NAME: LOOR JACOBSON

STUDENT NUMBER: 2000707806

REGISTRATION NUMBER: 20/U/7896/PS

ASSIGNMENT THREE

a)

FreeBSD:

It is a multitasking system in which when a user logs on to the system, the shell of the user's choice is run, awaiting commands and running programs the user requests. Since it is multitasking, the command interpreter may continue running while another program is executed. To start a new program, the shell executes a fork() system and then the selected program is loaded into memory via the exec() system call and then the program is executed. The shell then either waits for the process to finish or run the process in the background.

Compilers:

This refers to a program which translates the user's program (source code) into object code, which is stored in memory or mass storage and can therefore be understood by a specific CPU to aid in processing. If it is stored in memory, the operation is referred to as "compile, load, and go." If it is stored on tape, then the $LOAD instruction is required. Compilers have different phases involved in the translation process namely lexical analysis, syntax analysis, semantic analysis and the code generator. There exists different types of compilers including cross compiler, source-to-source compiler, just-in-time compiler, bytecode compiler, binary compilers among others.

Assemblers:

An assembler is a type of computer program that interprets software programs written in assembly language into machine language code and instructions that can be executed by the computer. It generates instructions by evaluating the symbols in operation field and find the value of symbol and literals to produce machine code. There of two types namely single pass assembler and multiple pass assembler.

Debuggers:

A debugger is a program which allows the user to view another program source code line by line. This in turn allows the user to identify incorrect code and find out how their program flows. It is very useful in finding semantic and syntax errors in the program. It may require one to run over and over but it Is well worth it as it reduces the time taken to perfect a program as errors are identifies and corrected as the source code accumulates. Examples of debuggers include: GNU Debugger, Microsoft Visual Studio Debugger, Radare2, LLDB debugger, Valgrind among others.

Interpreters:

This refers to a computer program which directly executes the instructions in a high-level language without converting it into machine code. There are three strategies that the interpreter follows. One is that it executes the source code directly and produce the output. Second is that it translates the source code into some intermediate code and then execute the code. Lastly, it uses an internal compiler to produce a precompiled code then execute the precompiled code. There are different types of interpreters but the major ones include Bytecode interpreters, threaded code interpreters, abstract syntax tree interpreters, and self-interpreters.

Absolute loader:

This refers to a loader which requires that a given load module always be loaded into the same location in main memory. Therefore, in the load module presented to the loader, all address references must be to specific, or absolute, main memory addresses. There are some drawbacks though to this approach one of which is that, every programmer would have to know the intended assignment strategy for placing modules into main memory. Also, if any modifications are made to the program that involve insertions or deletions in the body of the module, then all of the addresses will have to be altered.

Relocatable loader:

This refers to a loader in which the assembler or compiler produces not actual main memory addresses but addresses that are relative to some known point, such as the start of the program. The start of the load module is assigned the relative address and all other memory references within the module are expressed relative to the beginning of the module. The compilers or assembler uses the **relocation dictionary** to include information that tells the loader where the address references are and how they are to be interpreted.

Linkage Editor:

This refers to a linker that produces a relocatable load module. Each compiled or assembled object module is created with references relative to the beginning of the object module. All of these modules are put together into a single relocatable load module with all references relative to the origin of the load module

Overlay loader:

This refers to a loader that uses the concept such that whenever a process is running, it will not use the complete program at the same time but rather only some part of the program. So, to run the program that is for example bigger than the size of the physical memory, it loads into memory only the instructions and data that are needed at any given time and once

that part is done executing, it is unloaded and then new parts are loaded into memory and the process is repeated until the entire program executes.

Debugging systems:

These are systems whose purpose is finding and fixing errors in a system, both in hardware and in software. These errors are commonly known as bugs. These systems enable failure analysis by writing the error information to a log file to alert system administrators or users that the problem occurred. They may also capture the memory of the process (code dump)—and store it in a file for later analysis. They enable error information is saved to a log file, and the memory state to be saved to a crash dump when a crash occurs during program execution.

Key considerations in the design of an operating system:

There are majorly two key things to put into consideration when designing an operating system namely the definition of goals and specifications all each of which is affected by the choice of hardware and the type of system. The requirement specifications are divided into two: user goals—such as the system being convenient to use, easy to learn and to use, reliable, safe, and fast and system goals—such as being easy to design, implement, and maintain; and it should be flexible, reliable, error free, and efficient. Mechanisms which determine how to do something and policies which determine what will be done should also be looked at.

System programs:

These are programs which provide a convenient environment for program development and execution. They are quite a number with each serving a specific purpose. We have the file management programs for manipulating files and directories, programming language support programs like compilers, debuggers, interpreters among others. Communications programs, background services programs, application programs, program loading and execution programs are also among these system programs.

Micro Kernel:

This is a method that structures the operating system by removing all nonessential components from the kernel and implementing them as user level programs that reside in separate address space. Its main purpose is to provide communication between the client program and the various services that are also running in user space, together with minimal process and memory management. The microkernel approach makes extending the operating system easier. It also provides more security and reliability, since most services are running as user—rather than kernel—processes and if a service fails, the rest of the operating system remains untouched.

b) An operating system (OS) is a is a software program which manages computer hardware, software resources and provides common services for computer programs.

OPERATING SYSTEM DESIGN

It starts by first of all defining the goals and specifications of the system both of which depend on the choice of hardware and type of system. Goals can be **user goals** such as convenience, easy to learn and use reliable among others as well as system goals such as one which is easy to design, implement and maintain, flexible, reliable, error free among others.

The core issues to look at in the design of an OS are **mechanisms** which determine how to do something and **policies** which determine what will be done. Policies are likely to change over time or even across places but you should know that each change in policy would require a change in the underlying mechanism. For this reason, it is advisable that a general mechanism flexible enough to work across a range of policies be preferred rather than those which are not flexible. Most times though, a change in policy would require only redefinition of certain parameters of the system.

Policy decisions are very important for all resource allocation and so whenever it is necessary to decide whether or not to allocate a resource, a policy must be made. And whenever a question is how rather than what, a mechanism must be determined.

OPERATING SYSTEM IMPLEMENTATION

Operating systems are written by many people over a long period of time and so after designing the system, it must be implemented. But how do we go about with this? Its quite difficult to make general statements about how the development team will go about this.

Back in the day, operating systems were written in assembly language but of course with time, this has greatly deteriorated as now these operating systems are written in higher level languages such as C and C++. Sometimes it's a mixture of more than one High Level Language.

The lowest level of the **kernel** (a program in the OS which generally has complete control over everything in the system) might be written in assembly language and C. Higher level routines might be written in C and C++ and system libraries might be written in C++ or even other high-level languages.

Some advantages of using High level languages are that the code can be written faster, the code is more compact, easier to understand and debug. Improvements in compiler technology though help improve the generated code for the entire operating system by merely doing some simple recompilation.

It is also far easier to port the operating system to other hardware if it's written in High level languages which is particularly important for Operating Systems that are intended to run on several different hardware systems such as embedded devices like microwaves, intel X86 systems, RAM chips on mobile phones. One disadvantage though of implementing an operating system in a high-level language is that it results in reduced speed and most notably increased storage requirements.