CSC633 OPERATING SYSTEMS
ASSIGNMENT 1

Name: Keith Ah Sing | Student ID: s2009001714

Question 1: Discuss all the main purposes of an operating system?

An operating system is a program that acts as an interface between the computer user and computer hardware. It provides an environment that allows a user to execute application programs and manages the allocation of computer hardware resources between users and processes. The principle purposes of an operating system include:

Resource Management
- resources include cpu, memory, input/output devices
- operating system is responsible for the allocation of these resources to different users and processes.

Process Management
- manages which process the cpu is allocated to and for how long (achieved through scheduling)

Memory Management
- concerned with RAM
- before any program can be executed by the cpu, it must first be loaded into main memory (RAM)
- because main memory is finite, which process is loaded into memory or swapped out of memory is decided by the operating system.

I/O Management
- operating system allocates and manages all i/o devices i.e which process or user can access the I/O device and at what time is the responsibility of the operating system.

Storage Management
- concerned with the hard disk (secondary memory)
- operating system is responsible for how data is stored and accessed in secondary memory.

Security/Protection
- responsible for managing user accounts, storage of passwords and authenticating users at login.
- provides process level security i.e in cases where multiple processes are running, it ensures that each process is not interfered by another.

Question 2: Describe what are interrupts? How does an interrupt differ from a trap? Can traps be generated intentionally by a user program? If so, for what purpose?

Modern operating systems are interrupt driven. That is, instead of the checking on each hardware device to see if there is an event to be taken care of (polling), the operating system lays back, carrying out routine tasks until the hardware/software signals to the operating system via an interrupt that it need its attention.

There are of two types interrupts, hardware generated interrupts and software generated interrupts. A hardware generated interrupt provides a way for the hardware(mostly I/O devices) to alert the operating system that an event has taken place and it needs to be handled. For example, when a key is pressed on your keyboard and interrupt is generated, when you move your mouse or click on an icon an interrupt is generated. This interrupt comes in the form of a signal, the processor receives this signal and notifies the operating system that its attention is needed.

The operating system then pre-empts whatever process is currently running(i.e puts in a blocked state) attends to the interrupt via an interrupt handler - once finished, execution of the pre-empted process resumes.

A trap(or exception) is a software generated interrupt. It has a similar function to the hardware interrupt but is generated by application programs(software). These signals are generated when special events or errors occur during the execution of a program. The program can not handle it itself and thus alerts the operating system for assistance. For example when the program encounters a division by zero error or when the program tries to access a portion in memory it is not assigned to (memory segmentation). The difference between the two terminologies is blurred in most textbooks, with interrupts being the preferred term to describe both terminologies, however, when staking a difference between the two, interrupts are hardware generated and traps are software generated.

There are two common occasions where traps are intentionally generated. The first being for debugging purposes, such as catching arithmetic errors and the second is when a program requires some of its instructions to be executed in kernel mode. The request is sent as a system call which is implemented as a trap as shown in the diagram below.

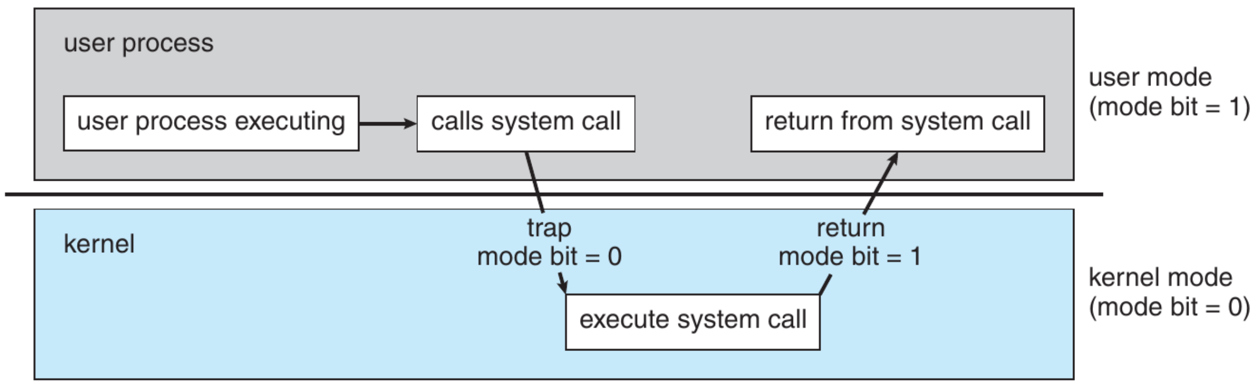


Illustration 1: Courtesy of Operating Systems Concepts 4th Edition

Question 3: Describe some of the challenges of designing operating systems for mobile devices compared with designing operating systems for traditional PCs.

The defining feature of a mobile device is its portability. It must be small enough to take around with you and still provide the same experience as a traditional PC for laptops or Hi-Fi Stereos for ipods and mp3 players. To achieve this portability, alot of trade-offs have to be made in-terms of the size of the components that power and drive the device. And this reduction in size poses challenges when designing operating systems to control and manage these devices. Below is a list of these challenges:

Battery Issues Because of the mobility of the device, it will not always be connected to a power source. Thus these devices rely on batteries and it is important that the operating system is designed in such a way that it manages the apps and the devices so that battery is not depleted unnecessarily.

To mitigate this, some mobile operating systems come pre-installed with utilities that inform you which apps are draining your battery.
Also the developer must ensure that the operating system is able to resume a session with an app if it is interrupted. Battery is consumed when the screen of your device is on, and the longer an app takes to complete a task, the longer the screen stays on and the more battery is depleted. Thus if a

session with an app is interrupted and cannot resume from where you left off, then starting your task all over again is an example of unnecessary consumption of battery.

Screen Size and Orientation Issues A mobile device has significantly less real estate to work with when it comes to screen size. Therefore alot of consideration has to be taken into account for interface design.

Navigation has to be planned and designed so that it is intuitive to the user. If a task has multiple steps, it must be split into distinct screens. The user must also be given the correct cues(can be visual or audio) so that he she is guided to complete the task.

Changing the screen orientation when the user turns the device has challenges of its own. Each element in the layout or view would have to re-organized itself according to the orientation of the screen. Some elements that were accessible in the horizontal orientation would be hidden in the vertical orientation. Which elements are not important enough to be hidden are all considerations that have to be taken care of.

Security Latest phones are incorporating hardware for bio-metric authentication(fingerprint sensors). Some are using existing hardware on the mobile device (facial recognition via phone’s camera). In this case the operating system has to keep up with these advances, ingenious ways must be introduce to distinguish between a real face and an image programmatically. If it can not make this distinction, then people will be gaining unauthorized access using photos of the owner. Also mobile devices are more susceptible to being stolen because of its portability. Thus the developer must incorporate services that allow remote wiping of data, or services to track a stolen phone.

Question 4: Discuss how system calls function?

A system call is when an application program requests a service from the operating system.

To better understand how a system call functions, we must first understand why is it required in the first place. Why does the application have to go through the extra step of requesting for the service? Why isn’t it given direct access?

There are two modes that a program can run in, user mode and kernel mode. If a program is running in user mode, then it does not have access to the hardware resources. If a program is running in kernel mode , it has direct access to hardware resources. The problem with running a program in kernel mode is that if it crashes, not only will the entire system come to a halt, it could cause irreversible or serious problems. Thus executing all application programs in user mode provides a “sandbox” that shields the kernel and system from this.
Only when a program requires system resources, it must first request permission from the operating system. This request is put forth in the form of a system call. The system call allows the application program to communicate to the operating system that it requires access to certain hardware resources/services.
The operating system then switches the program to kernel mode via a mode bit. When the process is done, another system call is made to the operating system to perform a switch back to user mode.

Note: every time a switch is made, a system call precedes this switch. System calls differ from one operating system to another, but the underlying concept remains the same.

Below is a diagram illustrating how the system calls function

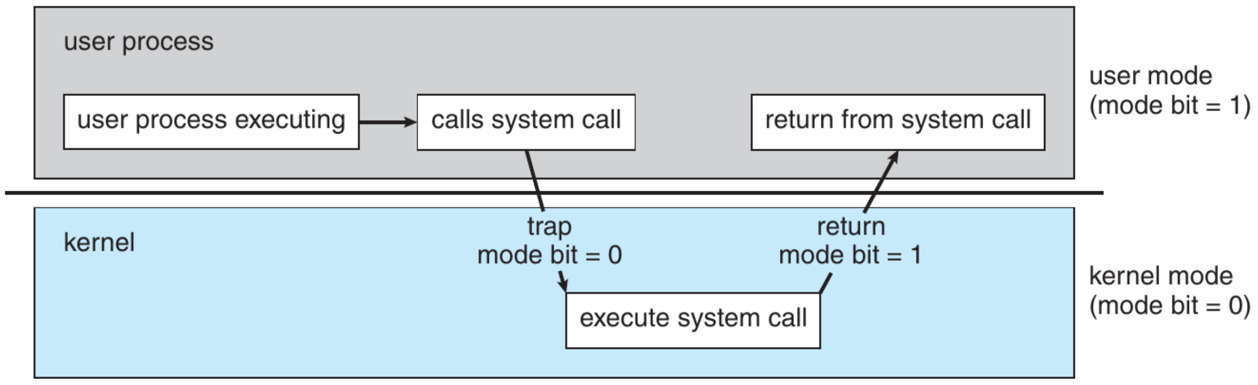


Illustration 2: Courtesy of Operating Systems Concepts 4th Edition

Question 5: Discuss three common ways of establishing relationship between the user and kernel thread?

In an operating system, threads are either implemented as user level threads or kernel level threads. User level threads are managed by the user and the operating system manages the latter. Some operating systems provide the facilities to implement both thus creating a multithreading system. In this type of system, multiple threads within the same application can run in parallel on multiple processors (Operating System - Multi-Threading - Tutorialspoint,2020). Thus in order for the two types of threads to work in tandem, a relationship must be established between the two and there are three models that accomplish this:

- many-to-many
- many-to-one
- one-to-one

Many-to-One Model In this model, many user-level threads is mapped to one kernel thread. The threads are managed by a thread library in the user space and when a single thread makes a blocking system call, the entire process will be blocked.

Drawbacks: Only one thread can access the kernel at a time - therefore multiple threads are unable to run in parallel despite the system having multiple cores. Because of this, only a handful of systems continue to use this model.

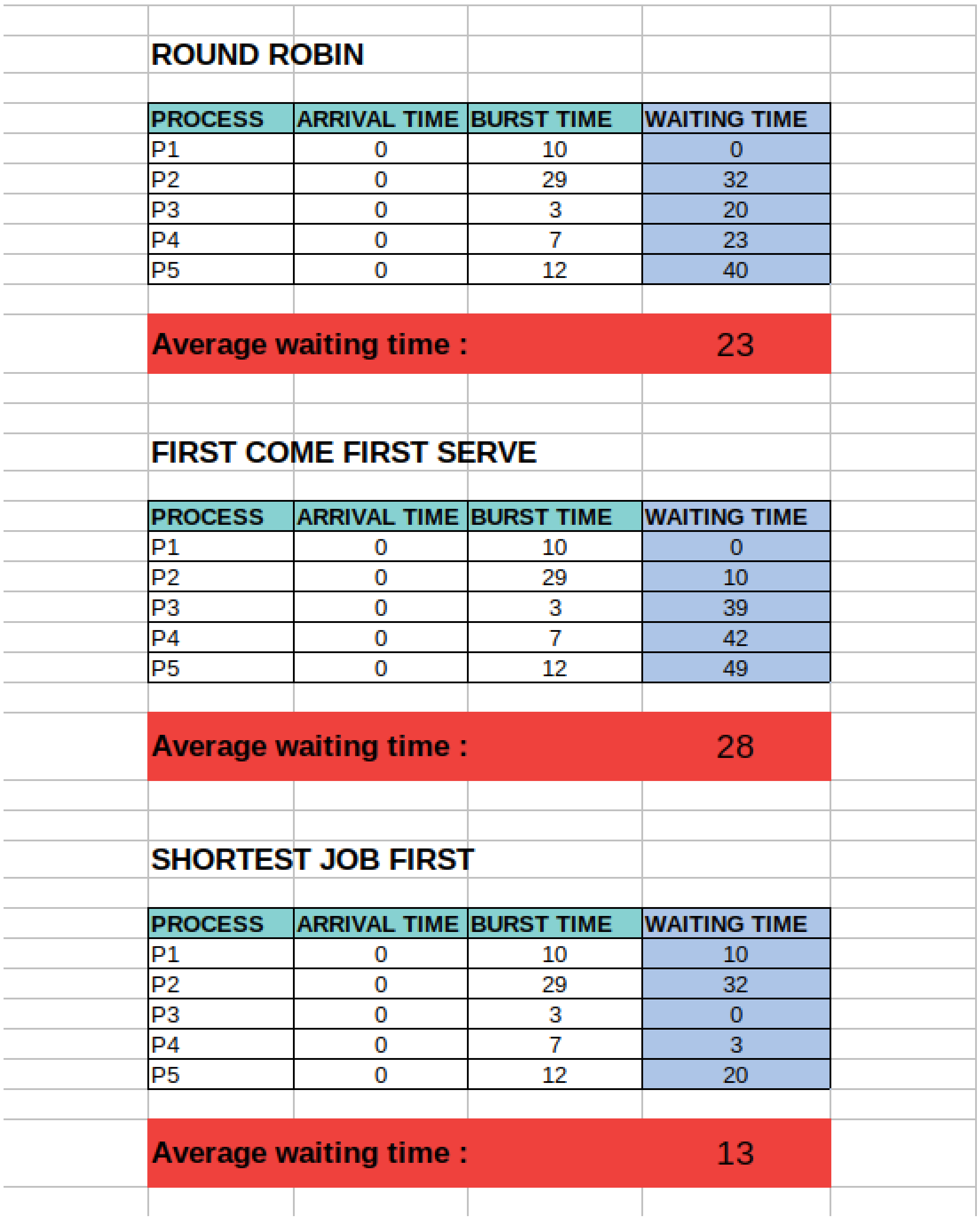
One-to-One Model Each user thread is mapped to a kernel thread. This model supports both concurrency and parallelism i.e while a thread makes a blocking system call, it is possible for another thread to be run. Multiple threads are also able to run on systems with multiple processors.

Drawbacks: For systems that implement this model, the number of threads that can be supported by the system is limited. The reason for this is that for each user thread that is created, a corresponding kernel thread must be created. This inturn creates alot of overhead which could affect the performance of an application.

Many-to-Many Model Here many user threads are mapped to an equal or lesser number of kernal threads. This model does not inherit any of the drawbacks inherent in the above models. There is no restrictions to the number of threads that can be created and the entire process is not block if a thread makes a system call. Kernel thread can also run in parallel on a multiprocessor system. The many-to-many model also has a variation model called the two level model. This model used to be implemented on older Solaris operating systems.

Question 6: Which algorithm would give the minimum average waiting time?

Shortest Job First (SJF) algorithm.



ROUND ROBIN

PROCESS	ARRIVAL TIME	BURST TIME	WAITING TIME
P1	0	10	0
P2	0	29	32
P3	0	3	20
P4	0	7	23
P5	0	12	40

Average waiting time : 23

FIRST COME FIRST SERVE

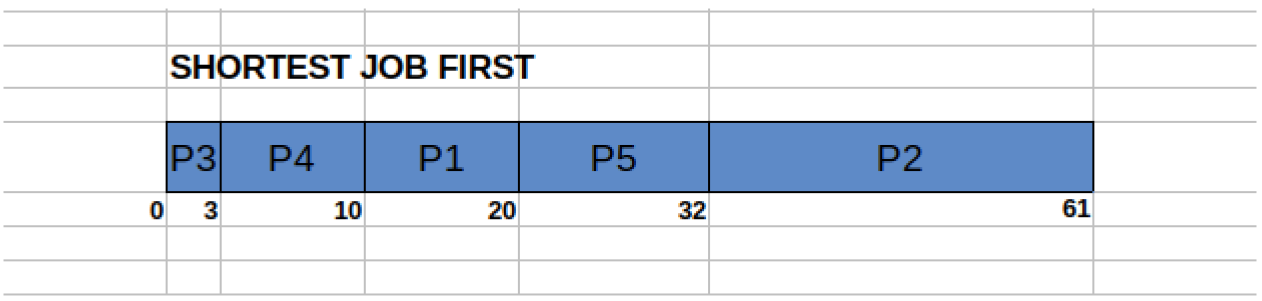
PROCESS	ARRIVAL TIME	BURST TIME	WAITING TIME
P1	0	10	0
P2	0	29	10
P3	0	3	39
P4	0	7	42
P5	0	12	49

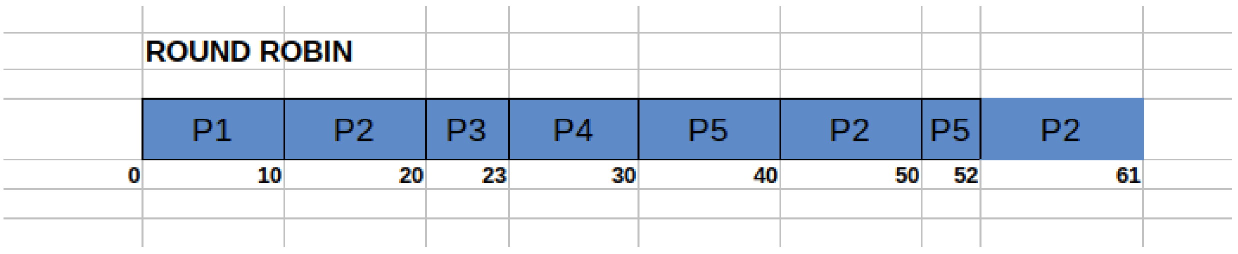
Average waiting time : 28

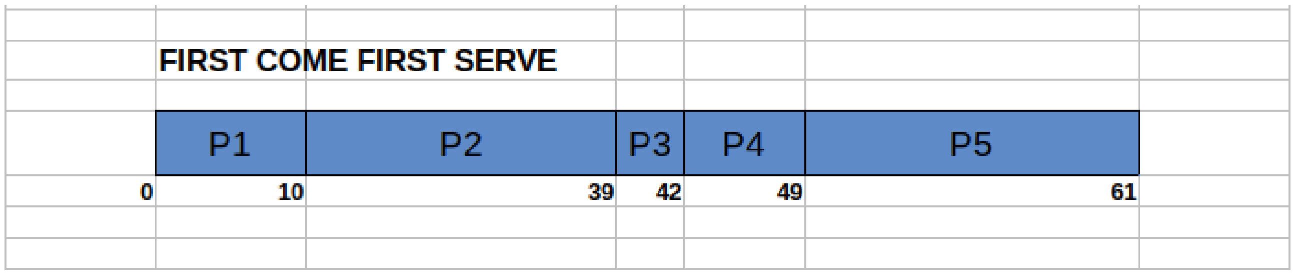
SHORTEST JOB FIRST

PROCESS	ARRIVAL TIME	BURST TIME	WAITING TIME
P1	0	10	10
P2	0	29	32
P3	0	3	0
P4	0	7	3
P5	0	12	20

Average waiting time : 13







Question 7: What are the possible states that a thread can be in?

A thread can be in any one of the following states (Tanenbaum and Bos, 2013):

- **Running** In a running state, the thread is active and is using cpu resources
- **Blocked** In a blocked state, the thread is waiting for an event, eg: I/O operation
- **Ready** Here the thread is scheduled to run and waiting on the cpu
- **Terminated** The thread has either reached the natural completion of its task or is aborted due to an error. In this state the thread is no longer eligible for execution.

Question 8: What are "zombie" threads? When does it get finally cleaned up?

Linux does not distinguish between a process and a thread and treats them the same way (Silberschatz, Galvin and Gagne, 2013). Thus to better understand the concept of a "zombie" thread, the context of a process will be used to explain it.

When a process is created, it is possible for that process to create another process. When this happens, it is called **process spawning**. The newly created process is called the child process and the process that spawned the child process is called its parent. On a Unix system, this is achieved using the fork() system call. After the child process is created, it is given a unique ID that identifies it in the processes table.

Now, when a child process is killed, all of the memory and resources associated with it is de-allocated so that it can be used by other newly created processes. However, the processes ID still remains in the process table and the process descriptor(information about the process attributes,

identification details and resource allocation entries that the process holds) remains in memory. It is at this instance that the process is in a zombie state - it is already killed yet life of its existence still exists in memory and in the processes table.
The process remains in this state until the parent process is notified via a SIGCHLD signal. It then executes the wait() system call to which the zombie process is then reaped or removed completely.

There are times when the parent process is not correctly programmed and it does not execute the wait() system call. In this case the zombie process continues to exist in memory unless the parent process is killed. When this happens, the init process inherit and becomes the new parent of the zombie processes. The init process periodically executes the wait() system call,and in turn, cleans all the zombie processes that it has inherited.

Question 9: Which function is used to put a thread to sleep?

The sleep system call places a thread into an inactive state for a period of time.

Windows OS On Windows, the Sleep() function takes a single parameter which is the number of milliseconds to sleep.

Unix On Unix or Posix operating systems, the sleep() function takes a single parameter which is the number of seconds to sleep. The data type of the argument must be an unsigned integer.

Question 10: As described in Section 4.7.2, Linux does not distinguish between processes and threads. Instead, Linux treats both in the same way, allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. However, other operating systems, such as Windows, treat processes and threads differently. Typically, such systems use a notation in which the data structure for a process contains pointers to the separate threads belonging to the process. Contrast these two approaches for modeling processes and threads within the kernel.

If a software engineer is writing code for a system that draws no distinction between a process and a thread, then it is safe to assume that the code will be much more simpler than for a system that differentiates between the two. It simplifies the job of the scheduler, as it does
not have to go the extra mile of obtaining special code to ascertain which thread is associated with which process for every step of the scheduling process. While this may seem flexible - consolidating a process and a thread into one simple task may from a programmer's perspective take away control from the programmer. Having a thread and a process as separate entities gives a handle to the programmer that allows him/her to better control a program or resources.

For example, a programmer would have less control over a textbox that supported only one data type – Numbers! When the user encounters the textbox, he is free to input any type of numbers, whether decimal or integers. But if the textbox supported more than one data type
(int, float, double), than the programmer would have better control over the textbox. He is now able to restrict the input of the textbox to only decimals. Thus the extra granularity gives the programmer better control.

Question 11: How does multiprogramming increase the utilization of resources?

When a process is executed by the CPU, eventually there comes a time when it needs to carry out some I/O operations. Since I/O operations do not require CPU resources - in a non-multiprogramming system, the CPU is left idle during this period.

However, this is not the case in a multiprogramming system, while I/O operations are being performed, the CPU fetches and executes the next process waiting in memory. Once the process waiting for I/O operations is ready, the currently executed process is put into a ready state and moved back to main memory. Execution then resumes for the former process. This “process ” continues as long as there is a process waiting in memory to be executed.

Should main memory become full (i.e the quota reserved in main memory to hold processes), a job pool is created in secondary memory. A job pool is a collection of processes that reside on the hard disk, waiting to be loaded into main memory as soon as space is available. This ensures the full utilization of main memory. Virtual memory also plays a role in memory utilization.

Question 12: Discuss the disadvantages of the Linux Systems

Steep Learning Curve Linux is not as easy to learn as the Windows or Mac operating systems. It has a steep learning curve and requires a broader base of computing knowledge than other operating systems. Transition from a Windows operating system to Linux may require you to unlearn things. The file structure isn’t the same i.e there is no “program files” directory or C or D drive. Setting permissions to files and directories aren’t the same, and installation of software aren’t always cut and dry. Some programs require dependencies before the actual program can be installed and mostly these dependencies need to be installed via the terminal(or commandline) which would require abit of effort on the user’s part to learning basic commands. While it is possible to gain understanding of the operating system through practice, it does require a technical threshold to be reached before one is proficient.

Portability Not all hardware and software are compatible with Linux. Common problems users face is the availability of drivers. A user may buy a new laptop which comes with the latest hardware eg: latest graphics card or a sub-woofer. In order for these components to play nice with the operating system, they require drivers - and drivers for new hardware may take alot of time before it surfaces. Most of them will have to be installed manually and also require a hunt for dependencies. Your favorite softwares like Microsoft office or PC games are not compatible with Linux. Although there are alternatives with similar functions available, they may take a bit of time getting use to.

Some propriety business software may not support Linux i.e software provided by the organization you work for may only support the Windows operating system – which may require the extra effort of dual booting with Windows or using a virtual machine in order to run.

Technical Issues Because Linux is open source, every now and then you are likely to run into some technical issues. There are frequent security patches and kernel updates and some of these updates do not work well with your existing software. Thus you might frequently find yourself doing one of two things, either waiting for a patch of that is compatible with the new kernel or you would have to revert back to the old kernel.

Too Many Variants There is a plethora of flavors to choose from when it comes to Linux and each flavor is known as a distribution. Distributions are mostly released targeting a specific audience. For example it you are a journalist and concerned about privacy and the anonymity of your sources, then you might prefer the Tails operating system; if you are a penetration tester or a computer enthusiast interested in computer security then you have the option of Kali Linux, Black Arch and Parrot OS. Each of these distributions have some utilities that make them better suited for a job then others and if you are currently running Kali Linux and encounter a job that is better suited for Black Arch, then you would find yourself doing alot of “distro hopping”. Thus to mitigate this, alot of developers, tech enthusiasts and penetration testers use virtual machines that allow them to host different distros.

References

Silberschatz, A., Galvin, P. and Gagne, G., 2014. Operating System Concepts. 9th ed. Hoboken: Wiley.

Iversen, J. and Eierman, M., 2014. Learning Mobile App Development. Harlow: Addison-Wesley.

Tanenbaum, A. and Bos, H., 2013. Modern Operating Systems. 4th ed. Upper Saddle River,New Jersey: Pearson Prentice-Hall, p.105.

Stallings, W., 2018. Operating Systems. 9th ed. Upper Saddle River: Pearson.

Tanenbaum, A. and Goodman, J., 2008. Structured Computer Organization. 5th ed. Englewood Cliffs, N.J.: Prentice-Hall International.

McHoes, A., 2013. Understanding Operating Systems. Boston, MA: Cengage Learning.

Tutorialspoint.com. 2020. Operating System - Multi-Threading - Tutorialspoint. [online] Available at: <https://www.tutorialspoint.com/operating_system/os_multi_threading.htm> [Accessed 22 March 2020].