Writing 1:
Comparison of Processes, Threads, and CPU Scheduling among Linux, Windows, and FreeBSD

Webster Cheng

April 17th, 2018

CS544

Operating Systems II

(Spring 2018)

Abstract

This article describe how Windows and FreeBSD implements processes, threads, and CPU scheduling, and compare each to Linux. The article contain Linux code to support the article.

1 INTODUCTION

A process, thread, and CPU scheduling are the fundamental of an operating system. Those managements will cause different CPU utilization, responding time and waiting time. Those are the important issues in the operating system field. To maximize their computer resources utilization, Windows, Linux, and FreeBSD have some similar designs and special algorithm for their system to allocate the computer resource. In the article, I will describe how they implement it, include some code to show how they do it, and compare their similarities and differences in scheduling.

2 PROCESS AND THREAD

2.1 Process

Processes are fundamental for an operating system to distribute their computer resource. The process contains a code and data for running. They also contain more computer resource such as "open files and pending signals, internal kernel data, processor state, a memory address space with one or more memory mappings, one or more threads of execution, and a data section containing global variables". [1]

According to Windows document, their process is a container that provides the resource and code to run the program. [2] They have two main processes structures which are EPROCESS and KPROCESS. The KPROCESS can be traced by Kernel and the EPROCESS can be traced by Executive Subsystems. Both of the structures belong to the Process Object that present the user process. The Windows process can be created by another process through CreateProcess function for a general process. This function make Windows also have relationship between two processes will be referred as a parent-child relationship.

FreeBSD process is created by fork family of system call and they use proc structure to store their information such as parent PID, PID, signal state, process tracing information, and timers. The fork function will completely copy the parent process that means duplicating the context of the parent for child process. The process states of FreeBSD are NEW, NORMAL, and ZOMBIE to schedule their process.[1][4]

Process in three of systems have some similar parts. For example, they use different process state to control their process. Moreover, they use process control block (PCB) to trace process status and to assign process id. However, the different part of three system is that Linux basic unit of schedule is process, and Windows and FreeBSD basic unit of schedule is thread. This means Windows and FreeBSD process only like a container that includes computer resources. The Linux process is not only for the resource but also for the scheduler.

2.2 Threads

Threads in Windows and FreeBSD are treated as a specific object from a process. They also treat thread as basic unit of schedule which is different from Linux. According to Windows Document, "a thread is the basic unit to which the operating system allocates processor time." [5] The thread in Windows also contains some information such as register set, private storage area, and stacks.[5] In FreeBSD, they have some similar thread implement as Windows. According to FreeBSD document, "a thread is the unit of execution of a process; it requires an address space and other resources, but it can share many of those resources with other threads".[8] Windows thread creation is through CreateThread function system call. This is different in the Linux. Threads are referred as tasks in Linux system. The creation of a thread is created through clone function which is defined in ¡linux/sched.h¿. This allows the address space sharing between a parent process and child thread.[1]

3 SCHEDULING

Windows, Linux, and FreeBSD have some similar parts of CPU scheduling. First of all, both of them are the preemptive multitasking system which is usually used in the modern operating system. The preemptive multitasking means that the scheduler can decide when the processes stop and resume in order to run new process or process in the waiting state. This can allow the process could not occupy the computer resource for a long time and give the fair allocating the computer resource. To approach the fair allocating, the Linux has the Completely Fair Scheduler (CFS) [1]. In Windows, they don't have the exact name of scheduling. But they have scheduling priority, quantum, thread state, and more to help them fairly scheduling. In FreeBSD, their scheduler is called ULE scheduler.

3.1 Priority policy

The priority policy between Linux and Windows also have similar ideas. All of them separate their priority into the normal and real-time. This design can help the user and developer easily control processes because real-time is usually for kernel process that need to run the process completely. In Linux, they have real-time priority (0 to 99) and nice value (–20 to +19) [1]. The priority level of Windows has 32 priority levels, ranging from 0 through 31 that has sixteen real-time levels (16 - 31), fifteen variable levels (1 - 15), and one system level (0) [6]. In FreeBSD, their process priority ranges from 0 to 127.

The process will depend on those priority to adjust the time that process can run. Also, scheduling will depend on the priority to cease the process for the other process which has higher priority. In addition, Windows and FreeBSD allow priority change by system call or kernel that make them have high flexibility to distribute computer resource that is different as Linux.[8]

3.2 General processes

For general processes scheduling, all systems have the similar concept to manage their processes. Windows uses Dispatcher to help them manage threads [6]. The dispatch will switch the context that has the highest priority. The highest priority thread is stored in the queue which has the order of highest to lowest priority. In Linux, the CFS scheduler finds the highest priority processes (Lower nice value) which is stored in a Red-Black.[1] Also, the process will have smallest virtual runtime which means the time a process can run.

```
static inline struct task_struct * pick_next_task(struct rq *rq)
{
  const struct sched_class *class;
  struct task_struct *p;

  if (likely(rq->nr_running == rq->cfs.nr_running))
  {
    p = fair_sched_class.pick_next_task(rq);
    if (likely(p))
      return p;
  }
  class = sched_class_highest;
  for (;;)
  {
    p = class->pick_next_task(rq);
    if (p)
      return p;

    class = class->next;
  }
}
```

Linux code, the function goes through the process RB tree and selects the highest priority process. [1]

3.3 Real-time processes

For real-time processes scheduling, they have more strict requirements that they need to complete their job before the time is ending. Therefore, Windows and Linux separate their scheduling method for real-time processes. For example, Windows use the real-time priority (16-31) to control processes [6]. Also, they never boost the thread with the real-time priority. This can prevent user break the system down accidentally. In Linux, they schedule real-time processes by following SCHED_FIFO and SCHED_RR policy instead of CFS. SCHED_RR perform a round-robin algorithm with specific timeslice. SCHED_FIFO perform a behavior, first in first out, that process needs to yield other processes to run.

3.4 Execution time

The execution time is also a big issue in the operating system. So, in the modern system, they design their system with dynamic execution time. For example, Linux uses virtual time as their process running which is calculated during the process running period. The virtual time is also modified by the nice value priority.[1] In Windows, they calculate the quantum value by "multiplying the processor speed in Hz with the number of seconds it takes for one clock tick to fire" which is similar to Linux. This dynamic modifying can help the system minimize the waiting time and maximize CPU utilization.[6] In FreeBSD, they also have the policy that fixes thread excution time.[8]

```
static inline void
__update_curr(struct cfs_rq *cfs_rq, struct sched_entity *curr, unsigned long delta_exec)
{
    unsigned long delta_exec_weighted;
    schedstat_set(curr->exec_max, max((u64)delta_exec, curr->exec_max));
    curr->sum_exec_runtime += delta_exec;
    schedstat_add(cfs_rq, exec_clock, delta_exec);
    delta_exec_weighted = calc_delta_fair(delta_exec, curr);
    curr->vruntime += delta_exec_weighted;
    update_min_vruntime(cfs_rq);
}
```

Linux code, __update_curr() weights the time by the number of runnable processes.The virtual time is modified by the weighted value. [1]

4 Conclusion

Three operating systems have their own features to maximize computer resource using. In their documents, it can find many different terms but with the same concept to approach the same ideas. Also, they continually add more feature to their system that does not exist at the first time. Therefore, the article talk more similar part of three operating systems. In order to compare correctly, the more article needs to be read, the more functions need to be tested, and the more factors need to be considered.

References

[1] L. Robert,*Linux Kernel Development*. Pearson Education, Inc. 2010.

[2] microsoft.com, *About Processes and Threads*. [online] https://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx/ [Accessed 16 Apr. 2018].

[3] microsoft.com, *Creating Processes*. [online] https://msdn.microsoft.com/en-us/library/windows/desktop/ms682512(v=vs.85).aspx [Accessed 16 Apr. 2018].

[4] M. K. McCusick,& V. George,*Design and Implementation of the FreeBSD Operating System*. Addison-Wesley, Inc. 2015.

[5] M. E. Russinovich,& D. A. Solomon,*Processes, Threads, and Jobs in the Windows Operating System*. 2009 [online] https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=7 [Accessed 16 Apr. 2018].

[6] M. E. Russinovich,& D. A. Solomon,& A. Ionescu,*Windows Internals, Part 2, 6th Edition*.Pearson Education, Inc. 2012.

[7] A. Silberschatz,*Operating System Concepts Ninth Edition*. John Wiley & Sons, Inc. 2012.

[8] M. K. McKusick,& G. V. Neville-Neil,& R. N. M. Watson,*Process Management in the FreeBSD Operating System*. 2014 [online] http://www.informit.com/articles/article.aspx?p=2249436 [Accessed 18 Apr. 2018].

[9] M. K. McKusick,& G. V. Neville-Neil, & R. N. M. Watson,*4.2 Process State*. 2014 [online] http://www.informit.com/articles/article.aspx?p=2249436&seqNum=2 [Accessed 18 Apr. 2018].