Q.2 a. Define the following: (8)

(i) Process
(ii) Process Control Block (PCB)
(iii) Multi programming
(iv) Time sharing

Answer:

(i) Process: Process is a program in execution; process execution must progress in sequential fashion. A process includes:

- program counter
- stack
- data section

(ii) Process Control Block (PCB): Information associated with each process is stored in Process control Block.

Process state
Program counter
CPU registers
CPU scheduling information
Memory-management information
Accounting information
I/O status information

(iii) Multiprogramming: A multiprogramming operating system is system that allows more than one active user program (or part of user program) to be stored in main memory simultaneously. Multi programmed operating systems are fairly sophisticated. All the jobs that enter the system are kept in the job pool. This pool consists of all processes residing on mass storage awaiting allocation of main memory. If several jobs are ready to be brought into memory, and there is not enough room for all of them, then the system must choose among them. A time-sharing system is a multiprogramming system.

(iv) Time Sharing: Sharing of a computing resource among many users by means of multiprogramming and multi-tasking is known as timesharing. By allowing a large number of users to interact concurrently with a single computer, time-sharing dramatically lowered the cost of providing computing capability, made it possible for individuals and organizations to use a computer without owning one, and promoted the interactive use of computers and the development of new interactive applications.

b. What is an operating system? List the typical functions of operating systems. **(8)**

Answer:

An **operating system** is system software that provides interface between user and hardware. The operating system provides the means for the proper use of resources (CPU, memory, I/O devices, data and so on) in the operation of the computer system. An operating system provides an environment within which other programs can do useful work.

Typical functions of operating system are as follows:

(1)Process management: A process is a program in execution. It is the job, which is currently being executed by the processor. During its execution a process would require certain system resources such as processor, time, main memory, files etc. OS supports multiple processes simultaneously. The process management module of the OS takes care of the creation and termination of the processes, assigning resources to the processes, scheduling processor time to different processes and communication among processes.

(2)Memory management module: It takes care of the allocation and deallocation of the main memory to the various processes. It allocates main and secondary memory to the system/user program and data. To execute a program, its binary image must be loaded into the main memory.

Operating System decides.

(a) Which part of memory are being currently used and by whom.

(b) which process to be allocated memory.

(c) Allocation and de allocation of memory space.

(3)I/O management: This module of the OS co-ordinates and assigns different I/O devices namely terminals, printers, disk drives, tape drives etc. It controls all I/O devices, keeps track of I/O request, issues command to these devices.

I/O subsystem consists of

(i) Memory management component that includes buffering, caching and spooling.

(ii) Device driver interface

(iii) Device drivers specific to hardware devices.

(4)File management: Data is stored in a computer system as files. The file management module of the OS would manage files held on various storage devices and transfer of files from one device to another. This module takes care of creation, organization, storage, naming, sharing, backup and protection of different files.

(5)Scheduling: The OS also establishes and enforces process priority. That is, it determines and maintains the order in which the jobs are to be executed by the computer system. This is so because the most important job must be executed first followed by less important jobs.

(6)Security management: This module of the OS ensures data security and integrity. That is, it protects data and program from destruction and unauthorized access. It keeps different programs and data which are executing concurrently in the memory in such a manner that they do not interfere with each other.

(7)Processor management: OS assigns processor to the different task that must be performed by the computer system. If the computer has more than one processor idle, one of the processes waiting to be executed is assigned to the idle processor. OS maintains internal time clock and log of system usage for all the users. It also creates error message and their debugging and error detecting codes for correcting programs.

Q.3 a. Differentiate between pre-emptive and non-pre-emptive scheduling. (4)

Answer:

In a **pre-emptive scheduling** approach, CPU can be taken away from a process if there is a need while in a non-pre-emptive approach if once a process has been given the CPU, the CPU cannot be taken away from that process, unless the process completes or leaves the CPU for performing an Input Output. Pre-emptive scheduling is more useful in high priority process which requires immediate response, for example in real time system. While in **nonpreemptive systems,** jobs are made to wait by longer jobs, but treatment of all processes is fairer.

b. What are the disadvantages of FCFS scheduling algorithm as compared to shortest job first (SJF) scheduling? (4)

Answer:

Disadvantages:

(i) Waiting time can be large if short requests wait behind the long ones.
(ii) It is not suitable for time sharing systems where it is important that each user should get the CPU for an equal amount of time interval.
(iii) A proper mix of jobs is needed to achieve good results from FCFS scheduling.

c. Define deadlock? Explain the necessary conditions for deadlock to occur. (4)

Answer:

Deadlock is a situation, in which processes never finish executing and system resources are tied up, preventing other jobs from starting. A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes, thereby causing deadlock.Necessary conditions for deadlock to occur are:

i. **Mutual exclusion:** At least one resource must be held in a nonsharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

ii. **Hold and wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

iii. **No pre-emption:** Resources cannot be pre-empted; that is, a resource can be released only voluntarily by the process holding it, after the process holding it has completed its task.

iv. **Circular wait:** A set{P0, P1,……, Pn) of waiting processes must exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, ……., Pn-1 is waiting for a resource that is held by Pn and Pn is waiting for a resource that is held by P0.

All four conditions must hold simultaneously for a deadlock to occur and conditions are not completely independent. For example, the circular-wait implies the hold-andwait condition.

d. Write an algorithm for deadlock detection. (4)

Answer:

An algorithm for deadlock detection:

1. Let Work and Finish be vectors of length m and n, respectively.
   Initialize:
   (a) Work = Available
   (b) For i = 1,2, …, n, if Allocation i $^1\neq$ 0, then
   Finish[i] = false; otherwise, Finish[i] = true.
2. Find an index i such that both:
   (a) Finish[i] == false
   i. (b) Requesti ≤ Work
   If no such i exists, go to step 4.
3. Work = Work + Allocationi
   Finish[i] = true
   go to step 2.
4. If Finish[i] == false, for some i, 1 ≤ i ≤ n, then the system is in deadlock state. Moreover, if Finish[i] == false, then Pi is deadlocked.

Q.4 a. What is a semaphore? Explain a binary semaphore with the help of an example. (4)

Answer:

A **semaphore** is a synchronization tool that provides a general-purpose solution to controlling access to critical sections.

A semaphore is an abstract data type (ADT) that defines a nonnegative integer variable which, apart from initialization, is accessed only through two standard operations: wait and signal. The classical definition of wait in pseudo code is

```
wait(S){
while(S<=0)
; // do nothing
S--;
}
```

The classical definitions of signal in pseudocode is

```
signal(S){
S++;
}
```

A binary semaphore is one that only takes the values 0 and 1. These semaphores are used to implement mutual exclusion.

b. What is a race condition? Explain how does a critical section avoid this condition. What are the properties which a data item should possess to implement a critical section? (6)

Answer:

Race condition: The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last. To prevent race conditions, concurrent processes must be **synchronized**.

Data consistency requires that only one processes should update the value of a data item at any time. This is ensured through the notion of a critical section. A critical section for data item d is a section of code, which cannot be executed concurrently with itself or with other critical sections for d. Consider a system of n processes (P_0, P_1,..., P_{n-1}), each process has a segment of code called a critical section, in which the proceses may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus the execution of critical sections by the processes is mutually exclusive in time.

Solution to the Critical Section Problem must satisfy the following three conditions:

1. **Mutual Exclusion**. If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress**. If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting**. A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

—Assume that each process executes at a nonzero speed
—No assumption concerning relative speed of the *n* processes.

c. Discuss the different techniques with which a file can be shared among different users. (6)

Answer:

Some popular techniques with which a file can be shared among different users are:

1. Sequential sharing: In this sharing technique, a file can be shared by only one program at a time, that is, file accesses by P1 and P2 are spaced out over time. A lock field can be used to implement this. Setting and resetting of the lock at file open and close ensures that only one program can use the file at any time.
2. Concurrent sharing: Here a number of programs may share a file concurrently. When this is the case, it is essential to avoid mutual interference between them. There are three categories of concurrent sharing:

a. **Immutable files**: If a file is shared in immutable mode, none of the sharing programs can modify it. This mode has the advantage that sharing programs are independent of one another.

b. **Single image immutable files**: Here the changes made by one program are immediately visible to other programs. The Unix file system uses this filesharing mode.

c. Multiple image mutable files: Here many programs can concurrently update the shared file. Each updating program creates a new version of the file, which is different from the version created by concurrent programs. This sharing mode can only be used in applications where concurrent updates and the existence of multiple versions are meaningful.

Q.5 a. Explain the differences between: (8)
(i) Logical and physical address space
(ii) Internal and external fragmentation

Answer:

Logical Vs physical address space

(1) An address generated by the CPU is commonly referred to as a logical a logical address. The set of all logical addresses generated by a program is known as logical address space. Whereas, an address seen by the memory unit- that is, the one loaded into the memory-address register of the memory- is commonly referred to as physical address. The set of all physical addresses corresponding to the logical addresses is known as physical address space.

(2) The compile-time and load-time address-binding methods generate identical logical and physical addresses. However, in the execution-time address-binding scheme, the logical and physical-address spaces differ.

(3) The user program never sees the physical addresses. The program creates a pointer to a logical address, say 346, stores it in memory, manipulate it, compares it to other logical addresses- all as the number 346.

Only when a logical address is used as memory address, it is relocated relative to the base/relocation register. The memory-mapping hardware device called the memorymanagement unit(MMU) converts logical addresses into physical addresses.

(4) Logical addresses range from 0 to max. User program that generates logical address thinks that the process runs in locations 0 to max.

Logical addresses must be mapped to physical addresses before they are used. Physical addresses range from (R+0) to (R + max) for a base/relocation register value R.

(ii) Internal and external fragmentation

(1) When memory allocated to a process is slightly larger than the requested memory, space at the end of a partition is unused and wasted. This wasted space within a partition is called as internal fragmentation. When enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes. This wasted space not allocated to any partition is called external fragmentation. (2) Internal fragmentation is found in multiple fixed partition schemes where all the
partitions are of the same size. That is, physical memory is broken into fixed-sized blocks.
External fragmentation is found in multiple variable partition schemes. Instead of dividing memory into a fixed set of partitions, an operating system can choose to allocate to a process the exact amount of unused memory space it requires.

(3) In multiple fixed partition scheme, the partition table needs to store either the starting address for each process or the number of the partition allocated to each process.
In multiple variable partition scheme, the overhead of managing more data increases. The partition table must store exact starting and ending location of each process and data about which memory locations are free must be maintained.

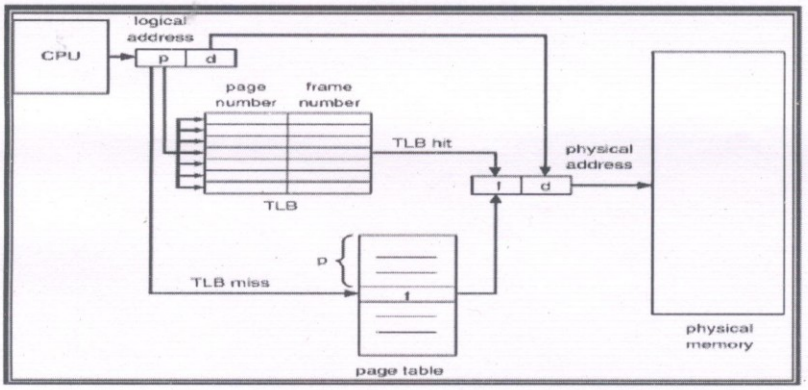
(4) In multiple fixed partition schemes, size/limit register is set at boot time and contains the partition size. Each time a process is allocated control of CPU, the operating system only needs to reset the relocation register. In multiple variable partition schemes, each time a different process is given control of the CPU, the operating system must reset the size/limit register in addition to the relocation register. The operating system must also make decisions on which partition it should allocate to a process.
(5) Internal fragmentation can be reduced using multiple variable partition method. However, this solution suffers from external fragmentation. External fragmentation can be solved using compaction where the goal is to shuffle the memory contents to place all free memory together in one large block. Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non contiguous. This solution is achieved by paging and segmentation.

b. Why are Translation Look-aside Buffers (TLBs) important? In a simple paging system, what information is stored in a typical TLB table entry? (8)

Answer:

The implementation of page-table is done in the following way:
- Page table is kept in main memory.
- *Page-table base register (*PTBR) points to the page table.
- *Page-table length register* (PRLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called *associative memory* or *translation look-aside buffers (TLBs)*. A set of associative registers is built of high-speed memory where each register consists of two parts: a key and a value. When the associative registers are presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is the output.



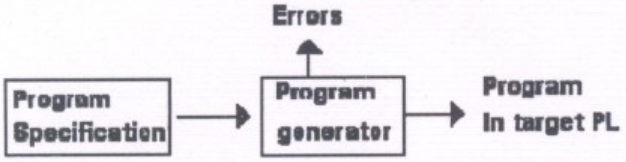
A typical TLB table entry consists of page# and frame#, when a logical address is generated by the CPU, its page number is presented to a set of associative registers that contain page number along with their corresponding frame numbers. If the page number is found in the associative registers, its frame number is available and is used to access memory. If the page number is not in the associated registers, a memory reference to the page table must be made. When the frame number is obtained, it can be used to access memory and the page number along with its frame number is added to the associated registers.

Q.6 a. Explain language processing activities. (8)

Answer:

There are two different types of language processing activities:
1. Program generation activities
2. Program execution activities

Program generation activities: A program generation activity aims at automatic generation of a program. The source language is a specification language of an application domain and the target language is typically a procedure oriented programming language. the following figure shows program generation activity

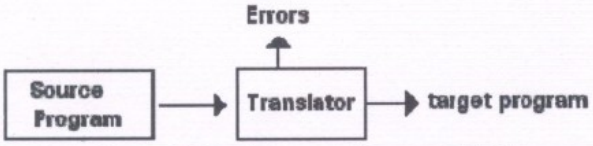


The program generator is a software system which accepts the specification of a program to be generated and generates a program in the target. PL. The program generator introduces a new domain between the application and PL domains. We call this the program generator domain. The specification gab is now gab between the application domain and the program generator domain. This gab is smaller than the gab between the application domain and PL domain.

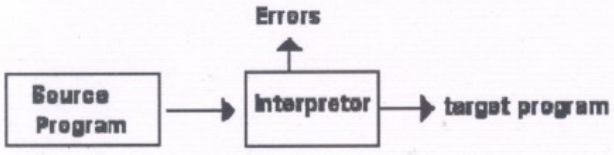
Program execution activities: A program execution activity organizes system. Two model program executions are:

1. Translation 2. Interpretation

Translation: The program translation models bridges execution gap by translating a program written in a PL, called the source program into an equivalent program in the machine or assembly language of the computer system.



Interpretation: The interpreter reads the source program and stores it in its memory. During interpretation it takes a statement, determines its meaning and performs actions which implement it.



b. How can be classified data structures used for language processors? (8)

Answer:

The data structures used in language processing can be classified on the basis of the following criteria:

1. **Nature of data structure** (whether a linear or non-linear data structure)
2. **Purpose of a data structure** (whether a search data structure or an allocation data structure)
3. **Life time of a data structure** (whether used during language processing or during target program execution)

A linear data structure consists of a linear arrangement of elements in the memory. A linear data structure requires a contiguous area of memory for its elements. This poses a problem in situations where the size of a data structure is difficult to predict. the elements of non linear data structures are accessed using pointers. Hence the elements need not occupy contiguous area of memory.

Search Data structures are used during language processing to maintain attribute information concerning different entities in the source program. In this the entry for an entity is created only once, but may be searched for large number of times.

Allocation data structures are characterized by the fact that the address of memory area allocated to an entity is known to the users. So no search operations are conducted.

Q.7 a. What is parsing? Explain any three parsing techniques. (8)

Answer:

Parsing is the process of analyzing a text, made of a sequence of tokens, to determine its grammatical structure with respect to a given formal grammer. Parsing is also known as syntactic analysis and parser is used for analyzing a text. The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar.

Following are three parsing techniques:

Top-down parsing - Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative righthand-
sides of grammar rules.

Bottom-up parsing - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing.

Recursive descent parsing- It is a top down parsing without backtracking. This parsing technique uses a set of recursive procedures to perform parsing. Salient advantages of recursive descent parsing are its simplicity and generality. It can be implemented in any language supporting recursive procedures.

b. Explain macro definition, macro call and macro expansion. (8)

Answer:

A unit of specification for a program generation is called a **macro**. It consists of name, set of formal parameters and body of code. When a macro name is used with a set of actual parameters it is replaced by a code generated from its body. This code is called **macro expansion**. There are two types of expansions:

1. lexical expansion
2. Semantic expansion

Lexical expansion: It means a replacement of character string by another string during program generation. It is generally used to replace occurrences of formal parameters by corresponding actual ones.

Semantic Expansion: It implies generation of instructions build to the requirements of specific usage. It is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.

The macro definition is located at the beginning of the program is enclosed between a macro header and macro end statement.

A macro statement contains macro name and parameters

< macro name > { < parameters>}

A macro call: A macro is called by writing the macro name in the mnemonic field of an assembly statement.

Q.8 a. What are the functions of passes used in two-pass assembler? Explain pass-1 algorithm. (8)

Answer:

Two pass translation of an assembly language program can handle forward references early.

The following tasks are performed by the passes of a two pass assembler are as follows:

Pass I: (i) Separate the symbol, mnemonic opcode and operand fields
(ii) Build the symbol table
(iii) Perform LC processing
(iv) Construct intermediate representation.

Pass II: Synthesize the target program

Pass I uses the following data structures:

OPTAB : A table of mnemonic opcodes and related information.

SYMTAB: symbol table, SYMTAB entry contains the fields address and length.

LITTAB: A table literally used in the program, A LITTAB entry contains literals and address.

OPTAB contains the fields mnemonic opcode, class and mnemonic information. The class field indicated whether the opcode corresponds to an imperative statement (IS), a declaration statement (DL) or an assembler directive (AD).

b. Describe Data structures used during passes of assembler and their use. (8)

Answer:

Data structure during passes of assembler and their use.

Pass 1 data base

1. Input source program
2. A location counter (LC)
3. A table, the machine-operation table (MOT), that indicates the symbolic mnemonic for each instruction and its length.
4. Pseudo- operation table
5. Symbol table
6. Literal table
7. Copy of the input to be used later by pass 2

Pass 2

1. Copy of source program input to pass 1
2. Location counter (LC)
3. MOT
4. POT
5. ST
6. Base table that indicates which registers are currently specified as base register.
7. A work space, INST, that's used to hold instruction as its various parts are being assembled together
8. Punch line, used to produce a printed listing.
9. Punch card for converting assembled instructions into the format needed by the loader.

Q.9 a. Explain analysis and synthesis phase of a compiler. (8)

Answer:

The analysis and synthesis phases of a compiler are:

Analysis Phase: Breaks the source program into constituent pieces and creates intermediate representation. The analysis part can be divided along the following phases:

1. **Lexical Analysis**- The program is considered as a unique sequence of characters. The Lexical Analyzer reads the program from left-to-right and sequence of characters is grouped into **tokens**–lexical units with a collective meaning.
2. **Syntax Analysis**- The **Syntactic Analysis** is also called **Parsing**. Tokens are grouped into grammatical phrases represented by a **Parse Tree, which** gives a hierarchical structure to the source program.
3. **Semantic Analysis**- The **Semantic Analysis** phase checks the program for semantic errors (**Type Checking**) and gathers type information for the successive phases. **Type Checking** check types of operands; No real number as index for array; etc.

Synthesis Phase: Generates the target program from the intermediate representation. The synthesis part can be divided along the following phases:

1. **Intermediate Code Generator**- An intermediate code is generated as a program for an abstract machine. The intermediate code should be easy to translate into the target program.
2. **Code Optimizer**- This phase attempts to improve the intermediate code so that faster-running machine code can be obtained. Different compilers adopt different optimization techniques.
3. **Code Generator**- This phase generates the target code consisting of assembly code. Here
   1. Memory locations are selected for each variable;
   2. Instructions are translated into a sequence of assembly instructions;
   3. Variables and intermediate results are assigned to memory registers.

b.Write short note on code optimization. (8)

Answer:

Code optimization is the optional phase designed to improve the intermediate code so that the
Ultimate object program runs faster or takes less space. Code optimization in compilers aims at improving the execution efficiency of a program by eliminating redundancies and by rearranging the computations in the program without affecting the real meaning of the program.
Scope – First optimization seeks to improve a program rather than the algorithm used in the program. Thus replacement of algorithm by a more efficient algorithm is beyond the scope of optimization. Also efficient code generation for a specific target machine also lies outside its scope.
The structure of program and the manner in which data is defined and used in it provide vital clues for optimization.
Optimization transformations are classified into local and global transformations.

TEXT BOOK

1. **Systems Programming and Operating Systems, D. M. Dhamdhere, Tata McGraw-Hill, Second Revised Edition, 2005**