International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 06 | June 2020 www.irjet.net p-ISSN: 2395-0072

Analysis of Process Structure in Windows Operating System

Mayuresh Kulkarni[1]

[1]*Computer Engineering (Pursuing), Department of Computer Engineering, Bharati Vidyapeeth College of Engineering, Navi Mumbai, Maharashtra, India*

Abstract - *Operating Systems have been one of the most intricate piece of softwares since they came into existence. Due to recent advancements in technology, the OSes too, have become extremely powerful. From responding within few milliseconds to parallelly computing many things in the background, it becomes difficult to understand how it works inside. Microsoft Windows handles programs differently than other operating systems. The reasons for which are discussed in this text. Analyzing of process structure and its management inside CPU and Kernel with the help of Windows, is the core matter. Components, types, process table, u-area and context are explained in detail.*

Key Words: operating system, process, Windows, analysis, kernel, u-area

1. INTRODUCTION

Operating Systems can be incredibly simple and easy-to-use at the user level and at the same time be the most intricate and complex piece of software. Microsoft Windows, like many other operating systems, is also divided into 3 sections:

1. User / Application Level
2. Kernel Level
3. Hardware Level

It is similar to the UNIX based systems, but it is crucial to note that Microsoft Windows is not a UNIX based operating system. Windows stopped using the 8-bit legacy code after the release of Windows XP in October 2001, and also the original CPM programs from the 1990s. Microsoft Windows now houses Windows NT (also popularly known as OS/2) in its code. Although previous versions of Windows were based on MS-DOS, it now is a mixture of MS-DOS and NT features. This helps to provide a much better code, which in-tern gives more efficiency and long-term usability from Microsoft's perspective.

Before diving into process structure and analysing how Windows handles them, it is essential to understand the basics of the 3 sections of the Windows Operating System. This will be immensely helpful to get a deeper understanding of later sections of this text.

The Operating System is the layer between the computer hardware and the end-user using the computer. Anything done on the computer (as simple as just moving the mouse pointer from one location to another) requires the 'attention' of the operating system. To explain in a better way, consider this example.

A user is accessing this computer and during the session, the user clicks/opens some program installed on his machine. Maybe a browser or something heavy which will consume little resources to function properly. The browser before opening is just an application stored in the computer (may or may not be ready to be opened. This depends on the machine to machine). When the user signals to open the application, it becomes a program. It is important to know the difference between a program and a process before understanding them at a deeper level of abstraction, and that too at Kernel level in operating systems. A program is a static sequence of instructions, whereas a process is a container for a set of resources used when executing the instance of the program.

When the operating system is instructed to open the program, it is converted into a process and the browser is now pushed into the process/mechanism of CPU scheduling. According to the definition, CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold(in 'waiting' state) due to the unavailability of any resource like I/O, etc, thereby making full use of CPU. CPU scheduling aims to make the system efficient, fast, and fair. There are different CPU scheduling algorithms like FCFS (First Come First Serve), SJF (Shortest Job First), Round Robin, etc. to increase the efficiency of the processor.

Now, that the browser program is converted into the process, it is officially handed over to Kernel by the user space (or user-level) for further computing and execution. What the Kernel does is none of the business of user-space, and it is only concerned with what the Kernel returns and what the next program is (for adding it to queue). The Kernel then checks the requirements demanded by the process and allows them for execution if they are available. If not, the process goes to the next task/job programmed if possible, otherwise, it goes into the 'waiting' state until the resources are available. The following figure shows the process state life cycle. It contains the different stages through which a process can go inside the Kernel.

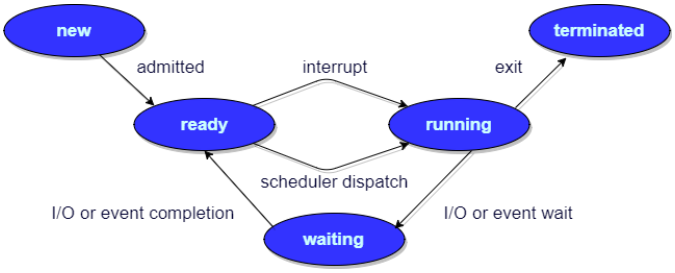

Fig -1: Process State Transition Diagram [1]

On a side note, a visible example of this is high graphics games played on low specifications computers. Since the requirements of the game (process) are high, most of the time they are not available to be allotted. So, they wait for a few milliseconds, and during this time, if the next frames are not loaded, the user experiences a delay in displaying them. This is also called 'lag' in layman language.

Once the resources are available, the process resumes execution until it is completed. This cycle continues according to the requirements of the user and the program's context/intention. The intent is also taken into consideration because in earlier times, viruses and other malware producing programs were also treated as other processes and the verification of authenticity was done once they entered the CPU [2].

Now that the overall life of a process inside a general operating system is clear, it will be easier to understand how the Microsoft Windows operating system handles this.

2. COMPONENTS OF PROCESS AND THREAD IN WINDOWS

At the highest level of abstraction, a Windows process comprises of the following information:

1. Private Virtual Space Address (PSVA): Set of virtual memory addresses that the process can use.
2. Executable Sub-Process: Defines initial code & data is mapped to process's PSVA.
3. List of Open Handles: Used to map various resources such as semaphores, synchronization objects, files shared by more than one threads in the process.
4. Security Context: *Access* token that defines the user, security groups, privileges, attributes, claims, User Account Control (UAC) virtualization state, session, etc.
5. Process ID (PID): Unique identifier which is internally a part of client ID.
6. List of Threads of execution: Every process is divided into sub-processes which are assigned some task to complete. This sub-process is known as thread. Every process has at least one thread. Therefore, it is theoretically not possible to have an 'empty' process.

The important thing to note here, is that, all the above-mentioned information can be tracked by knowing just one component: the *PID.* The process id can give access to the above-mentioned information to authorised users (users having appropriate rights inside the system, like administrator or, super user in case of Linux). There are several ways to get PID of any process, but Windows provides us with 2 default ways of getting this PID. One, using Task Manager (Only for Windows 8 and above). And second is using Command Prompt (Admin).

1. Task Manager: For this method, it is essential to have Windows version 8, 8.1 or 10 installed on your machine. Inside Task Manager, click on 'More Details' and navigate to 'Details' tab. Most of the information which a regular user will need, such as Name, PID, Status, Access Privilege, CPU (cores), Memory used (real-time) and a basic description of what the process does, is given there. From here, it is possible to get the process id of any application (as the name of the application is given under Name tab and not the system process name. To get the system process name, the 2nd method is more appropriate as this method is optimal for GUI usage and the next method is appropriate for Command-Line Interface usage).
2. Command Prompt: This method is applicable to all Windows versions above Vista. In the machine, open Command Prompt (CMD) with Admin Privileges. If the system is not inside current user's folder, then navigate to it. Then type '*tasklist*' command in the CMD. Depending on system, it may be quick or take time to gather the information. Again, here information such as Name (called Image Name), PID, Session Name (mostly Console or Services), Session ID (0,1,2,etc depending upon Session Name) and Memory usage (of the instance at which the Command Prompt tracked that particular process) is given.

In the recent versions of Windows 10 (after 2019), Microsoft has provided an additional option to see MANY more details of each and every process. In any Windows 10 version (build 19559 and above), go to Details tab and right click on any of the columns and click on 'Select Columns' option. There, one can find the currently selected columns and the other unselected ones which can be now enabled. Some of the unselected columns (by default) are: CPU time (time spent in CPU), Peak Working Set Memory (max memory used during CPU time), Commit Size (Amount of memory space reserved by the OS for a process), Paged Memory (Amount of pageable kernel memory allocated by kernel or drivers on behalf of the process), Threads (created and handles by the process), GDI object details, I/O Read-Writes (just mentions the number, not the source or the disk/drive it read/written from),

Architecture (x86 or x64; very important for hardware level Machine Learning Model Preparation), GPU (cores utilized) and GPU Memory used (real-time).

A thread is entity within a process that Windows schedules for execution. As mentioned above, a process cannot run without a thread. Like process, a thread includes the following components:

1. The contents of a set of CPU registers representing the state of the processor.
2. Two stacks–one for the thread to use while executing in kernel mode and one for executing in user mode.
3. A private storage area called *thread-local storage* (*TLS*) for use by subsystems, run-time libraries, and DLLs.
4. A unique identifier called a *thread ID* (part of an internal structure called a *client ID*; process IDs and thread IDs are generated out of the same namespace, so they never overlap).

Threads sometimes have their own security context, or token, which is often used by multithreaded server applications that impersonate the security context of the clients that they serve.

3. TYPES OF PROCESSES IN WINDOWS OPERATING SYSTEM

There are 4 basic types of user mode processes in Windows 10:

1. User Processes: 32-bit & 64-bit processes that are generated when launching an application (either by user or by another application) are included in User Processes category.
2. Service Processes: These are processes that host Windows services, such as the Task Scheduler and Print Spooler services. Services generally have the requirement that they run independently of user logons. Many Windows server applications, such as Microsoft SQL Server and Microsoft Exchange Server, also include components that run as services.
3. System Processes: These are fixed, or hardwired, processes, such as the logon process and the Session Manager, that are not Windows services. E.g. processes generated by Intel Networking module that communicates directly with the router (consider dealing with Wi-Fi here), then this process will come under System Process.
4. Environment subsystem server processes: These implement part of the support for the OS environment, or personality, presented to the user and programmer.

These processes are identified by the operating system at user level. At Kernel level, these processes are treated according to their context. Before understanding about the context of process and its usage in optimization, it is essential to understand about *u-area.* U-area is the next section of this text.

4. FIELDS IN PROCESS TABLE AND *U-AREA*

U-area is a section of process information which contains the execution time environment details about that particular process. This is valid if the process starts and completes its execution in the same environment. Otherwise, in most cases, there are different u-areas according to different threads as per the requirement of the program/process. It is essential to note that in majority of situations, more than one u-areas of process are there. And each u-area specifies the executing environment details. So, the operating system has to allot the requested resources and generate the environment necessary to complete execution of the process.

Every process that comes to CPU for execution is first registered in a 'process table' which is maintained by the Kernel of the operating system. This is true for each and every operating system and is not limited to Windows only. So, this process table contains different types of information about the process. The fields in the process table are [4]:

1. The *state* field identifies the process state. The states are according to Fig-1 given above.
2. The process table entry has fields that allow to locate the process and its *u area* in main memory (or in secondary storage). The kernel uses this information to do *context switch* to the process when the process moves from 'preempted' state to 'user running' state. Kernel also uses this info when swapping (or paging) process to and from main memory.
3. The process table has a field which gives process size, for the kernel to know exactly how much space it will take.
4. User IDs (or UIDs) determine various process privileges.
5. PIDs specify relationship of the process. ID fields are setup when the process enters the 'created' state in the *fork* system call. While the *fork* call is used by all operating systems, for external usage it is limited to Linux only.
6. Scheduling parameters allow the kernel to determine the order in which processes move from 'kernel running' and 'user running' states.
7. Various timers give process execution time and kernel resource utilisation, used for process accounting and for calculation of the process scheduling priority.

The urea contains the following fields that further characterise the process states:

1. A pointer to the process table identifies the entry that corresponds to the u area.
2. The real and effective user IDs determine various privileges allowed to process, such as file access rights.
3. Timer fields record the time the process spent executing in user mode & in kernel mode.
4. An array indicates how the process wishes to react to the signals.
5. The control terminal field identifies the "login terminal" associated with the process if one exists.
6. An 'error field' records errors encountered during system call (failure to allocate disk space, failure to communicate with external hardware like scanner, printer, etc).
7. A return value field contains the result of the system call.
8. I/O parameters Describe the amount of data to transfer, the address of source data array in user space file offsets for I/O, etc.
9. The current directory and current route describe the file system environment of the process.
10. User file descriptor table records the files the process has *open*.
11. Limit field restrict the size of a process and the size of a file it can *write*.
12. A permission modes field masks mode settings on files the process *creates*.

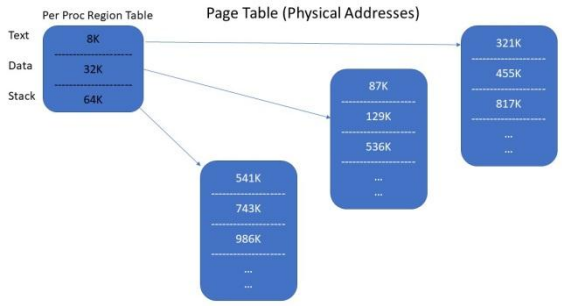

Fig -2: Virtual to Physical Address Mapping Diagram

The System-V kernel divides the virtual address space into logical *regions* (a contagious area of the virtual address space of a process that can be treated as a distinct object to be shared or protected). The kernel contains a region table and allocates an entry from the table for each active region in the system. The region table contains the information to determine where its contents are located in the physical memory. Every process contains a private *per process region table*, called *pregion*, which may exist in process table, u-area or maybe in some discrete location with a shared virtual address. Every *pregion* entry points to a region table entry & contains the starting virtual address of the process.

The above image shows 2 processes A & B, their regions, *j* and virtual addresses where the regions are connected. The processes share text region 'a' at virtual addresses 8K and 4K, respectively. If process A reads memory location 8K and process B reads memory location 4K then they actually read the memory location in region 'a'. The data regions & stack regions of the two processes are private.

A process can access its u-area only if it is in kernel mode. It cannot access u-area from user mode. This is because the u-area partially defines the context of the process and also because the kernel can access only 1 u-area at a time. When the kernel schedules the process for execution (in kernel mode), it also finds the corresponding u-area in the physical memory and makes it accessible to the process by mapping it to a virtual address and providing the virtual address to the process.

This is all about the u-area of processes in Windows. The next part of process analysis is understanding the Context of a process.

4. CONTEXT OF A PROCESS

The context of a process comprises of its *User-Address Space, Content of Hardware Registers & Content of Kernel Data Structures.* The context is actually the union of 3 sub-contexts:

1. User-Level Context
2. Register Context
3. System-Level Context

A table which gives contents of these 3 sub-contexts is given below:

User–Level Context	**Register Context**	**System–Level Context**
Process text, data, user stack, shared memory (virtual address details).	1. Program Counter – Virtual Address of the next instruction which will be executed by the CPU. 2. Process Status Register (PS) – Specifies the hardware status of the machine. 3. Stack Pointer – Follows PS. Contains current address of the next process in queue. 4. General Purpose Registers – Stores data generated by the	1]. Static Part: A. Process State B. Process Control Information C. *Pregion* Entries 2]. Dynamic Part: A. Stack frames of Kernel Procedures (which are required by the process) B. Stack of other system level contexts

	process during execution/during its lifetime.	

The system level context of a process has 2 parts [4], static and dynamic as shown above. It is interesting to note that every process has only 1 static part, which can be referenced by Kernel for high level use (for example, in a navigation application, a process which returns the current location of the device will be obviously using a system context and the process will be running all the times during the application's runtime. Therefore adding this data to static part of the process will not only be beneficial for application, but also for the kernel for repeated reference to same data and now at the same place, so the effective searching time is reduced giving a faster response to the application), but can have multiple dynamic parts for a single process (for example, consider the same navigation but this time the user is repeatedly changing the settings of, let's say, location accuracy. In this case, the location accuracy of the device is dependent on not only the satellites connected but also other technologies such as Bluetooth, internet connection type, whether the device is connected to Wi-Fi or is using mobile data from nearest coverage tower and so on. Due to continuous change in accuracy settings, the application has to change the requested resources again and again and eventually, the dynamic part of the system level context is also changing).

5. CONCLUSION

Computers earlier used to take minutes to do simple arithmetic, but the recent advancements in technology and the computer world has given so much power to these electronic devices that they can control some complicated machinery situated thousands of miles away by sitting at a coffee shop or while waiting in bus line. Parallelly, this is possible only because the operating systems are evolving at the same, or higher rate. In this text, an in-depth study of how Microsoft Windows operating system analyzes and handles processes is done. It is really astonishing to see something which seems to transparent, simple and elegant and be so complicated yet perfectly planned at the same time.

6. FUTURE SCOPE OF RESEARCH

The next stage of research is to study how Microsoft Windows operating system will execute the processes to reduce the time required to compute instructions and how the operating system dynamically does parallel processing while keeping the kernel load-free at all times.

Further, after execution, the study of how Windows manages to change contexts of processes without letting the system to bottleneck will also be interesting.

REFERENCES

[1] Neetu Goel, Dr. R. B. Garg, "A Comparative Study of CPU Scheduling Algorithms", International Journal of Graphics and Image Processing, Vol. 2, Issue – 4, November 2012

[2] Sukanya Suranauwarat, "A CPU Scheduling Algorithm Simulator", 37th ASEE/IEEE Frontiers in Education Conference, October 10 – 13, 2007, Milwaukee, WI

[3] Mayuresh Kulkarni, Torana Kamble, "INTEGRATION OF MACHINE LEARNING INTO OPERATING SYSTEMS: A SURVEY", International Journal of Creative Research Thoughts (IJCRT), ISSN:2320-2882, Volume.8, Issue 4, pp.1270-1273, April 2020

[4] Maurice Bach, "THE DESIGN OF THE UNIX OPERATING SYSTEM", ISBN: 81-203-0516-7