

Lab 6

Monitoring Processes and Threads Scheduling

ITSC205: Operating Systems Internals

NAME: **\_Coleton Sanheim\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Table of Contents

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

[Reading 3](#_Toc472470626)

[Introduction 2](#_Toc472470627)

[1.0 Process and Threads Scheduling with Performance Monitor tool 5](#_Toc472470628)

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

[3.0 Linux Process Scheduling –CFS 9](#_Toc472470630)

4.0 POSIX Threads…………..……………………………………………………………………..14

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies.

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Analyze threads scheduling using Windows performance Monitor | 15 |  |
| Analyze processes/threads using process explorer (Procexplorer) | 10 |  |
| Analyze round robin (SCHED\_RR) scheduling program | 15 |  |
| POSIX Threads | 10 |  |
| 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)

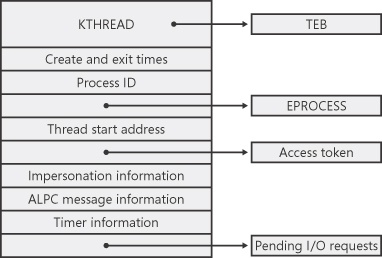
Introduction

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.

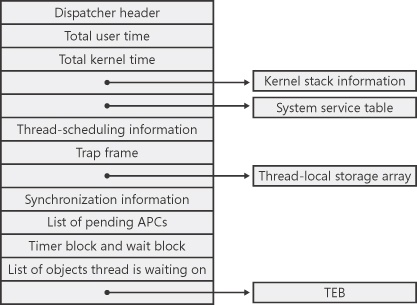
The objective of multiprogramming is to optimize CPU utilization. Processes are scheduled by a CPU scheduler such that CPU usage is maximized. The operating system scheduler in the kernel. All the scheduling routines in the kernel are called in windows. The scheduler or dispatcher determines what thread can execute next based on scheduling algorithms. The most common algorithms are **priority and Round Robin (RR).** Windows implements preemptive priority algorithm. Linux implements RR and FIFO policies for real time processes and SCHED\_OTHER policy for Time Sharing processes called also Completely Fair Scheduler (CFS) based on red-black tree data structure.

Process/ Threads Scheduling Concepts

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.



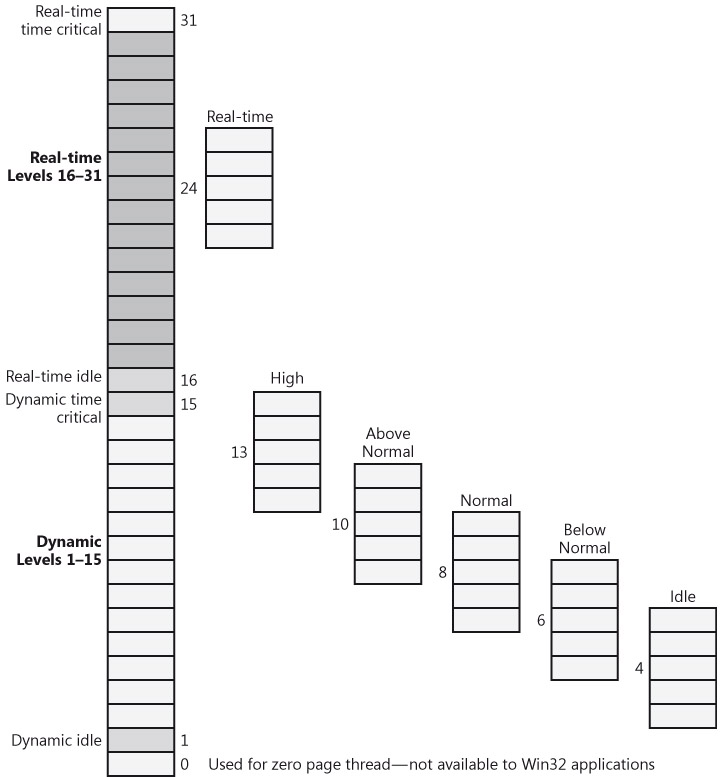
Source: <https://www.microsoftpressstore.com/articles/article.aspx?p=2233328&seqNum=4>



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

Windows implements 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 7th edition, Mark E. Russinovich and David A. Soloman, Microsoft Press, 2017, pg. 215.

**1.0 Performance Monitor to Analyze Thread scheduling \_\_\_15**

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>

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. Right-click on the window displaying the red line graph, and choose Properties.
4. 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.
5. Click the Add button on the toolbar to bring up the Add Counters dialog box.
6. 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.
7. 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.
8. 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. Click OK
9. To observe threads behaviour select the icon that change graph type and select Histogram bar. Based on chart result ( you can also click on report and verify respective chart values)
   1. What is the state of notepad process?

**5.000 (Waiting)**

* 1. What are the different threads states?

**0.000 (Init)**

**1.000 (Ready)**

**2.000 (Running)**

**3.000 (Standby)**

**4.000 (Terminate)**

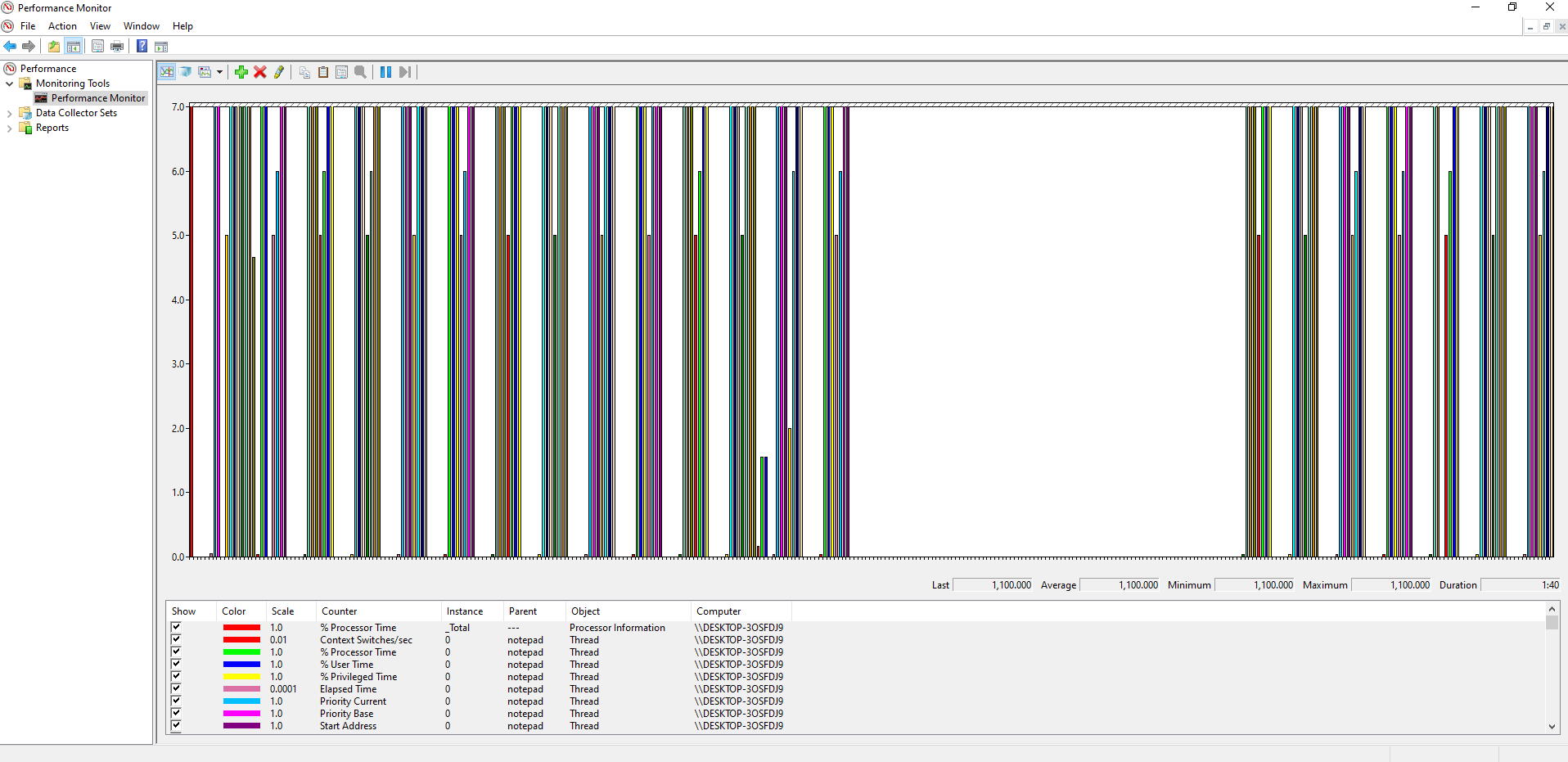
**5.000 (Waiting)**

**6.000 (Transition)**

**7.000 (Deferred ready)**

**8.000 (Gate waiting)**

1. Attach a screen capture with the graph of the threads and its different states



1. Start the browser and use task manager to identify the default priority. What is the priority of this process?

**Normal**

1. To analyze base and dynamic (current) priorities assigned to threads in windows start performance monitor and add a new **Thread counter**. Select **Priority Base** and **Priority current** for few instances of the browser.
2. Click on “change chart type” icon and select report. Analyze the results and answer the following questions:
   1. What is the difference between base and current priority?

**Base is what you assign to a task when you create it, sometimes it boosts priority to avoid starvation so current is typically higher than the base**

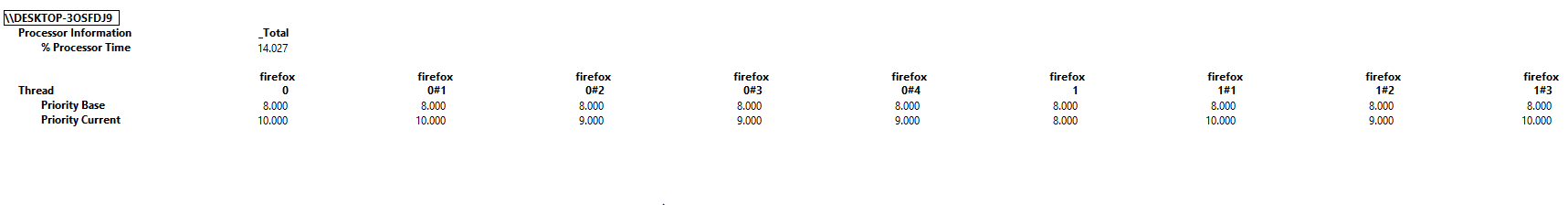
* 1. What number represent Normal priority?

**6-10, but normally 8**

* 1. Why do threads have two priorities?

**One to represent the base and one to represent the current**

1. Attach a screen capture with the report of the priorities assigned to the threads. Do not close the report you still need it.



1. Now use task manager to change for all browser instances the priority from Normal to High and go back and check what changed in the performance monitor report.
   1. What is the value of high priority?

**11 – 15, but normally 13**

* 1. Did the current (dynamic) priority change? why?

**Yes, because it is based off of the base priority which we changed**

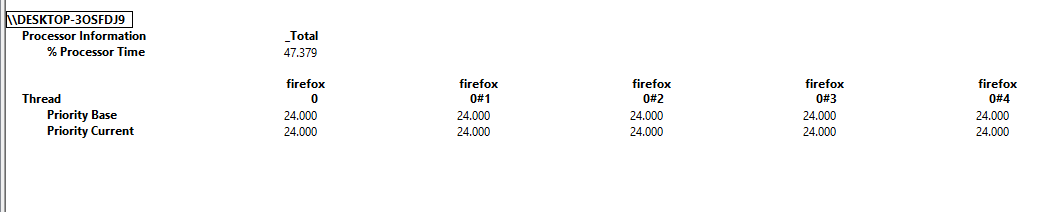
1. Now use task manager to change the browser priority to real time, observe the changes on performance monitor report and answer the following questions:
   1. What is the default value of real time priority?

**24**

* 1. Why is the base and current priority the same?

**Because windows doesn’t change the priority of a real-time process**

1. Attach a screen capture with the report that displays the changed priority



**2.0 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?

**<Non-existent Process>(1112)**

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.

**There are no .dll running without signature**

7. Start a Notepad process.

8. Use process explorer to verify the based priority of notepad process. What is the default priority of notepad?

**Base priority is 8**

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

**explorer.exe(5776)**

9. 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)\

10. Use process explorer tool to analyze notepad process. Double-click on Notepad.exe to show the process properties window, and then click on the Threads tab and answer the following questions. ( **use the first thread to answer the questions**)

1. How many treads are running in notepad process?

**6**

1. What is the thread state?

**Wait:WrUserRequest**

1. What is the Base priority and the dynamic priority of this thread?

**24 and 24**

1. Right click on cmd process and kill the process. Was notepad process (child of cmd) terminated? Explain.

**Notepad was not terminated. Notepad was not dependant on the cmd process to run**

11. There is a special process used by Windows called the **System Idle Process** that contains one thread per CPU. Use Process explorer tool to examine the properties of System Idle process and answer the following questions:

a. What is the purpose of System Idle Process?

**Keeps the CPU busy doing something while it waits for the next process to use a thread**

b. What is the priority of this process?

**0, or n/a**

c. How many threads are used by this process and what is the purpose of these

threads?

**2 threads, to hold threads in an idle state so there is always a thread available should something need it**

3.0 Linux Process Scheduling (CFS) \_\_\_\_/15

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: 1 -> 99

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

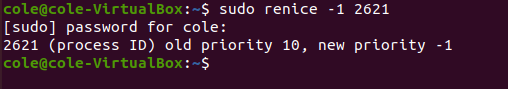
a.Priority values: 1 -> 99

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 highest and lowest priorities?

**-20 to 19**

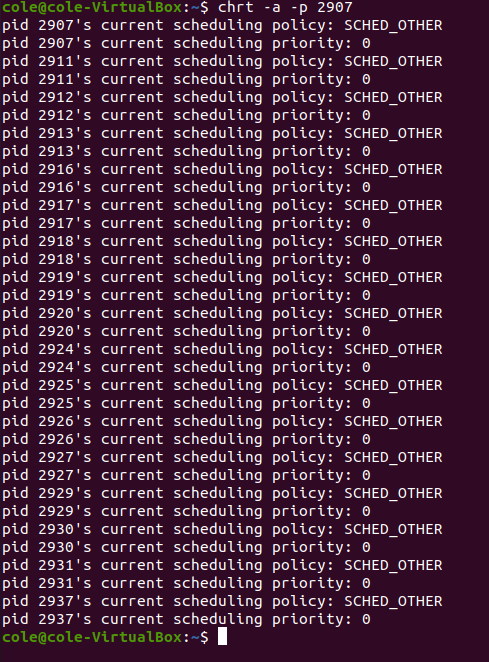
1. Use the command **nice** to create a process with **high priority** and **renice** it to lower the priority. Attach the screen that demo the results



1. Start the browser and use **pidof** command to find the PID of the process



1. Use Linux man to learn about **chrt** command and demo how to:
   1. Display current scheduling policy and priority of a calling process



* 1. (e.g. browser)
  2. Set the process(browser) priority to SCHED\_FIFO and SCHED\_RR and use ps or top to verify results
  3. Attach screen capture that demo resultsText

     Description automatically generated

**The ps/top command doesn’t show the priority so im not sure how you mean it will verify the results**

**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);

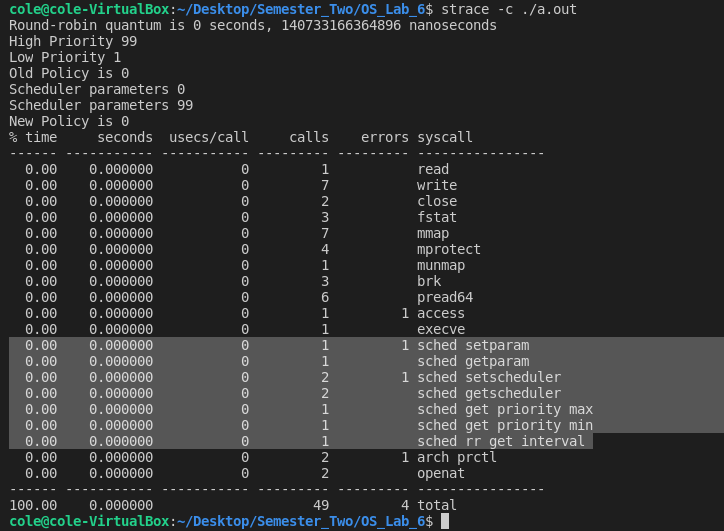
return(0);

}

6. What is the purpose of this program?

**Changes a process to the SHED\_RR policy and the highest priority then changes it to the lowest priority. Then finally it changes it back to the original policy and priority.**

7. Identify scheduling system calls?



8. Modify the program to:

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

b. print child process PID using 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;

}

}

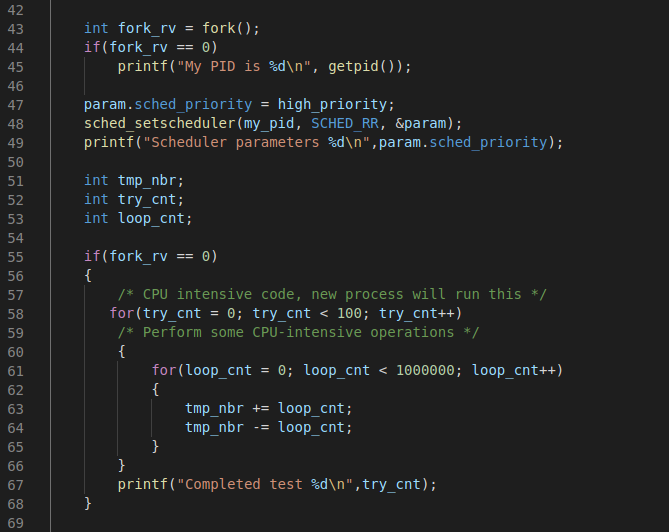
printf("Completed test %d\n",try\_cnt);

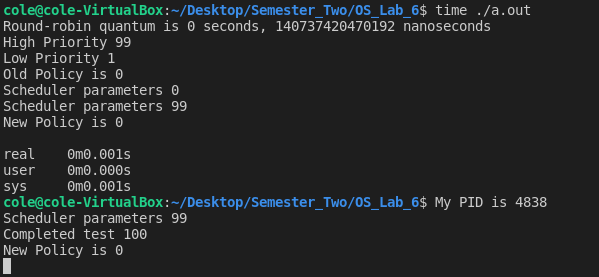
9. Compile and run the program using **time ./a.out** command to verify execution time

10 Use strace to trace system calls used by this program and compare it with the scheduling system calls provided above

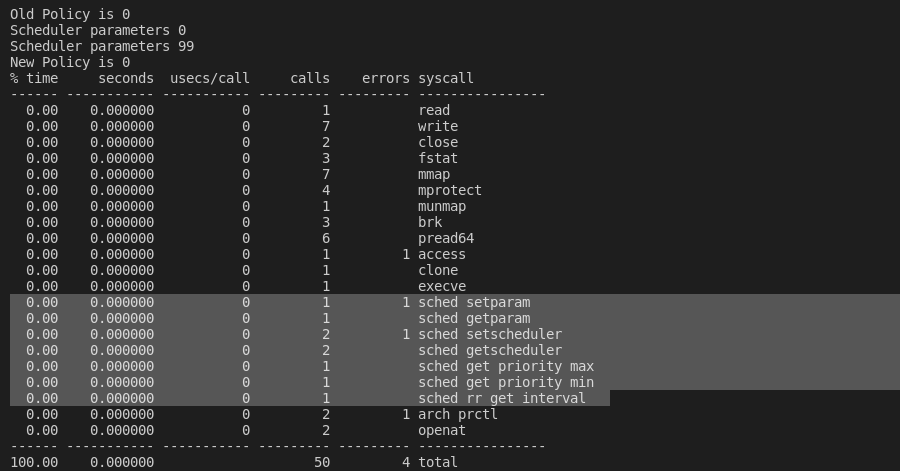
11. Attach the screen capture that demo the following:

a. Modified code and results after compiling and running the program





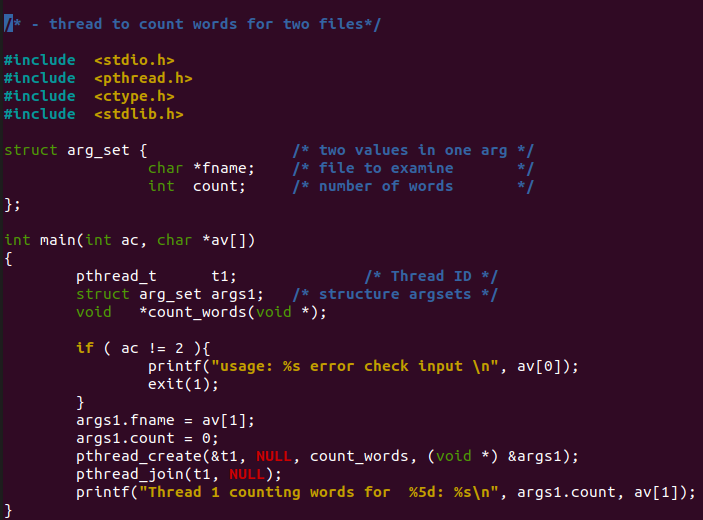
b. System calls after using strace

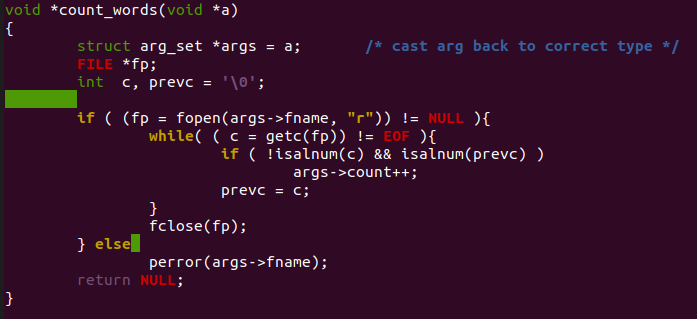


1. POSIX Threads \_\_\_\_/10

Modern operating systems are multithreaded. A thread is an instance (function) of a process. These functions (threads) run in parallel sharing resources.

1. Read Linux manual and class notes/slides to learn how to implement pthread\_create( ), pthread\_join( ) system calls. These system calls are in chapter (3). **man 3 pthread\_create .** Read Linux man pthreads and identify what is shared and what is unique by threads.
2. Compile and Run the following code. Remember when compiling POSIX threads is required to link pthread library **-lpthread**





1. What is the job of the thread in this program?

**To count the words in a given file**

1. In the printf ( ) function what is the purpose of **av[1]** ?

**The name of the file**

1. In this program add a new thread identified as t2 that executes same function **count\_words** to count the words for a second file
2. Attach modified code and results

