LAB 2 REPORT

Sahil Bhanderi(skb5400)
Osman Ali(oza5)

3.1.1 implemented RR , vLOT, mLOT in my_scheduler.c.

- RR: This scheduler functions by running threads in a cyclical fashion. This means that after each thread finishes running, it is sent to the end of the queue and the next thread in line is started. So in this way, the threads are completed in order one after another, restarting at the front of the queue each time the last thread finishes.

- vLOT: vLOT was implemented with help from OSTEP, we begin our implementation by getting the total weight of all threads in all nodes, once we have this cumulative value we calculate the weight of each thread in a node and use a random() function to generate a winning ticket. Now we keep on iterating the list and adding the weight as soon as the weight of a thread equals the ticket we assign it as the one to be scheduled.

- mLOT: This scheduler picks a random number between 0 and the list size and ignores weight during the selection process. Once a thread is selected, a global variable is used to keep track of the amount of times it runs which then looks at the weight of the process in order to determine the quanta. The process returns the same node until it matches the weighted quanta and then sends it to the back of the queue for selection of the next thread while also resetting the tracking variable.

3.1.2 tested using given TestFiles.

3.1.3 We created **TestFile6.c**, which can be used to run and test counter() and sleeping(). Moreover the file my_functions.c also contains the code for counter/sleeping but its use and testing should be referred to in **TestFile6.c**

3.2.1

A. Context switching time for **kernel threads:**

Host	OS	2p/0K ctxsw	2p/16K ctxsw	2p/64K ctxsw	8p/16K ctxsw	8p/64K ctxsw	16p/16K ctxsw	16p/64K ctxsw
cse-p204i	Linux 2.6.32-	7.2700	8.7900	11.6	20.7	25.5	20.9	20.5
cse-p204i	Linux 2.6.32-	7.5100	9.9600	12.4	18.8	22.8	20.4	25.9
cse-p204i	Linux 2.6.32-	7.0500	8.5400	12.6	19.9	24.8	20.5	24.7

cse-p204i Linux 2.6.32- 7.1600 6.7600 8.2000 20.2 21.4 20.2 22.9
cse-p204i Linux 2.6.32- 6.5700 5.3300 11.4 19.4 25.6 20.2 25.6

Context switching times for kernel threads

Times (microseconds)
7.2700
7.5100
7.0500
7.1600
6.5700

Sample Mean:
(7.27+7.51+7.05+7.16+6.57) / 5 = **7.112 microseconds**

Variance:
7.27 - 7.112 = 0.158^2 = 0.024964
7.51 - 7.112 = 0.398^2 = 0.158404
7.05 - 7.112 = -0.062^2 = 0.003844
7.16 - 7.112 = 0.048^2 = 0.002304
6.57 - 7.112 = -0.542^2 = 0.293764

Sum of squared numbers: 0.48328
Variance = 0.48328 / (5-1) = 0.12082 microseconds

B. Context switching time for **user-level threads:**

Terminal output upon run (in nanoseconds):

The number of context switches so far is: 1
The time taken by the context switch is: 2445
The number of context switches so far is: 2
The time taken by the context switch is: 2374
The number of context switches so far is: 3
The time taken by the context switch is: 1816
The number of context switches so far is: 4
The time taken by the context switch is: 2165
The number of context switches so far is: 5
The time taken by the context switch is: 1677

The number of context switches so far is: 6
The time taken by the context switch is: 2095
The number of context switches so far is: 7
The time taken by the context switch is: 2235
The number of context switches so far is: 8
The time taken by the context switch is: 2165
The number of context switches so far is: 9
The time taken by the context switch is: 1746
The number of context switches so far is: 10
The time taken by the context switch is: 2026
total rf = 8865240
The number of context switches so far is: 11
The time taken by the context switch is: 1815
total rf = 9300716
thread 0 info:
thread state = FINISHED
num_runs = 6
total_exec_time = 5010
total_sleep_time = 0
total_wait_time = 5000
avg_exec_time = 835
avg_wait_time = 714
num_continuous_runs = 6
avg_continuous_exec_time = 835

thread 1 info:
thread state = FINISHED
num_runs = 6
total_exec_time = 5240
total_sleep_time = 0
total_wait_time = 5010
avg_exec_time = 833
avg_wait_time = 715
num_continuous_runs = 6
avg_continuous_exec_time = 833

1. Total number of context switches.
= **11**

2. Time taken for each context switch in microseconds. Report the empirical average and variance across all context switches during your experiment.

Context switching times for user-level threads

Times (microseconds)
2.445
2.374
1.816
2.165
1.677
2.095
2.235
2.165
1.746
2.026
1.815

Sample Mean:

(2.445+2.374+1.816+2.165+1.677+2.095+2.235+2.165+1.746+2.026+1.815)/11 = **2.050 microseconds**

Variance:

2.445 - 2.050 = 0.395^2 = 0.156
2.374 - 2.050 = 0.324^2 = 0.105
1.816 - 2.050 = -0.234^2 = 0.055
2.165 - 2.050 = 0.115^2 = 0.013
1.677 - 2.050 = -0.383^2 = 0.147
2.095 - 2.050 = 0.045^2 = 0.002
2.235 - 2.050 = 0.185^2 = 0.034
2.165 - 2.050 =0.115^2 = 0.013
1.746 - 2.050 = -0.304^2 = 0.092
2.026 - 2.050 = -0.024^2 = 0.001
1.815 - 2.050 = -0.235^2 = 0.055

Sum of squared numbers = 0.673

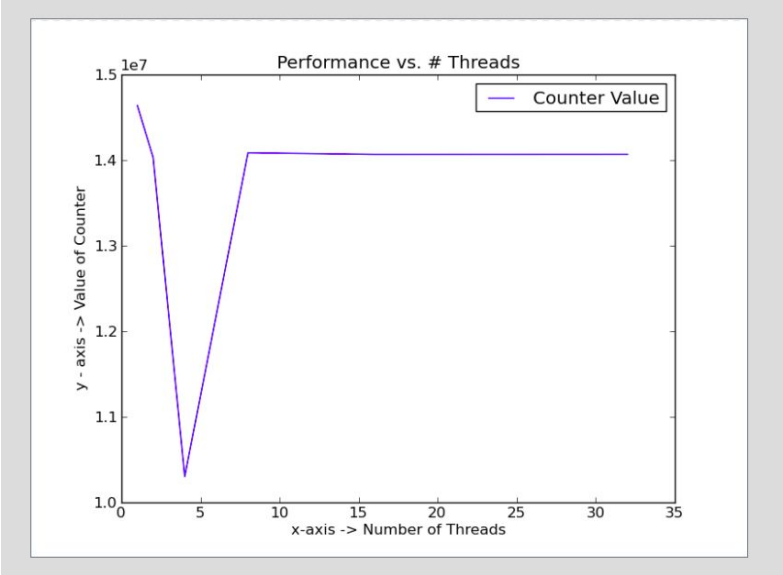
Variance = 0.673 / (11-1) = 0.067 microseconds

3.2.2

Effects of a blocking call for the Sleeping function

Kernel-Level	**User-Level**
5458920	1761220
964764	1761701
5542990	1760275
936646	3971409
Total work done: 12903320	**Total work done:** 9254605

Kernel level sleeping graph



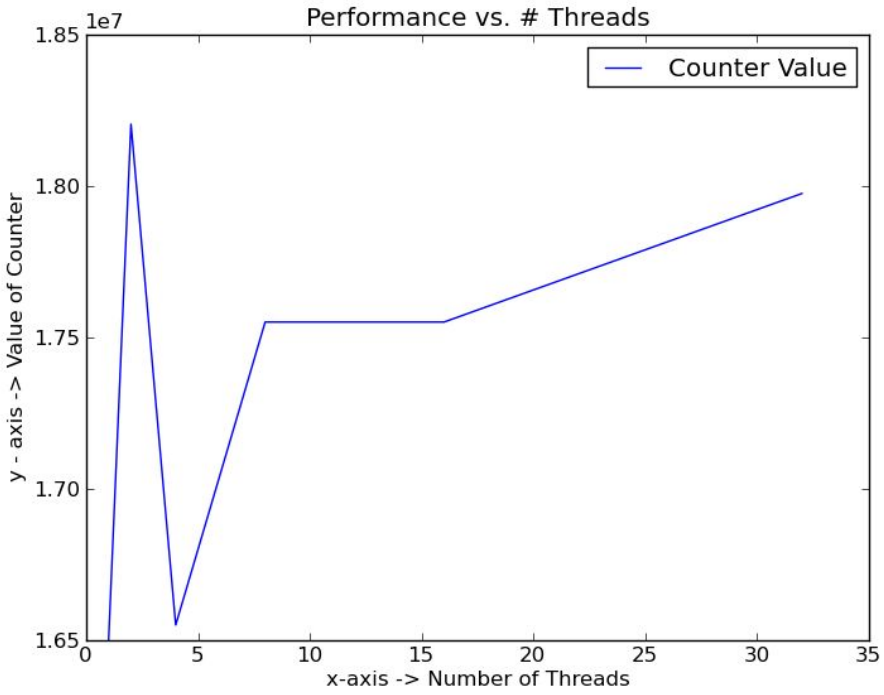
The kernel values switch from sleeping for two seconds to 7 seconds which explains the large differences in each consecutive value. The total work done is less for user-level than kernel which is the expected output.

3.2.3

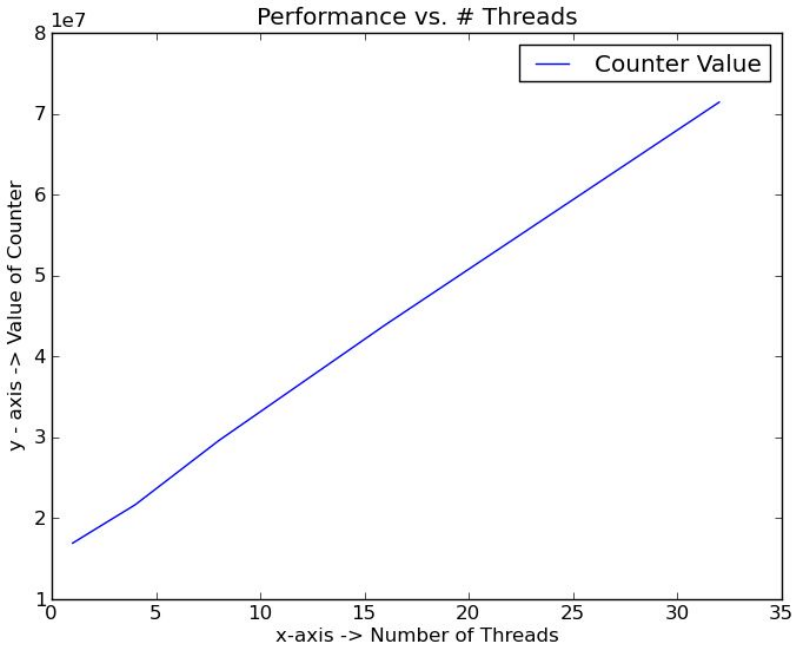
Effects of a blocking call for the Counter function

Kernel Counter	User Counter
16500631	16927353
18204545	18513991
16550604	21668895
17551020	29588435
16553000	43924940
17975500	71430133

Work Done Kernel-Level (Counter)



Work Done User-Level (Counter)



We notice in the User Level work done that as the number of threads increases the work done also increases at a constant rate which shows that the more the processors the more of threads can be efficiently handled.