Operating System Feature Comparison: Processes and Acheduling

by Sinan Topkaya

CS 444 - Spring 2016

Abstract: This paper will examine processes, threads and CPU scheduling for Windows and FreeBSD operating systems. It will mainly be about how these operating systems implement them and how is it different compared to Linux. Specifically, it will answer the questions: How do they differ? How are they the same? and Why do I think these similarities or differences exist?

WRITING ASSIGNMENT - PROCESSES AND SCHEDULING COMPARISON

Windows

Each Windows process is reptesented by an executive process(EPROCESS) structure. EPROCESS con tains and points to a number of other related data structures. Likewise if each process has o ne or more threads, each of them are repsented by and executive thread(ETHREAD) structure. To understand how Windows implements Processes and Threads we have to understand the structure EPROCESS and ETHREAD.

Processes: In this section I will write about how various data strctures involved in process state manipulation and management. Then I will write about how and when those data structures are created and filled out, while creating or terminating processes.

The EPROCESS and most of its data structures exist in system address space, except for process environment block(PEB), which exist in the process address space. The reason for that is because PEB contains information accessed by user-mode code. For each process that is executing Win32 program, the Win32 subsystem process (Csrss) maintains a parallel structure called the `CSR_PROCES`, kernel-mode part of the Win32 subsystem(Win32k.sys) maintains a per-process data structure, `W32PROCESS`. The W32PROCESS structure is created the first time a thread calls Windows USER or GDI function implemented in kernel code. Every EPROCESS structure is en capsulated as a process object by the executive object manager, but because processes are not named objects, they are not vissible in WinObj tool. Many other drivers and system components, can choose to create their own data structures to track information on a per-process basis. It is also important to take data structure sizes into considiration.

Process Object: `EPROCESS` structure has many key fields. APIs and components are divided into layered modules with their own naming. For example one of the key field(member) of a executive process structure is called Pcb, or in other words process control block. This is a struture of type `KPROCESS`, for kernel process. Although routines store information in EPROCESS, the dispatcher, scheduler and inturrept/time use KPROCESS. Therefore allowing a layer to exist between high-level functionality and its underlying low-level implementation of functions, this also helps prevent unwanted dependencies between layers. Data structure `PEB` lives in the user-mode adress space of the process, it contains information needed by image loader, the heap manager, and other Windows components that need to access it from user mode. The `CSR_PROCESS` structure contains information about processes that is specific to the Windows subsystem. `W32PROCESS` strcuture contains information that the Windows graphics and window management code in

kernel needs to maintain state information about GUI processes.Each of these data structures have different process structure and hold important information about the Process.

CrateProcess: Creating a process involves many steps

1) Validate parameters; coonver Windows subsystem flags and options to their native counterparts; parse, validate, and convert the attribute list to its native counterpart.
2) Open the image file(.exe) to be executed inside the process.
3) Create the Windows executive process object
4) Create the initial thread
5) Perform post-creation, Windows-subsystem-specific process initialization.
6) Start execution of the initial thread
7) In the context of the new process and thread, complete the initialization of the address space and begin execution of the program.

1) Stage 1: In CreateProcess, the priority class for the new process is specified in the CreationFlags parameter. If no priority class is pecified for the new process, the priority class defaults to Normal. The most important thing to know here is that CreateProcess does not fail jsut because the caller has insufficient privileges to create the process. All windows are associate with the graphical repsentation of a workspace and again if no desktop is specified, the process in associated with the caller current desktop.This stage maninly deals with validating parameters and converting flags and options to their native counterparts.
2) Stage 2: `NtCreateUserProcess` specifies the Windows image that will run the executable file. But just because the a section object has been succesfully created does not mean that it is a valied Windows image. It could be DLL or a POSIX executable. In case if the file is POSIX executable, the iamge to be run becomes Posix.exe and CreateProcess starts from stage 1. Incase if the file is DLL then the creation fails. Once the executable is found it looks inthe registry to see whether a subkey with the file name and extension exists there. If it does PspAllocateProcess looks for that value named Debugger for that key. If the value is there the image run becomes the string in that value and the process creation starts from Stage 1 again.
3) Stage 3: At this point NtCrateUserProcess has opened a valid Windows executable, and created a section object to map it into the new process. Iin this stage PspAllcateProcess is being called to run the image. This internal system function involves seeting up the EPROCESS object, creating the initial process address space, initializing the kernel process strcuture(KPROCESS), setting up the PEB and

concluding the setup of the process address space.

4) Stage 4: At this point Windows executive process object is set up. The process still has no threads so it will not be doing anything. Now we have to create or intert a thread by suing PspAllocateThread and PspInsertThread. PspAllocateThread handles the actualy creation and initialization of the executive thread object while PspInsetThread handles the creation of the thread handle and security attributes. Also KeStartThread call can be used to turn the object into a schedulable thread on the system.
5) Stage 5: Once NtCreateUserProcess returns success, all execitove process and thread objects have been created. At this point Kernel32.dll send a message to the Windows subsystem so that is can set up information. This message includes information on process and thread handles, entries in the creation flags, ID of the process creator, flag indicating whether if this process belongs to a Windows application, UI language information, DLL redirection .local flags and manifest file information. Once this message is recieved b the subsystem, Csrss process structure, thread structure, inserts, count of process are all allocated and inialized.
6) Stage 6: This isthe stage when the process is environment is determined, resources for its threads to use have been allocated, the process has a thread and the subsystem knows about the new process. Execution of the initial thread starts.
7) Stage 7: The new thread begins life running the kernel-mode thread startup routine KiThreadStartup. Then PspUserThreadStartup cehcks whether the debugger notifications have already been send for this process, then DbgkCreateThread then waits for a reply from the debugger. After the debugger is notifies, PspUserThreadStartup looks at the result of the initial check on the thread life. After this PspUserThreadStartup checks whether the systemwide cookie in the SharedUserData structure has been set up yet. If is has not, then it generates one based on hash system information. Finally PspUserThreadStartup sets up the initial context to run the image-loader

Threads: Similarly Threads are formed with many Data Structures as well. The executive thread object encapsualtes an ETHREAD structure, whcih in return contains `KTHREAD` strcuture as its first member. Like Processes, ETHREAD strcuture and the other structures it points exist in the system address soace, while the thread environment block(TEB) exists in the process address space. The Windows subsytem process maintains a paralel structure for each thread created, calling `CSR_THREAD`. The data structures have almost the same attributes as the process, for threads.

A life cycle of a Thread:

1) CreateThread converts the Windows API parameters to native flags and build native strcuture describ-

ing object parameters. `OBJECT_ATTRIBUTES`.

2) CreateThread builds an attribute list with two entries: client ID and TEB address. This allows those values to be recived once the thread has been created.
3) NtCreateThreadEx is called to create user-mode context and capture the attribute list. After this PspCreateThread has been called to create a suspended executive thread object.
4) CreateThread allocates a activation context for the thread used by side-by-side assembly support. It then queries the activation stack to see if it requires activation, and it does so if needed. The stack is is saved in TEB.
5) CreateThread notifies the Windows subsystem about the new thread, and the subsystem does some setup work for the new thread.
6) The thread handle and the thread ID are returned to the caller.
7) Thread is now resumed so that is can be scheduled for execution.

CPU Scheduling: Windows CPU scheduling uses priority-based preemptive scheduling, the highest-priority runs next. The scheduler is Dispatcher. Thread runs until blocks, uses time slice and preempted by higher priority thread. If no run-able thread exists then it runs the idle thread. There is always a queue for each priority. Win32 API identifies several priority classes to which a process can belong to such as: `REALTIME_PRIORITY_CLASS`, `HIGH_PRIORITY_CLASS` or `ABOVE_NORMAL_PRIORITY_CLASS`. For example the high priority class will be the first in queue comapred to the above normal priority class. A thread is also given priority class such as `NORMAL` or `BELOW_NORMAL`. In this case the normal priority thread will be the first in queue. The default priority is NORMAL within the class.

FreeBSD

Processes:

Threads:

CPU Scheduling:

Answer to the questions

Windows

How is it different compared to Linux?:

How is it similar to Linux?:

Why do I think these similarities or differences exist?:

FreeBSD

How is it didderent compared to Linux?:

How is it similar to Linux?:

Why do I think these similarities or differences exist?: