Understanding Enforcement of Flow of Processes in Operating Systems

Author's Details: Dr. Zaffar Ahmed Shaikh[1], Dr. Intzar Ali Lashari[2]
[1]Assistant Professor, Faculty of CS & IT, Benazir Bhutto Shaheed University, Lyari, Karachi
[2]Assistant Professor, Institute of Business Administration, University of Sindh, Jamshoro

Abstract
The core functionality of Operating Systems (OSs) is to keep flow of processes or execution of system running. The OS enforces flow of execution and smooth running of processes through implementing several hardware techniques, software applications and algorithms. This paper discusses those techniques in detail. The functionality of OS from context switching level to hardware issues, threats and their solutions have been explained and examined. Thus, it helps to understand how flow of processes has been enforced in OSs through several basic level core steps to advanced third party applications.
Keywords: *data flow, device management, memory management, operating system, processes, process flow, process management*

1. Introduction
Whenever OSs are being discussed, the discussion has normally been done about the GUI based functionalities or discussions about the flow of data. But whenever the discussion has lead one to the scenario where processes are being discussed, that has been extensive. Process Management is an extensive and huge task for an OS (Pinedo, 2015; Toporkov et al., 2014). The OS assigns resources for processes to complete their execution hence achieving the goal of overall system stability and reliability (Tanenbaum & Bos, 2014). The OS makes it possible with the help of several algorithms, memory management, and other proven techniques (Peter et al., 2016). OS enforces flow of processes by allowing a mechanism for them to share information when required. For proper synchronization and functionality, the OS provides them with the specific data structure and allocates memory when required (Lin, Wang, & Zhong, 2014; Rakib, 2016). There seem to be only two tasks which are required by an OS to enforce the flow and those are creation and termination of the process (Choi & Park, 2015; Pinedo, 2015). But in real the case is not so simple. Creation is dependent on several parameters while deletion and switching require other measures to be kept in view (Quigley et al., 2009).

We shall now see how process management is done is OSs to effectively keep the flow of the system. We shall also observe and discuss OS states like deadlocks to see how the system manages those issues.

2. Process and Management Models
Before getting into the discussion, we must discuss the process. A process is an instance of a program. Here program means a computer program. A process can be active or having other states too about which we shall discuss later in the paper (Guzek et al., 2014; Li et al., 2014). When we talk in terms of computer sciences we see that functionality of computer system is basically instructions that are executed. And the execution of those instructions is called the process (Peter et al., 2016; Rakib, 2016).

The question comes in mind whether one process is related to one task that OS assigns or another question can be about one task with several processes. There can also be a question about several processes for several tasks. We observe that OS handles the processes for achieving the desired goal and several hardware related implementations which may lead us outside of out the desired topic like distributed computing and parallel computing (Talk, 2016; Toporkov et al., 2014). Those terms discuss improvement of the process rather than enforcing flow of the process. Process management improvement is itself a separate topic having its own boundaries discussing several other parameters but our focus is discussing the flow of the process (Agne et al., 2014; Pinedo, 2015). Let us discuss few process models and see how the OS handles processes and keeps its flow with discussing the problems that it handles under this heading (Belay et al., 2014; Strange & Morrison, 2014).

2.1. Two-State Process Management Model
One of the major roles of an OS is to allocate and deallocate resources to the processes. If we talk about only one process running, then it seems to be easy for the OS to handle that process by allocating resources and keeping the record in some table or block which we shall discuss. Two-State Model has only two states which are Running and Not-Running (Li et al., 2014; Peter et al., 2016).

Either one process is running or not running (Tanenbaum & Bos, 2014). There is no third concept. If one process is running it is having all the resources it requires to it and when it is in Not-Running state, the resources are detached and allocated to the running process (Li et al., 2014; Pinedo, 2015).

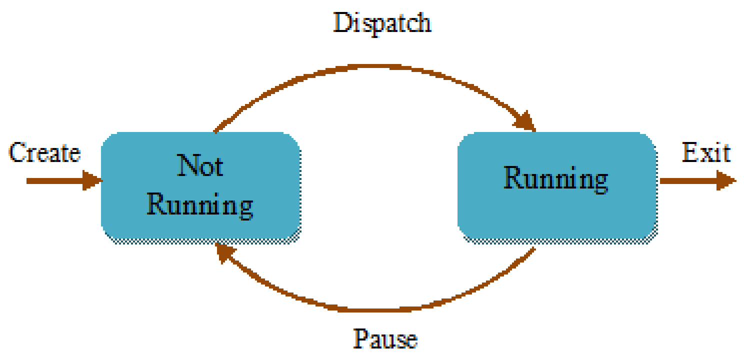

Fig. 1. Two-State Process Management Model

After the execution of the running process, it will change its state and move to Not-Running state. This seems to be very simple model hence easy to deal with (Panda & Bhoi, 2014). If this model is implemented, it is easy to handle for the system to keep the flow of the processes without being handing all those allocations (Strange & Morrison, 2014; Tanenbaum & Bos, 2014). But considering the requirement of fast systems with extra memory, systems need to be upgraded with the responsibility of keeping the process flow (Guzek et al., 2014; Quigley et al., 2009). Design elements of the OS under the heading of Two-State models are Not-Running processes queue, the state of processes as Running or Not-Running, keeping the process under view and keeping its track. This process seems simple but considering the fact of having fast systems with extra memory does not suit for such model to handle (Hambarde, Varma, & Jha, 2014; Stallings, 2014). Hence leaving with the requirement of much more refined models which are further discussed and it is observed that those models are a far better choice (Li et al., 2014).

2.2. Three-State Process Management Model

If only the execution remains the major goal of an OS leaving everything behind, then previously discussed model could also be suitable for all the cases. But in real this is not practical approach considering the fact of having need of fast systems doing multiple tasks (Belay et al., 2014; Pinedo, 2015). And when discussion about fast system starts, we need a model with better functionality that may execute its processes quickly along with having the capability of handling multiple processes at the same time (Guzek et al., 2014; Rakib, 2016).

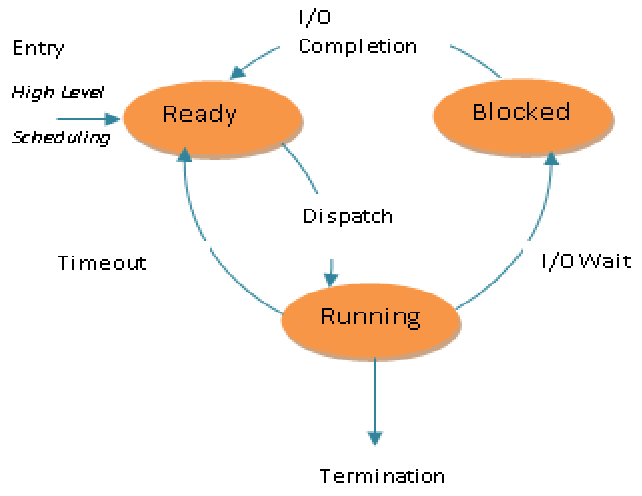

Fig. 2. Three-State Process Management Model

No doubt that handling multiple processes is a complex task. But OS does it effectively. With the above-mentioned model, OS keeps the flow of process with the help of enforcing a type of protocol. There are three types of states of a process in such model (Choi & Park, 2015; Pinedo, 2015).

Running: The process is active and running and being executed.
Ready: The process is in a Ready state and waiting for its execution.
Blocked: The process is in a state that cannot start its execution until some specific interruption occurs. That could be an I/O completion or any other interrupt.

Now in such model, the OS manages the processes in such a way that only one process is in Running state. But there can be more than one processes that are there waiting for their execution. Processes that need to be in Running state first become available in the Ready state (Hambarde, Varma, & Jha, 2014; Talk, 2016). Then after being in the Ready state, they are put to be in Running state. By following the criteria, the flow of processes is enforced (Quigley et al., 2009).

One more consideration is the transition of a process that is in the Running state into the Ready state. Such transition may take place due to several reasons one of which is timeout scenario (Strange & Morrison, 2014). Another scenario can be when some priority has been assigned for several processes by an OS and process having a higher priority enters the system. This transition is from a Running state to Ready state. There can be other transitions too (Bovet & Cesati, 2005; Panda & Bhoi, 2014). The OS keeps the flow of processes and may put a process that is in the Running state into a Blocked state for some time depending upon the data or memory area it is demanding or it has demanding something for which the resource is not available (Toporkov et al., 2014). If the resource is not available and the process has demanding something, then the system has to put it into a Blocked state allowing the next process in the Ready state to start its execution (Mathew, Sekaran, & Jose, 2014). This helps in keeping the flow of the system (Belay et al., 2014; Rakib, 2016).

2.3. Five-State Process Management Model

When the improvement in functionality of OS keeps on happening, the further enhancements in process models were made. Not only for handling processes but enhancements in all other fields from designing phase of SE to hardware infrastructure and the whole architecture was made (Li et al., 2014; Silberschatz et al., 1998).

The enhancement in terms of process management model takes us to a concept of using a portion of the hard disk with the same states but with extended functionality (Mathew, Sekaran, & Jose, 2014). The concept of virtual memory comes here along with several other extended concepts like suspend block and suspend ready.

This process model is much more efficient than the models described earlier. It is a refined form and the processes behave in an efficient way. OS enforces all such models effectively enforcing the flow of processes hence resulting in overall system performance (Guzek et al., 2014; Quigley et al., 2009).

In Five-State process management model, the process may be suspended from an Active (Running), Ready or Blocked state to two new states which are Ready-Suspend and Blocked-Suspend. A process goes to Suspended state from the Running state is called Ready-Suspended while the process that goes to Suspended state from Blocked is called Blocked-Suspended (Agne et al., 2014; Hambarde, Varma, & Jha, 2014).

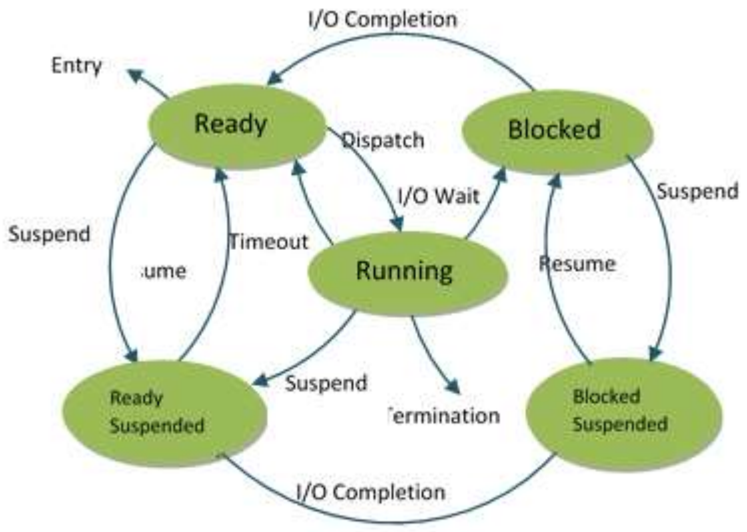

Fig. 3. Five-State Process Management Model

Memory management is the main reason behind suspension of several processes (Bovet & Cesati, 2005; Madhavapeddy et al., 2015). Now let us understand how the flow of processes is enforced in this model.

Suspended but Ready: It talks about suspension of a blocked process (Talk, 2016). It is obvious that we never want to suspend a running process implicitly or a process which is ready. Preference in terms of suspension will be given to those processes which are blocked (Agne et al., 2014; Pinedo, 2015).

***Suspend-Ready then Ready*:** All such scenarios have been implemented by the OS to keep the proper flow of processes and this transition talks about the scenario when there is no ready process in memory. Now to start the execution, a process is required (Guzek et al., 2014; Panda & Bhoi, 2014). There can be another case too which is to have a higher priority by a process in a Ready-Suspend state. OS is designed in a way to keep priorities and states in view to do swapping and other acts (Stallings, 2014).

Blocked then Suspend-Blocked: It is all about memory management when we talk in terms of such models. Considering memory is the core role of an OS (Mathew, Sekaran, & Jose, 2014). Requiring more memory by a running process is normal. And when the running process will require more memory, obviously, it is a better choice for an OS to terminate the blocked process because the OS will efficiently want to free the memory. For this purpose, blocked process will be wiped out of the memory (Madhavapeddy et al., 2015; Silberschatz et al., 1998). Such transitions may also be made for those processes that are in the Blocked state (Agne et al., 2014).

Suspend-Blocked then Suspend-Ready: Several implementations in an OS is made considering the fact that something would occur. Like an interrupt about which we shall discuss later. Just like that scenario, this transition takes place when an event occurs for which there is a wait (Quigley et al., 2009).

3. Kernel and Process Flow

Discussing the kernel and its functionalities is itself a huge topic but discussing its functionalities related to enforcement of process flow can be discussed without a boundary. The kernel is a core program that controls everything (Mathew, Sekaran, & Jose, 2014; Rakib, 2016). It can be said to be the most important process manager within the computer system. It controls everything. It does a lot of data processing, handles I/O, manages memory, handles peripherals, etc. Its interface is requested through a system call (Stallings, 2014; Talk, 2016). How does Kernel help in keeping the flow of processes? Is Kernel the only part of computer system responsible for the flow of processes? Let us discuss how it performs its functionality (Bovet & Cesati, 2005; Li et al., 2014).

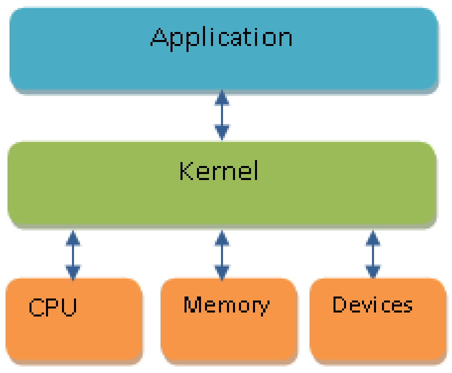

Fig. 4. Kernel and Process Flow

When we discuss major functionalities of the Kernel we observe that access to resources, defining the rule to access and allocation is the core functionality of the OS and there must be a core part of specific functionality on how to perform this task. In OSs, the Central Processing Unit (CPU) is responsible for the execution of the processes. And it is the only main part of a system in terms of hardware by a naive user but this is not the case if observed deeply (Agne et al., 2014; Panda & Bhoi, 2014). There are several other areas in computer systems that are performing their functionality. There must be a mechanism of deciding which process will start its execution. And this must be decided considering several attributes to keep the flow of the process (Pinedo, 2015; Tanenbaum & Bos, 2014). It becomes the responsibility of Kernel to decide among several processes available that who to start its execution. This is decided considering several aspects (Mathew, Sekaran, & Jose, 2014).

Another major goal of OS is to manage memory. Without managing memory and deciding whom to allocate memory and how much memory, the OS cannot keep the flow of the process (Li et al., 2014). Random access memory is responsible within an OS to keep instructions of the program and data if required. But it is a limited memory to some extent. Several processes may demand the memory with or without required need. There can be a scenario that some blocked and suspended processes may have hold of the memory in some cases (Reshetova et al., 2014; Quigley et al., 2009). This becomes the responsibility of Kernel to decide which process to allocate how much memory. And if a process requires more memory, it decides how to do some memory management and allocate to required process (Guzek et al., 2014; Rakib, 2016). This is how Kernel helps in keeping the flow of processes hence resulting in proper flow.

Another core functionality that Kernel performs is to allocate the resources to the peripheral devices like keyboard, mouse, disk drives, printers, etc. (Madhavapeddy et al., 2015). Without managing all such resources, the OS cannot keep the flow of the system. Now one may ask a question about interconnectivity and communication of processes. The kernel does it when required through methods (Li et al., 2014; Tanenbaum & Bos, 2014).

4. Memory and Device Management

When we observe the flow of processes, memory and device management becomes the core area to discuss. Without managing memory and resources including devices, the OS cannot keep the flow of processes. If memory management is not done in an efficient way, the whole system will go into the deadlock or blocked kind of state (Hambarde, Varma, & Jha, 2014; Pinedo, 2015). The kernel provides the access of memory to the processes as they require (Strange & Morrison, 2014). The kernel uses virtual memory, paging and segmentation types of mechanism to managing memory.

Through virtual memory mechanism, each process behave like it is the only process running within the system and that helps in avoiding system crashes (Bovet & Cesati, 2005; Li et al., 2014). The kernel uses virtual memory concept in such a way that more memory can be used than available physically. All such implementations help in keeping the proper flow of the system (Agne et al., 2014; Panda & Bhoi, 2014).

Another core functionality that Kernel performs within an OS is to manage devices. And kernel does it with device drivers. Without managing the devices and keeping the scenario in view when devices require the attention of CPU, one may not think of a continuous flow of the process ((Tanenbaum & Bos, 2014; Toporkov et al., 2014). There is a basic concept of assigning priorities to the devices and when one device of higher priority requests something, it is given preference (Peterson & Silberschatz 1985). The kernel provides a mechanism of letting OS know on how to deal with a specific type of device in terms of allocation of resources and attention. Taking an example of showing something on the screen, the request is made to the Kernel to show something on the screen, and Kernel further forwards the request (Madhavapeddy et al., 2015; Rakib, 2016).

5. Interrupt, System Call and Exception

A flow of processes can easily be implemented if the requests generate in a serial manner. But practically it is not possible. A system resource and attention or CPU is expensive (Reshetova et al., 2014). One process with a lower priority may use those resources when no higher priority process requiring the access.

When processes request system services, that mode is called system call. A process does not have access to the resources within the computer system (Panda & Bhoi, 2014). To keep the flow of processes and execution, the OS has implemented within a mechanism of system calls that are generated by the processes for the Kernel requesting access to some resources (Li et al., 2014; Quigley et al., 2009). This mechanism helps in enforcing the flow. There can be two types of processes. One which is system level processes may have higher privileges and accesses to the system resources as compared to the processes that are user processes (Criswell, Dautenhahn, & Adve, 2014). System calls are generated to communicate to other devices or external IO entities (Peterson & Silberschatz 1985).

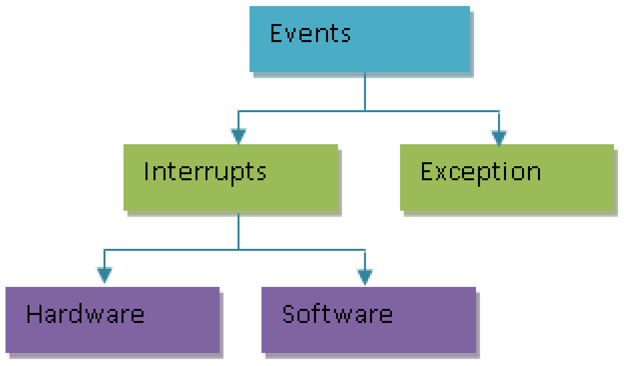

Fig. 5. Interrupt, System Call and Exception

When discussing an interrupt as the name suggest is an interruption for the CPU. When the CPU is busy doing some other task or doing nothing, and a process needs the attention of CPU, it generates an interrupt for getting the attention of CPU. There can be two types of interrupts mainly with categorizing deeply (Criswell, Dautenhahn, & Adve, 2014; Li et al., 2014). One interrupts are those which are hardware interrupts like pressing any key from keyboard generates an interrupt. Those are called interrupts while on the other hand if an interrupt is generated through software is called an exception (Peterson & Silberschatz 1985; Stallings, 2014).

Interrupts that are generated through software are also called traps (Quigley et al., 2009). How the flow of processes is maintained when an interrupt is generated. There is a special routine which is called ISR (Interrupt Service Routine). The control is transferred to this routine. This decides through a mechanism on how to deal with this interrupt and hence performs a further action. Hardware interrupts are generated by some hardware which resides outside of CPU (Silberschatz et al., 1998). Those interrupts can be generated by peripheral devices, ports, various cards like sound or network etc. (Hofmann et al., 2013; Madhavapeddy et al., 2015).

It is necessary to mention here that the system called is a kind of built-in system functionality and subroutine which has been implemented while the interrupt is an event that occurs (Belay et al., 2014). Also, one major difference is that interrupts are hardware events and those events are asynchronous due to its nature. For example, a key of hardware can be pressed asynchronously (Peterson & Silberschatz 1985; Rakib, 2016). While on the other hand system calls are synchronous (Criswell, Dautenhahn, & Adve, 2014; Li et al., 2014). By going to the subroutine and deciding on how to deal with the interrupt remembering the last state, and then coming back to continue the execution ensures the proper flow of the processes going on in an OS (Reshetova et al., 2014; Tanenbaum & Bos, 2014). A mapping is done and a mechanism is designed for interrupt vector.

We must realize that exclusion cannot be made for external entities. The system is connected to several external entities and the OS has to manage all the resources including hardware too (Panda & Bhoi, 2014). It is the responsibility of the OS to keep the flow of processes to avoid deadlocks and waste of resources. The devices attached to the system may require services time to time (Peterson & Silberschatz 1985). And there is no synchronous way of determining when the attention will be demanding of those resources. There can be self-determining and query to several services whether they need attention or not but that surely is a waste of time in many cases (Criswell, Dautenhahn, & Adve, 2014; Tanenbaum & Bos, 2014).

Interrupt itself has two types. One type of interrupts is that can be masked and other which cannot be masked. There can be very critical sections and parts of interrupts that are critical and require urgent attention of CPU (Madhavapeddy et al., 2015).

6. Process Control Block or PCB

Our focus of discussion is on how OS maintains the flow of the system by using several data structures, techniques and does those implementations. Discussing PCB becomes crucial when we discuss how processes are handled (Strange & Morrison, 2014). When we shall talk about all the process handling mechanisms related to effective flow, then it will be clear how OS performs its role in terms of handling the processes (Madhavapeddy et al., 2015).

There is not a single process that is active in the system. There are multiple processes that are active in the system (Mathew, Sekaran, & Jose, 2014). Also, they have related information with them. Each process has some information associated with it about which we shall discuss. Without knowing that specific information, the OS cannot utilize itself in allocating resources or managing the whole system (Belay et al., 2014; Stallings, 2014). There must be a mechanism for storing all the information related to processes at a place which shall have all the required information about the process from timing to resources it requires (Hofmann et al., 2013; Rakib, 2016).

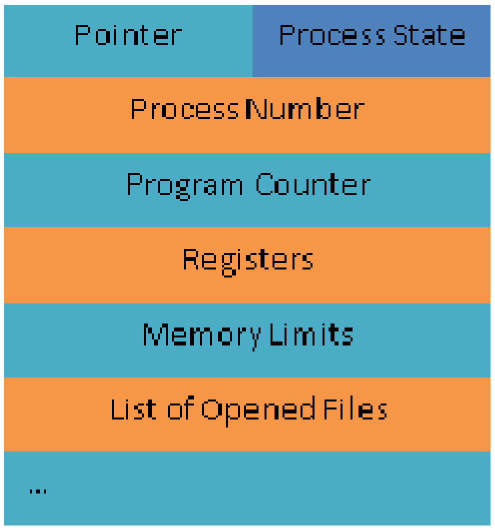

Fig. 6. Process Control Block or PCB

What if we say that combinations of PCBs describe the current state of the whole OS? This statement would be correct. It contains the core central functionality in process management. When interconnectivity of processes is discussed then the term of pointers from one PCB to another is discussed too (Reshetova et al., 2014). Every kind of process management and tasks related to processes are managed by PCB. When we talk about the actual functionality of PCB and the tasks it performs, then we come up with the core tasks it performs (Lin, Wang, & Zhong, 2014; Talk, 2016). And the job that system is performing cannot be excluded from OS's view to keep the flow of the processes. PCB contains basic information about a job in an OS and some other information too like it keeps the information about how much processing it has completed and information about the remaining process.

Its storage location, including the time it has spent during execution and remaining time (Quigley et al., 2009). Whenever OS needs to check the processes and resources, it takes help from such infrastructure it has formed which are independent in a sense but relying on OS core functionalities like Kernel etc. The OS uses such structures to keep a flow of the processes (Criswell, Dautenhahn, & Adve, 2014; Rakib, 2016). Whenever PCBs are discussed deeply, it is observed that some more concepts are related to PCBs in terms of processes it keeps track of (Agne et al., 2014; Bovet & Cesati, 2005; Hofmann et al., 2013).

Each process is unique in terms of address and identifier (Belay et al., 2014; Choi & Park, 2015). It is uniquely identified thus resulting in the identification of the resources it is using and all other functionalities that are associated with it.

There is also a record of the Current state of the process. Without keeping track of the Current state, the further steps cannot be taken. Also, the state of the process is identified whether the process is in Running state, Ready state, Blocked state, Ready-

Suspended or Blocked-Suspended. Also, the information about a process related to the resources is kept, for the duration, it has used a resource (Lin, Wang, & Zhong, 2014).

CPU keeps track of several other measures associated with processes and hence uses those measures in deciding how to keep the flow of processes. The OS keeps track of parent and child processes and their queues to ensure the flow of processes (Talk, 2016). It also decides about several rights that need to be granted in terms of resources. Each process is assigned a unique identifier. It keeps track of the next instruction which is to be executed after the current instruction.

The OS also keeps track of several registers to keep track of processes that are using registers. Also, a list of devices associated or allocated to a resource is maintained and used when required. The OS keeps track of all the information associated with a process and information regarding how they are linked with each other to keep the flow of the processes (Stallings, 2014).

7. Process Termination

The OS keeps the flow of processes but ideal scenario is not there always. There can be situations when the process terminates without completing its execution. There can be many reasons for that but the OS is designed in a way that if process termination happens, the system does not go into a Loop or Halt state (Rakib, 2016). OS gracefully suspends the current execution, displays the message if required and shifting to the next execution. The termination of the processes can have many reasons but the role of OS becomes much more important when it keeps the flow in this situation (Agne et al., 2014; Tanenbaum & Bos, 2014).

- Situation when a user logoffs and ongoing process is terminated
- Process itself sends a terminate request due to some reason
- A situation when a process terminates due to an invalid instruction
- Terminates when access of write is demanded a read-only file
- The process may terminate when access to an outer boundary element is demanded. Like the extra bit that resides outside of boundary
- When misuse of data cause in termination of the running process
- When the parent requests and the child has to be terminated in terms of such requests that require termination. In this case, parent resides
- Parent terminates and then child terminates automatically
- Termination occurs when timeout scenario takes place
- Process terminations is done for prevention of deadlocks or for helping the system resuming from a deadlock
- When memory for the process is unavailable and termination is required for allowing another process to start its execution
- Termination due to invalid arithmetic operation i.e. divide by zero
- Few errors may cause the process to terminate its execution
- Also, OS terminates the process upon its execution

Process termination does not mean that system will go into a Halt state. Instead, those measures are kept in view to deciding which process to start the execution (Hofmann et al., 2013)..

8. Context Switching

The OS maintains information regarding processes and all information regarding its states (Lin, Wang, & Zhong, 2014). But there is not an ideal scenario of executing one single process until its completion. That is not a practical approach. If this happens only one process will be able to execute at one time and all other processes will suffer. By this, there will not be any ideal scenario of using a system (Strange & Morrison, 2014).

Context switching is the name of storing and restoring states. By this, the simple definition becomes when an OS when considers about one process to be stored with its state and then moving to another process to start its execution. After its execution when required, storing its state with parameters required using system architecture moving to another process. This is called context switching (Choi & Park, 2015; Silberschatz et al., 1998). In real how we see a computer system is like it is performing all the operations at the same time. We start media player, use the internet and there are a lot of functionality that keeps on performing at the backend. But in real no single application is using all the resources or CPU (Hofmann et al., 2013; Rakib, 2016). In real the context switching is being done to enforce flow of processes. Context switching is being done in such a fast way that a user does not notice that (Stallings, 2014).

Context switching is not a simple task to do. It stores memory mapping, values, registers save and load, table and lists if required to return to a process and continue its execution (Lin, Wang, & Zhong, 2014). It performs the same functionality for the processes it switches to.

It must be understood that context switching is done to make the flow of processes intact without halting the system and there can be several measures that enforce and implement the concept of context switching. One of which is when the system needs to handle the interrupt. If an interrupt is generated, then system through context switching moves to next process and start its execution or perform the action depending upon the interrupt that has been occurred (Agne et al., 2014; Reshetova et al., 2014).

The major role of an OS is multi-tasking. Without multitasking, the need of using a computer system may feel unnecessary. As discussed earlier, execution of one process until its completion does not provide us with required functionality (Tanenbaum & Bos, 2014). So, that multitasking is required. There can be several reasons for this to happen from an I/O operation to pre-emption of CPU but whenever this happens, the old state including required data is stored for returning and resuming (Talk, 2016).

9. Scheduling and Process Selection Algorithms

To make a proper flow of the processes, the OS makes use of scheduling and selects the lucky process for a Running state. A process moves between various queues. If we discuss scheduling queues, then we come up with a ready queue which discusses ready processes in main memory that are waiting for their turn to start execution (Silberschatz et al., 1998). Also, there is a wait queue where processes are residing and waiting for some action or interrupt. They reside there until a specific interrupt or action is performed. Also, there is a device queue where processes wait for some I/O operation. All such structures and methods are only for the enforcing proper flow of processes (Choi & Park, 2015; Rakib, 2016).

The dispatcher is something which needs to be discussed here. We have discussed a process which is ready and then executes. Dispatcher plays its role by taking the first process from the ready queue and puts it forward for execution. Dispatcher provides a process with the control of CPU (Strange & Morrison, 2014; Tanenbaum & Bos, 2014).

Taking an example of design model during a software design, we have several models but we observe that not all those models fit in the system at the same time. We must understand the nature of the system and then implement the desired model (Quigley et al., 2009). Just like that, we have several classical scheduling algorithms and select that algorithm that fits best as per our requirement that may help to enforce the flow of the process (Hofmann et al., 2013; Stallings, 2014). We have FCFS (First Come First Serve) which is a nonpreemptive scheduling and serves without considering any other parameter instead it considers the arrival time (Lin, Wang, & Zhong, 2014; Rakib, 2016). The process that first comes into for execution will be served first. We have SJF (Shortest Job First) which may be preemptive or nonpreemptive (Reshetova et al., 2014). It during the execution decides and switches among processes based on its size (Agne et al., 2014). Also, we have priority based scheduling algorithms, multilevel feedback, Round-robin and other algorithms and with the help of those algorithms based on the scenario, we enforce the flow of the process (Hambarde, Varma, & Jha, 2014; Tanenbaum & Bos, 2014).

10. Conclusion

This paper discussed core functionality of OSs, which is to keep the flow of processes or the execution of the OS running. The paper explained how OS enforces flow of processes through implementing several proven techniques. Several techniques of using algorithms for enforcing flow of execution and smooth running of processes within OS were discussed. This paper also detailed on how OS uses different methodologies and algorithms including implementing several hardware techniques as well as software applications that further helped enforcing proper flow of processes in OSs. Also, OS functionality from very basic level i.e., context switching level to hardware issues, threats and their solutions were reviewed. The paper helped understand how proper flow of processes have been enforced in OSs by explaining several basic level core steps to advance third party applications.

References

Agne, A., Happe, M., Keller, A., Lubbers, E., Plattner, B., Platzner, M., & Plessl, C. (2014). ReconOS: An operating system approach for reconfigurable computing. *IEEE Micro*, *34*(1), 60-71.

Belay, A., Prekas, G., Klimovic, A., Grossman, S., Kozyrakis, C., & Bugnion, E. (2014, October). IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *OSDI* (Vol. 14, pp. 49-65).

Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel: from I/O ports to process management*. "O'Reilly Media, Inc.".

Choi, H., & Park, S. (2015). Evaluations of Hardware and Software-Based Context Switching Methods in Cortex-M3 for Embedded Applications. *Cortex*, *6*, 7.

Criswell, J., Dautenhahn, N., & Adve, V. (2014, May). KCoFI: Complete control-flow integrity for commodity operating system kernels. In *Security and Privacy (SP), 2014 IEEE Symposium on* (pp. 292-307). IEEE.

Guzek, M., Pecero, J. E., Dorronsoro, B., & Bouvry, P. (2014). Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. *Applied Soft Computing*, *24*, 432-446.

Hambarde, P., Varma, R., & Jha, S. (2014, January). The survey of real time operating system: RTOS. In *Electronic Systems, Signal Processing and Computing Technologies (ICESC), 2014 International Conference on* (pp. 34-39). IEEE.

Hofmann, O. S., Kim, S., Dunn, A. M., Lee, M. Z., & Witchel, E. (2013, March). Inktag: Secure applications on an untrusted operating system. In *ACM SIGARCH Computer Architecture News* (Vol. 41, No. 1, pp. 265-278). ACM.

Li, S., Yang, J., Chen, W. H., & Chen, X. (2014). Disturbance Observer-Based Control. *International Standard Book Number*, *13*.

Lin, F. X., Wang, Z., & Zhong, L. (2014). K2: a mobile operating system for heterogeneous coherence domains. *ACM SIGPLAN Notices*, *49*(4), 285-300.

Madhavapeddy, A., Leonard, T., Skjegstad, M., Gazagnaire, T., Sheets, D., Scott, D. J., ... & Crowcroft, J. (2015, May). Jitsu: Just-In-Time Summoning of Unikernels. In *NSDI* (pp. 559-573).

Mathew, T., Sekaran, K. C., & Jose, J. (2014, September). Study and analysis of various task scheduling algorithms in the cloud computing environment. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on* (pp. 658-664). IEEE.

Panda, S. K., & Bhoi, S. K. (2014). An effective round robin algorithm using min-max dispersion measure. *arXiv preprint arXiv:1404.5869*.

Peter, S., Li, J., Zhang, I., Ports, D. R., Woos, D., Krishnamurthy, A., ... & Roscoe, T. (2016). Arrakis: The operating system is the control plane. *ACM Transactions on Computer Systems (TOCS)*, *33*(4), 11.

Peterson, J. L., & Silberschatz, A. (1985). *Operating system concepts* (Vol. 2). Reading, MA: Addison-Wesley.

Pinedo, M. (2015). *Scheduling*. Springer.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).

Rakib, S. S. (2016). *U.S. Patent Application No. 15/083,215*.

Reshetova, E., Karhunen, J., Nyman, T., & Asokan, N. (2014, October). Security of OS-level virtualization technologies. In *Nordic Conference on Secure IT Systems* (pp. 77-93). Springer International Publishing.

Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). *Operating system concepts* (Vol. 4). Reading: Addison-wesley.

Stallings, W. (2014). *Operating Systems: Internals and Design Principles| Edition: 8*. Pearson.

Strange, J. A., & Morrison, J. A. (2014). *U.S. Patent No. 8,719,531*. Washington, DC: U.S. Patent and Trademark Office.

Talk, O. M. (2016). Operations management.

Tanenbaum, A. S., & Bos, H. (2014). *Modern operating systems*. Prentice Hall Press.

Toporkov, V., Toporkova, A., Tselishchev, A., & Yemelyanov, D. (2014). Slot selection algorithms in distributed computing. *The Journal of Supercomputing*, *69*(1), 53-60.