CS303 SYSTEM SOFTWARE

MODULE I

System Software Vs. Application Software

Operating System is the **System Software** that makes the Computer work. We can say that an Operating System (OS) is Software that acts as an **interface between you and the hardware**. It not only contains drivers used to speak the hardware's language, but also offers you a very specific graphical user interface (GUI) to control the computer.

OS can also act as an interface (from the hardware) to the other software. A **complex OS like Windows or Linux or Mac OS offers the services of an OS**, but also has applications built in. **Solitaire, Paint, Messenger, etc. are all applications**.

Application software is the software that you install onto your Operating System. It consists of the programs that actually let you do things with your computer. These Applications are written to run under the various Operating Systems.

These include things like your word processing programs, spread sheets, email clients, web browser, games, etc. Many programs, such as most of the Microsoft Office suite of programs, are written in both Mac and Windows versions, but you still have to have the right version for your OS.

For example - Tally for Accounting, MS-Word for Word Processing etc.

So, the Operating system of a Computer is the Software that allows the Computer work. It provides the framework under which the Applications run. An operating system is the type of Computer system you have such as Window XP or Window 95, 98, Mac, etc.

The Applications are the Software that actually allows the user to do something with the Computer. Without the applications, all you can do is change settings and navigate among the folders. You can purchase its CD from a software company or download from a software company's web site.

Examples of System Software are - Operating Systems, Language Translators etc.

Different System Software– Assembler, Linker, Loader, Macro Processor, Text Editor,

Debugger, Device Driver, Compiler, Interpreter, Operating System

Assembler: A computer will not understand any program written in a language, other than its machine language. The programs written in other languages must be translated into the machine language. Such translation is performed with the help of software. A program which translates an assembly language program into a machine language program is called an assembler. If an assembler which runs on a computer and produces the machine codes for the same computer then it is called self assembler or resident assembler. If an assembler that runs on a computer and produces the machine codes for other computer then it is called Cross Assembler.

Assemblers are further divided into two types: One Pass Assembler and Two Pass Assembler. One pass assembler is the assembler which assigns the memory addresses to the variables and translates the source code into machine code in the first pass simultaneously. A Two Pass Assembler is the assembler which reads the

source code twice. In the first pass, it reads all the variables and assigns them memory addresses. In the second pass, it reads the source code and translates the code into object code.

Compiler: It is a program which translates a high level language program into a machine language program. A compiler is more intelligent than an assembler. It checks all kinds of limits, ranges, errors etc. But its program run time is more and occupies a larger part of the memory. It has slow speed. Because a compiler goes through the entire program and then translates the entire program into machine codes. If a compiler runs on a computer and produces the machine codes for the same computer then it is known as a self compiler or resident compiler. On the other hand, if a compiler runs on a computer and produces the machine codes for other computer then it is known as a cross compiler.

Interpreter: An interpreter is a program which translates statements of a program into machine code. It translates only one statement of the program at a time. It reads only one statement of program, translates it and executes it. Then it reads the next statement of the program again translates it and executes it. In this way it proceeds further till all the statements are translated and executed. On the other hand, a compiler goes through the entire program and then translates the entire program into machine codes. A compiler is 5 to 25 times faster than an interpreter.

By the compiler, the machine codes are saved permanently for future reference. On the other hand, the machine codes produced by interpreter are not saved. An interpreter is a small program as compared to compiler. It occupies less memory space, so it can be used in a smaller system which has limited memory space.

Linker: In high level languages, some built in header files or libraries are stored. These libraries are predefined and these contain basic functions which are essential for executing the program. These functions are linked to the libraries by a program called Linker. If linker does not find a library of a function then it informs to compiler and then compiler generates an error. The compiler automatically invokes the linker as the last step in compiling a program.

Not built in libraries, it also links the user defined functions to the user defined libraries. Usually a longer program is divided into smaller subprograms called modules. And these modules must be combined to execute the program. The process of combining the modules is done by the linker.

Loader: Loader is a program that loads machine codes of a program into the system memory. In Computing, a **loader** is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory. Once loading is complete, the operating system starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders. In many operating systems the loader is permanently resident in memory.

macro processor

A **general-purpose macro processor** or **general purpose preprocessor** is a macro processor that is not tied to or integrated with a particular language or piece of software.

A macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any kind of text.

Macro processors have been used for language expansion (defining new language constructs that can be expressed in terms of existing language components), for systematic text replacements that require decision making, and for text reformatting

Text Editors

A *text editor* is a tool that allows a user to create and revise *documents* in a computer. Though this task can be carried out in other modes, the word text editor commonly refers to the tool that does this interactively. Earlier computer documents used to be primarily plain text documents, but nowadays due to improved input-output mechanisms and file formats, a document frequently contains pictures along with texts whose appearance (script, size, colour and style) can be varied within the document. Apart from producing output of such wide variety, text editors today provide many advanced features of interactiveness and output.

Types of Text Editors

Depending on how editing is performed, and the type of output that can be generated, editors can be broadly classified as -

1. Line Editors - During original creation lines of text are recognised and delimited by end-of-line markers, and during subsequent revision, the line must be explicitly specified by line number or by some pattern context. eg. edlin editor in early MS-DOS systems.
2. Stream Editors - The idea here is similar to line editor, but the entire text is treated as a single stream of characters. Hence the location for revision cannot be specified using line numbers. Locations for revision are either specified by explicit positioning or by using pattern context. eg. sed in Unix/Linux.

   Line editors and stream editors are suitable for text-only documents.

3. Screen Editors - These allow the document to be viewed and operated upon as a two dimensional plane, of which a portion may be displayed at a time. Any portion may be specified for display and location for revision can be specified anywhere within the displayed portion. eg. vi, emacs, etc.

4. Word Processors - Provides additional features to basic screen editors. Usually support non-textual contents and choice of fonts, style, etc.
5. Structure Editors - These are editors for specific types of documents, so that the editor recognises the structure/syntax of the document being prepared and helps in maintaining that structure/syntax.

Debugger

Debugging means locating (and then removing) *bugs*, i.e., faults, in programs. In the entire process of program development errors may occur at various stages and efforts to detect and remove them may also be made at various stages. However, the word debugging is usually in context of errors that manifest while running the program during testing or during actual use. The most common steps taken in debugging are to examine the flow of control during execution of the program, examine values of variables at different points in the program, examine the values of parameters passed to functions and values returned by the functions, examine the function call sequence, etc. In the absence of other mechanisms, one usually inserts statements in the program at various carefully chosen points, that prints values of significant variables or parameters, or some message that indicates the flow of control (or function call sequence). When such a modified version of the program is run, the information output by the extra statements gives clue to the errors.

Using *print* statements for debugging a program is often not adequate or convenient. For example, the programmer may want to change the values of certain variables (or parameters) after observing the execution of the program till some point. For a large program it may be difficult to go back to the source program, make the necessary changes (maybe temporarily) and rerun the program. Again, if such *print*statements are placed inside loops, it will produce output everytime the loop is executed though the programmer may be interested in only certain iterations of the loop. To overcome several such drawbacks of

debugging by inserting extra statements in the program, there are a kind of tool called *debugger* that helps in debugging programs by giving the programmer some control over the execution of the program and some means of examining and modifying different program variables during runtime.

Device Drivers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

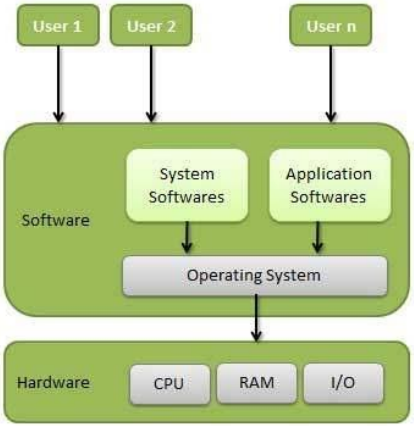
A device driver performs the following jobs −

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Operating System

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

SIC & SIC/XE Architecture, SIC & SIC/XE Architecture, Addressing modes, SIC & SIC/XE Instruction set, Assembler Directives and Programming.

http://solomon.ipv6.club.tw/Course/SP/sp1-1.pdf

Instruction Formats – SIC

- 24 bit

8 bit opcode	1 bit addressing mode	15 bit address

- Load and Store Registers
  - LDA, LDX, STA, STX, etc.

- Integer Arithmetic (all involve register A)
  - Add, SUB, MUL, DIV
- Compare
  - COMP – compares A with a word in memory
  - Sets the CC in the SW
- Jump instructions
  - JLT, JEQ, JGT – based on the CC as set by COMP
- Subroutine Linkage
  - JSUB – jumps to subroutine, places return address in L
  - RSUB – returns, using the address in L

Input/Output - SIC

- TD – test device is ready to send/receive data
  - CC of < means device is ready
  - CC of = means device is not ready
- RD – read data, when the device is ready
- WD – write data
- Transfers 1 byte at a time to or from the rightmost 8 bits of register A.
- Each device has a unique 8-bit code as an operand.

Addressing Modes – SIC

- Direct x = 0 Target address = address
- Indexed x = 1 Target address + (X)

(X) is the contents of register X

MODULE II

Assemblers

Basic Functions of Assembler. Assembler output format – Header, Text and End Records- Assembler data structures, Two pass assembler algorithm, Hand assembly of SIC/XE program, Machine dependent assembler features.


Assemblers.pptx


CH1H.ppt