- Scheduling is used to decide which process is carried out next - endures multi-tasking operates efficiently
- One task of a higher priority can interrupt another currently running task

Interrupt Handling

- transferring control to another routine when a service is required

Process States

- Ready – program is waiting to run on the CPU for allocated amount of time
- Running – program is currently running/being executed on the CPU
- Blocked – program execution is halted, waiting for an event to occur

Process Changes

- Running → Ready
    - ➔ The time slice of the running process expires
    - ➔ There is a process with a higher priority in the ready queue (the running process gets preempted)
    - ➔ An interrupt arrives at the CPU – the process running on the CPU gets preempted
- *Ready → Running*
    - ➔ *Process is selected by CPU to be executed – the OS scheduler has allocated this given time to the process*
- *Running → Blocked*
    - ➔ *Process needs to carry out an input/output operation or wait for a resource to be available*
    - ➔ *Process cannot proceed with execution before a specified event takes place*
- *Blocked → Ready*
    - ➔ *Event or input/output operation has occurred, program can resume*

Scheduling – managing the processes running on the CPU

- Allows more than one task to appear to be executed at the same time (enables multi-tasking)
- Allows high priority jobs to be completed first
- Keeps the CPU busy at all times – this ensures all processes execute efficiently and reduces wait times for all processes

Scheduling Routines

Shortest Job First

- Short processes are executed first and followed by longer processes (executed in ascending order of CPU time required)
- Leads to an increased throughput – as more processes can be executes in a smaller amount of time

Round Robin

- Each process is served by the CPU for a fixed time
- Starvation doesn't occur – as each process is given a fixed time to be executed every round robin cycle

First Come First Served

- **No complex logic – each process is executed one by one**
- **Received processes are queued**
- **Starvation doesn't occur – every process will eventually get a chance to run**

Shortest Remaining Time

- ***Processes are placed in queue (consists of a ready and blocked queue)***
- ***If a process with a shorter execution/burst time is queued, the process currently in line is preempted and the shorter process runs first***
- ***Processes will run until executed or a process with a shorter execution time is added***
- ***Starvation may occur – if processes with a shorter burst time keep queuing, then the longer processes may never run***

How the OS Kernel acts as an Interrupt Handler

- **Receives a signal when an interrupt is generated**
- **Checks priority of interrupt and reviews status of current interrupts**
- **Consults the interrupt dispatch table (IDT) - saves the contents of the registers on the kernel stack**
- **Restored the register contents once the interrupt is serviced**

Virtual Memory

- **is created temporarily**
- **secondary storage is used to simulate additional main memory**
- **extends RAM – means the CPU appears to be able to access more memory space than the actual RAM available**
- **only data in use needs to be in main memory – data can be swapped between RAM and virtual memory as necessary**

Reasons for Use

- **when RAM is running low – e.g. when a computer is running many processes at once**

- for more efficient use of RAM – if programs are not immediately needed, they can be moved from RAM to virtual memory

Paging – reading/writing blocks of data from/to secondary storage when required

- memory is divided into fixed size blocks
- dividing of memory into pages done by the operating system
- faster access times than segmentation

Segmentation

- In segmented memory, the virtual address space is broken into varying sized blocks (sections) - segment size is calculated by the compiler
- Each segment has a name, size and is numbered - its number is used as an index in the segment map table
- During execution segments from virtual memory are loaded into physical memory
- Address is specified by the user - contains the segment name and offset value
- An offset value determines the size of the segment
- Segment map table maps virtual addresses to physical addresses (contains segment number and offset value)
- Slower access times than paging

Page Replacement

- *'Page Fault' – an interrupt raised by hardware that occurs when a requested page is not yet loaded into main memory*
- *The OS replaces an existing page with a new one – done by swapping pages between secondary and main/primary memory*

FIFO Algorithm (First In, First Out)

- *OS keeps track of all pages in memory using a queue structure*
- *The oldest page is at the front of the queue and is the first to be removed when a new page is added*
- *Page usage is not considered during replacement*
- *Suffers from Belady's Anomaly – more page faults with an increasing number of page frames*

LRU Algorithm (Last Recently Used Replacement)

- *Replaces the page which has not been used for the longest time*

- ***Implemented using a linked list consisting of all pages in memory – most recently used page is at the front, least recently used at the back***

Clock Page Replacement

- ***Uses a circular queue structure – has one single pointer that acts as a head and tail***
- ***Has an R-flag which can be 0 or 1***
    - ***If R=0 – the page being pointed to is removed and a new page is inserted into its place , else a new page is inspected***
    - ***If R=1 – the next/following page is inspected, the pointer moving until a page where R=0 if found***

Disk Thrashing

- **Problem that occurs when virtual memory is being used (due to frequent transfers between virtual and physical memory)**
    - **As main memory fills up, more pages need to be swapped between virtual and physical memory**
    - **This swapping leads to a very high rate of hard disk access - results in excessive head movements**
    - **Latency increases as hard disk head movements take a relatively long time**
    - **Eventually, more time is spent swapping the pages/data than processing it - program may freeze or not run**

16.2 Translation Software

Interpreter

- **examines source code one statement at a time**
- **checks each statement for errors**
    - **...if no error is found, the statement is executed**
    - **...if an error is found, it is reported and the interpreter halts**
- **interpretation is repeated for every iteration in loops**
- **interpretation has to be repeated every time the program is run**

Compilation Stages

Lexical Analysis

- **converting a sequence of characters into a sequence of tokens**

Syntax Analysis

- uses parsing algorithms to interpret the meaning of a sequence of tokens
- checks code matches the grammar of the language
- syntax errors are reported
- a parse tree is produced

Code generation

- converting an intermediate representation of source code into an executable form

Optimisation

- minimising a program's execution time and memory requirement

Backus-Naur Form Notation

- Is a formal method of defining the grammatical rules of a programming language

  <variable_name> ::= *option1* | *option2* | *option3* ... ;

  - ➔ <> is used to enclose an item
  - ➔ ::= is basically a = or ← assigning options to a variable
  - ➔ | means OR – allows a variable to have multiple choices

Example

- To define a hexadecimal number, the following definitions can be made:

  <digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 ;
  <letter> ::= A | B | C | D | E | F ;
  <hexnum> ::= <digit> | <letter> | <hexnum> <digit> | <hexnum> <letter>

  - ➔ This states that only digits between 0-9 and letters from A-F are valid
  - ➔ Many BNF definitions are recursive, allowing the hexadecimal number to have multiple digits and letters

Reverse Polish Notation

- Used to carry out evaluation of expressions
  - ➔ Provides an unambiguous method of representing an expression
  - ➔ Reads from left to right
  - ➔ Doesn't require brackets or rules of precedence (BODMAS)

Data Structures used to evaluate in RPN

- Stack – operands are popped from stack in reverse order to how they were pushed
- Binary Tree – allows both infix and postfix to be evaluated (tree traversal)

Infix vs Postfix

- Infix refers to regular algebraic equations e.g. 3 + 4 evaluates to 7
- Postfix is used in RPN e.g. 3 4 + evaluates to 7
  - ➔ When writing postfix notation, the operator comes after the two numbers it is being performed on (2 4 * → 8 and 15 3 / → 5)
  - ➔ Brackets are not used – 5 3 + 7 2 - * evaluates to (5 + 3) * (7 - 2)
  - ➔ Numbers are grouped with the operator to their right – 8 - 2 3 * is 8 - 2*3

RPN evaluation using stacks

- Expression is read from left to right, one item at a time
- Each element is checked to see whether it is an operator or a value
- Values are pushed onto a stack until an operator is found
- Operator is then applied to the last two values on the stack
- Result is pushed back onto the stack
- Process repeats until only one value remains (the result)

Eg. a b * 2 / c d / * (where a = 20, b = 3, c = 10 and d = 5)

						5		
	3		2		10	10	2	
20	20	60	60	30	30	30	30	60