**Name:** Haneef Adebayo Salaudeen

Matric Number: 222512

**QUESTION 1**

**UNIX OPERATING SYSTEM**

UNIX, short for UNiplexed Information Computing System (also known as UNICS), was developed in 1969. A highly popular and multitasking operating system, UNIX is powerful. Unix refers to the original operating system developed by AT&T. More generally, it refers to family of derived operating systems. Original code was developed by Linus and the GNU Foundation. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops. Unix was one of the first OSes to be written in the C programming language. Since its introduction, the Unix operating system and its offshoots have had a profound effect on the computer and electronics industry, offering portability, stability and interoperability across a range of heterogeneous environments and device types.

**What is Unix used for?**

Unix is a modular OS made up of a number of essential components, including the kernel, shell, file system and a core set of utilities or programs.

At the heart of the Unix OS is the kernel, a master control program that provides services to start and end programs. It also handles low-level operations, such as allocating memory, managing files, responding to system calls and scheduling tasks. Task scheduling is necessary to avoid conflicts when multiple programs attempt to access the same resource at the same time.

Users interact with the Unix environment through the shell, a CLI for entering commands that are passed to the kernel for execution. A command is used to invoke one of the available utilities. Each utility carries out a specific operation, such as creating files, deleting directories, retrieving system information or configuring the user environment.

Some Unix commands take one or more arguments, which provide a way to refine the utility's behavior. For example, a user might enter the command *rm* OldFIle.txt. The command is calling the *rm* utility, which deletes files in a directory. The command also includes the argument OldFIle.txt, which is the file to be deleted. When the user enters this command in the shell, the kernel runs the *rm* program and deletes the specified file.

### Unix-like operating systems

The term Unix-like is often used to describe the different Unix variants, but there is no clear definition of what this term means. In general, it can refer to any OS that has some relation to Unix, no matter how distant, including free and open source variations. Some software developers assert that there are three types of Unix-like systems:

1. OSes historically connected to the original codebase from Bell Labs, such as the BSD systems developed by researchers at Berkeley.
2. Trademarked and branded Unix-like systems that meet SUS, such as HP-UX and IBM AIX. The Open Group has determined that these systems are allowed to use the Unix name.
3. Functional Unix-like systems, such as Linux and Minix, that behave in a manner consistent with the Unix specification. For example, they must have a program that manages the login and command-line sessions.

**Unix-like operating system:Linux**

Linux refers to the kernel of the GNU/Linux operating system. More generally, it refers to the family of derived distributions.

“Linux" properly refers to the OS kernel.

By "different flavors of Linux", you are referring to different Linux distributions. A "distribution" is a complete operating system -- a particular packaging of the kernel with various other utilities, services and shells that provide the rest of what users expect from an operating system. It provides some good links to start with.

The differences between distributions are in the particulars of

* what a distribution is designed to accomplish,
* which versions of what tools it includes to achieve that goal, and
* how it chooses to handle configuration and packaging.

A distribution aimed at desktop users typically includes the X-window system, a desktop environment, a printing subsystem, networking components, etc. One designed for servers often doesn't include X or a desktop environment, but adds services that aren't needed on a desktop (webserver, database, network file systems, etc.).

**Difference between Linux flavors**

The first major difference between various Linux distributions is their target audiences and systems. For example, some distributions are customized for desktop systems, some distributions are customized for server systems, and some distributions are customized for old machines, and so on.

Usually, distributions are designed for specific purposes or to provide an environment to perform particular functions.

The second major difference between distributions is the process of installing and updating applications. Different distributions use different application installation and management tools, called package management tools.

The next big difference is how distributions provide supports and updates. Some distributions are maintained by a community of volunteers while others are maintained and supported by a commercial vendor.

If a distribution is maintained by a community, you will get all updates and supports free of cost. But if a distribution is maintained by a commercial vendor, you may have to pay for support services, updates, and maintenance.

List of linux flavors include:

1. Red Hat Enterprise Linux
2. CentOs Linux
3. Fedora Linux
4. SUSE Linux
5. Debian Linux
6. Ubuntu Linux

**QUESTION 2**

**SOFTWARE FUNCTIONAL REQUIREMENTS**

A Functional Requirement (FR) is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional Requirements in Software Engineering are also called *Functional Specification*.

In software engineering and systems engineering, a Functional Requirement can range from the high-level abstract statement of the sender’s necessity to detailed mathematical functional requirement specifications. Software functional requirements help you to capture the intended behaviour of the system.

Functional Requirements of a system should include the following things:

* Details of operations conducted in every screen
* Data handling logic should be entered into the system
* It should have descriptions of system reports or other outputs
* Complete information about the workflows performed by the system
* It should clearly define who will be allowed to create/modify/delete the data in the system
* How the system will fulfill applicable regulatory and compliance needs should be captured in the functional document

**Benefits of Functional Requirement**

Here, are the pros/advantages of creating a typical functional requirement document-

* Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
* A functional requirement document helps you to define the functionality of a system or one of its subsystems.
* Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
* Errors caught in the Functional requirement gathering stage are the cheapest to fix.
* Support user goals, tasks, or activities

**Types of Functional Requirements**

Here are the most common functional requirement types:

* Transaction Handling
* Business Rules
* Certification Requirements
* Reporting Requirements
* Administrative functions
* Authorization levels
* Audit Tracking
* External Interfaces
* Historical Data management
* Legal and Regulatory Requirements

**What should be included in the Functional Requirements Document?**

Here is how to write functional requirements document:

**Example of Functional Requirements**

Below are the popular functional requirements examples:

* The software automatically validates customers against the ABC Contact Management System
* The Sales system should allow users to record customers’ sales
* The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
* Only Managerial level employees have the right to view revenue data.
* The software system should be integrated with banking API
* The software system should pass Section 508 accessibility requirement.

**Best practice of Functional Requirement**

Important best practice for developing functional requirement document is as follows:

* Do not combine two requirements into one. Keep the requirements granular.
* You should make each requirement as complete and accurate as possible.
* The document should draft all the technical requirements.
* Map all requirements to the objectives and principles which contributes to successful software delivery
* Elicit requirements using interviews, workshops and casual communications.
* If there is any known, verified constraint which materially affects a requirement then it is a critical state that should be documented.
* It is necessary that you document all the assumption in the document.

**Mistakes While Creating a Functional Requirement**

Here, are some common mistakes made while creating function requirement document:

* Putting in unjustified extra information that may confuse developers
* Not putting sufficient detail in the requirement document.
* You add rules or examples, scoping statements or objectives anything except the requirement itself.
* Left out a piece of important information that is an absolute must to fully, accurately, and definitively state the requirement.
* Some professionals start to defend the requirements they have documented when the requirement is modified, instead of finding the correct truth.
* Requirements which are not mapped to an objective or principle.

**KEY LEARNINGS**

* Explain functional requirements in Software Engineering: A Functional Requirements define a system or its components
* Functional Requirements Document should contain Data handling logic and complete information about the workflows performed by the system
* Functional requirements along with requirement analysis help identify missing requirements
* Transaction corrections, adjustments, and cancellations, Business Rules, Certification Requirements, Reporting Requirements, Administrative functions, Authorization levels, Audit Tracking, External Interfaces, Historical Data management, Legal or Regulatory Requirements are various types of functional requirements
* As a good practice do not combine two requirements into one. Keep the requirements granular.
* Putting in unjustified extra information that may confuse developers should be avoided in the functional requirement document.

**QUESTION 3**

Why do programmers prefer Unix?

The answer is simple: Many developers find it's a refreshing alternative to monolithic tools like IDEs and languages like Java.

Ever since Unix burst onto the scene in the early '70s, observers in the computer world have been quick to write it off as a quirky operating system designed by and for expert programmers. Despite their proclamations, Unix refuses to die. Way back in 1985, Stewart Cheifet wondered if Unix would become the standard operating system of the future on the PBS show "The Computer Chronicles," even though MS-DOS was well in its heyday. In 2018, it's clear that Unix really is the standard operating system, not on desktop PCs, but on smartphones and tablets.

It's also the standard system for web servers. The fact is, millions of people around the world have interacted with Linux and Unix systems every day, most of whom have never written a line of code in their lives.

So what makes Unix so beloved by programmers and other techie types? Let's take a look at some of things this operating system has going for it.

**QUESTION 4**

Why is Unix being referred to as a scientist OS?

UNIX is a **multi-user, multi-tasking operating system**. Multiple users may have multiple tasks running simultaneously. This is very different from PC operating systems such as MS-DOS or MS-Windows (which allows multiple tasks to be carried out simultaneously but not multiple users).

In the late 20th century UNIX was widely used for Internet servers, workstations, and mainframe computers. The main features of UNIX were its **simplicity, portability (the ability to run on many different systems), multitasking and multiuser capabilities, extensive library of software, and hierarchical file system**.

What advantage does UNIX have over other operating system?

UNIX **use less memory while running sophisticated programs**. UNIX OS can handle virtual memory nicely. The virtual memory expands as more programs come into the main memory. Most of the tasks in UNIX is done by using fewer resources.

**QUESTION 5**

What type of programming language is C?

C is an **imperative procedural language** supporting structured programming, lexical variable scope, and recursion, with a static type system. It was designed to be compiled to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support.

*Why is it called a structured programming language?*

**C is called structured programming language** because a program in c language can be divided into small logical functional modules or structures with the help of function procedure.

**QUESTION 6**

Detailed Structure of a C Programming Language

Most programming languages have a structure, including the C language. A **C program** is divided into six sections: Documentation, Link, Definition, Global Declaration, Main() Function, and Subprograms. While the main section is compulsory, the rest are optional in the structure of the C program.

Basic Structure of the C Program

Section Description

Documentation Consists of the description of the program, programmer's name, and creation date. These are generally written in the form of comments.

Link All header files are included in this section which contains different functions from the libraries. A copy of these header files is inserted into your code before compilation.

Definition Includes preprocessor directive, which contains symbolic constants. E.g.: #define allows us to use constants in our code. It replaces all the constants with its value in the code.

Global Declaration Includes declaration of global variables, function declarations, static global variables, and functions.

Main() Function For every C program, the execution starts from the main() function. It is mandatory to include a main() function in every C program.

Subprograms Includes all user-defined functions (functions the user provides). They can contain the inbuilt functions and the function definitions declared in the Global Declaration section. These are called in the main() function.

**QUESTION 7**

How to create a C Programming file on Linux

Linux

1.Open terminal

Use the vim editor

Open file using,

2.vim file.c (file name can be anything but it should end with dot c extension) command.

To Edit the file:

3.Press i to go to insert mode.

Type your program.

4.To save the file:

Press Esc button and then type :wq. It will save the file.

To compile the program:

Type,

5.gcc file.c

To Run the program:

Type,

6. ./a.out