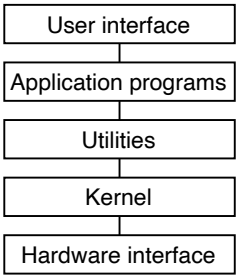
ALevel CS C20 System Software (1)

aspects relating to the use of OS:
1. the operating system programs are **stored on disk** so there is no operating system.the computer has stored in ROM a **basic input/output system (BIOS)** which starts a bootstrap program. It is this bootstrap program that loads the **operating system into memory** and sets it running.
2. An operating system can provide facilities to have more than one program stored in memory.
3. The internal viewpoint concerns how the activities of the operating system are organised to best use the resources available. The external viewpoint concerns the facilities made available for the user of the system.
4. The major resources associated with the internal viewpoint are the CPU, the memory and the I/O system.



User mode: one available for the user or an application program.
Kernel mode: sole access to part of the memory and to certain system functions that user mode cannot access.

How os can maximise the use of resources:
Memory
1. Moving frequently accessed instructions to cache for faster recall as SRAM is used rather than DRAM for cache
2. Making use of virtual memory with paging or segmentation to swap memory to and from a disk
3. Partitioning memory dividing main memory into static/dynamic partitions to allow for more than one program/task to be available //multiprogramming
4, Removing unused items/tasks from RAM by marking a partition as available as soon as the process using it has terminated

Disk:
1. Disk caching a disk cache holds data that is frequently transferred to/from the disk the cache can be held on disk or in RAM
2. Compression utility decreasing the size of a file stored on disk in order fit more / larger files on the disk
3. Defragmentation utility files are rearranged to occupy contiguous disk space this reduces the time taken to access files// decreases latency

Memory management aspects:
1. The provision of protected memory space for the OS kernel.
2. The loading of a program into memory requires defining the memory addresses for the program itself, for associated procedures and for the data required by the program.
3. The storage of processes in main memory can get fragmented in the same way as happens for files stored on a hard disk.
4. Decisions have to be made about how large a part of memory should be allocated to individual processes sharing memory.

Partition: different processes were loaded into memory simultaneously need to partition memory
Dynamic partition: allowed to adjust to match the process size
Segmentation: where a large process is divided into segments for loading into memory but the segments are **not constrained to be the same size**
factors of limited the efficiency by segmentation:
1. the segments were not constrained to be the same size.
2. the size of process did not allow all of the segments for one process to be in memory at the same time.
3. These two factors combined to cause fragmentation both of the memory and of the disk storage. This resulted in degradation in the performance of the system.

Paging: where a large process is divided into pages which have to be **the same size**
1. The process is divided into equal-sized pages
2. memory is divided into frames of the same size
3. The secondary storage can also be divided into frames.
4. A large program is likely to have optional routes for the execution.

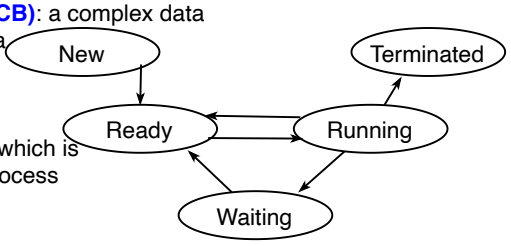
Virtual memory: a paging mechanism that allows a program to **use more memory addresses** than are available in main memory
An address has to comprise two parts: the **page number** plus the **offset** from the start of the page.
Page number: the page number **in logical memory**. The logic page number is **continuous**. The logical address stores the **page number in the four most significant bits** and the **page offset in the four least significant bits**.
Page frame number: the page number **in physical memory**. The page frames used **do not have to be adjacent**.
Presence flag: indicating whether or not the page is in memory.
Paging step:
1. the set of pages comprising a process are stored on disk.
2. One or more of these **pages is loaded into memory** when the process is changing to the ready state.
3. When the process is dispatched to the running state, the process starts executing.
4. the process will need access to a page that the page table indicates is **not in memory**. This is called a **page fault condition**.
5. in order to bring in the required page from secondary storage, **a page will need to be taken out of memory first.** use **first-in first-out method** or **least-recently-used method.**

Disk thrashing: when paging is being used and a repetitive state has been reached where loading one page causes a need for another page to be loaded almost immediately but the loading of this new page causes the same immediate need

Multitasking: allows computer to carry out multiple tasks at the same time.
1. Multitasking is actually multiple tasks processed at the same time, but actually processor can process one task at a time, so it has to swap between processes called scheduling.
2. Swapping happens so fast that it appears that all processes are running at the same time.
3. When there are too many processes, or some of them are making the CPU work especially hard, it can look as though some or all of them have stopped.
4. Multitasking doesn't mean that an unlimited number of tasks (process) can be juggled at the same time.
5. A process is a program that has started to be executed. A task that is to be executed or being executed by CPU is called process Each task consumes system storage and other resources. As more tasks are started, the system may slow down or begin to run out of shared storage.
6. Multitasking ensures the best use of computer resources by monitoring the state of each process.
7. The processes can be in running, ready or blocked state.
8. The Kernel of Operating system overlaps the execution of each process based on scheduling algorithm.

High-level scheduler: makes decisions about which program stored on disk should be moved into memory
Low-level scheduler: makes decisions about which process stored in memory should have access to the CPU
Process: a program in memory that has an associated process control block
Transitions between the status:
1. A new process arrives in memory and a **PCB is created**; it changes to the **ready state**.
2. A process in the ready state is **given access to the CPU** by the dispatcher; it changes to the **running state**.
3. A process in the running state is **halted by an interrupt**; it returns to the **ready state**.
4. A process in the running state cannot progress until **some event has occurred (I/O perhaps)**; it changes to the **waiting state** (sometimes called the 'suspended' or 'blocked' state).
5. A process in the waiting state is notified that **an event is completed**; it returns to the **ready state**.
6. A process in the running state **completes execution**; it changes to the **terminated state**.

Process control block (PCB): a complex data structure containing all data relevant to the running of a process
Thread
Thread: part of a program which is handled as an individual process when being executed
two reason for interrupts:



1. Processes consist of alternating periods of CPU usage and I/O usage. **I/O takes far too long for the CPU** to remain idle waiting for it to complete.
2. The **scheduler** decides to halt the process

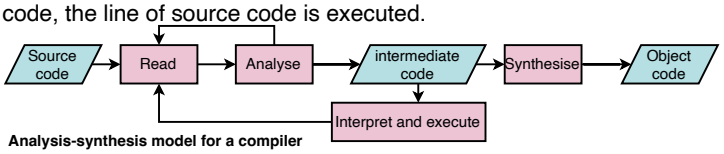
Scheduling method:
preemptive: can halt a process that would otherwise continue running undisturbed.
non-preemptive:
Shortest job firs:
FCFS(first come first served):
1. **non-preemptive** algorithm
2. can be implemented by placing the process in a **first-in first-out(FIFO) queue**.
3. very **inefficient**

Round-robin:
1. **allocates a time slice** to each process
2. **preemptive** algorithm.
3. A process will be halted when its time slice has run out.
4. it can be implemented as a **FIFO queue**.

Priority-based scheduling Algorithm:
1. every time a new process enters the ready queue or when a running process is halted, the priorities for the processes may have to be re-evaluated.
2. **possible criteria**:
   1. estimated time of process execution
   2. estimated remaining time for execution
   3. length of time already spent in the ready queue
   4. whether the process is I/O bound or CPU bound.

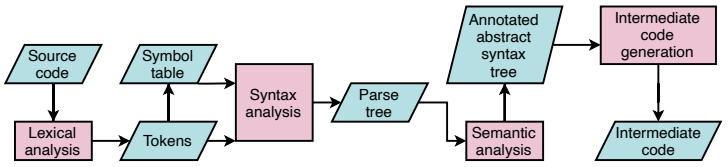
ALevel CS C20 System Software (2)

Backend program: takes this intermediate code as input and performs synthesis of object code.
Analyse -> Read: the source code is read line-by-line
intermediate code -> interpret and execute: once a line of source code has been found to be error free and therefore converted to intermediate code, the line of source code is executed.



Analysis-synthesis model for a compiler

Front-end program: performs analysis of the source code and unless errors are found produces an intermediate code that expresses completely the semantics (the meaning) of the source code.



Front-end analysis

Lexeme: Each meaningful individual character or collection of characters. A lexeme may be an identifier used by the programmer or may be a keyword, operator or symbol that is defined by the programming language.
lexical analysis: first to remove all white space and all comments then to take each line of source code and identify each lexeme.

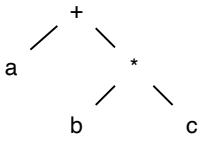
RPN:

1. An assignment statement often has an algebraic expression defining a new value for an identifier.
2. The expression can be evaluated by first converting the infix representation in the code to Reverse Polish Notation (RPN).

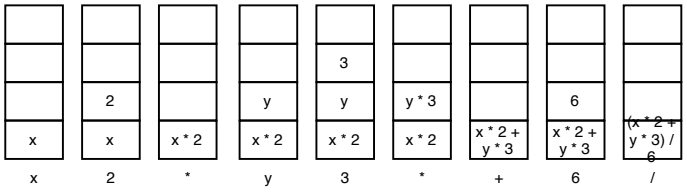
step:
1. Working from left to right in the expression
2. If element is a number PUSH that number onto the stack
3. If element is an operator then POP the first two numbers from stack
4. perform that operation on those numbers
5. PUSH result back onto stack
6. End once the last item in the expression has been dealt with

infix to RPN:
a + b * c --> a + b c * --> a b c * +

syntax tree



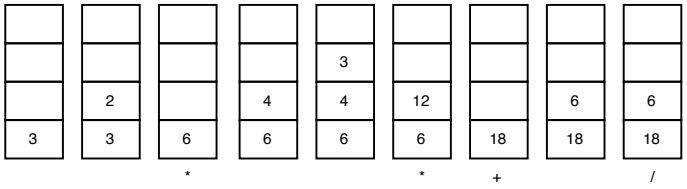
RPN to infix:
x 2 * y 3 * + 6 / --> (x * 2) y 3 * + 6 / --> (x * 2) (y * 3)+ 6 / -->
((x*2) + (y*3))6/ --> ((x*2) + (y*3)) / 6 --> (x * 2 + y * 3) / 6



(x * 2 + y * 3) / 6

Evaluating an RPN expression:

x 2 * y 3 * + 6 / (x = 3 ; y = 4)



Symbol table: a data structure in which each record contains the name and attributes of an identifier

1. For each identifier recognized there must be an entry made in the symbol table
2. contains identifier attributes such as the data type, where it is declared and where it is assigned a value.
3. The symbol table is an important data structure for a compiler.

```
Counter ← 0
INPUT Password
REPEAT
    IF Password <> "Cambridge"
        THEN
            INPUT Password
    ENDIF
    Counter ← Counter + 1
UNTIL Password = "Cambridge"
OUTPUT Counter
```

Keyword Token

Keyword	Token
←	01
+	02
+	02
IF	4A
THEN	4B
ENDIF	4C
ELSE	4D
REPEAT	4E
UNTIL	4F
TO	50
INPUT	51
OUTPUT	52
ENDFOR	53

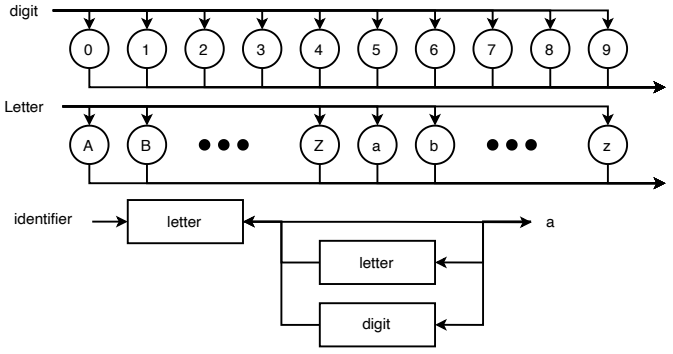
Symbol table

Symbol	Value(token)	Type(token)
Counter	60	Variable
0	61	Constant
Password	62	Variable
"Cambridge"	63	Constant
1	64	Constant

output from the lexical analysis

60	01	61	51	62	4E	4A	62
04	63	4B	51	62	4C	60	...

Syntax Diagram: defined the syntax allowed for an identifier



BNF(Naur Form notation): meta-language, used to describe the syntax and composition of statements which make up a high-level programming language.
1. programming languages
2. document formats
3. communication protocols

<Identifier> ::= <letter> | <identifier><letter>|<identifier><letter>
<Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Letter> ::= A | B | C | ... | Z | a | b | c | ... | z
<Letter> ::= <UpperCaseLetter>|<LowerCaseLetter>
<UpperCaseLetter> ::= A | B | C | ... | Z
<LowerCaseLetter> ::= a | b | c | ... | z
<binary digit> ::= 0 | 1

1. | is to separate individual options
2. ::= read as 'is defined as'
3. the recursive definition of <Identifier> in this particular version of BNF

Optimisation:
the main back-end stage is machine code generation from the intermediate code.
The aim of optimisation is to create an efficient program. One type of optimisation focuses on features that were inherent in the original source code and have been propagated into the intermediate code.

x := (a + b) * (a – b)
y := (a + 2 * b) * (a – b)

optimisation:
temp := (a – b)
x := (a + b) * temp
y := x + temp * b

1. Optimisation means that the code will have fewer instructions
2. Optimised code occupies less space in memory
3. Fewer instructions reduces the execution time of the program