Assignment 1 Operating Systems	
Name: Zubair Abid	Roll No: 20171076

1. (a) *Explain the context of "probe instructions" in Dijkstra's paper entitled "My recollections of OS design". List the problems caused by probe instructions.* [5]

   The simplest idea often suggested for process control was to have a **"physical machine" simulate a (small) number of logical machines** - each coupled to its own device so its state would determine if the corresponding logical machine could proceed. Each logical machine had its own priority. The idea was that the logical machine of highest priority would be coupled to the device with short, urgent, frequent messages and those with lower priority would be coupled to devices that could be served more leisurely; some "background computation", say, the generation of prime numbers.

   The system was all fine with logical machines coupled with independent experiments, but **things got difficult** when they had to communicate with each other, for instance to deal with different aspects of the same experiment. Safe communication was tricky, if not impossible. There was also less flexibility for general purpose computing, for situations like when processes of very different urgencies alternated.

   Introduction of "probe instructions" allowed more flexibility in process control that enabled the central processor to test whether a concurrent activity it had started earlier had, in the mean time, been completed.

   The issues caused by probe instructions included:
   1. Probes are tricky: while the answer "Yes" is reliable, the answer "No" can become obsolete as soon as it has been given.
   2. Since the execution of the probe instruction takes time, probing with high frequency could noticeably slow down the computation, while probing with low frequency could increase the computer's reaction time too much
   3. If you wanted to probe every 100 instructions, would you insert a probe into a loop body of 10 instructions? Doing so might probe with unnecessarily high frequency, or unnecessarily low frequency otherwise
   4. The subroutine library poses another problem. One might find themselves tempted to introduce each subroutine in two versions, one with probe instructions and one without.

   (b) *Discuss the solution proposed by Dijkstra for the problems caused by probe instructions* [5]

   Dijkstra suggested an arrangement known as an "interrupt". The idea behind an interrupt was that while the computer calculates at full speed, a piece of dedicated hardware monitors the outside world for completion signals from communication devices. When a completion is detected, the program under execution is interrupted after the current instruction and in such a fashion that it can be resumed at a later moment as if nothing had happened, thus instantaneously freeing the central processor for a suddenly more urgent task. After the interrupt the processor would execute a standard program establishing the source of the interruption and taking appropriate action.

   Dijkstra called it "a great invention, but also a Box of Pandora", because the exact moments of the interrupts were unpredictable and outside operator's control. Which meant that the interrupt mechanism turned the computer into a nondeterministic machine with a nonreproducible behavior

2. *There are five levels in THE operating system. The functions at each level are assigned in an hierarchical manner. Discuss the advantages of hierarchical organization by considering the sample functions or operations of level 0 and level 1. What would have happened, if the services of level 0 and level 1 would have been written in a single level ?* [10]

In a hierarchical organization, a **primary advantage** is that due to (proper implementation of) **abstraction**, higher levels need not worry about the lower level tasks, and can perform their own, without any dependence of the internal structure of the previous hierarchy. The operating system also is **easy to test and maintain**.

At level 0 processors are allocated their processes whose dynamic progress is logically permissible (i.e. in view of explicit mutual synchronization). At this level the interrupt of the realtime clock is processed and introduced to prevent any process from monopolizing processing power, and a priority rule is incorporated to achieve quick response of the system where this is needed. Abstraction at this level has been achieved: above here, the number of processors actually shared is no longer relevant. At higher levels we find the activity of the different sequential processes, the actual processor that had lost its identity having disappeared from the picture.

At level 1 there is a "segment controller," a sequential process synchronized with respect to the drum (memory) interrupt and the sequential processes on higher levels. Here we find that the responsibility to cater to the bookkeeping resulting from the automatic backing store. At all higher levels identification of information takes place in terms of segments, the actual storage pages have lost their identity, having disappeared from the picture.

Then we look at testing: testing level 0 (the real-time clock and processor allocation) implied testing a number of sequential processes on top of it, making sure that under all circumstances processor time was divided among them according to the rules. Testing the segment controller at level 1 meant that all "relevant states" could be formulated in terms of sequential processes making demands on core pages, situations that could be provoked by explicit synchronization among the testing programs.
Due to the hierarchical architecture, the existence of the real-time clock—although interrupting all the time, was immaterial. By that time the correct reaction upon the (mutually unsynchronized) interrupts from the real-time clock and the drum had been implemented.

If levels 0 and 1 were not separated, and if a terminology (i.e. that of the rather abstract sequential processes) in which the existence of the clock interrupt could be discarded had not been created, but instead was built as a non-hierarchical construction, to test the central processor by making it react directly upon any weird time succession of these two interrupts, the number of "relevant states" would have exploded to such a height that exhaustive testing would have been an illusion, and since the drum and clock speed were out of the tester’s control, it would be a near-impossible task to generate all test cases for the system.

3.(a) *In the UNIX system, devices are treated as files. List the corresponding advantages.* [5]

In UNIX systems, devices are also treated as files. They are read and written just like ordinary disk files, but requests to read or write result in activation of the associated device. An entry for each file lives in the directory /dev.

The advantages of treating devices as files in UNIX systems include:

1. File and Device I/O are as similar as possible
2. File and device names have the same syntax and meaning, so that a program expecting a file name as a parameter can be passed a device name
3. Special files are subject to the same protection mechanism as regular files.

(b) *What are the advantages and the problems of deferred block I/O system in UNIX.* [5]

The block I/O devices are accessed through a layer of buffering software. The system maintains a list of buffers each assigned a device name and a device address. This buffer pool constitutes a data cache for the block devices. On a read request, the cache is searched for the desired block. If the block is found, the data are made available to the requester without any physical I/O. If the block is not in the cache, the least recently used block in the cache is renamed, the correct device driver is called to fill up the renamed buffer, and then the data are made available. Write requests are handled in an analogous manner. The correct buffer is found and re-labeled if necessary. The write is performed simply by marking the buffer as ‘‘dirty.’’ The physical I/O is then deferred until the buffer is renamed.

Advantages:
The reduction in involvement of the actual physical I/O device due to the block I/O device of this scheme are substantial, especially considering the file system implementation.

The problems:

1. The asynchronous nature of the algorithm makes error reporting and meaningful user error handling almost impossible. The inelegant I/O error handling in the UNIX system is partly due to the asynchronous nature of the block I/O system.

2. Delayed writes: If the system stops unexpectedly, it is almost certain that there is a lot of logically complete, but physically incomplete, I/O in the buffers. There is a system primitive to flush all outstanding I/O activity from the buffers. Periodic use of this primitive helps, but does not solve, the problem.

3. The associativity in the buffers can alter the physical I/O sequence from that of the logical I/O sequence. This means that there are times when data structures on disk are inconsistent, even though the software is careful to perform I/O in the correct order. On non-random devices, notably magnetic tape, the inversions of writes can be disastrous.

4. (a) *Discuss how the notions of hierarchy and information hiding are helpful in easing the development of operating system ?* [5]

Hierarchy :
The idea is to create a **hierarchy of abstraction** levels so that at any level we can **ignore the details** of what is going on at all lower levels.

At the highest level are system users who are ideally insulated from everything except what they want to accomplish. **Each level adds new operations** to the machine and **hides selected operations** at lower levels. The operations visible at a given level form the instruction set of an abstract machine.

Hence, a program written at a given level can, via well-defined interfaces, **invoke visible operations of lower levels, but no operations of higher levels**. This can help in **easier fault correction**, since an error can only happen in a particular level, and the levels have been written in well defined modules.

Information Hiding :
Information hiding helps in solving one of great problems faced by operating system designers, the problem to **manage the complexity** of operations at many levels of detail, from hardware operations that take one billionth of a second to software operations that take tens of seconds.

Information hiding refers to **confining the details** of managing a class of "objects" within a module that has a good interface with its users. With information hiding, designers can **protect themselves from extensive reprogramming**. If the hardware or some part of the software changes, the change affects only the small portion of the software interfacing directly with that system component, and hence this helps in **modularizing** the entire OS. This allows for better **software maintenance**.

(b) *Consider the following levels in a hypothetical OS. Take any two operations and discuss the corresponding differences at each level.* [5]

LEVEL	NAME	OBJECTS	OPERATIONS
Level 11	Devices (access to external devices)	Printers, displays, and key boards	Create, destroy, open, close, read, write
Level 10	File system	Files	Create, destroy, open, close, read, write
Level 9	Communications	Pipes	Create, destroy, open, close, read, write

Operation	**Create**	**Destroy**
Level 9	Creates a new empty pipe and returns a capability for it. (If the caller is a user process it can store this capability in a directory entry and make the pipe available throughout the system.)	Destroys the given pipe (undoes a create-pipe operation).
Level 10	Creates a new empty file and returns a capability for it. (If the caller is a user process, it can store this capability in a directory entry and make the file available throughout the system.)	Destroys the given file (undoes a create-file operation).
Level 11	Returns a capability for a device of the given type at the given address. The access code of the returned capability will not include "W" if the device is read only or " R" if the device is write-only	Detaches the given device from the system (undoes a create-device operation).