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.

Threads:

CPU Scheduling: My implementation of the concurrency problem was done in C language. I have used Mersenne Twister to generate my random numbers. The problem has some contraints such as, while an item is being added or removed from the buffer, the buffer must be in an inconsistent state, I have acheived this through locking my the buffer for the specific thread that has to use it. Also if a consumer thread arrives while buffer is empty, it has to clock until producer adds a new item, I have achieved this by creating a checkempty funtion, this function check if the buffer is empty, and initalizes it to 1 or in other words true if so. Another constraint was that a producer thread should be able to put more items if the buffer is full, I have achieved this by using the same function but in !checkempty format. I have used getrandinit32 funtion for creating my random numbers.

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?: