Q 2 (a) Compare and contrast Multiprogramming system with time sharing system.

Answer

Multiprogramming vs Time Sharing Systems: Multiprogramming is the allocation of more than one concurrent program on a computer system and its resources. Multiprogramming allows using the CPU effectively by allowing various users to use the CPU and I/O devices effectively. Multiprogramming makes sure that the CPU always has something to execute, thus increases the CPU utilization. On the other hand, Time sharing is the sharing of computing resources among several users at the same time. Since this will allow a large number of users to work in a single computer system at the same time, it would lower the cost of providing computing capabilities.

Difference between Multiprogramming System and Time Sharing System: Main difference between multiprogramming and time sharing is that multiprogramming is the effective utilization of CPU time, by allowing several programs to use the CPU at the same time but time sharing is the sharing of a computing facility by several users that want to use the same facility at the same time. Each user on a time sharing system gets her own terminal and gets the feeling that she is using the CPU alone. Actually, time sharing systems use the concept of multiprogramming to share the CPU time between multiple users at the same time.

Q2 (b) With the help of suitable diagram, list the various elements of a Process Control Block.

Answer Process Page No.320 and 325 of textbook.

Q3 (a) What is process scheduling? Explain the different sub-functions of process scheduling.

Answer

Scheduling is a key part of the workload management software which usually perform some or all of:

- Queuing
- Scheduling
- Monitoring
- Resource management
- Accounting

The difficult part of scheduling is to balance policy enforcement with resource optimization in order to pick the *best* job to run. Essentially one can think of the

scheduler performing the following loop:

- Select the *best* job to run, according to policy and available resources.
- Start the job.
- Stop the job and/or clean up after completion.
- repeat.

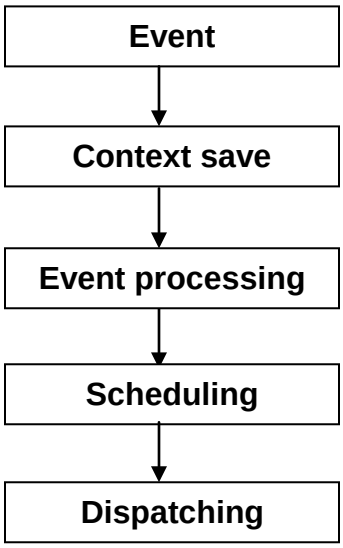
Process scheduling consists of the following sub-functions:

1. Scheduling: Selects the process to be executed next on the CPU. The scheduling function uses information from the PCB's and selects a process based on the scheduling policy in force.

2. Dispatching: Sets up execution of the selected process on the CPU. This function involves setting up the execution environment of the selected process, and loading information from the PSR and registers fields of the PCB into the CPU.

3. Context save: Saves the status of a running process when its execution is to be suspended. This function performs housekeeping whenever a process releases the CPU or is pre-empted.

Use of scheduling sub functions: Occurrence of an event invokes the context save function. The kernel now processes the event that has occurred. The scheduling function is now invoked to select a process for execution on the CPU. The dispatching function arranges execution of the selected function on the CPU.The dispatching function arranges execution of the selected function on the CPU.



Q 3 (b) Define deadlock. Explain the conditions that are required for a deadlock to occur.

Answer Page Number 376-377 of Textbook

Q 4 (a) Explain critical section problem in relation to process synchronization. List various requirements that critical section problem solution must satisfy.

Answer

Critical Section Problem:

- Consider a system consisting of *n* processes {Po, P_1, P_{n-1}.} Each process has a segment of code, caIled a *critical sectio'*. .in which the process 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. The critical-section problem is to design a protocol at the processes can use to cooperate. Each process request permission to enter its critical section.
- The section of code implementing this request is the *entry* section. The critical section may be followed by an exit section. The remaining code is the remainder section·

Solutions to Critical Section Problem:

A solution to the critical-section problem must satisfy the following three requirements:

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 sections, then only those processes that are not executing in their remainder section can participate in the decision of which will enter its critic a section next and this selection cannot be postponed indefinitely.

3. Bounded Waiting: There exist a bound 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.

Q 4 (b) Explain with the help of examples, the two disk allocation methods: linked and indexed.

Answer Page Number 510 of textbook.

Q 5 (a) Using suitable example, explain any two page replacement algorithms.

Answer Page Number 471-479 of Text Book

Q5 (b) List various approaches used for realization of virtual memory. List advantages and disadvantages of virtual memory.

Answer Page Number 481 of Text Book

Q6 (a) What do you mean by language processing? Describe language processing activities.

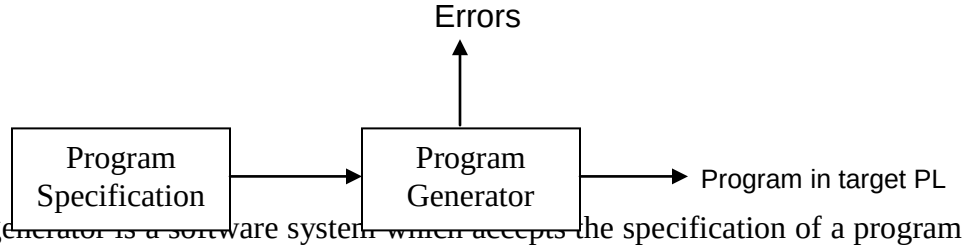
Answer

A language processor is software which bridges a specification or execution gap. Language processing describes the activity performed by a language processor. Input program to a language processor is known as source program and the output program of a language processor is called a target program.

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

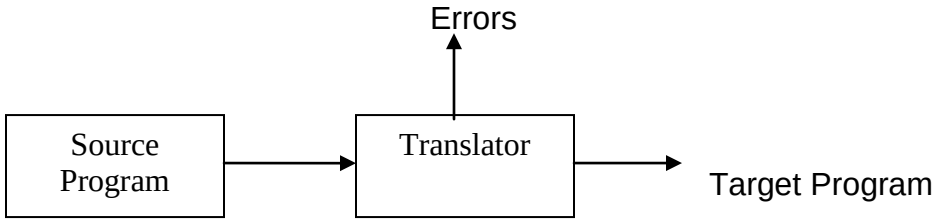
introduces a new domain between the application and PL domains. We call this the program generator domain. The specification gab is now gap between the application domain and the program generator domain. This gab is smaller than the gap between the application domain and PL domain.

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

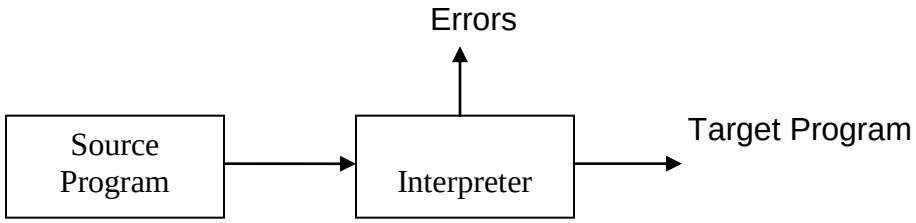
1. Translation

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.



Q6 (b) How the data structures used for language processors are classified? Explain.

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) Define Parsing. What are the goals of parsing? Explain its various types.

Answer

Parsing: Source programmed statements are regarded as tokens, building block of language the task of scanning the source statement, recognizing and classifying the various tokens is known as lexical analysis. The part of the compiler that performs this task is commonly called a scanner. After the token scan, each statement in the program must be recognized as some language constructs, such as declaration or an assignment statement described by the grammar. This process is called Syntactic analysis or parsing and is performed by the part of compiler called parser.

Goals:

1. To check the validity of a source string, and
2. To determine the syntactic structure of a source string.

   i,e. given an input program:

   a. Find all syntax errors; for each, produce an appropriate diagnostic message, and recover quickly.
   b. Produce the parse tree, or at least a trace of the parse tree, for the program

Types of Parsing:

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.

Q7 (b) What is macro-expansion? List the key notions concerning macro expansion. Write an algorithm to outline the macro-expansion using macro-expansion counter.

Answer

macro call leads to macro expansion. During macro expansion, the macro call statement is replaced by a sequence of assembly statements. Two key notions concerning macro expansion are:

1.**Expansion time control flow-** this determines the order in which model statements are visited during macro expansion.

2.**Lexical substitution:** Lexical substitution is used to generate an assembly statement from a modal statement.

The flow of control during macro expansion can be implemented using a macroexpansion counter (MEC). The outline of algorithm is as follows:

1. MEC:=statement number of first statement following the prototype statement;

2. While statement pointed by MEC is not a MEND statement

(a) If a model statement then

(i) expand the statement.

(ii) MEC:=MEC+1;

(b) Else (i.e. a pre processor statement)

(i) MEC:=new value specified in the statement;

3. Exit from macro expansion**.**

Q8 (a) What is assembly language? What kinds of statements are present in an assembly language program? Discuss.

Answer

Assembly language is a family of low-level language for programming computers, microprocessors, microcontrollers etc. They implement a symbolic representation of the numeric machine codes and other constants needed to program a particular CPU architecture. This representation is usually defined by the hardware manufacturer, and is based on abbreviations (called mnemonic) that help the programmer remember individual instruction, register etc. Assembly language programming is writing machine instructions in mnemonic form, using an assembler to convert these mnemonics into actual processor instructions and associated data.

An assembly program contains following three kinds of statements:

1. **Imperative statements**: These indicate an action to be performed during execution of the assembled program. Each imperative statement typically translates into one machine instruction.

2. **Declaration statements**: The syntax of declaration statements is as follows:

[Label] DS<constant>

[Label] DC '<value>'

The DS statement reserves areas of memory and associates names with them.

The DC statement constructs memory words containing constants.

3. **Assembler directives**: These instruct the assembler to perform certain actions during the assembly of a program. For example

START <constant> directive indicates that the first word of the target program generated by the assembler should be placed in the memory word with address <constant>.

Advantages of assembly language program:

- reduced errors
- faster translation times

□□ changes could be made easier and faster

Q8 (b) Explain the stepwise approach to arrive at a design specification for an assembler.

Answer

Four step approach is used to arrive at a design specifications for an assembler:

Step 1: Identify the information necessary to perform a task.

Step 2: Select a suitable data structure to hold the information.

Step 3: Determine the processing necessary to maintain the information in the data structure.

Step 4: Determine the processing necessary to perform the task.

The fundamental information requirements are determined by the synthesis phase of assembly. Hence it is the best to begin by considering synthesis of the target program. We then consider how best this information can be made available i.e. whether it should be collected during analysis or generated during the synthesis phase.

Q 9 (a) Define and explain memory allocation. What are different approaches of memory allocation?

Answer

Memory Allocation:-

- The run time representation of a variable is a string of bits in some memory word or CPU register. The value of the variable is determined from the bit string in accordance with the variable's type. A compiler ensures type integrity by generating type specific code, i.e. the value of a variable of $type_i$ is only manipulated through instructions which know values of $type_i$ are represented.
- Each instruction is type specific, i.e. it expects the operand to be of a specific type. Type integrity is guaranteed by fact that the choice of the instruction opcode is based on the type of its operand.

Memory allocation involves the following tasks:-

1. Determine the amount of memory required to represent the value of a data item.

2. Use an appropriated memory allocation model to implement the lifetimes and scoped of data items.
3. Determine appropriate memory mapping to access the values in a non scalar data item, e.g. values in an array.

The first task is simple since semantic analysis of data declaration statement would have extracted all necessary information.

Static and Dynamic Allocation:-

- A memory binding is an association between the 'memory address' attribute of a data item and the address of a memory area.
- Memory allocation is the procedure used to perform memory binding. The binding ceases to exist when memory is deallocated. Two forms of memory binding are static and dynamic binding. The corresponding memory allocation models are called static and dynamic memory allocation. In static memory allocation, memory is allocated to a variable before the execution of a program begins. This is typically performed during compilation. Thus no memory allocation/deallocation takes place during the execution of a program, and variables remain permanently allocated irrespective of their accessibility at any point in a program's execution. For example, allocation to a variable exists even if the program unit in which it is defined is not active. In dynamic memory allocation memory binding are established and destroyed during the execution of a program. Typical examples of the use of these memory allocation models are Fortran for static allocation and block structured language like PL/I, Pascal, Ada, etc., for dynamic allocation.
- Dynamic allocation is of two types namely:
    1. Automatic allocation/deallocation and
    2. Program controlled allocation/deallocation
    - The automatic allocation/deallocation implies memory binding performed at execution init time of a program unit, while the Program controlled allocation/deallocation implies memory binding performed during the execution of a program unit.

- In automatic dynamic allocation, memory is allocated to the variable - declared in a program unit when the program unit is entered during execution and the same is de allocated when the program unit exited. Thus the same memory area may be used for the variables of different program units. In program controlled dynamic allocation, a program can allocate/deallocate memory at arbitrary points during execution.
- Dynamic memory allocation is implemented using stacks and heaps, thus necessitating pointer based access to variables. This tends to make it slower in execution than static memory allocation. Automatic dynamic allocation is implemented using a stack since entry /exit from program units is LIFO in nature. When a program unit is entered during the execution of a program, a record is created in the stack to contain its variables. A pointer is set to point to this record. Individual variables of the program unit are accessed using displacements from this pointer. Program controlled dynamic allocation is implemented using a heap. A pointer is needed to point to each allocated memory area.

Q 9 (b) Explain various parameter passing techniques.

Answer

Parameter passing:-

Language rules for parameter passing define the semantics of parameter usage inside a function, thereby defining the kind of side effects a function can produce on its actual parameter.

1. **Call by value:-**
   - In this mechanism, values of actual parameter are passed to the called function. These values are assigned to the corresponding formal parameters. The passing of values only takes place in one direction-from the value of a formal parameter, the change is not reflected on the corresponding actual parameter. Thus a function cannot produce any side effects on its parameters.

- Call by value is commonly used for built in functions of language. Its main advantage is its simplicity. A called function may allocate memory to a formal parameter and copy the value of the actual parameter into this location at every call. During execution, the function need not distinguish between a formal parameter and a local variable. This mechanism is very efficient if parameters are scalar variables.

Call by value-result: This mechanism extends the capabilities of the call by value mechanism by copying the values of formal parameters back into corresponding actual parameters at return. Thus, side effects are realized at return. This mechanism inherits the simplicity of the call by value mechanism.

2. **Call by reference: -** In this mechanism, the address of an actual parameter is passed to the called function. If the parameter is an expression, its value is computed and stored in a temporary location and the address of the temporary location is passed to the called function. If the parameter is an array element, its address is similarly computed at the time of call. The parameter list is thus a list of addresses of actual parameters. At every access of a formal parameter in the function, the address of the corresponding actual parameter is obtained from the parameter list. This used as the address of the formal parameter.
3. **Call by name:-** This parameter transmission mechanism has the same effect as if every occurrence of a formal parameter in the body of the called function is replaced by the name of the corresponding actual parameter.

Text Book

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