

Lab 4

Monitoring Linux and Windows Processes and Threads Scheduling

ITSC205: Operating Systems Internals

NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Table of Contents

[Lab Outcome(s) 3](#_Toc472470625)

[Reading 3](#_Toc472470626)

[Introduction 2](#_Toc472470627)

[1.0 Process and Threads Scheduling 3](#_Toc472470628)

[2.0 Process Explorer to analyze processes/Threads behavior 7](#_Toc472470629)

[3.0 Process Monitor to trace processes 9](#_Toc472470630)

4.0 Linux Process Scheduling……………………………………………………………………..12

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Analyze threads scheduling using Windows performance Monitor | 10 |  |
| Analyze processes/threads using process explorer (Procexplorer) | 10 |  |
| Analysis of process behavior using Process Monitor tool (Procmon) | 10 |  |
| Modify time-sharing process priority - process scheduling | 5 |  |
| Analyze round robin (SCHED\_RR) scheduling program | 15 |  |
| TOTAL MARK | 50 |  |

Lab Outcome(s)

* Examine process and threads activity using various windows monitoring tools.
* Analyze process scheduling policy and priorities

Reading

* Textbook sections 20.4.2 (Processes and Threads), 20.5 Scheduling (Tread and Real Time Scheduling), 21.3.4.3 (Windows Threads) and 21.3.4.4(Thread Scheduling)

Introduction

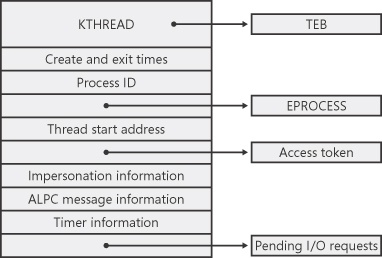
The key to current operating systems is the ability to execute multiple processes simultaneously. It is essential that the operating system optimizes the execution order of the processes to ensure efficient and equitable use of the CPU resources.

The objective of multiprogramming is to have some processes running on the CPU at all times. Processes are scheduled by a CPU scheduler such that CPU usage is maximized. CPU scheduling uses measurements of the I/O - CPU activity burst cycle.

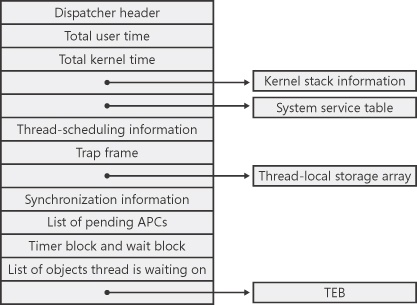
The primary objective of CPU scheduling is to have some process running at all times – i.e., maximize CPU utilization.

1. Process/ Threads Scheduling

Windows processes is merely a container, threads do the work and consume resources. Every process has at least one thread and the thread is the entity within a process that Windows schedules for execution. Threads are kept in executive thread blocks (ETHREAD) and the thread environment blocks (TEB) in user space.



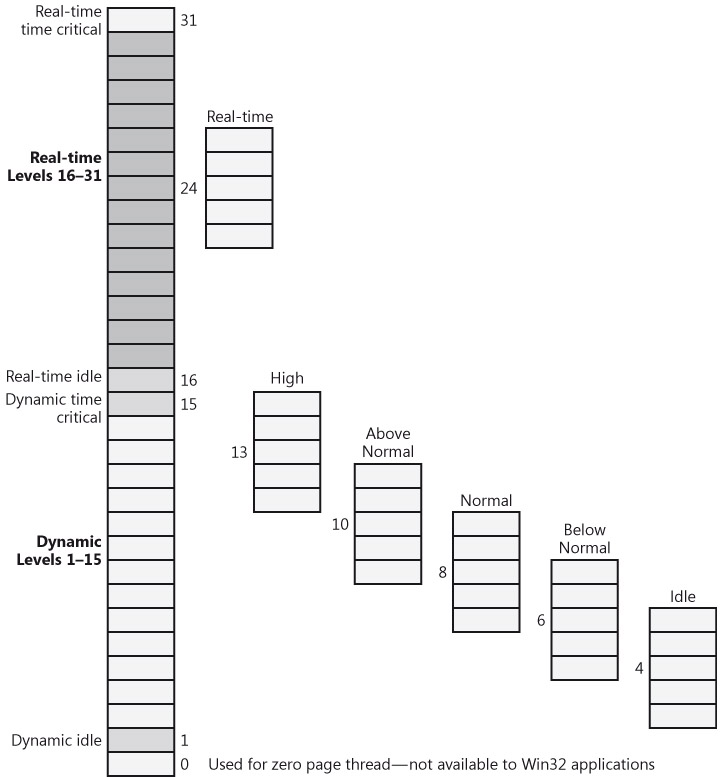
Source: <https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=4>



Source: <https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=4>

Windows uses a variation of pre-emptive priority scheduling algorithm where priority is calculated based on the process priority class and the thread priority level.

The scheduler has a “Priority Boost” feature where a thread’s priority is increased after waiting for I/O – the amount depends of what type of I/O. (e.g. keyboard would receive a greater boost than disk.) A thread’s priority is decreased if it used up its quantum. A background process that is brought to the foreground will have 3x the quantum to execute.



Source: Windows Internals 5th edition, Mark E. Russinovich and David A. Soloman, Microsoft Press, 2009, pg. 394.

**Windows Performance Monitor to Analyze Thread scheduling\_\_\_10**

1. To learn how Windows schedules threads and the different priority values assigned to threads, read the following Microsoft (msdn) reference: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms685100%28v=vs.85%29.aspx>
2. Read handouts provided in class.

Windows Performance Monitor tool can be useful when debugging a multithreaded application and to verify the state of the threads running in the process. To analyze thread-scheduling state changes by using Performance Monitor tool, follow these steps:

1. Start Notepad (Notepad.exe) process.
2. Start the Windows Performance Monitor
3. Select chart view
4. Right-click on the graph, and choose Properties.
5. Click the Graph tab, and change the chart vertical scale maximum to 7. (As you'll see from show description for the performance counter, thread states are numbered from 0 through 7.) Click OK.
6. Click the Add button on the toolbar to bring up the Add Counters dialog box.
7. Select the **Thread** performance object, and then select the Thread State counter. **Check** the **Show Description** box to see the definition of the thread state values. Read the thread state definition and the different state values.
8. In the Instances box, select <All instances> and click Search. Scroll down until you see the **Notepad** process (notepad/0); select it, and click the Add button.
9. Scroll back up in the Instances box to the **mmc** process (the Microsoft Management Console process running the System Monitor), select all the threads (mmc/0, mmc/1, and so on), and add them to the chart by clicking the Add button.
10. Now close the Add Counters dialog box by clicking OK
11. Based on chart result ( you can also click on report and verify respective chart values)
    1. What is the state of notepad process? Why?
    2. What are the different threads states?
    3. Are there threads in transition state?Why? (read handout provided in class to differentiate a thread in transition state)
12. Create a process and use performance monitor tool to analyze its (instances) threads behavior. Demonstrate and explain to the instructor:
    1. Thread context switch
    2. Thread states
    3. Thread priorities ( Base and current)
    4. Thread wait reasons

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**Process Explorer to analyze Processes/Threads behavior \_\_\_/10**

1. Microsoft Reference <https://technet.microsoft.com/en-us/sysinternals/processexplorer>

2. From System Internals run Process Explorer and identify processes running in your system

3. Under **Options** click on **Verify Image Signature** to display which processes have signature. Explore that column and identify processes without signature ( This information can be used for security purposes)

4. Under **view** click on:

1. **system information** and select CPU tab to verify total handles, threads, processes and context switches in the system. Analyze what is going on in each CPU. Red color is kernel usage and green is combination of user and kernel usage
2. **Lower pane view** to display the DLL (Dynamic Link Libraries) associated with each process.
3. **Select columns.** Explore columns that can be added to analyze process behavior. Under process performance tab select threads, CPU cycles, Base priority, Handle count and context switch

5. Click twice on explorer.exe process to display process properties. What is the parent of explorer.exe?

6. Identify and analyze DLLs associated with explorer.exe process. Are there DLLs running without signature? This information may help to identify suspicious DLLs running in the system or for troubleshooting purposes.

5. Start a Notepad process.

6. Run cmd as administrator and use process explorer to verify threads, priority, handles, context switch of this process. What is the parent of cmd?

6. Use the following command to start another Notepad process in real-time (must run as administrator) – At cmd type:**start /realtime notepad**. (Notepad should open)\

7. Use process explorer tool to analyze Notepad processes. Double-click on Notepad.exe to show the process properties window, and then click on the Threads tab and answer the following questions.

* + 1. How many treads are running in notepad process?
    2. What is the thread state?
    3. What is the Base priority and the dynamic priority of this thread? What is the difference between base and dynamic priority? (Refer to handout provided in class)
    4. Right click on cmd process and kill the process. Was notepad process (child of cmd) terminated? Explain.

8. Task Manager can show you similar information. Press Ctrl+Shift+Esc to start Task Manager, and go to the Processes tab. Right-click on the Notepad.exe process, and select the Set Priority option. What are the priority of the 2 notepad processes?

9. There is a special process used by Windows called the System Idle Process, examine its properties and you will notice its CPU usage is quite high. Research and explain the purpose of this process to the instructor.

10. Start a web browser process (e.g. Firefox). Use process explorer to analyze all the resources used by this process. Demo and **Explain** to the instructor the following:

a. process image path

b. process handles

c. process CPU and memory usage

d. process priority

e. Process environment,

f. DLLs associated with this process and Process and DLLs signatures

g. Threads running in this process, threads state, threads base and dynamic

priority, threads context switch, threads stack

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

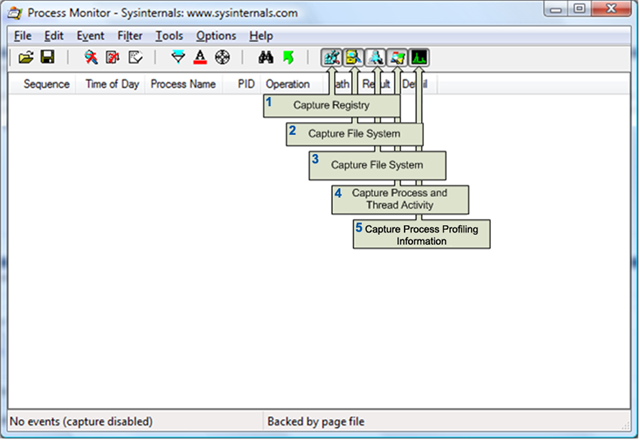
3.0 Process Monitor (Procmon) to Trace Processes

Process Explorer is a great tool for observing processes in the system but it’s a moving snapshot of the system. Sysinternals has another utility that captures detailed information – Process Monitor. Procmon is an advanced logging tool that can tell you what low-level operations the process is performing and is essential to diagnostics.

Procmon captures information about the registry, files, processes and network activity, these can add up to millions of operations in just a few seconds. This tool has powerful features that can be used for troubleshooting and as a malware hunting toolkit.

The most important aspect to using procmon is to know how to filter the capture to drill down to the suspect component for analysis.

1. Start Process Monitor (procmon.exe) from the Sysinternals folder. The main window will immediately begin filling up with event capture data. Each row in the table represents one low-level event that had occurred in the system.

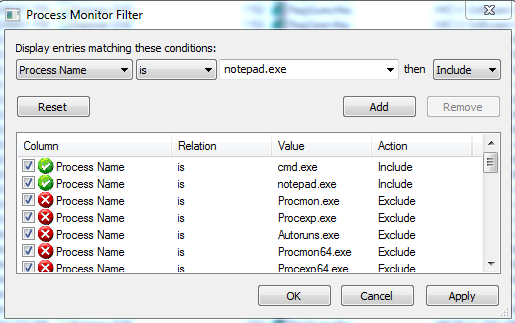


1. Stop/disabled event capture by selecting File on the menu and unchecking Capture Events or click the magnifying glass icon on the toolbar.
2. Clear the event list using the Edit menu or the button on the toolbar (hover over the buttons to view the function and the shortcut key).

In this lab we will explore processes and threads. **Turn on only the registry process and threads activity buttons**. Turn off the other buttons.

We're going to be looking at a very simple executable—Notepad.exe—and we will be launching it from a Command Prompt window (cmd.exe). It's important that we look both at the operations inside cmd.exe and those inside Notepad.exe. Recall that a lot of the user-mode work is performed by *CreateProcess*, which is called by the parent process before the kernel has created a new process object.

1. Click on Filter button and include only **cmd** and notepad process as follows: Under **Display entries matching these options**: Select **Process Name is cmd.exe**  and click on Add button. Select **Process Name is notepad.exe** and click on Add button



a. Start cmd and type **Notepad.exe** and press Enter. You will see all the instances of cmd and the registry keys that were modified when the process cmd was created.

1. More columns can be added by clicking on Options 🡪 Select Columns. You can add TID, PPID, completion time.
2. Stop the capture and analyze the following results:
   * cmd process and thread creation time.
   * What is the path where the image was loaded?
   * What was the first registry key read by cmd and what was the result?
   * What libraries (DLL) were (loaded) used by the process?
   * What operations were SUCCESS and which were not?
   * What is the PID, PPID and TID of cmd?
   * At what point was notepad process and thread created?
   * Are there more threads created by notepad?
   * What is the PID, PPID and TID of notepad?
   * What other libraries (DLL) are required by notepad process?
3. Every process and thread has its own **stack**. You can analyze **process/threads stack** and verify what part of the process creation flow was performed in **user mode** or **kernel mode**. Click twice on the **first** cmd event ( Process Start) to display event properties
   1. Analyze the modules running in the **stack** in user and kernel mode. What mode (user or kernel) is cmd process and DLL running on?
   2. Explore and analyze the notepad process (Load image event) properties

6. Demonstrate and explain the results to the instructor.

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

4.0 Linux Process Scheduling

A program can request a scheduler policy and/or a priority through system calls. Three scheduling policies are specified by POSIX, and available with Linux.

1. **SCHED\_OTHER** The normal time-sharing policy that is the default for all user space processes.

a.Niceness values: -20 (highest priority) 0 (default) 19 (lowest priority)

2. **SCHED\_FIFO** Real time First-In First-Out scheduling

a.Priority values: 0 -> 99

3. **SCHED\_RR** Real time Round-Robin scheduling

a.Priority values: 0 -> 99

The normal time-sharing policy, **SCHED\_OTHER** will give preference to higher priority processes but will not guarantee that the process will run without being preempted. Process priority values are specified as **niceness** level. Time-sharing processes can be nice. Only processes running with root privilege (usually System processes) can request a niceness value lower than zero

* 1. Use Linux **man to learn about nice command. F**ind out the range of nice values. What is the value for the highest and lowest priorities?
  2. Use the command **nice** to create a process with high priority and command **renice** to lower the priority of an existing process. Demo the results
  3. Start the browser and use **pidof** command to find the PID of the process
  4. Use Linux man to learn about **chrt** command and demo how to:
     1. Display current scheduling policy and priority of a calling process

(e.g. browser)

* + 1. Set the process(browser) priority to SCHED\_FIFO and SCHED\_RR and use ps or top to verify results

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**Scheduling System calls**: (Refer to Linux man pages for details.)

getpriority Get a processes current time-sharing priority

setpriority Set a processes current time-sharing priority

sched\_getscheduler Returns the scheduling policy of a specified process

sched\_getparam Returns the scheduling priority of a specified process

sched\_get\_priority\_max Returns the maximum priority allowed for a scheduling policy

sched\_get\_priority\_min Returns the minimum priority allowed for a scheduling policy

sched\_rr\_get\_interval Returns the current quantum for the round-robin scheduling policy

sched\_setscheduler Sets the scheduling policy and priority of a specified process

sched\_setparam Sets the scheduling priority of a specified process

sched\_yield Yields execution to another process

5. Compile, run and analyze the results for the following program.

/\* Change to the SCHED\_RR policy and the highest priority, then \*/

/\* lowest priority, then back to the original policy and priority. \*/

#include <unistd.h>

#include <sched.h>

#include <stdio.h>

main ()

{

struct sched\_param param; //copy sched\_param structure to param

struct timespec rr\_interval; //copy timespec structure to rr\_interval

int my\_pid = 0;

int old\_policy, new\_policy, old\_priority;

int low\_priority, high\_priority;

/\* Determine the round-robin quantum \*/

sched\_rr\_get\_interval(my\_pid, &rr\_interval);

printf("Round-robin quantum is %lu seconds, %ld nanoseconds\n",rr\_interval.tv\_nsec);

/\* Get parameters to use later. Do this now \*/

/\* Avoid overhead during time-critical phases.\*/

high\_priority = sched\_get\_priority\_max(SCHED\_RR);

printf("High Priority %d\n",high\_priority);

low\_priority = sched\_get\_priority\_min(SCHED\_RR);

printf("Low Priority %d\n",low\_priority);

/\* Save the old policy for when it is restored. \*/

old\_policy = sched\_getscheduler(my\_pid);

printf("Old Policy is %d\n",old\_policy);

/\* Get all fields of the param structure. This is where \*/

/\* fields other than priority get filled in. \*/

sched\_getparam(my\_pid, &param);

printf("Scheduler parameters %d\n",param.sched\_priority);

/\* Keep track of the old priority. \*/

old\_priority = param.sched\_priority;

/\* Change to SCHED\_RR, highest priority. The param \*/

/\* fields other than priority get used here.\*/

param.sched\_priority = high\_priority;

sched\_setscheduler(my\_pid, SCHED\_RR, &param);

printf("Scheduler parameters %d\n",param.sched\_priority);

/\* Print the value associated with the RR policy \*/

new\_policy = sched\_getscheduler(my\_pid);

printf("New Policy is %d\n",new\_policy);

/\* Change to SCHED\_RR, lowest priority. The param \*/

/\* fields other than priority get used here, too. \*/

param.sched\_priority = low\_priority;

sched\_setparam(my\_pid, &param);

/\* Restore original policy, parameters. Again, other \*/

/\* param fields are used here. \*/

param.sched\_priority = old\_priority;

sched\_setscheduler(my\_pid, old\_policy, &param);

return(0);

}

6. What is the purpose of this program?

7. Identify scheduling system calls?

8. Modify the program to:

a. create (clone a process fork()) before the scheduler is changed to the maximum priority.

b. print child process PID and make my\_pid= getpid()

d. After the scheduler is set to maximum priority make the new process

(child process) to execute in a loop as follows

/\* CPU intensive code, new process will run this \*/

for (try\_cnt = 0; try\_cnt < 100; try\_cnt++)

/\* Perform some CPU-intensive operations \*/

{

for(loop\_cnt = 0; loop\_cnt < 1000000; loop\_cnt++)

{

tmp\_nbr+=loop\_cnt;

tmp\_nbr-=loop\_cnt;

}

}

printf("Completed test %d\n",try\_cnt);

9. After the job or test is completed the program should print the new policy and new scheduler parameters.

10. Compile, run and demo the results to instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |