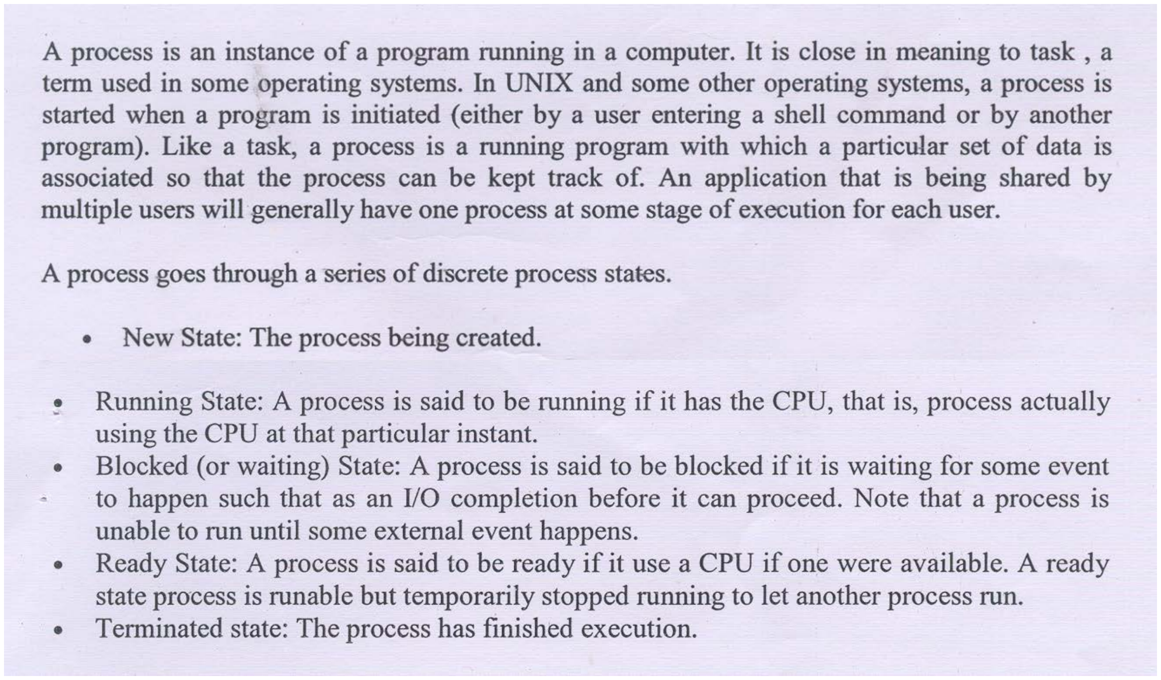
Q2 (a) What are the different states of a process?

Answer



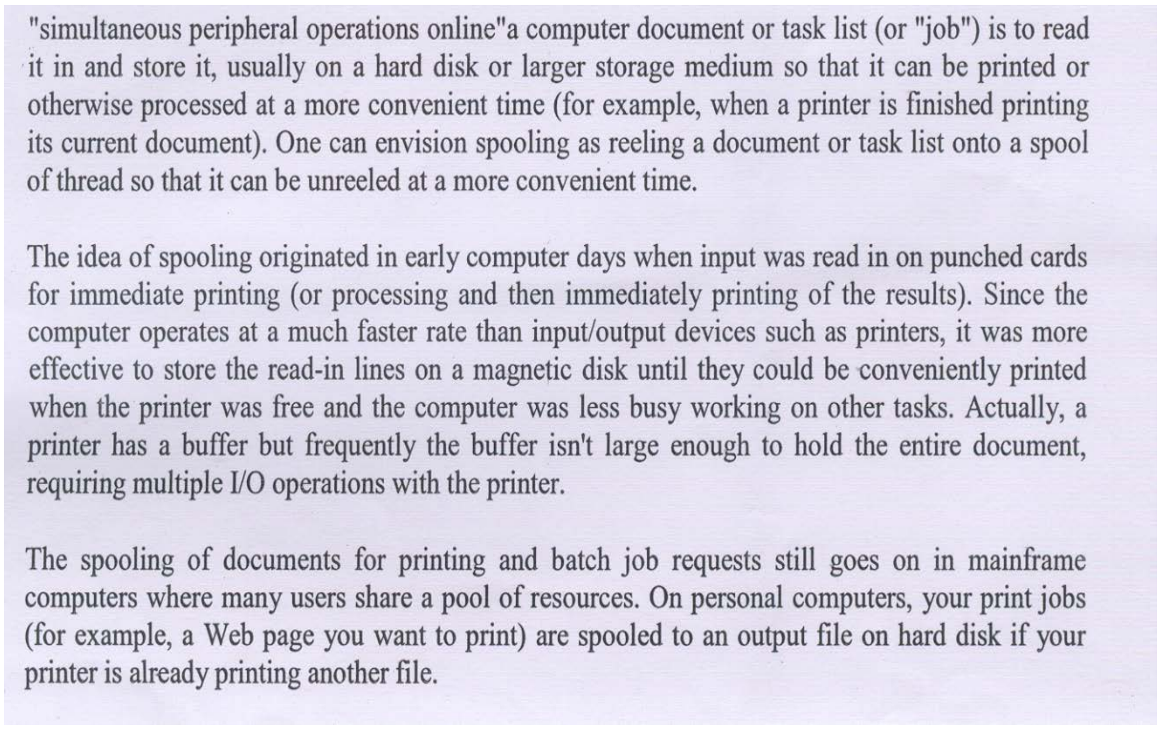
A process is an instance of a program running in a computer. It is close in meaning to task , a term used in some operating systems. In UNIX and some other operating systems, a process is started when a program is initiated (either by a user entering a shell command or by another program). Like a task, a process is a running program with which a particular set of data is associated so that the process can be kept track of. An application that is being shared by multiple users will generally have one process at some stage of execution for each user.

A process goes through a series of discrete process states.

- New State: The process being created.
- Running State: A process is said to be running if it has the CPU, that is, process actually using the CPU at that particular instant.
- Blocked (or waiting) State: A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that a process is unable to run until some external event happens.
- Ready State: A process is said to be ready if it use a CPU if one were available. A ready state process is runable but temporarily stopped running to let another process run.
- Terminated state: The process has finished execution.

Q2 (b) Explain the spooling technology. **

Answer



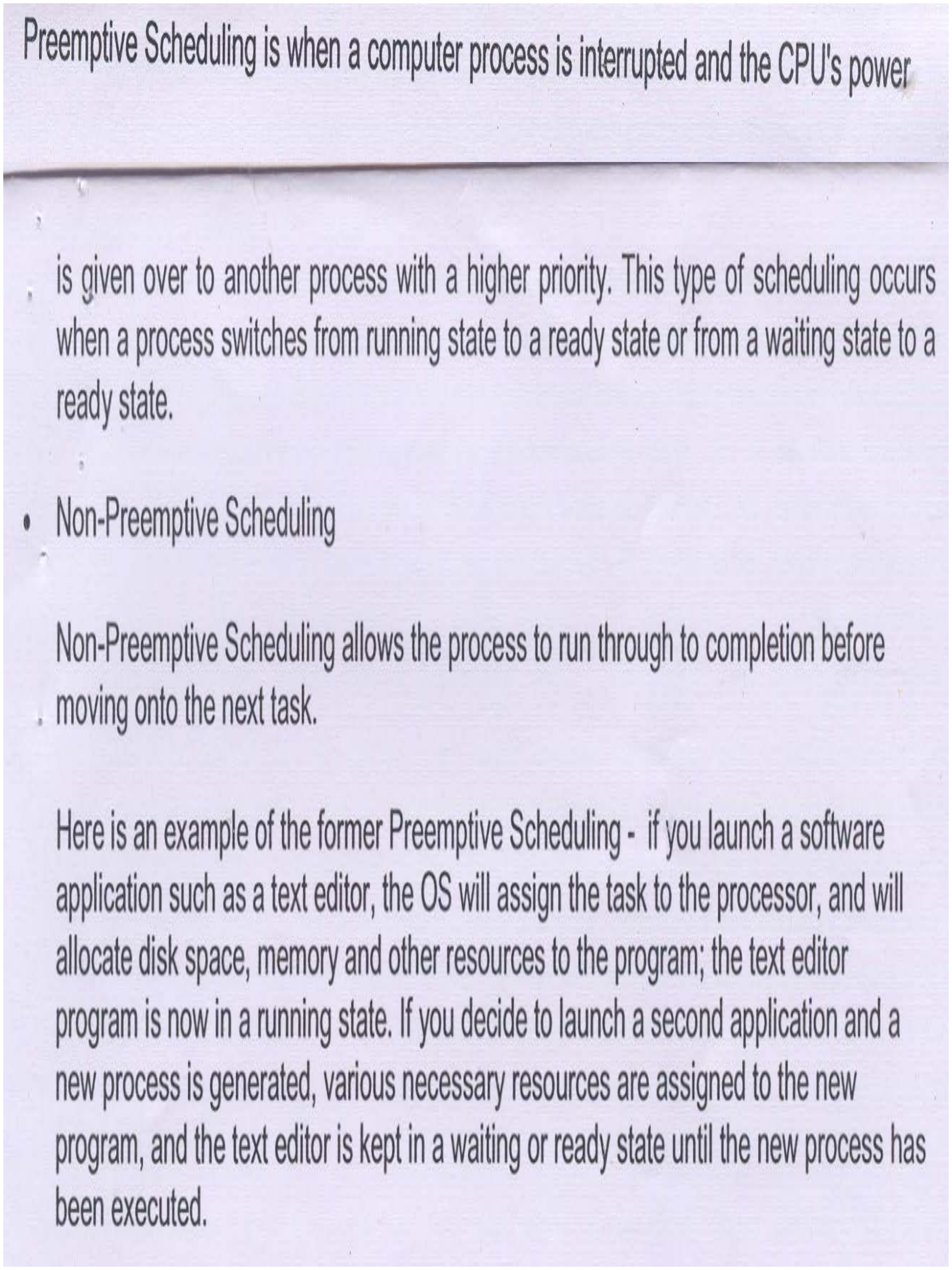
"simultaneous peripheral operations online"a computer document or task list (or "job") is to read it in and store it, usually on a hard disk or larger storage medium so that it can be printed or otherwise processed at a more convenient time (for example, when a printer is finished printing its current document). One can envision spooling as reeling a document or task list onto a spool of thread so that it can be unreeled at a more convenient time.

The idea of spooling originated in early computer days when input was read in on punched cards for immediate printing (or processing and then immediately printing of the results). Since the computer operates at a much faster rate than input/output devices such as printers, it was more effective to store the read-in lines on a magnetic disk until they could be conveniently printed when the printer was free and the computer was less busy working on other tasks. Actually, a printer has a buffer but frequently the buffer isn't large enough to hold the entire document, requiring multiple I/O operations with the printer.

The spooling of documents for printing and batch job requests still goes on in mainframe computers where many users share a pool of resources. On personal computers, your print jobs (for example, a Web page you want to print) are spooled to an output file on hard disk if your printer is already printing another file.

Q3 (a) Differentiate between preemptive and non-preemptive scheduling.

Answer



Preemptive Scheduling is when a computer process is interrupted and the CPU's power

is given over to another process with a higher priority. This type of scheduling occurs when a process switches from running state to a ready state or from a waiting state to a ready state.

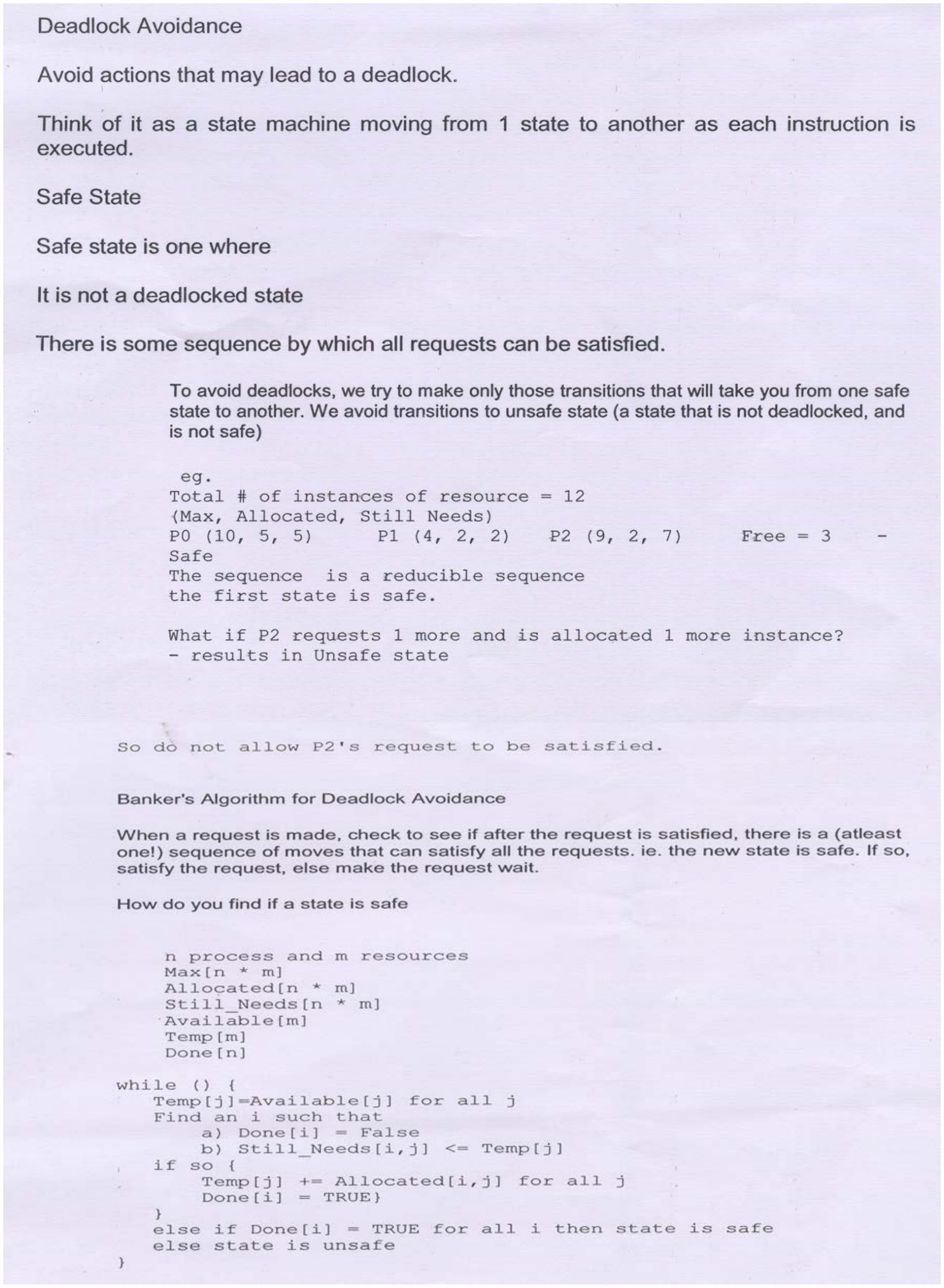
- Non-Preemptive Scheduling

Non-Preemptive Scheduling allows the process to run through to completion before moving onto the next task.

Here is an example of the former Preemptive Scheduling - if you launch a software application such as a text editor, the OS will assign the task to the processor, and will allocate disk space, memory and other resources to the program; the text editor program is now in a running state. If you decide to launch a second application and a new process is generated, various necessary resources are assigned to the new program, and the text editor is kept in a waiting or ready state until the new process has been executed.

Q3 (b) What do you mean by deadlock avoidance? Explain.

Answer



Deadlock Avoidance

Avoid actions that may lead to a deadlock.

Think of it as a state machine moving from 1 state to another as each instruction is executed.

Safe State

Safe state is one where

It is not a deadlocked state

There is some sequence by which all requests can be satisfied.

To avoid deadlocks, we try to make only those transitions that will take you from one safe state to another. We avoid transitions to unsafe state (a state that is not deadlocked, and is not safe)

```
 eg.
Total # of instances of resource = 12
(Max, Allocated, Still Needs)
P0 (10, 5, 5)      P1 (4, 2, 2)    P2 (9, 2, 7)      Free = 3    -
Safe
The sequence  is a reducible sequence
the first state is safe.

What if P2 requests 1 more and is allocated 1 more instance?
- results in Unsafe state
```

```
So do not allow P2's request to be satisfied.
```

Banker's Algorithm for Deadlock Avoidance

When a request is made, check to see if after the request is satisfied, there is a (atleast one!) sequence of moves that can satisfy all the requests. ie. the new state is safe. If so, satisfy the request, else make the request wait.

How do you find if a state is safe

```
    n process and m resources
    Max[n * m]
    Allocated[n * m]
    Still_Needs[n * m]
    Available[m]
    Temp[m]
    Done[n]

while () {
   Temp[j]=Available[j] for all j
   Find an i such that
       a) Done[i] = False
       b) Still_Needs[i,j] <= Temp[j]
   if so {
       Temp[j] += Allocated[i,j] for all j
       Done[i] = TRUE}
   }
   else if Done[i] = TRUE for all i then state is safe
   else state is unsafe
}
```

Q3 (c) An OS contains 3 resource classes. The number of resource units in these classes is 7, 7 and 10, respectively. The current resource allocation state is as shown below:

	Allocated resources			**Maximum requirements**		
	R_1	R_2	R_3	R_1	R_2	R_3
Process p_1	**2**	**2**	**3**	**3**	**6**	**8**
Process p_2	**2**	**0**	**3**	**4**	**3**	**3**
Process p_3	**1**	**2**	**4**	**3**	**4**	**4**

(i) Is the current allocation state safe?

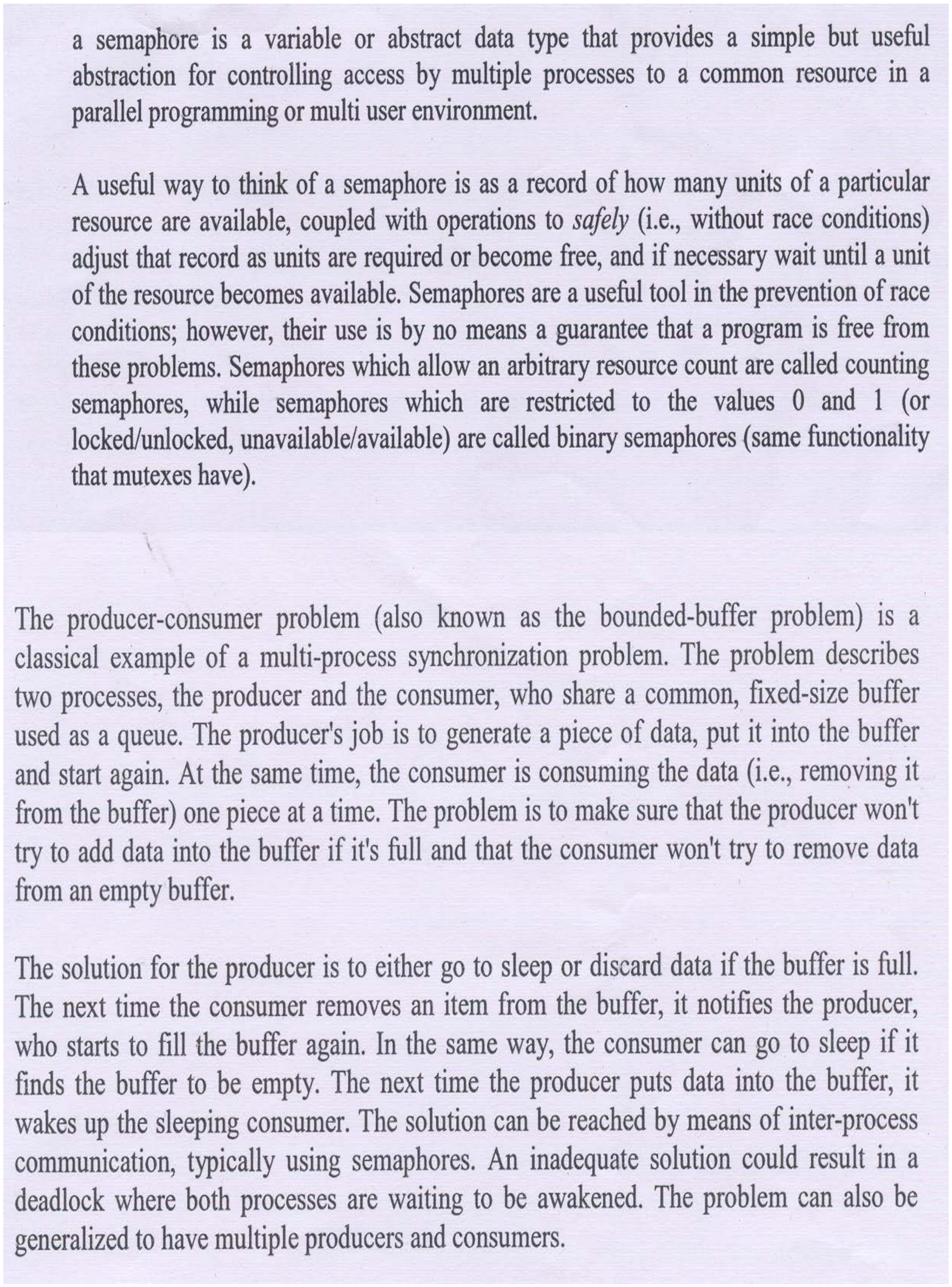
(ii) Would the following requests be granted in the current state?

- **Process** p_1 **requests (1, 1, 0)**
- **Process** p_2 **requests (0, 1, 0)**
- **Process** p_3 **requests (0, 1, 0)**

Answer Page Number 394 of text book

Q4 (a) What is Semaphore? Write the code for Producer-Consumer problem using Semaphore.

Answer



a semaphore is a variable or abstract data type that provides a simple but useful abstraction for controlling access by multiple processes to a common resource in a parallel programming or multi user environment.

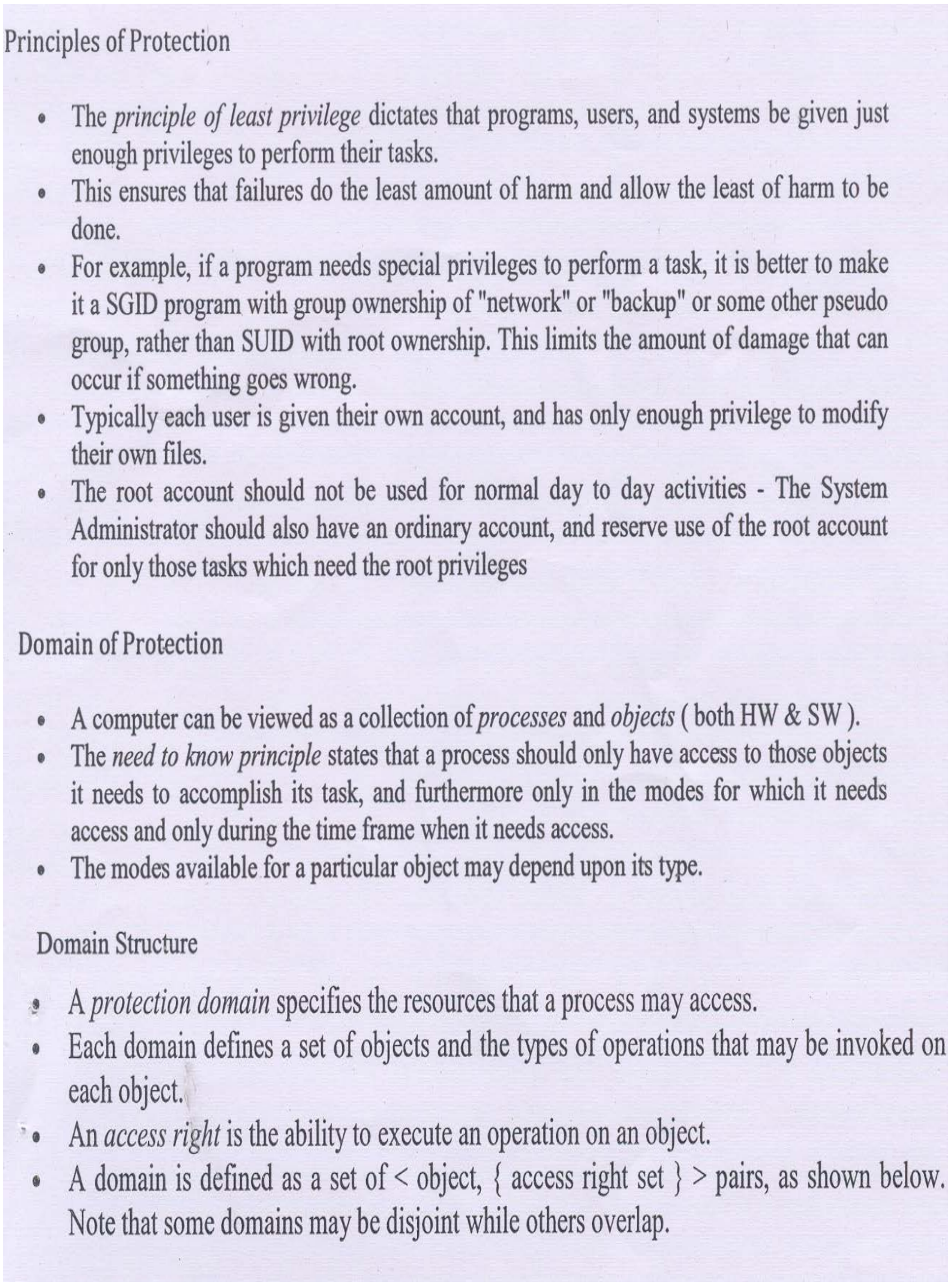
A useful way to think of a semaphore is as a record of how many units of a particular resource are available, coupled with operations to *safely* (i.e., without race conditions) adjust that record as units are required or become free, and if necessary wait until a unit of the resource becomes available. Semaphores are a useful tool in the prevention of race conditions; however, their use is by no means a guarantee that a program is free from these problems. Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores (same functionality that mutexes have).

The producer-consumer problem (also known as the bounded-buffer problem) is a classical example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

Q4 (b) Describe principle and domain of protection used to protect a file.

Answer



Principles of Protection

- The *principle of least privilege* dictates that programs, users, and systems be given just enough privileges to perform their tasks.
- This ensures that failures do the least amount of harm and allow the least of harm to be done.
- For example, if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership. This limits the amount of damage that can occur if something goes wrong.
- Typically each user is given their own account, and has only enough privilege to modify their own files.
- The root account should not be used for normal day to day activities - The System Administrator should also have an ordinary account, and reserve use of the root account for only those tasks which need the root privileges

Domain of Protection

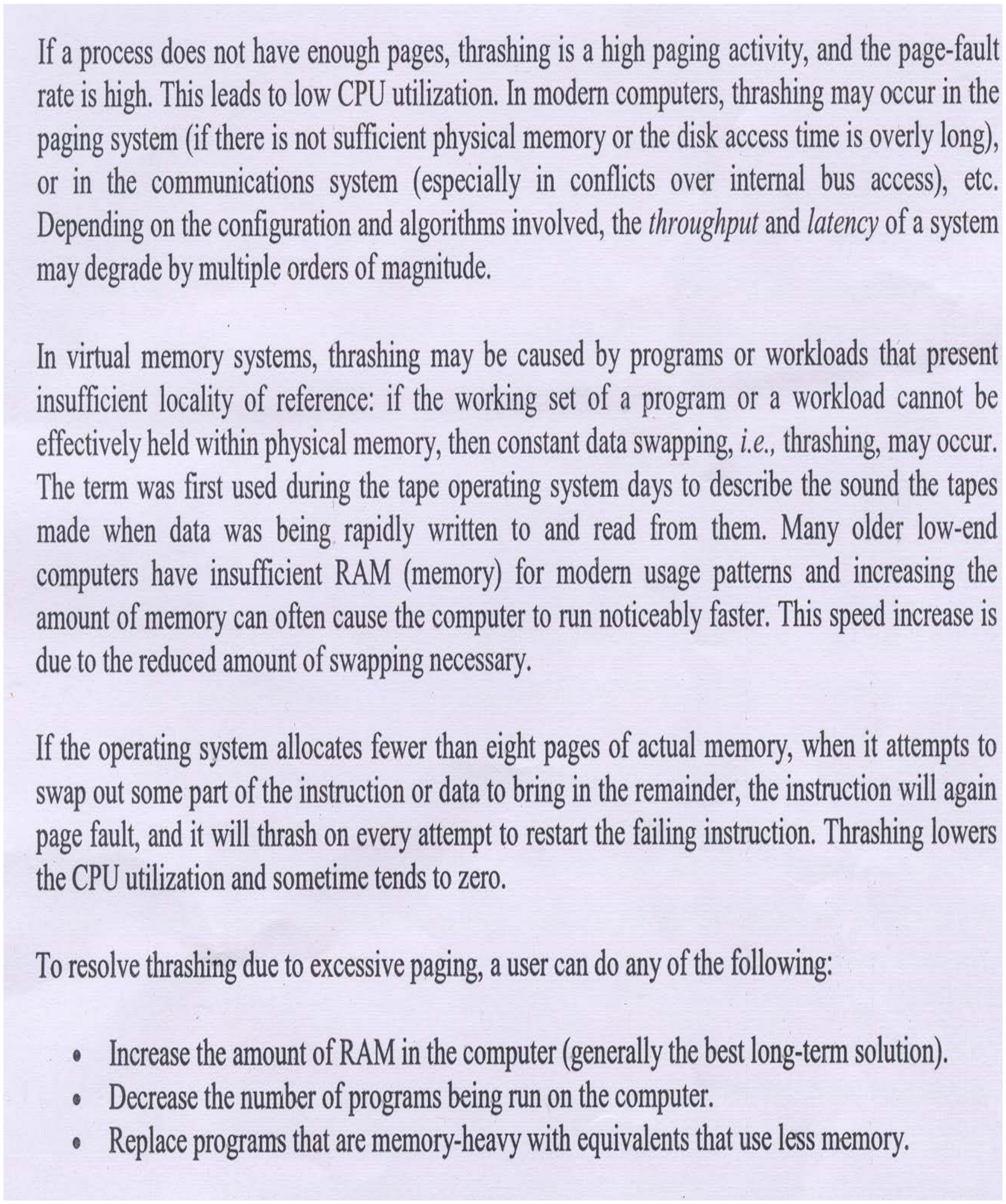
- A computer can be viewed as a collection of *processes* and *objects* (both HW & SW).
- The *need to know principle* states that a process should only have access to those objects it needs to accomplish its task, and furthermore only in the modes for which it needs access and only during the time frame when it needs access.
- The modes available for a particular object may depend upon its type.

Domain Structure

- A *protection domain* specifies the resources that a process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- An *access right* is the ability to execute an operation on an object.
- A domain is defined as a set of < object, { access right set } > pairs, as shown below. Note that some domains may be disjoint while others overlap.

Q5 (a) What is thrashing? When does it happen and how does it affect performance? What a usershould do to resolve thrashing due to excessive paging?

Answer



If a process does not have enough pages, thrashing is a high paging activity, and the page-fault rate is high. This leads to low CPU utilization. In modern computers, thrashing may occur in the paging system (if there is not sufficient physical memory or the disk access time is overly long), or in the communications system (especially in conflicts over internal bus access), etc. Depending on the configuration and algorithms involved, the *throughput* and *latency* of a system may degrade by multiple orders of magnitude.

In virtual memory systems, thrashing may be caused by programs or workloads that present insufficient locality of reference: if the working set of a program or a workload cannot be effectively held within physical memory, then constant data swapping, *i.e.,* thrashing, may occur. The term was first used during the tape operating system days to describe the sound the tapes made when data was being rapidly written to and read from them. Many older low-end computers have insufficient RAM (memory) for modern usage patterns and increasing the amount of memory can often cause the computer to run noticeably faster. This speed increase is due to the reduced amount of swapping necessary.

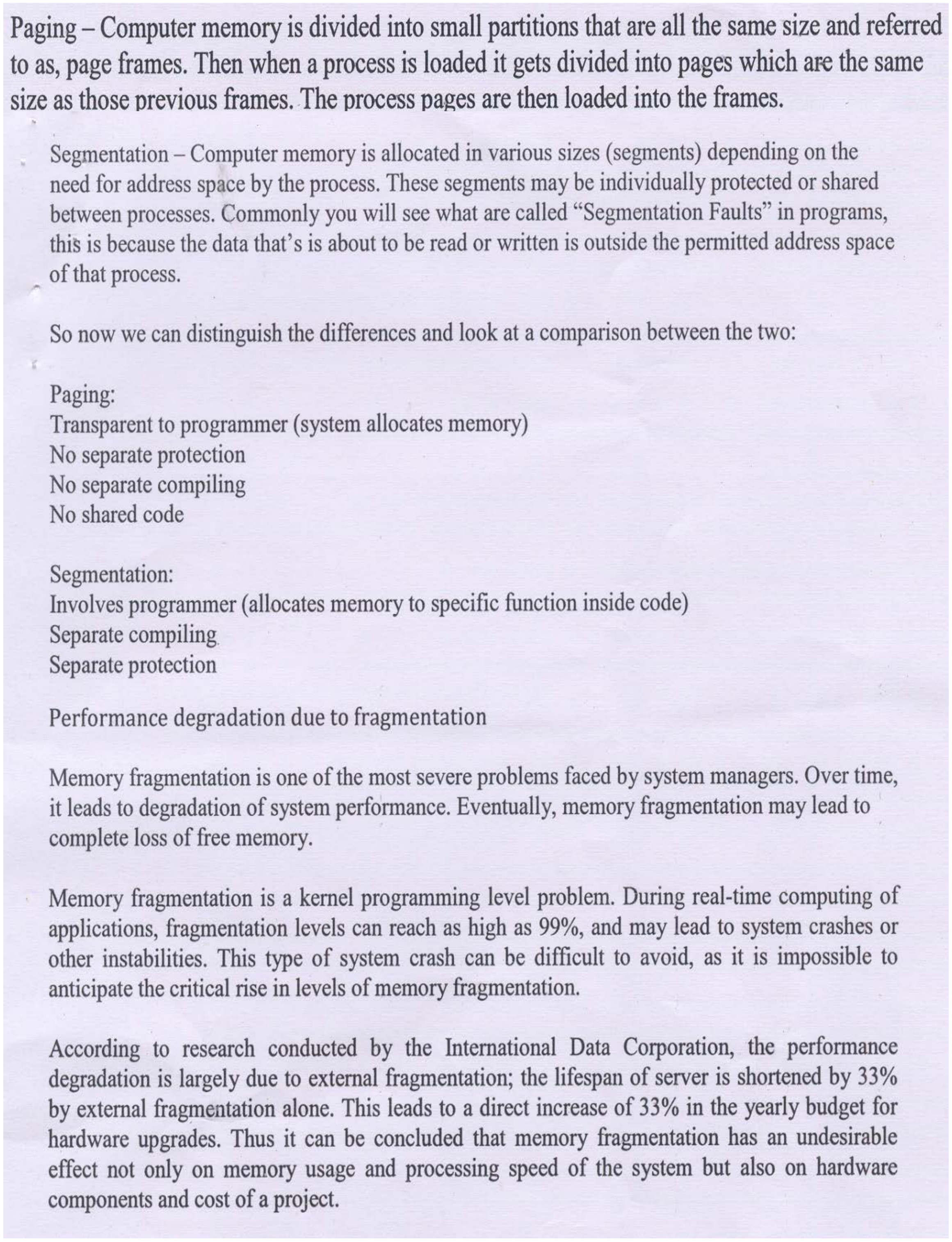
If the operating system allocates fewer than eight pages of actual memory, when it attempts to swap out some part of the instruction or data to bring in the remainder, the instruction will again page fault, and it will thrash on every attempt to restart the failing instruction. Thrashing lowers the CPU utilization and sometime tends to zero.

To resolve thrashing due to excessive paging, a user can do any of the following:

- Increase the amount of RAM in the computer (generally the best long-term solution).
- Decrease the number of programs being run on the computer.
- Replace programs that are memory-heavy with equivalents that use less memory.

Q5 (b) Compare and contrast the paging with segmentation.

Answer



Paging – Computer memory is divided into small partitions that are all the same size and referred to as, page frames. Then when a process is loaded it gets divided into pages which are the same size as those previous frames. The process pages are then loaded into the frames.

Segmentation – Computer memory is allocated in various sizes (segments) depending on the need for address space by the process. These segments may be individually protected or shared between processes. Commonly you will see what are called "Segmentation Faults" in programs, this is because the data that's is about to be read or written is outside the permitted address space of that process.

So now we can distinguish the differences and look at a comparison between the two:

Paging:
Transparent to programmer (system allocates memory)
No separate protection
No separate compiling
No shared code

Segmentation:
Involves programmer (allocates memory to specific function inside code)
Separate compiling
Separate protection

Performance degradation due to fragmentation

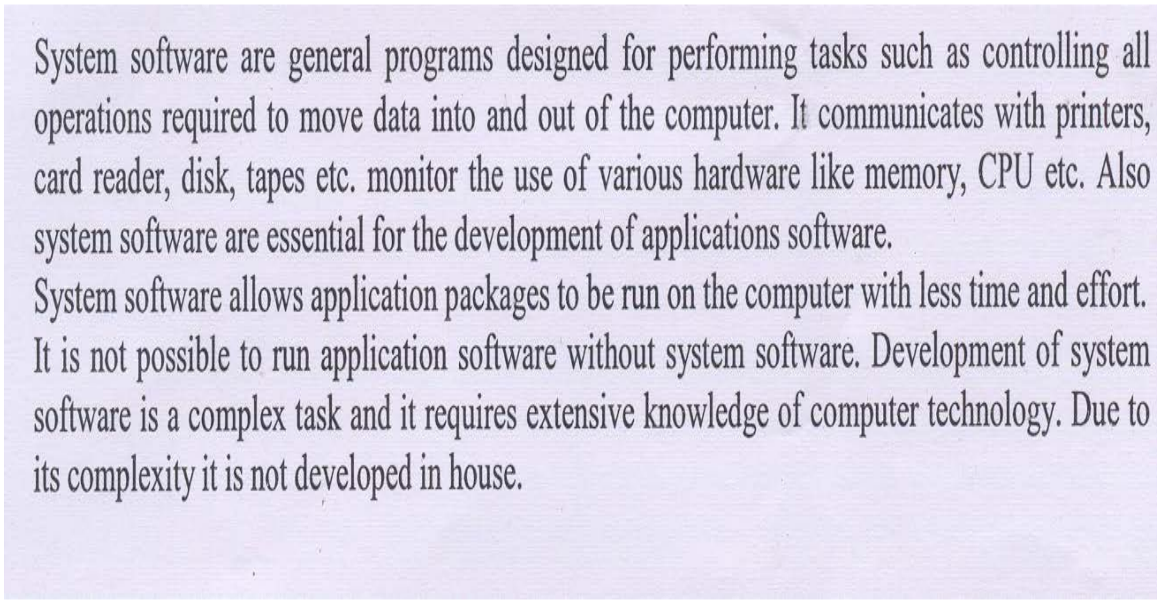
Memory fragmentation is one of the most severe problems faced by system managers. Over time, it leads to degradation of system performance. Eventually, memory fragmentation may lead to complete loss of free memory.

Memory fragmentation is a kernel programming level problem. During real-time computing of applications, fragmentation levels can reach as high as 99%, and may lead to system crashes or other instabilities. This type of system crash can be difficult to avoid, as it is impossible to anticipate the critical rise in levels of memory fragmentation.

According to research conducted by the International Data Corporation, the performance degradation is largely due to external fragmentation; the lifespan of server is shortened by 33% by external fragmentation alone. This leads to a direct increase of 33% in the yearly budget for hardware upgrades. Thus it can be concluded that memory fragmentation has an undesirable effect not only on memory usage and processing speed of the system but also on hardware components and cost of a project.

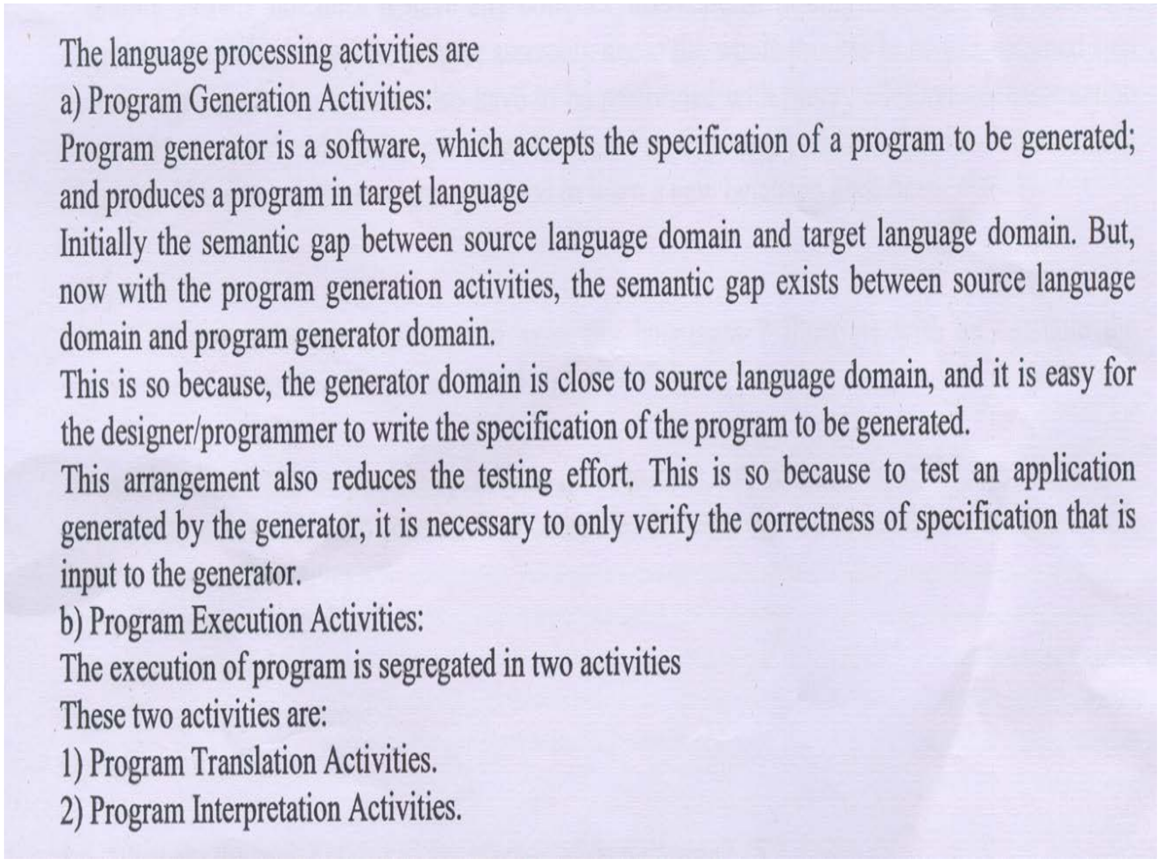
Q6 (b) What do you understand by the term System Software?

Answer


System software are general programs designed for performing tasks such as controlling all operations required to move data into and out of the computer. It communicates with printers, card reader, disk, tapes etc. monitor the use of various hardware like memory, CPU etc. Also system software are essential for the development of applications software.
System software allows application packages to be run on the computer with less time and effort. It is not possible to run application software without system software. Development of system software is a complex task and it requires extensive knowledge of computer technology. Due to its complexity it is not developed in house.

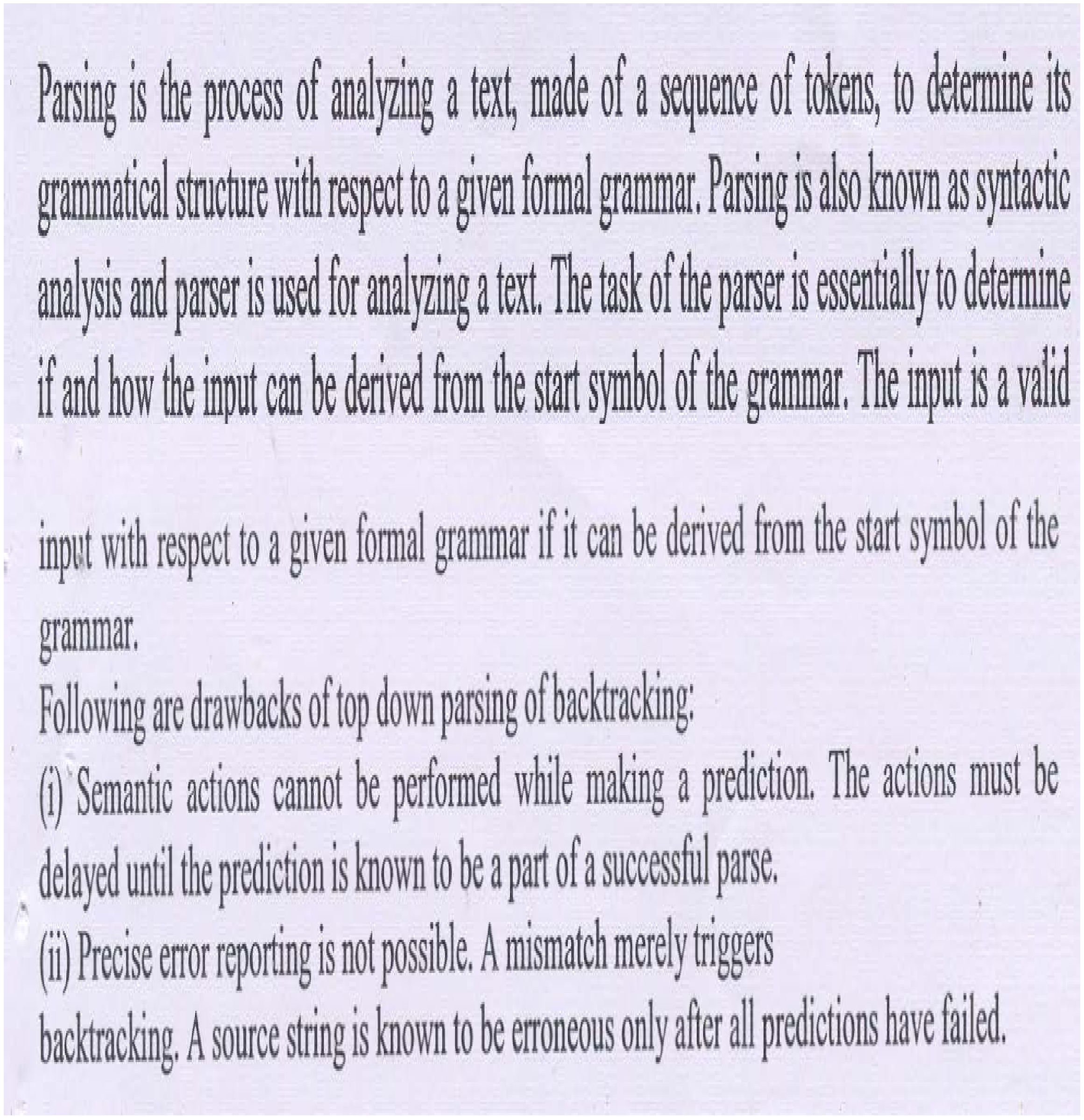
Q6 (c) What are the various *language processing* activities in the domain of system software ?

Answer


The language processing activities are
a) Program Generation Activities:
Program generator is a software, which accepts the specification of a program to be generated; and produces a program in target language
Initially the semantic gap between source language domain and target language domain. But, now with the program generation activities, the semantic gap exists between source language domain and program generator domain.
This is so because, the generator domain is close to source language domain, and it is easy for the designer/programmer to write the specification of the program to be generated.
This arrangement also reduces the testing effort. This is so because to test an application generated by the generator, it is necessary to only verify the correctness of specification that is input to the generator.
b) Program Execution Activities:
The execution of program is segregated in two activities
These two activities are:
1) Program Translation Activities.
2) Program Interpretation Activities.

Q7 (a) What is parsing? Write down the drawbacks of top down parsing with backtracking.

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 grammar. 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. The input is a valid

input with respect to a given formal grammar if it can be derived from the start symbol of the grammar.

Following are drawbacks of top down parsing of backtracking:

(i) Semantic actions cannot be performed while making a prediction. The actions must be delayed until the prediction is known to be a part of a successful parse.

(ii) Precise error reporting is not possible. A mismatch merely triggers backtracking. A source string is known to be erroneous only after all predictions have failed.

Q7 (b) Explain Nested Macro calls using suitable example.

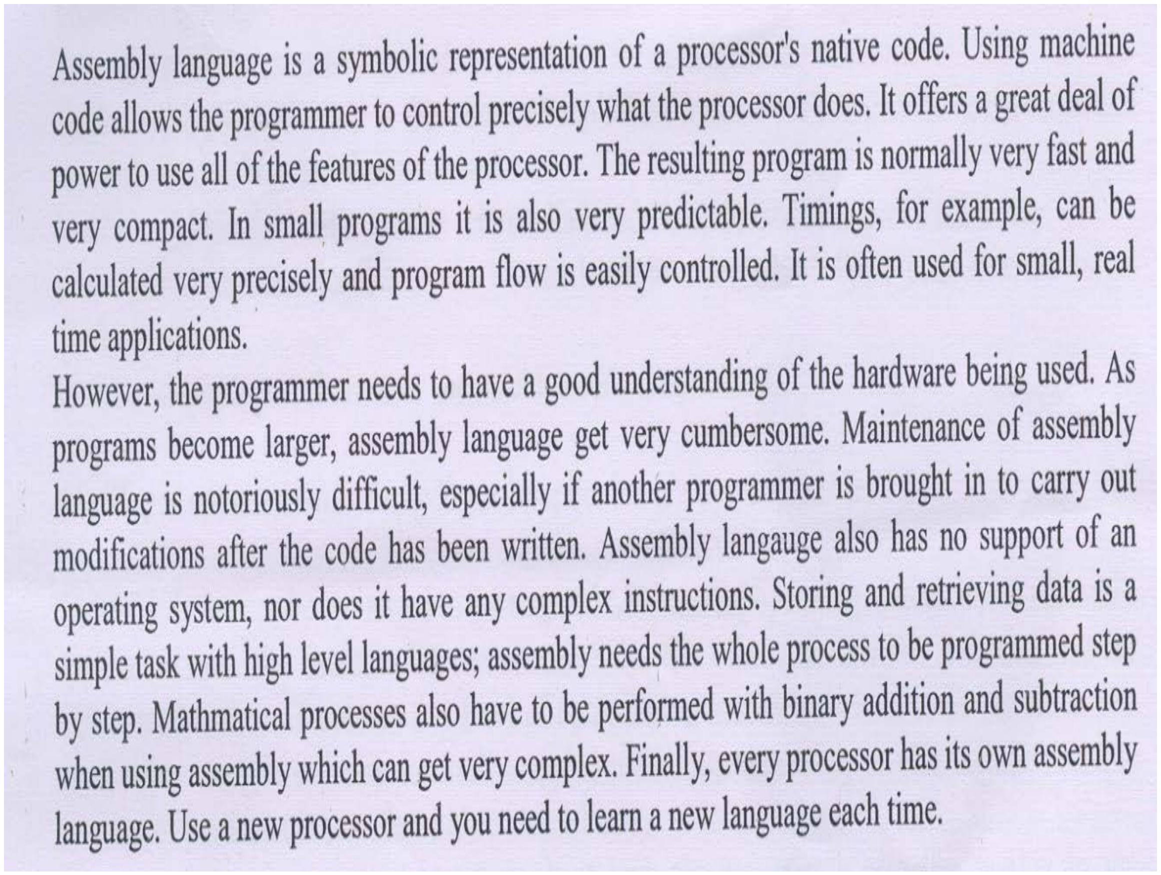
Answer Page Number 137 of Text Book

Q7 (c) Explain program relocation algorithm.

Answer Page Number 229 of Text Book

Q8 (a) Mention some advantages of assembly language over machine language.

Answer

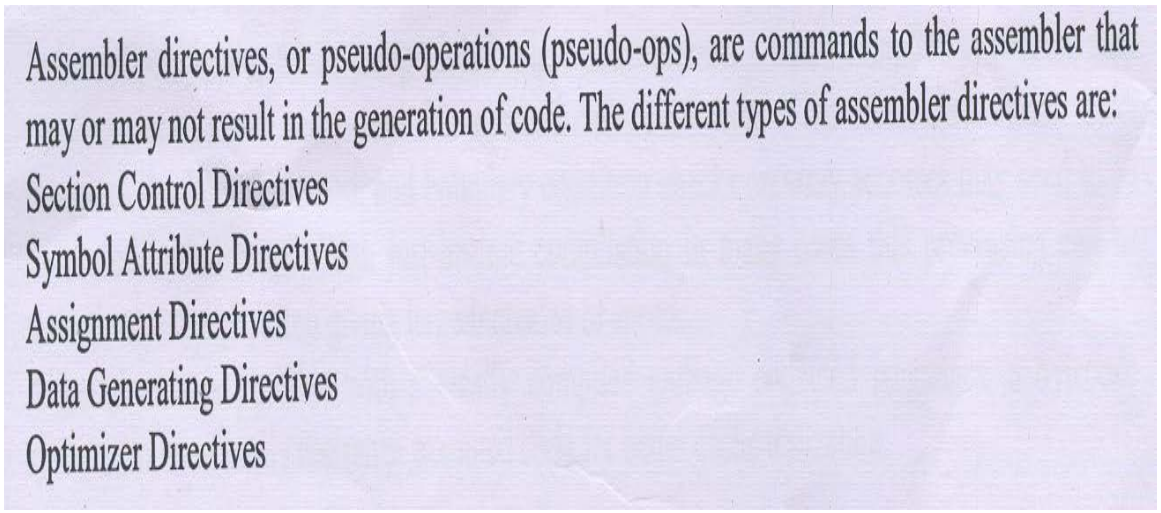


Assembly language is a symbolic representation of a processor's native code. Using machine code allows the programmer to control precisely what the processor does. It offers a great deal of power to use all of the features of the processor. The resulting program is normally very fast and very compact. In small programs it is also very predictable. Timings, for example, can be calculated very precisely and program flow is easily controlled. It is often used for small, real time applications.

However, the programmer needs to have a good understanding of the hardware being used. As programs become larger, assembly language get very cumbersome. Maintenance of assembly language is notoriously difficult, especially if another programmer is brought in to carry out modifications after the code has been written. Assembly langauge also has no support of an operating system, nor does it have any complex instructions. Storing and retrieving data is a simple task with high level languages; assembly needs the whole process to be programmed step by step. Mathmatical processes also have to be performed with binary addition and subtraction when using assembly which can get very complex. Finally, every processor has its own assembly language. Use a new processor and you need to learn a new language each time.

Q8 (b) What are *assembler directives* in assembly languages? Explain using suitable examples.

Answer



Assembler directives, or pseudo-operations (pseudo-ops), are commands to the assembler that may or may not result in the generation of code. The different types of assembler directives are:

Section Control Directives

Symbol Attribute Directives

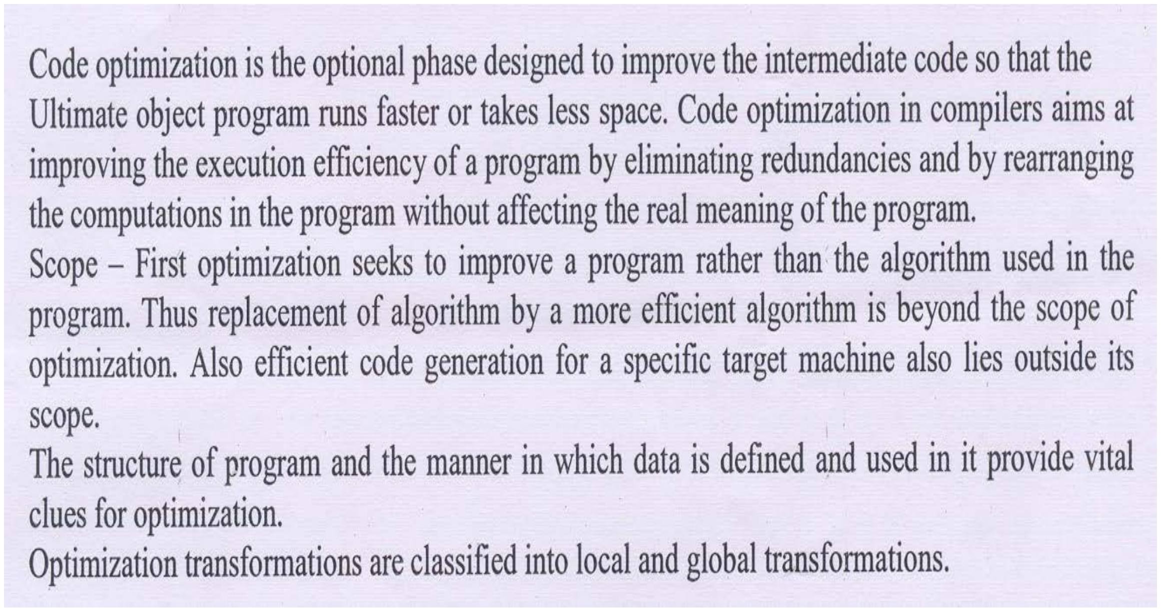
Assignment Directives

Data Generating Directives

Optimizer Directives

Q9 (a) Write short note on code optimization.

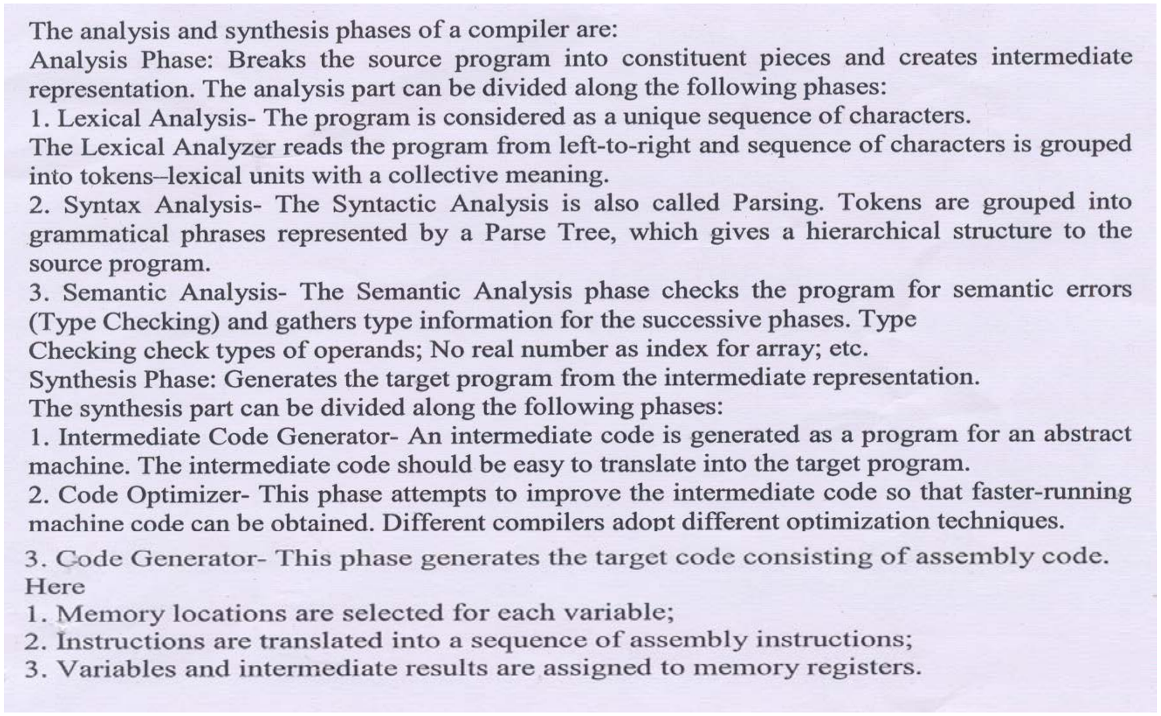
Answer



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.

Q9 (b) Explain analysis and synthesis phase of a compiler.

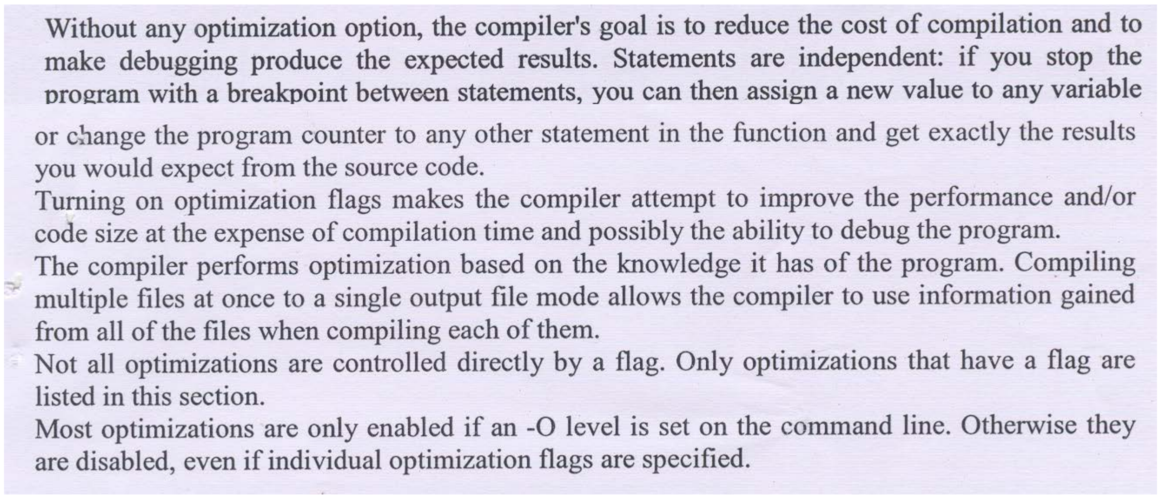
Answer



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.

Q9 (c) Which kind of optimisation is more effective inside loops - space optimisation or time optimisation? Why?

Answer



Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code.
Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program.
The compiler performs optimization based on the knowledge it has of the program. Compiling multiple files at once to a single output file mode allows the compiler to use information gained from all of the files when compiling each of them.
Not all optimizations are controlled directly by a flag. Only optimizations that have a flag are listed in this section.
Most optimizations are only enabled if an -O level is set on the command line. Otherwise they are disabled, even if individual optimization flags are specified.

<u>Text Book</u>

Systems Programming & Operating Systems, D.M. Dhamdhere, Tata Mc Graw-Hill, II Revised Edition, 2005