Q.2 a. Define process. What are the states of 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 run able but temporarily stopped running to let another process run.
- Terminated state: The process has finished execution.

b. Explain the spooling technology in details.

Answer:

"Simultaneous peripheral operations online". a computer document or task list 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.

c. Explain the following: (4×2)

(i) Distributed System
(ii) Parallel System
(iii) Real Time System
(iv) Threads

Answer:

A distributed operating system is software over a collection of independent, networked, communicating, and physically separate computational nodes. Individual nodes each hold a specific software subset of the global aggregate operating system. Each subset is a composite of two distinct services provisionary. The first is a ubiquitous minimal kernel, or microkernel, that directly controls that node's hardware. Second is a higher-level collection of system management components that coordinate the node's individual and collaborative activities. These components abstract microkernel functions and support user applications.

The microkernel and the management components collection work together. They support the system's goal of integrating multiple resources and processing functionality into an efficient and stable system.[4] This seamless integration of individual nodes into a global system is referred to as transparency, or single system image; describing the illusion provided to users of the global system's appearance as a single computational entity.

Parallel operating systems are used to interface multiple networked computers to complete tasks in parallel. The architecture of the software is often a UNIX-based platform, which allows it to coordinate distributed loads between multiple computers in a network. Parallel operating systems are able to use software to manage all of the different resources of the computers running in parallel, such as memory, caches, storage space, and processing power. Parallel operating systems also allow a user to directly interface with all of the computers in the network.

A parallel operating system works by dividing sets of calculations into smaller parts and distributing them between the machines on a network. To facilitate communication between the processor cores and memory arrays, routing software has to either share its memory by assigning the same address space to all of the networked computers, or distribute its memory by assigning a different address space to each processing core. Sharing memory allows the operating system to run very quickly, but it is usually not as powerful. When using distributed shared memory, processors have access to both their own local memory and the memory of other processors; this distribution may slow the operating system, but it is often more flexible and efficient.

A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time application requests. It must be able to process data as it comes in, typically without buffering delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter.

A key characteristic of an RTOS is the level of its consistency concerning the amount of time it

takes to accept and complete an application's task; the variability is jitter. A hard real-time operating system has less jitter than a soft real-time operating system. The chief design goal is not high throughput, but rather a guarantee of a soft or hard performance category. An RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS.

An RTOS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more frequently dedicated to a narrow set of applications. Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency; a real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

Q.3a. 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 allows the process to run through to completion before moving onto the next task. 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.

b.What do you mean by deadlock avoidance? Write the Banker's algorithm for multiple resources.

Answer: Page 391-92 of text book

Q.4a. Explain and also write the code for Producer-Consumer problem using Semaphore.

Answer:

The producer-consumer 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.

b.Describe the different mechanisms 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.

Q.5 a. What is memory allocation? Differentiate between contiguous and non contiguous memory allocation. Explain the concept of virtual memory.

Answer:
Contiguous memory allocation is a classical memory allocation model that assigns a process consecutive memory blocks (that is, memory blocks having consecutive addresses).

Contiguous memory allocation is one of the oldest memory allocation schemes. When a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process. If sufficient contiguous memory is found, the process is allocated memory to start its execution. Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available.

Virtual memory is a feature of an operating system that enables a process to use a memory (RAM) address space that is independent of other processes running in the same system, and use a space that is larger than the actual amount of RAM present, temporarily relegating some contents from RAM to a disk, with little or no overhead.
In a system using virtual memory, the physical memory is divided into equally-sized pages. The memory addressed by a process is also divided into logical pages of the same size. When a process references a memory address, the memory manager fetches from disk the page that includes the referenced address, and places it in a vacant physical page in the RAM. Subsequent references within that logical page are routed to the physical page. When the process references an address from another logical page, it too is fetched into a vacant physical page and becomes the target of subsequent similar references.

b.Compare and contrast paging with segmentation. In particular, describe issues related to fragmentation. (8)

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.

Q.6a. What are the benefits of using "language processors"? (5)

Answer:
Natural language processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and human (natural) languages; it began as a branch of artificial intelligence. In theory, natural language processing is a very attractive method of human–computer interaction. Natural language understanding is sometimes referred to as an AI-complete problem because it seems to require extensive knowledge about the outside world and the ability to manipulate it.

Whether NLP is distinct from, or identical to, the field of computational linguistics is a matter of perspective. The Association for Computational Linguistics defines the latter as focusing on the theoretical aspects of NLP. On the other hand, the open-access journal "Computational Linguistics", styles itself as "the longest running publication devoted exclusively to the design and analysis of natural language processing systems"

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.

c. What are the various language processing activities in the domain of system software? What do you understand by cross-compilation?

Answer:
The language processing activities are
a) Program Generation Activities:
1) Program generator is a software, which accepts the specification of a program to be generated; and produces a program in target language
2) 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.
3) 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.
4) 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.

Q.7 a. Explain the difference between scanning and parsing.

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.
A parser is one of the components in an interpreter or compiler, which checks for correct syntax and builds a data structure implicit in the input tokens.

A scanner is a device that optically scans images, printed text, handwriting, or an object, and converts it to a digital image. Common examples found in offices are variations of the desktop (or flatbed) scanner where the document is placed on a glass window for scanning.

b.Explain the following:
(i) Macro definition
(ii) Macro call

Answer: Page 132, 133 of text book

c. Briefly describe why programming language influence the linking requirements of programs?

Answer: Page 230 of text book

Q.8 a. Mention some advantages of assembly language over machine language.

Answer:
Stages from Source to Executable
Compilation: source code
Linking: many relocatable binaries
Loading: relocatable
Execution: control is transferred to the first instruction of the program

At compile time, absolute addresses of variables and statement labels are not known.
In static languages, absolute addresses are bound at load time (LT).
In block-structured languages, bindings can change at run time (RT).
Phases of the Compilation Process
Lexical analysis (scanning): the source text is broken into tokens.
Syntactic analysis (parsing): tokens are combined to form syntactic structures, typically represented by a parse tree.
The parser may be replaced by a syntax-directed editor, which directly generates a parse tree as a product of editing.
Semantic analysis: intermediate code is generated for each syntactic structure.
Type checking is performed in this phase. Complicated features such as generic declarations and operator over loading are also processed.
Machine-independent optimization: intermediate code is optimized to improve efficiency.
Code generation: intermediate code is translated to relocatable object code for the target machine.
Machine-dependent optimization: the machine code is optimized.

b. What are assembler directives in assembly languages? Illustrate with an example the importance of assembler directives.

Answer:
The compilation can be optimized to the targeted CPU and the operating system model where the application runs. For example JIT can choose SSE2 CPU instructions when it detects that the CPU supports them. To obtain this level of optimization specificity with a static compiler, one must either compile a binary for each intended platform/architecture, or else include multiple versions of portions of the code within a single binary.

The system is able to collect statistics about how the program is actually running in the environment it is in, and it can rearrange and recompile for optimum performance. However, some static compilers can also take profile information as input.

The system can do global code optimizations (e.g. inlining of library functions) without losing the

advantages of dynamic linking and without the overheads inherent to static compilers and linkers. Specifically, when doing global inline substitutions, a static compilation process may need run-time checks and ensure that a virtual call would occur if the actual class of the object overrides the inlined method, and boundary condition checks on array accesses may need to be processed within loops. With just-in-time compilation in many cases this processing can be moved out of loops, often giving large increases of speed.

Although this is possible with statically compiled garbage collected languages, a bytecode system can more easily rearrange executed code for better cache utilization.

c. Explain the differences between two pass and single pass translation.

Answer: Page 94 of text book

Q.9 a. What are the major stages in the process of compilation?

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.

b. Write short note on code optimization.

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.

c. Compare and contrast the following parameter passing mechanisms in terms of execution efficiency and power to produce side effects:
(i) call by value-result
(ii) call by reference
(iii) call by name

Answer: Page 196-198 of text book

TEXTBOOK

1. Systems Programming and Operating Systems, D. M. Dhamdhere, Tata McGraw-Hill, Second Revised Edition, 2005 (TB-I)