B.E.

Sixth Semester Examination, Dec-2008

Systems Programming & System Administration (IT-303-E)

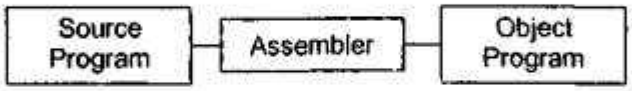
Note : Attempt any five questions.

Q. 1. What is system programming? Discuss various components of system programming How these components are helpful? List out various advantages and disadvantages of each component in respect/context to programming environment. **20**

Ans. System programming refers to the mechanism of developing programs and making coordination between these programs in order to make the system functional.

Following are the components of system programming :

(i) Assemblers : Programmers found it difficult to write or read programs in machine language. In their quest for a more convenient language they began to use a mnemonic (symbol) for each machine instruction, which they would subsequently translate into machine language. Such a mnemonic machine language is now called an assembly language. Programs known as assemblers were written to automate the translation of assembly language into machine language. The input to an assembler program is called the source program, the output is a machine language translation object program.



(ii) Loaders : A loader is a program that places programs into memory and prepares them for execution. In simple loading scheme, the assembler outputs the machine language translation of a program on a secondary storage device and a loader is placed in core.

It would be more efficient if subroutines could be translated into an object from that the loader could "relocate" directly behind the user's program. The task of adjusting programs so they may be placed in arbitrary core locations is called relocation. Relocating loaders perform four functions :

(i) Allocate space in memory for the programs (allocation)

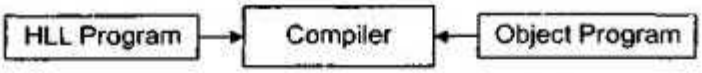
(ii) Resolve symbolic reference between object [illegible]king).

(iii) Adjust all address-dependent locations, such as address constants, to correspond to the allocated space (relocation).

(iv) Physically place the machine instructions and [illegible] memory (loading).

(iii) Macros : To relieve programmers of the need to repeat identical parts of their program, operating systems provide a macro processing facility, which permits the programmer to define an abbreviation for a part of his program and to use the abbreviation in his program. The macro processor treats the identical parts of the program defines by the abbreviation as a macro definition and saves the definition.

(iv) Compilers : A compiler is a program that accepts a program written in a high level language and produces an object program. A compiler compiles the program in one go and display results at the end. Compilers are faster than interpreters.



(v) Formal Systems : A formal system is an uninterpreted calculus. It consists of an alphabet, a set of words called axioms, and a finite set of relations called rules of inference. Formal systems are becoming important in the design, implementation and study of programming languages. Specifically, they can be used to specify the syntax and the semantics of programming languages. They have been used in syntax-directed compilation, compiler verification and complexity studies of languages.

Q. 2. What is Compiler? Describe compilation process. Also discuss incremental compiler. 20

Ans. A compiler is a program that accepts the program written in high level language and converts it into an equivalent machine level language program.

The compilation process takes place in following phases :

(i) Lexical Phase : The three tasks of the lexical analysis phase are :

(i) To parse the source program into the basic elements or tokens of the language.

(ii) To build a literal table and an identifier table.

(iii) To build a uniform symbol table.

(ii) Syntax Phase : The function of the syntax phase is to recognize the major constructs of the language and to call the appropriate action routines that will generate the intermediate form or matrix for these constructs. In some compilers this phase is implemented by one large program that recognize each construct.

(iii) Interpretation Phase : The interpretation phase is typically a collection of routines that are called when a construct is recognized in the syntactic phase. The purpose of these routines called action routines is to create an intermediate form of the source program and add information to the identifier table.

The separation of the syntactic phase from the interpretation phase is a logical division. The former phase recognizes syntactic constructs while the latter interprets the precise meaning into the matrix or identifier table.

(iv) Optimization : The two types of optimization is performed by the compiler-machine-dependent and machine-independent.

Machine-dependent optimization is so intimately related to the instructions that get generated that it was incorporated into the code generation phase, whereas machine-independent optimization was done in a separate optimization phase.

(v) Storage Assignment : The purpose of this phase is to :

(i) Assign storage to all variable referenced in the source program.

(ii) Assign storage to all temporary locations that are necessary for intermediate results. The storage references were reserved by the interpretation phase and did not appear in the source code.

(iii) Assign storage to literals.

(iv) Ensure that the storage is allocated and appropriate locations are initialized.

(vi) Code Generation : The purpose of the code generation phase is to produce the appropriate code. The code generation phase has the matrix as input. It uses the code productions which define the operators that may appear in the matrix to produce code. It also references the identifier tables and literal tables in order to generate proper address and code conversions.

(vii) Assembly Phase : The task of the assembly phase depends on how much has been done in code generation. If a lot of work has been done in code generation, then the assembly phase must resolve label references in the object program, format the object deck, and format the appropriate information for the loader. At the other extreme, if code generation has simply generated symbolic machine instructions and labels, the assembly phase must :

(i) Resolve label references.

(ii) Calculate addresses.

(iii) Generate binary machine instructions.

(iv) Generate storage, convert literals.

Incremental Compiler : Incremental compiler is part of a running program that uses that compiler. This allows new program items to be compiled at anytime, either extending the previously compiled programs or replacing some parts of the program. Because an incremental compiler is part of the runtime system, source code can be read in at anytime, from the terminal, from a file or possibly from a data structure constructed by the running program and translated into a machine block or function and the newly compiled program fragment is then immediately available for use by running system.

Q. 3. What is Macro? Differentiate macro language and macro instruction? Discuss feature of macro facility : 20

(i) Macro calls with macro instruction defining macros.

(ii) Conditional macro expansion.

Ans. In simplest form, macro is an abbreviation for a sequence of operations. Macro instructions (often called macros) are single-line abbreviations for groups of instructions. In employing a macro, the programmer essentially defines a single "instruction" to represent a block of code. By defining the appropriate macro instructions, an assembly language programmer can tailor his own higher level facility in a convenient manner, at no cost in control over the structure of his program. Macro instructions are usually considered an extension of the basic assembler language and the macro processor is viewed as an extension of the basic assembler algorithm. As a form of programming language, however, macro instruction languages differ significantly from assembly languages and compiled algebraic languages.

(i) Macro Instruction Defining Macros : Macros are generalized abbreviations for instruction sequences, nothing that it seems reasonable to permit any valid statements in the abbreviated sequence, including macro definitions. In this manner a single macro instruction might be used to simply the process of defining a group of similar macros. It is important to realize that the inner macro definition is not defined until after the outer macro has been called. The following example defines a macro instruction DEFINE, which when called with a subroutine name defines a macro with the same name as the subroutine.

Example :.....

```
Definition of macro DEFINE
    MACRO
    DEFINE  & SUB
    MACRO
    & SUB   & Y
    CNOP    0, 4
    BAL     1, 4
    DC      A(& Y)
    L       15, = V(& SUB)
    BALR    14, 15
    MEND
Definition of macro & SUB
    MEND
```

(ii) Conditional Macro Expansion : Two important macro processor pseudo-ops, AIF and AGO, permit conditional reordering of the

sequence of macro expansion. This allows conditional selection of the machine instructions that appear in expansions of a macro call.

Example :

```
            MACRO
& ARGO      VARY    &COUNT, &ARG1, &ARG2, &ARG3
& ARGO      A       1, &ARG1
            AIF     (&COUNT EQ1), FINI
            A       2, &ARG2
            AIF     (&COUNT EQ2), FINI
            A       3, &ARG3
FINI        MEND
            :
LOOP 1      VARY    3, DATA1, DATA2, DATA3
            :
LOOP 2      VARY    2, DATA3, DATA2
            :
LOOP 3      VARY    1, DATA1
            :
DATA 1      DC      F'5'
DATA 2      DC      F'10'
DATA 3      DC      F'15'
```

Labels starting with a period (.), such as FINI, are macro labels and do not appear in the output of the macro processor. The statement AIF (&COUNT EQ1) FINI directs the macro processor to skip to the statement labelled. FINI if the parameter corresponding to &COUNT is a 1; otherwise the macro processor is to continue with the statement following the AIF pseudo-op.

AIF is a conditional branch pseudo-op; it performs an arithmetic test and branches only if the test condition is true. The AGO is an unconditional branch pseudo-op or 'goto' statement.

Q. 4. Describe the following concept of unix operating system : **20**

(i) Swapping

(ii) Demand paging

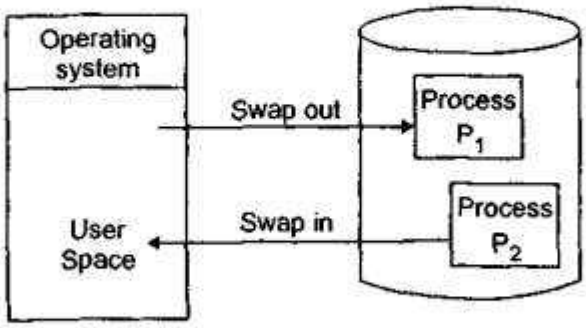
(iii) Fragmentation

(iv) User to user communication

Also discuss advantage and disadvantage of each concept.

Ans. (i) Swapping : A process needs to be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store, and then brought back into the memory for continued execution.

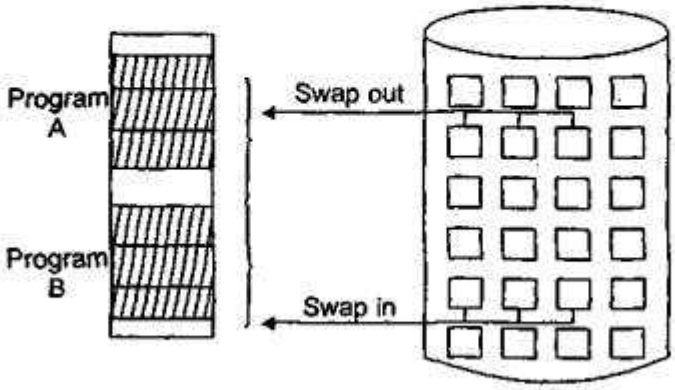
Normally, a process that is swapped out will be swapped back into the same memory space that it occupied previously. This restriction is dictated by the method of address binding. If execution-time binding is being used, then a process can be swapped into a different memory space.



Swapping requires a backing store. The backing store is commonly a fast disk, it must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images. The system maintains a ready queue, consisting of all processes whose memory images are on the backing store or in memory and are ready to run. The dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers as normal and transfers control to the selected process.

The context-switch time in such a swapping is fairly high.

(ii) Demand Paging : A demand-paging system is similar to a paging system with swapping. Processes reside on secondary memory. When we execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed. A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process.



When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only these necessary pages into

memory. Thus, it avoids reading into memory, pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

(iii) Fragmentation : Segmentation may cause external fragmentation, when all blocks of free memory are too small to accomodate a segment.

Fragmentation refers to a problem in memory management scheme. There are two types of fragmentation :

(i) Internal fragmentation

(ii) External fragmentation

External fragmentation refers to the condition when we have a free blocks of memory but these blocks are scattered in such a way that inspite of having an adequate space for a process we can not allocate it to a process. Internal fragmentation refers to the memory space which is acquired by the process but never used, i.e., their is a memory for drivers, etc. in each process which may not be used by the process and results in internal fragmentation.

(iv) User to User Communication : There are tools that allow users to communicate with one another through the shell-**wall, write, talk** and **mesg**.

Wall : Wall command allow a system administrator as root to message all users on the system. This is used by script such as "shutdown" or "reboot", telling everyone to get off. Use is simple,

Example : root@quad~$wall
System Maintainence Tongiht!
^D (ctrl D).

Write : Write command sends a message directly to a single user.

Example : kmr@quad~$write kumar11
Hello how are you?
^D

Talk : Talk command is the realt-time user communication tool. It is said real time because we can see each character the other user enters. The 'talk' tool requires a daemon process.

Mesg : Mesg command takes the option y or n. It simply determines whether or not other people can put you using write and talk.

Example : kmr@quad~$mesg n

The main use is to avoid messages from annoying users.

Q. 5. How unix operating system is different from ordinary operating system? Discuss following commands format : **20**

(i) Unix documentation **(ii) Basic file operations** **(iii) Changing your password**

Ans. UNIX : Unix operating system is different from ordinary operating system in the following ways :

(i) The File and Process : UNIX doesn't really care to know the type of file we are using. It considers even directories and devices as members of the file system. A file to UNIX is just an array of bytes and can contain virtually anything-text, object code or a directory structure. The second entity is the process, which is the name given to a file when it is executed as a program.

(ii) A Multiuser System : UNIX is a multiprogramming system. It permits multiple programs to run and compete for the attention of the CPU. This can happen in two ways : Multiple users can run separate jobs

that share the system's CPU and resources, but a single user can also run multiple jobs. UNIX is also a multiuser system.

(iii) A Multitasking System : A single user can also run multiple tasks concurrently. UNIX is a multitasking system. It is usual for a user to edit a file, print another one on the printer, send emai to a friend and browse the WWW-all without leaving any of the applications. The kernel is designed to handle a user's multiple needs.

(a) UNIX Documentation : UNIX documentation is no longer the sore point it once was. Even though its sometimes uneven, at most times the treatment is quite lucid. The principal on-line facility available is the man command, which remains the most important reference for commands and their configuration files.

UNIX offers an on-line help facility in the man command. Man displays the documentation-often called the man documentation-of practically every command on the system.

Man presents the first page and pauses. It does by sending its output to a pager program which displays the contents of a file on page at a time. The pager is actually a UNIX command, and man is always preconfigured to be used with a specific pager.

(b) Basic File Operations :

(i) Creating File : Cat command is used to create a file.

$ cat > filename

Example : $ cat > abc

(ii) Displaying File : Cat command is also used to display the contents of a file.

$ cat Filename

Example : $ cat abc

(iii) Deleting a File : rm command is used to delete the file.

$ rm filename

Example : $ rm abc

(iv) Renaming Files : mv renames (moves) files. It has two functions :

(i) It renames a file or directory.

(ii) It moves a group of files to a different directory.

$ mv oldfilename newfilename

Example : $ mv abc xyz

mv doesn't create a copy of the file.

(c) Changing Your Password : Password command is used to change user password.

$ password

Password : Changing password for user 1.

Enter login password : ******* {Ask for old password}

New password : ******

Re-enter new password : ******

passwd (SYSTEM) : passwd successfully changed for user 1.

Q. 6. How the following test manipulation takes places : **20**

(i) File statistics **(ii) Searching for patterns**

(iii) AWK utility **(iv) Translating characters**

Ans. (a) File Statistics : UNIX provides the file command to determine the type of file, especially of an ordinary file

```
$ file*
```

This command identifes the file type by examining the magic number that is embedded in the first few bytes of the file. Every file type has a unique magic number.

UNIX features a universal word-counting program. The wc command counts lines, words and characters, depending on the options used. It takes one or more filenames as its arguments, and displays a four columnar output.

```
$ wc filename
```

Example :

```
$ wc abc
3    20 103 abc
```

wc counts 3 lines, 20 words and 103 characters. The filename has also been shown in the fourth column.

wc options :

l : Option counts only the number of lines.

w : Option counts only the number of words.

c : Option counts only the number of characters.

(b) Searching for Patterns : grep scans its input for a pattern, and can display the selected pattern, the line numbers or the filenames where the pattern occurs.

Syntax :

```
$ grep options pattern filenames(s)
```

grep stands for **global regular expression.** grep searches for pattern in one or more filenames. The first argument is the pattern and ones remaining are filenames.

Example :

```
$ grep "sales" emp.lst
```

grep options :

(i) Ignoring Case (–i) : When we look for a name, but are not sure of the case, grep offers the –i option.

(ii) Deleting Lines (–v) : The –v option selects all except lines containing the pattern.

(iii) Displaying Line Numbers (–n) : The – n (number) option displays the line numbers containing the pattern, along with the lines

(iv) Counting Lines Containing Pattern (– c) : The – c (count) option counts the number of lines containing the pattern.

(v) Displaying Filenames (– l) : The – l (list) option displays only the names of files containing the pattern.

(c) AWK Utility : The awk command made a late entry into the UNIX system in 1977 to augment the tool kit with suitable report formatting capabilities. Named after its authors, Aho, Weinberger and Kernighan, awk, until the advent of perl, was the most powerful utility for text manipulation. awk appears a gawk (GNU awk) in Linux.

awk is not just a command, but a programming langauge too. It uses an unusual syntax that uses two components and requires single quotes and curly braces.

Syntax :

awk options 'selection_criteria {action}' file(s)

The selection_criteria (a form addressing) filters input and selects lines for the action component to act upon. This component is enclosed within curly braces. The selection_criteria and action constitute an awk program that is surrounded by a set of single quotes.

(d) Translating Characters : The tr (translate) filter manipulates individual characters in a line. More specifically, it translates characters using one or two compact expressions :

Syntax : tr options expression 1 expression 2 standard input tr takes input only from the standard input; it doesn't take a filename as argument. By default, it translates each character in expression 1 on its mapped counterpart in expression 2.

- **tr options :**

(i) Deleting Characters (-d) : -d option is used to delete the characters/and/from the file.

(ii) Compressing Multiple Consecutive Characters (-s) : UNIX tools work best with fields rather than columns, so its preferable to use files with delimited fields. In that case, lines need not be of fixed length, we can eliminate all redundant spaces with the -s (squeeze) option, which squeezes multiple consecutive occurrences of its argument to a single character.

(iii) Complementing Values of Expression (-c) : Finally, the -c (complement) option complements the set of characters in the expression. Thus, to delete all characters except the/and/. We can combine the -d and -c options.

Q. 7. What is shell? How shells are different kernel? Why shell prog are necessary (if so justify)? Discuss the following : 20

(i) Wild cards (ii) Shell constructs

Ans. A Unix shell, is a command interpreter and script host that provides a traditional user interface for the Unix operating system and for Unix-like systems. Users direct the operation of computer by entering command input as text for a command line interpreter to execute or by creating text scripts of one or more such commands. The most generic sense of term shell means any program that users use to type commands.

The kernel of an operating system is something you will never see. It basically enables your programs to execute. It handles events generated by hardware and software, and manages access to resources. The kernel usually defines a few abstractions like files, processes, sockets, directories, etc. Which correspond to an internal state it remembers about last operations, so that a program may issue a session of operation more efficiently.

A shell is a special program that is usually integrated in any OS distribution and which offers humans an interface with the kernel. The way it appears to users may vary from system to system, but the concept is always the same :

(i) Allow the user to select a program to be started and optionally give it session-specific arguments.

(ii) Allow trivial operation on the local storage listing the content of directories, moving and copying files across the system.

(i) Wild Cards : The meta characters that are used to construct the generalized pattern for matching filenames belong to a category called wild-cards.

Following are the wild cards.

* → Any number of characters including none.

? → A single character.

[ijk] → A single character-either an i, j or k.

[x-z] → A single character within the ASCII range of the characters x and z.

[!ijk] → A single character that is not i, j or k.

[!x-z] → A single character that is not within ASCII range of the character x and z.

{pat 1, pat 2 ...} → pat 1, pat 2, etc.

(ii) Shell Constructs : The term shell also refers to a particular program, such as the Bourne shell, sh. The Bourne shell was the shell used in early versions of UNIX and became a de facto standard; every UNIX-like system has at least on shell compatible with the Bourne shell. The Bourne shell program is located in the UNIX file hierarchy at/bin/sh. On some systems, such as BSD, /bin/sh is a Bourne shell or equivalent, but on other systems such as LINUX/bin/sh is likely to be a link to a compatible, but more feature rich shell. POSIX specifies its standard shell as a strict subset of the Korn shell.

Q. 8. Discuss basic feature of LINUX operating system. How LINUX operating system differ from unix operating system? Discuss advantage and disadvantage of LINUX. **20**

Ans. Features of LINUX :

(i) Linux is Technically Advanced : Linux, if not the most advanced OS out there, at least among the most advanced operating systems.

(ii) Linux is highly Configurable : You can customize Linux exactly to your liking. You can make it work exactly the way you want and on the platform you want.

(iii) Linux is Secure : Linux has many features that protect your system from intruders.

(iv) Software Dependencies : Software dependencies is an excellent thing because it helps you keep things small and simple.

Differences between Unix and Linux : Following are the difference between Unix and Linux :

(i) Unix was developed by AT and T in 1969. Whereas Linux was first released by its inventor Linux Torvalds in 1991.

(ii) Unix requires a more powerful hardware configuration. It will work in large mainframe computers but will not work in an x86 based personal computer. Linux, however, has small hardware requirements and it will work on both a large mainframe computer and an x86 based personal computer.

(iii) Unix treats everything as a file, it provides greater security for users. Linux uses a Unix architecture as its basis and provides more facilities and applications.

(iv) Unix distribution is POSIX whereas Linux distributions are Redhat, Fedora, Susee, Mandriva and Ubuntu.

(v) Unix is the foundation for a number of operating systems, with Linux being the most popular one.

Advantages of Linux :

(i) Stability : Linux can crash, but it is much harder to do. If an application crashes in Linux, it will usually not harm the kernel or other processes.

(ii) Free Software : Most software can be obtained without cost for Linux.

(iii) Runs on Old Hardware : Linux runs even of old 386 or 486 processors.

(iv) Security : Linux has the advantage of the code being in the public domain. This can be a double-edged sword; while you can look at the code, and developers can fix holes rapidly, it also means hackers can find bad code.

Disadvantages of Linux :

(i) Learning Curve : Linux is very difficult to learn.

(ii) No Support for all Programs : Linux do not support all the programs which may be useful for user. For example, Linux does not support office suite program.

(iii) Not all Hardware Compatible : Some of the latest and greatest hardware that is being produced is not compatible with Linux.