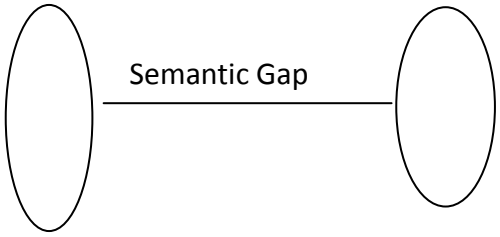
UNIT – 1

Introduction to Language Processor

UNIT -01/Lecture 01

What is the need of language processing activities that take place in computer system?

Ans. Language processing activities arise due to the differences between the manner in which a software designer describes the ideas concerning the behavior of the software and the way it is actually implemented inside the computer system. The designer expresses these ideas in the application domain. These ideas are actually implemented in the execution domain. The gap between the application domain and execution domain is called semantic gap.



What are the bad/worse effects if semantic gap is covered by programmer?

Ans. Since the semantic gap is to be covered by the programmer, it means that the designer has to specify his/her ideas directly in the machine language of the computer. This obviously has three bad/worse effects as far as software design is concerned:

- Large development time
- Large development effort
- Poor quality and less reliable software

How do bad effects of large semantic gap tackled?

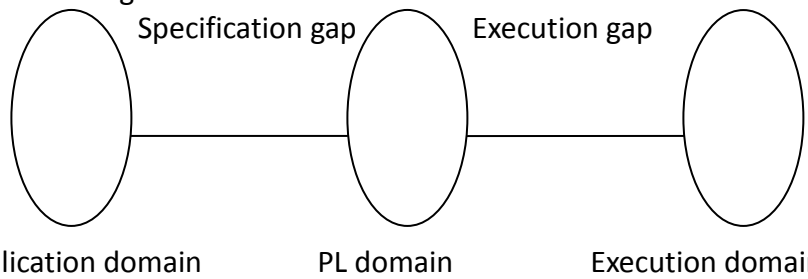
Ans. These issues are tackled by well defined steps of software engineering and the development of programming languages (PL). The software engineering aimed at the use of PL can be grouped into

- Specification, design and coding steps.
- PL implementation steps

Software implementation using PL introduces new domain, the PL domain between application and execution domain. The semantic gap is now split into:

1. Specification gap between the application domain and the PL domain.
2. Execution gap between the PL domain and execution domain.

This is shown in figure below:



The programme is now responsible only to translate his specifications in PL language.

Therefore he has to bridge a very small specification gap as compared to the large semantic gap between application and execution domain. He need not worry about the direct translation of his specifications in machine language. The introduction of PL domain provides two advantages to the programmer:

- Since the programme is now responsible only to bridge the specification gap between application domain and PL domain, this reduces the severe consequences of large semantic gap that existed between application and execution domain prior to the advent of programming languages.
- The language processor provides the diagnostic capability which detects and indicates errors in the input. This helps in improving the quality of software.

What is the specification language of application domain, PL domain and execution domain?

Ans. The specification language for the application domain is some specification language of the user (Structured English, data flow diagrams, flow-charts, decision tables and decision tree). The specification language of PL domain is the programming language itself. The specification language of execution domain is the machine language of the computer system. We will use the term execution gap if one of the two specification languages is machine language. In all other cases we will introduced the term specification gap.

What do you understand by the term language processor? Explain its four versions.

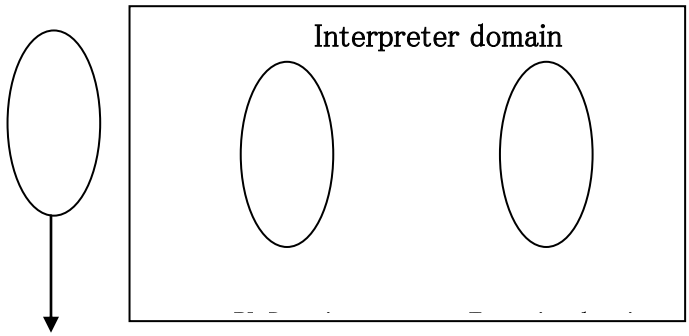
Ans. Language processor is software that bridges the specification gap or execution gap. The input to the language processor is the source program while the output of the language processor is the target program. The languages in which these programs are written are known as source language and target language respectively. The language processor has four important versions:

1. Language translator: It bridges the execution gap to machine language of the computer system. An assembler is a language translator whose source language is an assembly language. The compiler is a language translator that is not an assembler (source language is high-level language instead of assembly language). In both cases, the target language is machine level language.
2. De-translator: It bridges the same execution gap as the language translator but in reverse direction.
3. Pre-processor: It is a language processor that bridges the execution gap but it is not the language translator.
4. Language migratory: It bridges the specification gap between two programming languages.

Explain interpreter in details.

Ans. An interpreter is a language processor which bridges an execution gap without generating a machine language program. An interpreter is a language translator that does not have any target language/target program.

The absence of target program implies an absence of output interface for an interpreter. Thus the language processing activities of an interpreter cannot be separated from its program execution activities. Hence we can say that interpreter 'executes' the program written in a programming language. Thus execution gap vanishes completely.



Application domain

The figure above shows the schematic representation of an interpreter, wherein the interpreter domain encompasses PL domain and execution domain. Thus the specification language of an interpreter domain is the specification of PL domain. Since an interpreter also incorporates an execution domain, it is as if we have a computer that is capable of understanding the programming language.

What do you understand by the term language processing activities? Illustrate.

Ans. The fundamental language processing activities are those that bridges:

****Specification gap, ****Execution gap

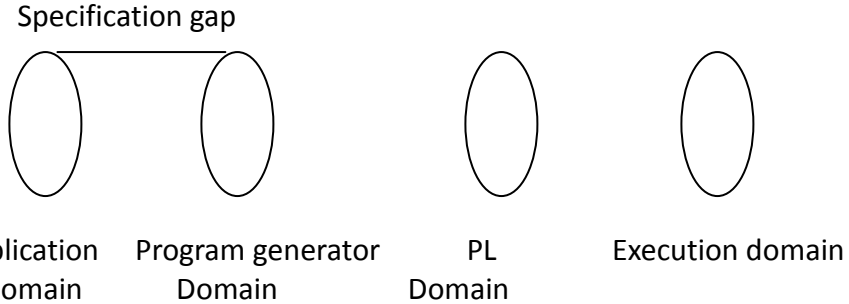
Accordingly we have two activities of the language processors:

1. Program generation activities
2. Program execution activities.

The program generation activity aims at an automatic generation of a program. The source language here is the specification language of application domain while the target language is typically the procedure oriented programming language. The program execution activity organizes the execution of the program written in the programming language on a computer system. The source language here is procedure or problem oriented language.

What do you understand by the term program generation activity? What is program generator domain?

Ans. The program generation activity is a language processing activity that aims at automatic generation of a program. The program generator is software that accepts the specification of the program and generates the target program in some target PL. The program generator introduces new domain between the application domain and the PL domain. The specification gap is now the gap between the application domain and the program generator domain. This gap is smaller as compared to the specification gap between the application domain and PL domain.



S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What do you understand by the term language processor	DEC 2009	

Unit-01/Lecture-02

What are the advantages of introducing program generator domain between application and PL domain? Discuss.

Ans. There are following advantages of introducing program generator domain between application and PL domain:

1. Since the programmer needs to give the specifications directly to the computer, rather than coding them in Pl, the specification gap now between the application domain and the program generator domain is very small as compared to the specification gap between application domain and the PL domain earlier. The harder task of bridging the gap to the target programming language is now the responsibility of the program generator.

2. The above arrangement also reduces the testing effort because proving the correctness of the program amounts to proving the correctness of the specifications that are input to program generator. This is much simpler than verifying the correctness of the generated program. This task can be further simplified by providing good diagnostic capability (i.e error generation)in the program generator which would detect inconsistencies in specifications.

Why is it more economical to develop program generator domain as compared to development of problem oriented language?

Ans. This is because the problem oriented language suffers a large execution gap between PL domain and execution domain whereas the program generator has a small semantic gap to the target PL domain which is the domain of the target procedure oriented programming language. The execution gap between the target programming language domain and the execution domain is bridged by the compiler or interpreter for the PL.

Discuss the program execution activities in details.

Ans. The program execution activities aim at execution of a program. There are two program execution activities:

Program translation: The program translation model bridges the execution gap between the source program written in source PL language into target program written in machine or assembly language. There are following characteristic features of the program translation model:

1. The program must be translated before it is executed.
2. The translated program may be saved in a file. The saved program may be executed repeatedly.
3. The program may be translated following modifications.

The machine level language program is executed by the CPU through the program counter (PC) in following steps:

a. Fetch an instruction and PC incremented to the address of next instruction.
b. Decode the instruction.
c. Execute the instruction.

Now the control returns to the next fetch phase. This is an instruction execution cycle.

Program interpretation:

Interpreter reads the source program statement by statement manner. It then

determines the meaning of the statement and performs actions that implement its meaning. This includes computational and input/output actions. The program is interpreted statement by statement basis through interpretation cycle that has following steps:

a. Fetch the source statement
b. Analyze the statement and determine its meaning (operands and computation to be performed)
c. Execute the meaning of the statement.

Finally we identify two important characteristics of interpretation:

a. The source program is retained in source form only. There is no target program generated.
b. A statement is analyzed during its interpretation.

Compare translator and interpreter.

Ans. Following are the differences between translator (assembler/compiler) and the interpreter:

Translator	Interpreter
1. A fixed cost is incurred in translation model. This is because the target machine level language program so generated incurs an effort by translator and occupies memory	1. No translation overhead is incurred in the interpretation model because the source program is retained in source form itself and target code is not generated. This is advantageous if a program is modified between executions as in program testing and debugging.
2. Translation model is fast as compared to interpretation model because now the target machine language program is directly executed and there is no need to refer to source program	2. Since the source program is retained in source form, analyzing the meaning of the source statement is quiet slow as compared to the direct execution of machine language program.

What do you understand by the term language processing? Illustrate.

Answer. Language processing = Analysis of source program + Synthesis of target program. The collection of language processor components engaged in the analysis of source program constitutes analysis phase of a language processor. Components engaged in synthesizing a target program constitutes synthesis phase of a language processor. The specification of source language forms the basis of source program analysis. The specification consists of three components.

1. Lexical rules: These govern the formation of valid lexical units in the source program.
2. Syntax rules: These govern the formation of valid statements in source language.
3. Semantic rules: These associate meaning with valid statements of the language.

The analysis phase uses each component of source language specification to determine the relevant information concerning the statement in the source program. Thus the analysis of source program consists of lexical, syntax and semantic analysis.

The synthesis phase is concerned with the construction of target language statements which have same meaning as source statements. This consists of two main activities:

- Creation of data structures in the target program.
- Generation of target code.

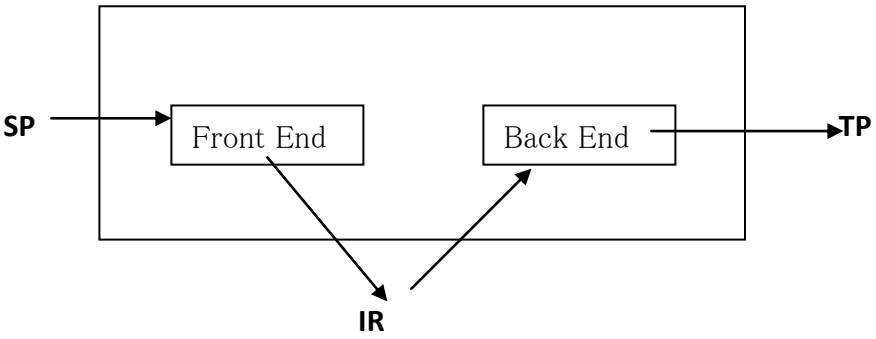
What is language processing pass? Illustrate.

Ans. Language processing pass is the processing of every statement in a source program, or its equivalent representation, to perform a set of language processing function. For example, for a two pass translator, following activities are carried out in two passes:

- Pass 1: It performs analysis of source program and note relevant information.
- Pass 2: Performs synthesis of target program.

What is an intermediate representation of source program? What is its use? What are its desirable properties?

Ans. Intermediate representation of source program is that representation of source program that reflects the effects of some, but not all analysis and synthesis tasks performed during language processing. In this case, first pass performs analysis of source program and reflects its effects in intermediate representation. The second pass reads and analyses the IR, instead of source program, to synthesize target program. This avoids repeated processing of source program. The first pass is concerned exclusively with the source language issues. Hence it is called the front end of language processor. The second pass is concerned with program synthesis for a specific target language. Hence it is called back end of a language processor. Note that front and back end need not coexist in memory. This reduces the memory requirements of a language processor.



Desirable properties of language processor:

- Ease of use: IR should be easy to construct and analyze.
- Processing efficiency: Efficient algorithms must exists for constructing and analyzing IR.
- Memory efficiency: IR must be compact.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Short notes on compiler, interpreter, assembler	Dec 2013 Dec 2012	7 7

Unit-01/Lecture-03

What are semantic actions?

Ans. All actions performed by the front end except the lexical and syntax analysis are called semantic actions. These include actions for the following:

- Checking semantic validity of constructs in SP.
- Determining the meaning of SP.
- Constructing an IR.

Discuss the functions of front end of language processor for a toy compiler.

Ans. The front end performs lexical, syntax and semantic analysis of the source program. Each kind of analysis involves the following functions:

1. Determine the validity of source statement from the viewpoint of analysis.
2. Determine the contents of source statements.
3. Construct a suitable representation of source statement for use by subsequent analysis functions, or by the synthesis phase of the language processor.

The word 'contents' has different meanings for three types of analysis of the source statement. In lexical analysis, 'contents' means the lexical class to which each lexical unit belongs. For syntax analysis, 'contents' means the syntactic structure of source statement. For semantic analysis, 'contents' denote the meaning of the statement-for declaration statement, it is the set of attributes for the declared variables, while for imperative statement, it is the sequence of actions implied by the statement.

Each analysis represents the 'contents' of source statement in the form of

1. Tables of information.
2. Description of the source statement.

Subsequent analysis uses this information for its own purpose and adds information to these tables or descriptions, or constructs its own tables or descriptions. For example, the syntax analysis uses information concerning the lexical class of lexical units and constructs a representation for the syntactic structure of source statement. Semantic analysis uses information concerning syntactic structure and constructs the meaning of the statement. The tables and descriptions at the end of semantic analysis form the intermediate representation (IR) of the front end.

Discuss the outputs of the front end of language processor.

Ans. The IR produced by the front end consists of following two components:

1. Tables of information: Most important table of information constructed by front end is the symbol table which contains information concerning all the identifiers used in the source program. This table is constructed during the lexical analysis. Semantic analysis adds information concerning symbol attributes while processing declaration statements. It may also adds new names designating temporary results.
2. Intermediate code (IC): IC is a sequence of IC units. Each IC represents the meaning of one action in SP. IC may also contain references to the information in various tables.

What are macros and macro processors? Explain in brief.

A macro is similar to a subroutine (or a procedure), but there are important Differences between them. A subroutine is a section of the program that is written Once, and can be used many times by simply calling it from any point in the program. Similarly, a macro is a section of code that the programmer writes (defines) once, and then can use many times. The main difference between a subroutine and a Macro is that the former is stored in memory once (just one copy), whereas the Latter is duplicated as many times as necessary. Macros involve two separate phases. Handling the definition and Handling the expansions. A macro can only be defined once but it can be expanded

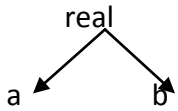
many times. Handling the definition is a relatively simple process. The assembler reads the Definition from the source files and saves it in a special table, the Macro Definition Table (MDT).The assembler does not try to check the definition for errors, to assemble it, execute it, or do anything else with it.

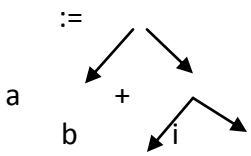
A macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so. Macro processors are often embedded in other programs, such as assemblers and compilers. Sometimes they are standalone programs that can be used to process any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

Discuss lexical analysis (scanning), syntax analysis (parsing) and semantic analysis?
Ans. Lexical analysis (Scanning) Lexical analysis identifies the lexical units in a source statement. It then classifies the units into different lexical classes, e,g, constants, reserved id's etc. Each identifier during this phase is described in terms of token for that identifier. The token for that identifier take the following form (class code, number in the class). Code refer to id for identifier, C for constant and Op for operator and number in the class is the entry number of the identifier in the relevant table. For example, the statement a:=b+i; after the lexical analysis is written as:(id #2) (op #5) (id #3) (op #2) (id #1) (op #10).

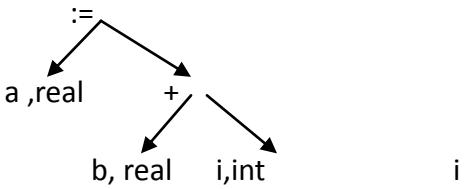
Syntax Analysis (Parsing): Syntax analysis processes the string of tokens, build by lexical analysis to determine the statement class, e.g assignment statement, if statement etc. It then builds an IC which represents the structure of the statement. The IC is passed to semantic analysis to determine the meaning of the statement.
Consider the above example: We can draw the IC tree for **a,b:real** and a:=b+i; as follows:



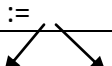


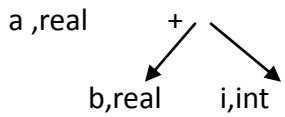
Semantic analysis: Semantic analysis of declaration statement adds information to the symbol table about the variables (i.e type length and dimensionality). Semantic analysis of imperative statement identifies the sequence of actions necessary to implement the meaning of the source statement. In both cases the structure of source statement guides the application of the semantic rules.
As per above example, we will here demonstrate the steps to carry out semantic analysis of a:=b+i;

1. Information regarding the type of operands is added to IC tree. IC tree now looks as under.

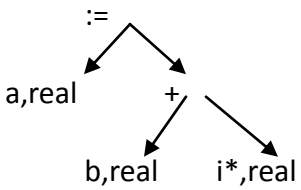


2. Rules of meaning governing an assignment statement indicate that the expression on the right hand side should be evaluated first. Hence the focus shifts to the right subtree rooted at +.

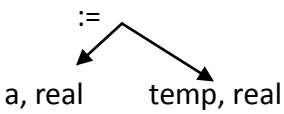




3. Rules of addition says that type conversion of i should be performed to ensure the type compatibility of the operands. This leads to the action
   A1. Convert i to real giving i*.
   The IC tree now becomes



4. Rules of addition indicate that addition is now feasible. This leads to the action.
   A2. Add i* to b giving temp.
   IC tree now becomes:



5. The assignment can now be performed. This leads to the action
   A3. Store temp in a.
   The analysis of the statement is complete after step 5.

Discuss the functions of the back end for the compiler

Ans. The back end performs memory allocation and code generation.

Memory allocation: At this stage the memory requirements of an identifier is computed from the type, length and dimensionality. Accordingly, the memory is allocated and address filled in the address field. At this particular point of time a decision whether or not to allocate the memory to temporary variables is taken. In our particular example, the memory will not be allocated to newly added variables i* and temp. The other variables are allocated the memory and address is filled in address field.

S.No.	Symbol	Type	Length	Address
1.	I	int	2	2000
2.	A	real	4	2002
3.	B	real	4	2006

Code generation: The code generation uses the knowledge of target architecture to exploit its instruction sets and addressing modes. The important issues in code generation are:

- Determine the places where the intermediate results (whether in memory or registers) are to be placed.
- Determine the instructions to be used for type conversion operations.
- Determine the addressing modes to be used.

For our example for the assignment statement a:-b+i;

Following actions were taken

1. Convert i to real giving i*.
2. Add i* to b giving temp
3. Store temp in a.

The synthesis phase may decide to hold i* and temp in machine registers and may generate the following assembly code.

```
ADD_R    AREG,  B
MOVEM    AREG, A
```

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Short notes on Macros and Macro Processor	Dec 2009 June 2010	3 4
Q.2	Explain Macro processor & its features	Dec 2013	7

Unit-01/Lecture-04

Static and dynamic binding:

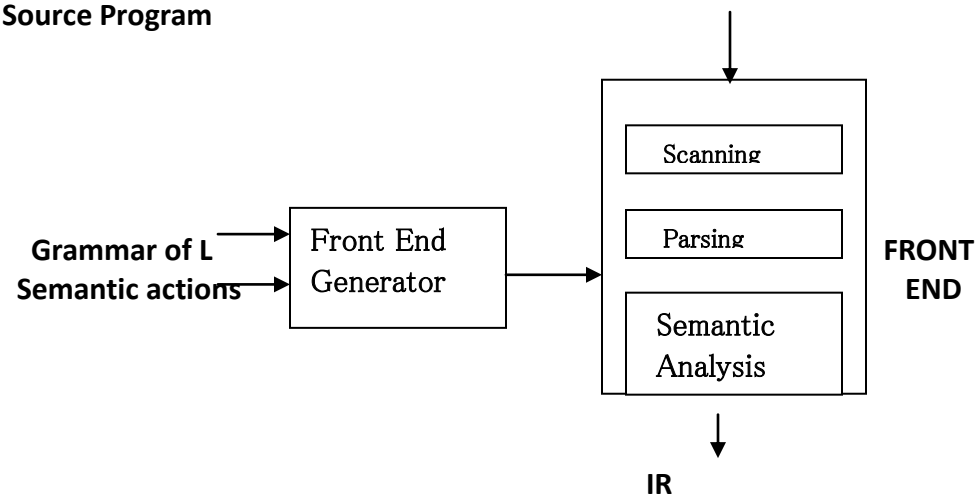
Static binding: The binding performed before the execution of program begins is called static binding.
Dynamic binding: The binding performed after the execution of program has started is called late binding.

Static binding provides greater execution efficiency while dynamic binding provides greater flexibility in program writing.

Write note on language processor development tools.

Ans. **Need for language processor development tools**: The analysis phase of the language processor has a standard form. The source text is subjected to lexical, syntax and semantic analysis. The result of the analysis phase is IR. Thus writing of language processors has become very important. This has lead to the development of a set of language processor development tools (LPDT).

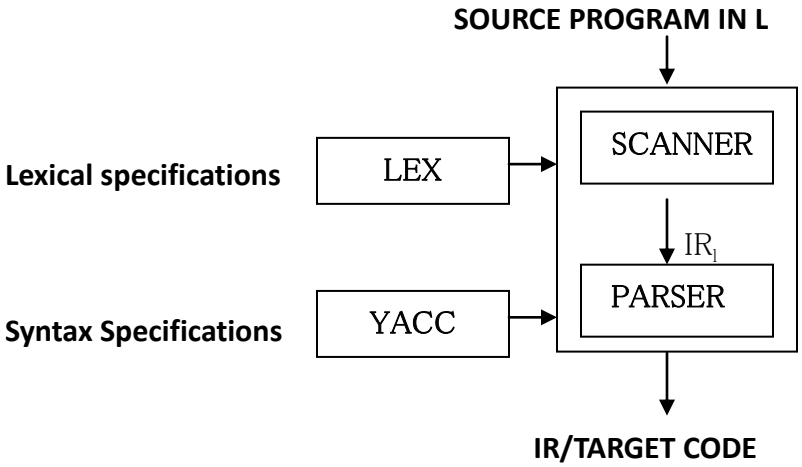
Schematic of LPDT: The figure below shows a schematic of language processor development tool that generates the analysis phase of language processor whose source language is L.



The language processor development tool requires two set of inputs:

1. Specification of grammar of a language L.
2. Specification of semantic actions to be performed in the analysis phase.

Introduction to LEX and YACC: There are two language processor development tools namely LEX and YACC. LEX is the lexical analyzer that performs lexical analysis of SP written in a programming language L. YACC is the parser generator that constructs the syntactic structure of the source statement. The input to these tools is the specification of lexical and syntactic constructs of L, and semantic actions to be performed on recognizing the constructs. The specification consists of a set of translation rules of the form <string specification> {<semantic action>}. Here semantic action> consist of C code that is executed when a string matching <string specification> is encountered in the input. The figure below shows the schematic for the development of analysis phase of a compiler for language L using LEX and YACC.



The analysis phase processes the source program to build an intermediate representation. A single pass compiler can be built using LEX and YACC if the semantic actions are aimed at generating target code instead of IR. Scanner also generates an intermediate representation of a source program for use by the parser. We call it IR_I to differentiate from the IR of analysis phase.

LEX: LEX accepts an input specification which consists of two components. The first component is the specification of strings representing lexical units in L. e.g, id, constants etc. The second component is the semantic actions aimed at building the IR. As already seen, the IR consists of tables of information (symbol table) and a sequence of tokens representing source statement. Accordingly semantic actions make new entries in the tables and build tokens for the lexical units.

Example: Following is a sample input to LEX.

```
%{
Letter                    [A-Z a-z]
Digit                     [0-9]
}%

%%
Begin                      {return(BEGIN);}
 End                       {return(END);}
 ":="                      {return(ASGOP);}
{letter} ({letter}|{digit})*    {yylval=enter_id(); return(ID);}
{digit}+                  {yylval=enter_num(); return(NUM);}
%%
enter_id()
{/* enters the id in symbol table and returns entry number */}
enter_num()
{
  /* enters the number in the constants table and return entry number */
}
```

The sample input consists of following components:

1. The first component between %{ and }% defines the symbols in specifying the strings of L. It defines the symbol 'Letter' for any uppercase or lowercase letter and 'Digit' to stand for any digit.
2. The second component between %% and %% are the translation rules.
3. The third component contains auxiliary routines which can be used in semantic actions.

YACC: Each string specification in the input to YACC resembles a grammar production. The

parser generated by YACC performs reductions according to this grammar. The actions associated with the string specification are executed when a reduction is made according to the specification. An attribute is associated with every non-terminal symbol. The value of this attribute can be manipulated during parsing. The attribute can be given any user-designed structure. The symbol '$n' in the action part of a translation rule refers to n^{th} symbol in the RHS of the string specification. '$$' represent the attribute of the LHS symbol of the string specification.

Example: Here is the sample input to the YACC.

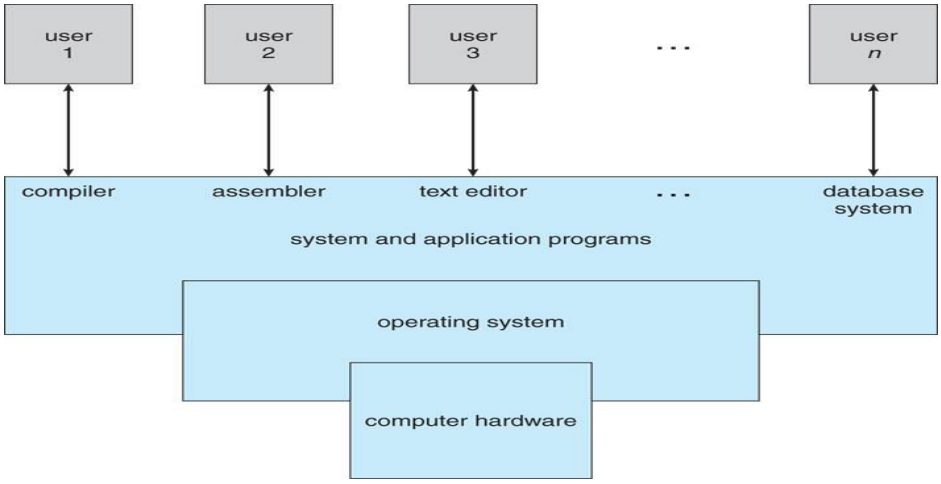
```
%%
E : E + T   {$$ = gencode('+', $1, $3);}
   | T       {$$ = $1;}
   ;
T : T * V  {$$ = gencode('*', $1, $3);}
   | V       {$$ = $1;}
  ;
V : id       { $$ = gendesc($1);}
 ;
%%
gencode(operator, operand_1, operand_2)
{ /* Generates code using operand descriptors. Returns descriptor for result */

gendesc(symbol)
{  /* Refer to symbol/constant table entry. Build and return descriptor for the symbol */}
```

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Write note on language processor development tools	Dec 2012	7
Q.2	Short notes on Software Tools	June 2010	4

Unit-01/Lecture-05

Introduction to Operating System

What Operating Systems Do - For Users, For Applications, etc.



Abstract view of the components of a computer system

- Computer = HW + OS + Apps + Users
- OS serves as interface between HW and (Apps & Users)
- OS provides services for Apps & Users
- OS manages resources

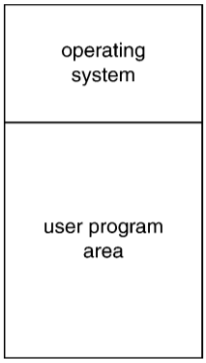
Operating System Definitions

Resource allocator – manages and allocates resources.
Control program – controls the execution of user programs and operations of I/O devices .
Kernel – the one program running at all times (all else being application programs).

What is an Operating System?
A program that acts as an intermediary between a user of a computer and the computer hardware.
Operating system goals:
Execute user programs and make solving user problems easier.
Make the computer system convenient to use.
Use the computer hardware in an efficient manner

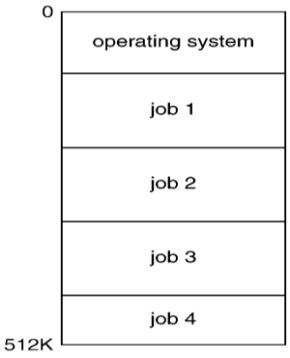
Mainframe Systems
* Reduce setup time by batching similar jobs
* Automatic job sequencing – automatically transfers control from one job to another.
* Resident monitor
*initial control in monitor
*control transfers to job
* when job completes control transfers pack to monitor

Memory Layout for a Simple Batch System



Multiprogrammed Batch Systems

Several jobs are kept in main memory at the same time, and the CPU is multiplexed among them.



OS Features Needed for Multiprogramming

*I/O routine supplied by the system.
*Memory management – the system must allocate the memory to several jobs.
*CPU scheduling – the system must choose among several jobs ready to run.
*Allocation of devices.

Time-Sharing Systems–Interactive Computing

*The CPU is multiplexed among several jobs that are kept in memory and on disk (the CPU is allocated to a job only if the job is in memory).
*A job swapped in and out of memory to the disk.
*On-line communication between the user and the system is provided; when the operating system finishes the execution of one command, it seeks the next "control statement" from the user's keyboard.
*On-line system must be available for users to access data and code.

Desktop Systems

*Personal computers – computer system dedicated to a single user.
*I/O devices – keyboards, mice, display screens, small printers.
*User convenience and responsiveness.
*Can adopt technology developed for larger operating system' often individuals have sole use of computer and do not need advanced CPU utilization of protection features.
*May run several different types of operating systems (Windows, MacOS, UNIX, Linux)

Parallel Systems

*Multiprocessor systems with more than on CPU in close communication.
*Tightly coupled system – processors share memory and a clock; communication usually takes place through the shared memory.
*Advantages of parallel system:
*Increased throughput
*Economical
*Increased reliability
*graceful degradation

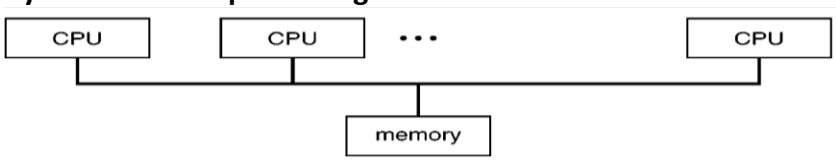
Symmetric multiprocessing (SMP)
*Each processor runs and identical copy of the operating system.
*Many processes can run at once without performance deterioration.
*Most modern operating systems support SMP

Asymmetric multiprocessing
*Each processor is assigned a specific task; master processor schedules and allocated work to slave processors.
*More common in extremely large systems

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What are Operating System components	Dec 2009	5
Q.2	Types of Operating System	Dec 2009 June 2011 Dec 2013	10 10 7
Q.3	Features of Operating System	Dec 2011 Dec 2012	10 7

Unit-01/Lecture-06

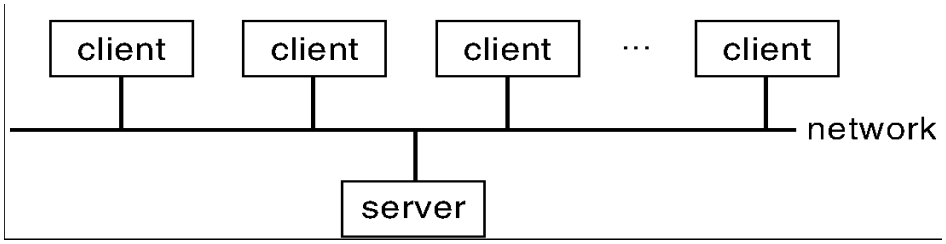
Symmetric Multiprocessing Architecture



Distributed Systems

*Distribute the computation among several physical processors.
*Loosely coupled system – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or telephone lines.
*Requires networking infrastructure.
*Local area networks (LAN) or Wide area networks (WAN)
*May be either client-server or peer-to-peer systems.
Advantages of distributed systems.
*Resources Sharing
*Computation speed up – load sharing
*Reliability
*Communications

General Structure of Client-Server



Clustered Systems

*Clustering allows two or more systems to share storage.
*Provides high reliability.
Asymmetric clustering: one server runs the application while other servers standby.
Symmetric clustering: all N hosts are running the application.

Real-Time Systems

*Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
*Well-defined fixed-time constraints.
*Real-Time systems may be either hard or soft real-time.
Hard real-time:
*Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
*Conflicts with time-sharing systems, not supported by general-purpose operating systems.

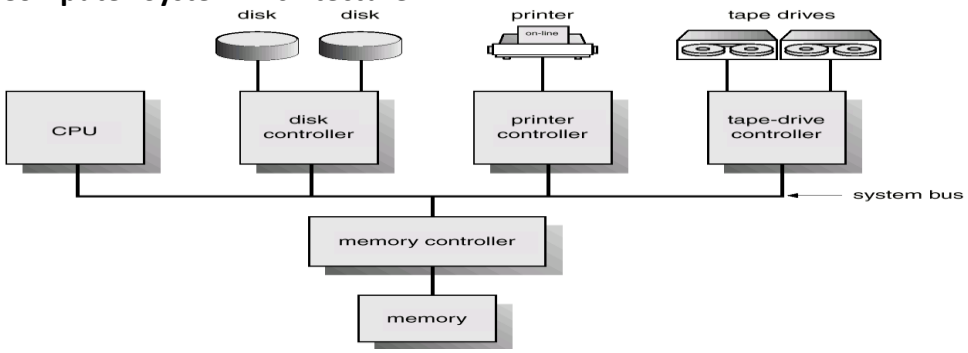
Soft real-time
*Limited utility in industrial control of robotics

*Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

Handheld Systems

* Personal Digital Assistants (PDAs)
* Cellular telephones
Issues:
* Limited memory
* Slow processors
* Small display screens.

Computer-System Architecture



Computer-System Operation

*I/O devices and the CPU can execute concurrently.
*Each device controller is in charge of a particular device type.
*Each device controller has a local buffer.
*CPU moves data from/to main memory to/from local buffers
*I/O is from the device to local buffer of controller.
*Device controller informs CPU that it has finished its operation by causing an interrupt.

Common Functions of Interrupts

*Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
*Interrupt architecture must save the address of the interrupted instruction.
*Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
*A trap is a software-generated interrupt caused either by an error or a user request.
*An operating system is interrupt driven.

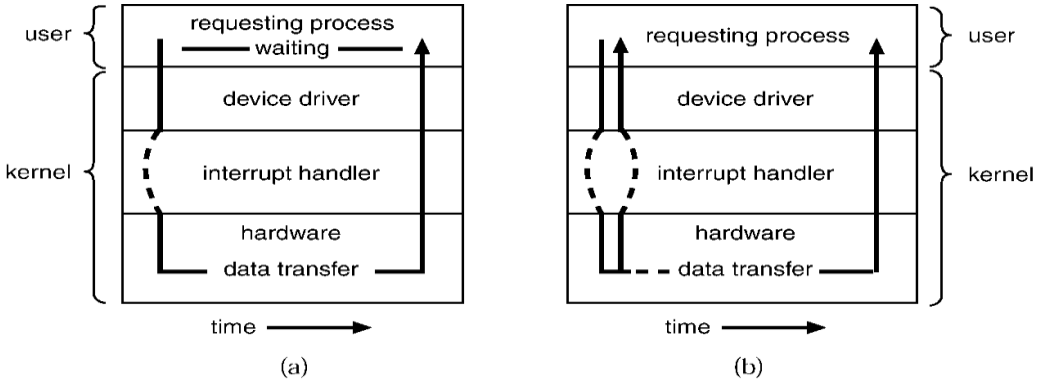
Interrupt Handling

*The operating system preserves the state of the CPU by storing registers and the program counter.
*Determines which type of interrupt has occurred:
*polling
*vectored interrupt system
*Separate segments of code determine what action should be taken for each type of interrupt

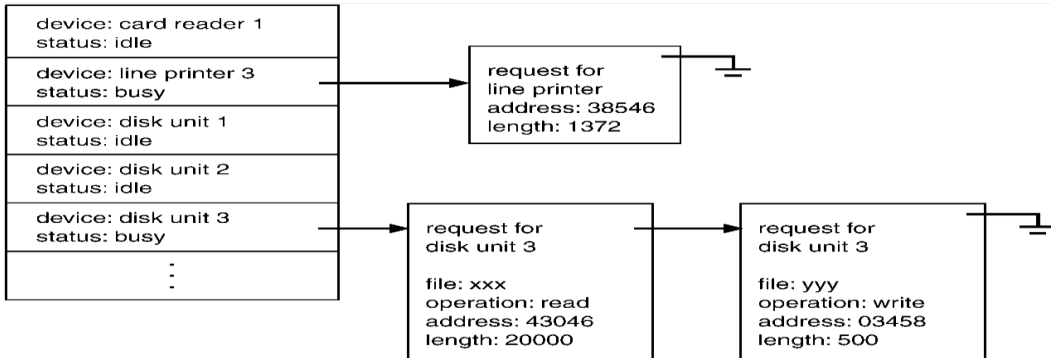
I/O Structure

*After I/O starts, control returns to user program only upon I/O completion.
*Wait instruction idles the CPU until the next interrupt
*Wait loop (contention for memory access).
*At most one I/O request is outstanding at a time, no simultaneous I/O processing.
*After I/O starts, control returns to user program without waiting for I/O completion.
*System call – request to the operating system to allow user to wait for I/O completion.
*Device-status table contains entry for each I/O device indicating its type, address, and state.
*Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Two I/O Methods



Device-Status Table

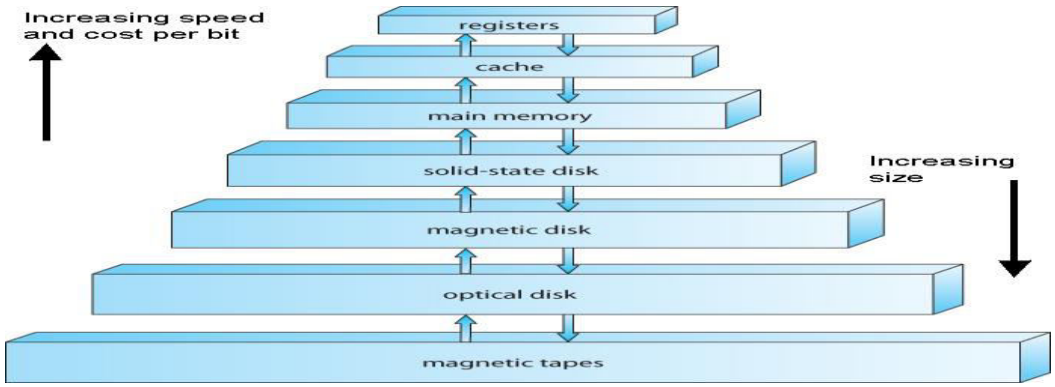


S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain Symmetric Asymmetric Multiprocessing	Dec 2012	10

Unit-01/Lecture-07

Storage Structure

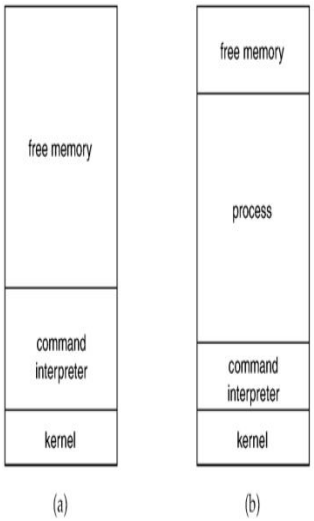
- Main memory (RAM)
  - Programs must be loaded into RAM to run.
  - Instructions and data fetched from RAM into registers.
  - RAM is volatile
  - "Medium" size and speed
- Other electronic (volatile) memory is faster, smaller, and more expensive per bit:
  - Registers
  - CPU Cache
- Non-volatile memory ("permanent" storage) is slower, larger, and less expensive per bit:
  - Electronic disks
  - Magnetic disks
  - Optical disks
  - Magnetic Tapes



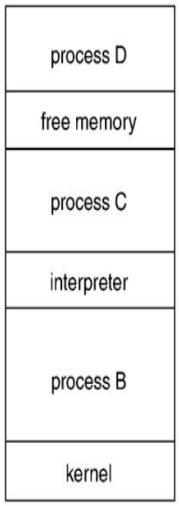
Storage-device hierarchy

Operating System Structures:-

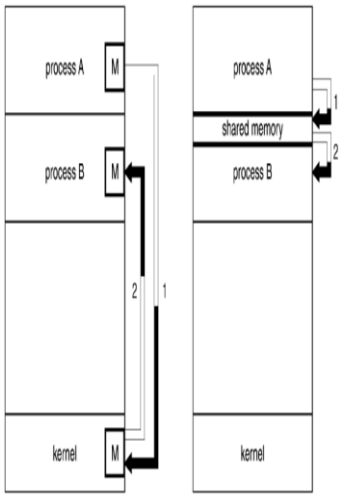
MS-DOS Execution



UNIX Running Multiple Programs



Communication Models



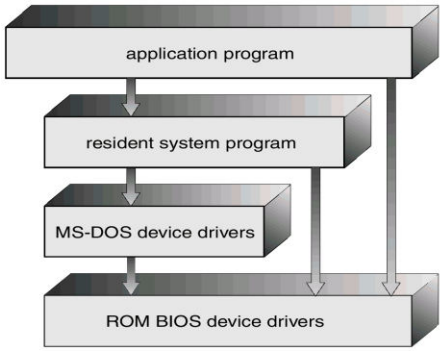
MS-DOS System Structure

*MS-DOS – written to provide the most functionality in the least space
*not divided into modules
*Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.

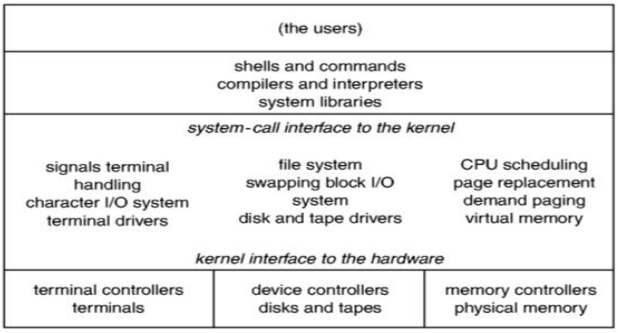
UNIX System Structure

*UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.
*Systems programs
*The kernel
*Consists of everything below the system-call interface and above the physical hardware
*Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

MS-DOS Layer Structure



UNIX System Structure

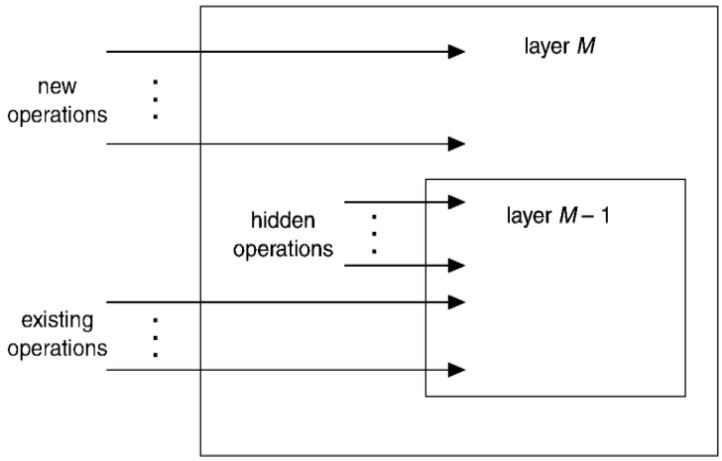


Unit-01/Lecture-08

Layered Approach

*The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
*With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.

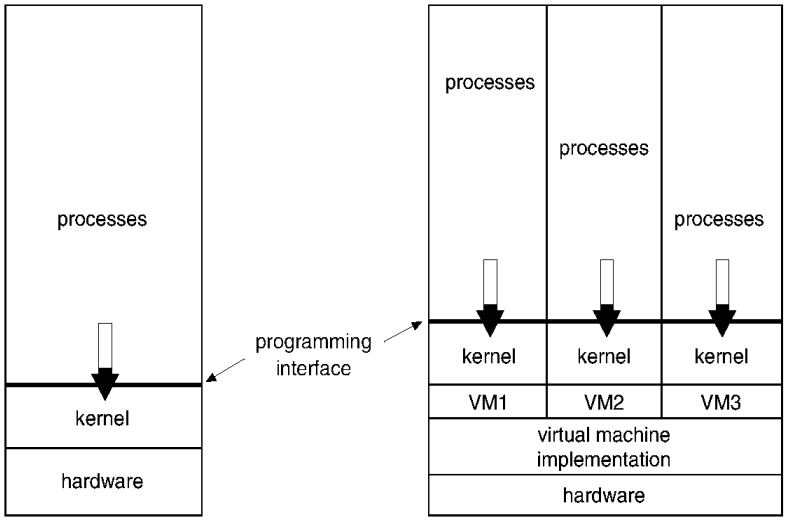
An Operating System Layer



Virtual Machines

*A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
*A virtual machine provides an interface identical to the underlying bare hardware.
*The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
*The resources of the physical computer are shared to create the virtual machines.
*CPU scheduling can create the appearance that users have their own processor.
*Spooling and a file system can provide virtual card readers and virtual line printers.
*A normal user time-sharing terminal serves as the virtual machine operator's console.

System Models



Advantages/Disadvantages of Virtual Machines

*The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.

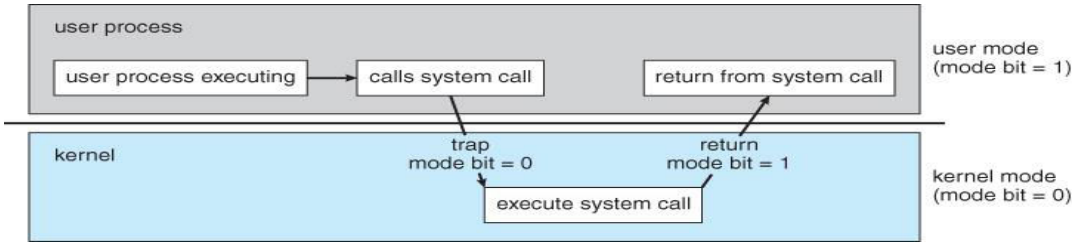
*A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

*The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine.

Operating-System Operations

Interrupt-driven nature of modern OSes requires that erroneous processes not be able to disturb anything else.

Dual-Mode and Multimode Operation



Transition from user to kernel mode

- User mode when executing harmless code in user applications
- Kernel mode (a.k.a. system mode, supervisor mode, privileged mode) when executing potentially dangerous code in the system kernel.
- Certain machine instructions can only be executed in kernel mode.
- Kernel mode can only be entered by making system calls. User code cannot flip the mode switch.

Operating System Services

- Program execution – system capability to load a program into memory and to run it.
- I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.
- Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via shared memory or message passing.
- Error detection – ensure correct computing by detecting errors in the CPU and memory ardware, in I/O devices, or in user programs.

Additional Operating System Functions

Additional functions exist not for helping the user, but rather for ensuring efficient system

operations.

1. Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.
2. Accounting – keep track of and record which users use how much and what kinds of compute resources for account billing or for accumulating usage statistics.
3. Protection – ensuring that all access to system resources is controlled.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain virtual machine concept.	Dec 2012	10
Q.2	Services of Operating System	Dec 2009 Dec 2011 Dec 2013	5 10 7

Unit-01/Lecture-09

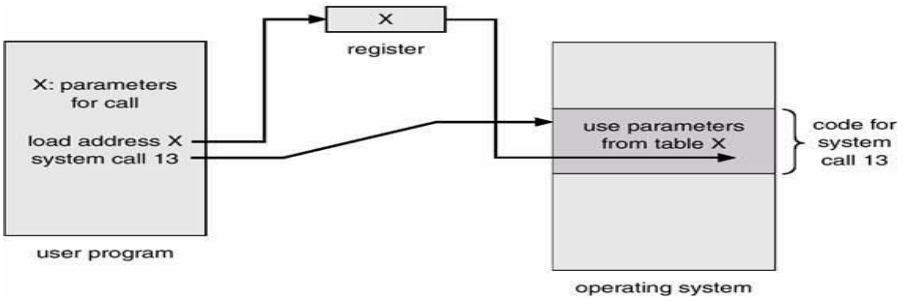
System Calls

System calls provide the interface between a running program and the operating system.

1. Generally available as assembly-language instructions.
2. Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)

Three general methods are used to pass parameters between a running program and the operating system.

1. Pass parameters in registers.
2. Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
3. Push (store) the parameters onto the stack by the program, and pop off the stack by operating system. Passing of Parameters As A Table



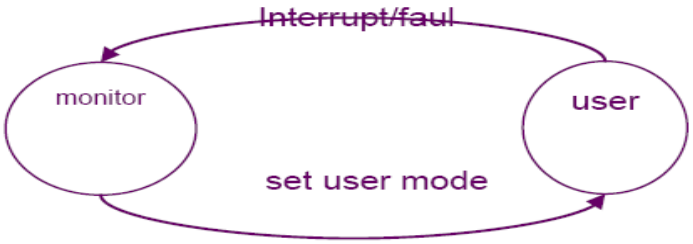
Types of System Calls

1)Process control, 2) File management, 3)Device management, 4)Information maintenance,

5)Communications

Hardware Protection
*Dual-Mode Operation
*I/O Protection
*Memory Protection
*CPU Protection

Dual-Mode Operation
*Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
*Provide hardware support to differentiate between at least two modes of operations.
1. User mode – execution done on behalf of a user.
2. Monitor mode (also kernel mode or system mode) – execution done on behalf of operating system.
*Mode bit added to computer hardware to indicate the current mode:
monitor (0) or user (1).
*When an interrupt or fault occurs hardware switches to monitor mode.
*Privileged instructions can be issued only in monitor mode.



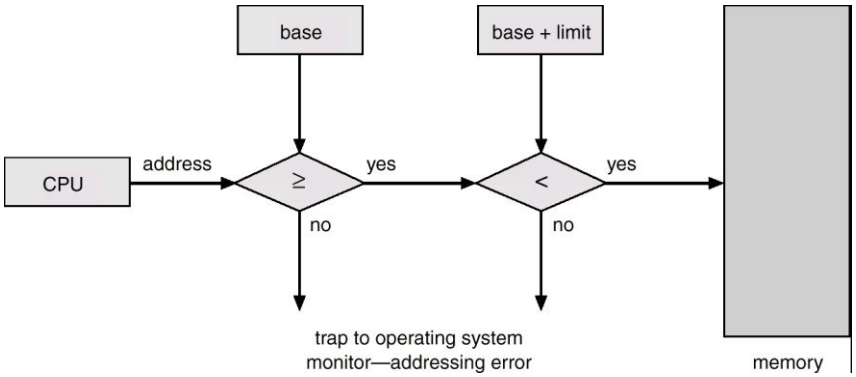
I/O Protection

*All I/O instructions are privileged instructions.
*Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

Memory Protection

*Must provide memory protection at least for the interrupt vector and the interrupt service routines.
*In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
***Base register** – holds the smallest legal physical memory address.
***Limit register** – contains the size of the range
*Memory outside the defined range is protected.

Hardware Address Protection



Hardware Protection

*When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.
*The load instructions for the base and limit registers are privileged instructions.

CPU Protection

*Timer – interrupts computer after specified period to ensure operating system maintains control.
*Timer is decremented every clock tick.
*When timer reaches the value 0, an interrupt occurs.
*Timer commonly used to implement time sharing.
*Time also used to compute the current time.
*Load-timer is a privileged instruction.

S.NO	RGPV QUESTION	YEAR	MARKS
Q.1	Explain System calls	Dec 2009	4
Q.2	Explain system protection in OS	Dec 2009	4