**Header: Grade: FOR /20 DES /40 EXP /30 ORI /10 / TOT _100**
*Lab # 4*

*CS 5390 Summer 2017, Date of Submission: 08/04/17*
*Students Names: Adeel Malik (amalik@utep.edu)*
*Instructor: Yadira Jacquez*

## Section 1: Effort: 6 hours

- Planning and preparation: 1 hour
- Experiment: 4 hours (on simulator)
- Report writing: 1 hour1

## Section 2: Objectives

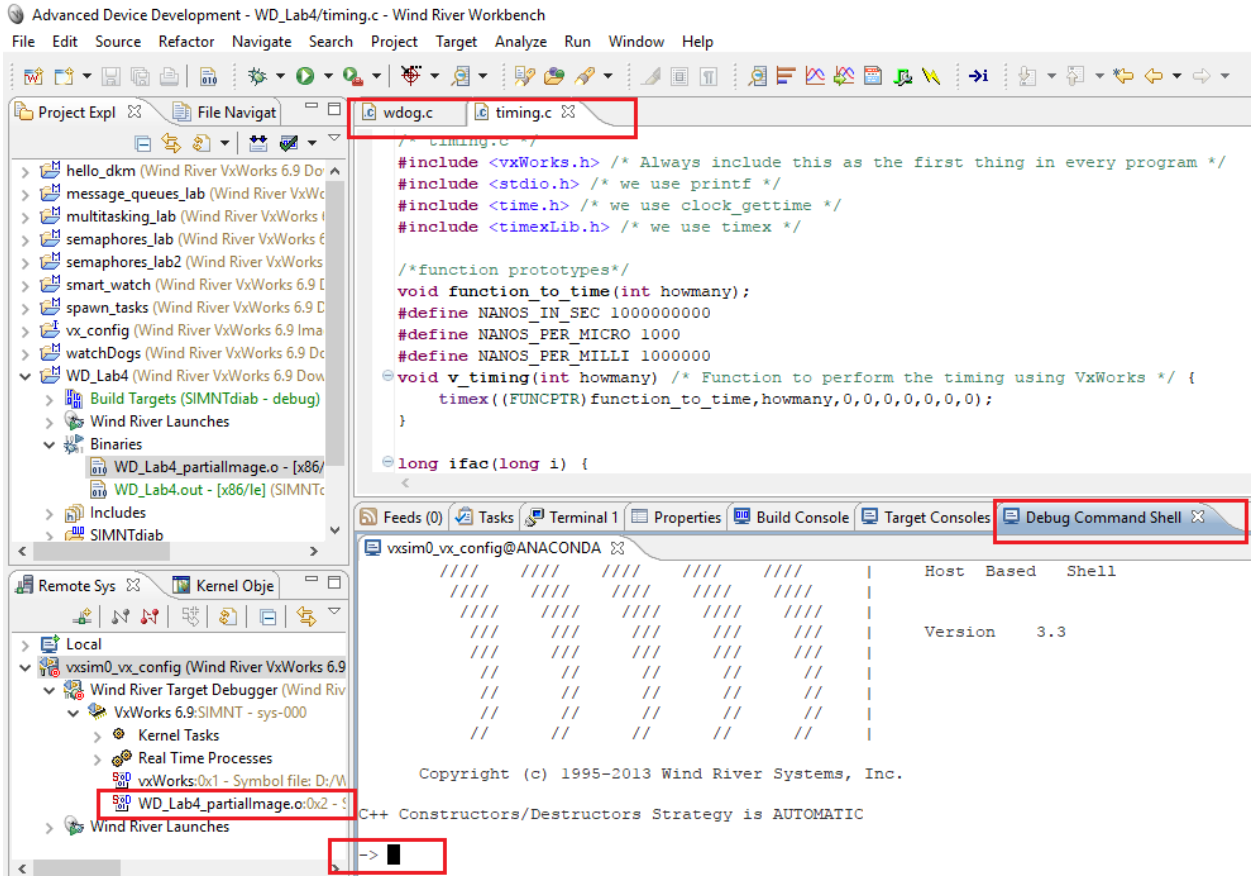
The objective of this experiment is to learn the concept of watchdog timer and use POSIX 1003.1b and VxWorks OS routines to delay task execution.

## Section 3: Procedures and Results

### Part A:

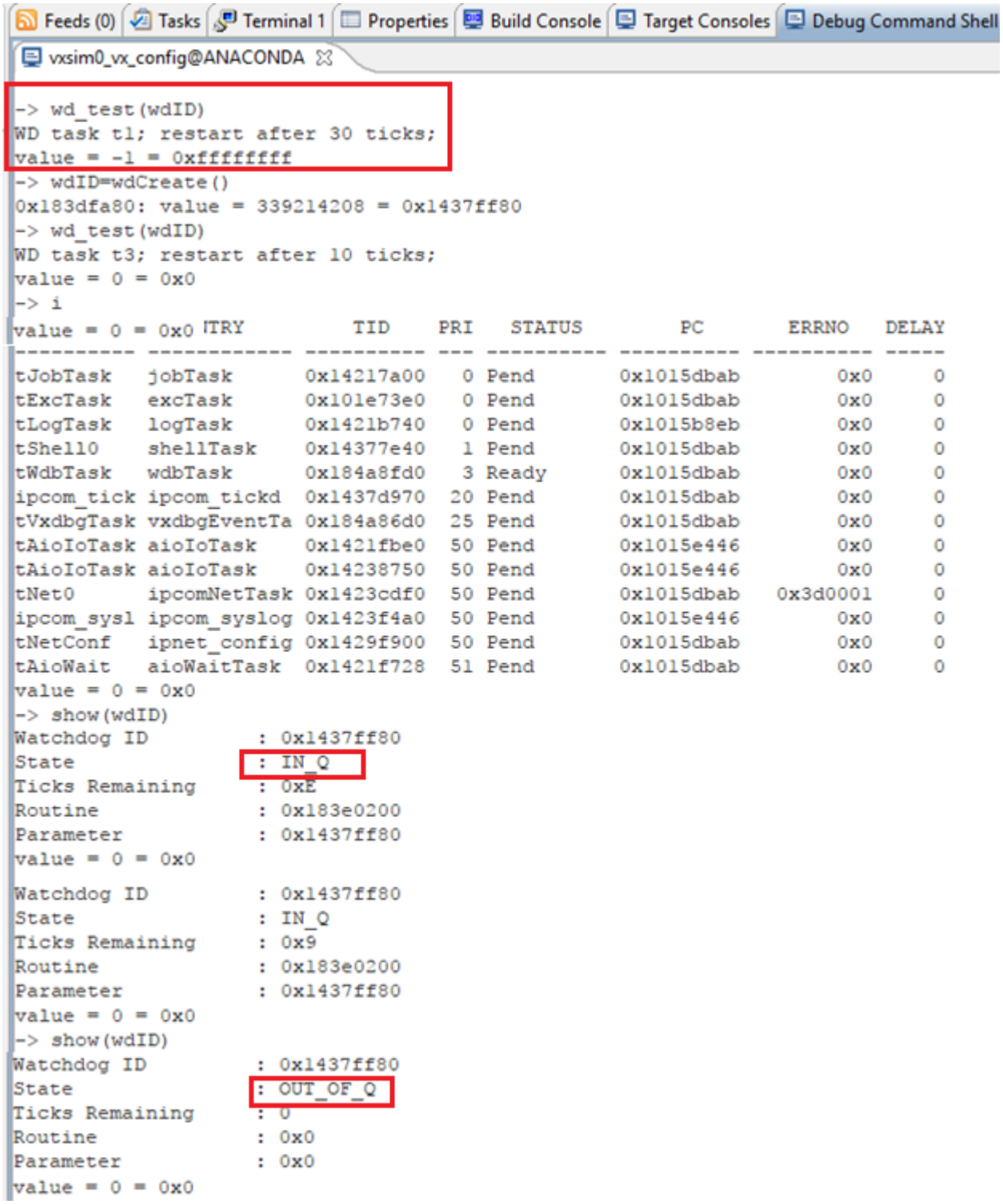
**A1.** Add the example source code to the project, compile, and download to the target. Start the shell.

**Answer:** Attached the provided source code, compiled and downloaded the target, running shell.



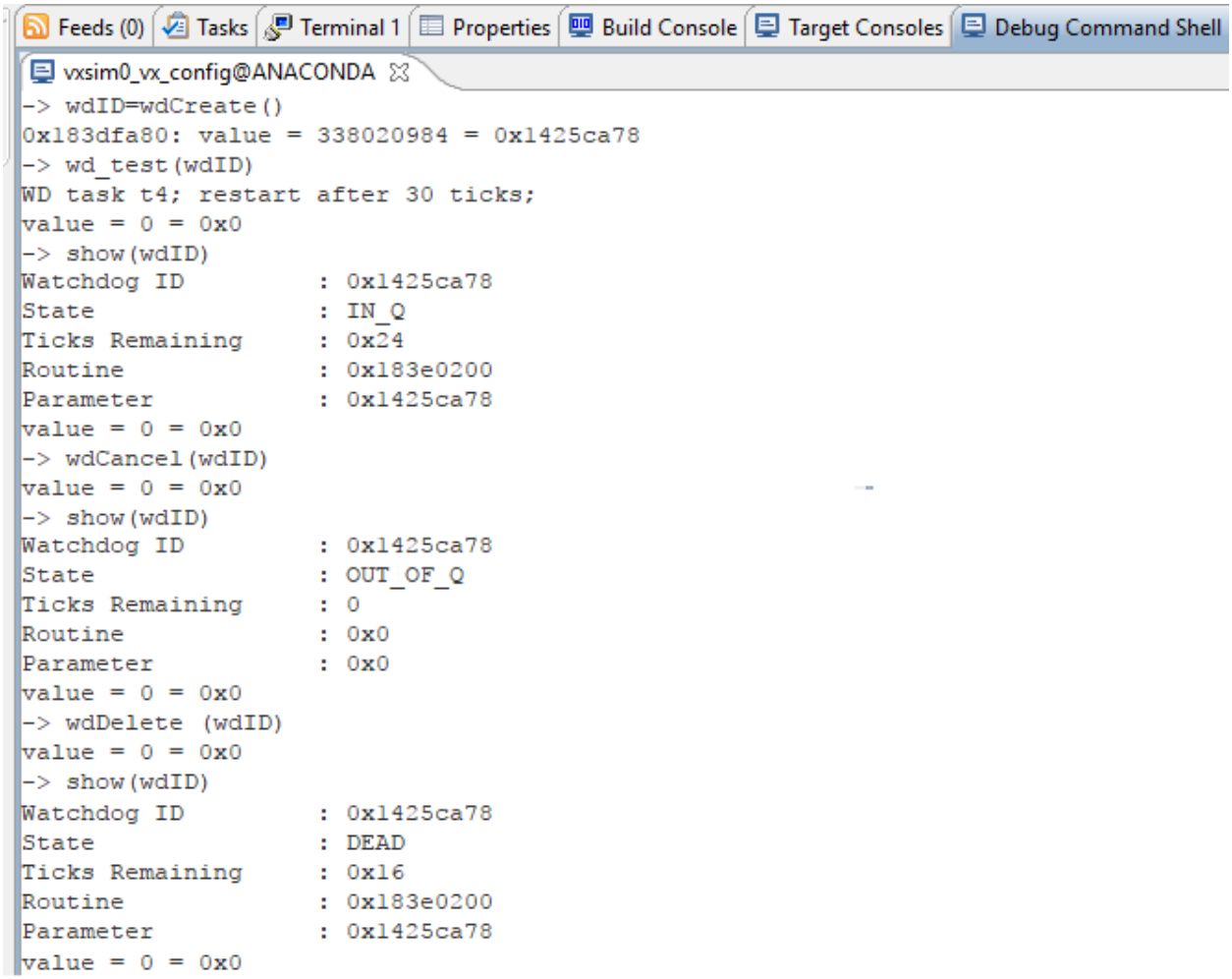

**A2.** Execute function wd_test(wdID) from command line. Observe the output and the currently running tasks. (i). Where is the output displayed? What tasks are running? Explain. Create required watchdog timer wdID = wdCreate() and re-execute. Check the status of watchdog using “show wdID”. Show the results and comment on them.

**Answer:** Running the command wd_test(wdID) returns an error because the watch dog wdID is not yet created. Output is displayed in the shell. After creating watch dog with the command wdID=wdCreate() and re-executing the command wd_test(wdID) keeps the state of watch dog “IN_Q” as long as there are remaining ticks, once the ticks are over, the state changes to “OUT_OF_Q” and ticks are random every time. Results are displayed in the shell.



**A3.** Experiment with wdCancel, wdStart, and wdDelete from the shell line. Explain and show your results.

**Answer:** wdStart starts the watchdog timer. The image below shows that a watchdog timer is created to execute after 30 ticks. wdCancel cancels the started watchdog timer and changes the state to OUT_OF_Q. wdDelete de-allocates a watchdog timer and cancels any previous start.




**A4.** What VxWorks system function you must use to display clock resolution? Use it, show result and explain.

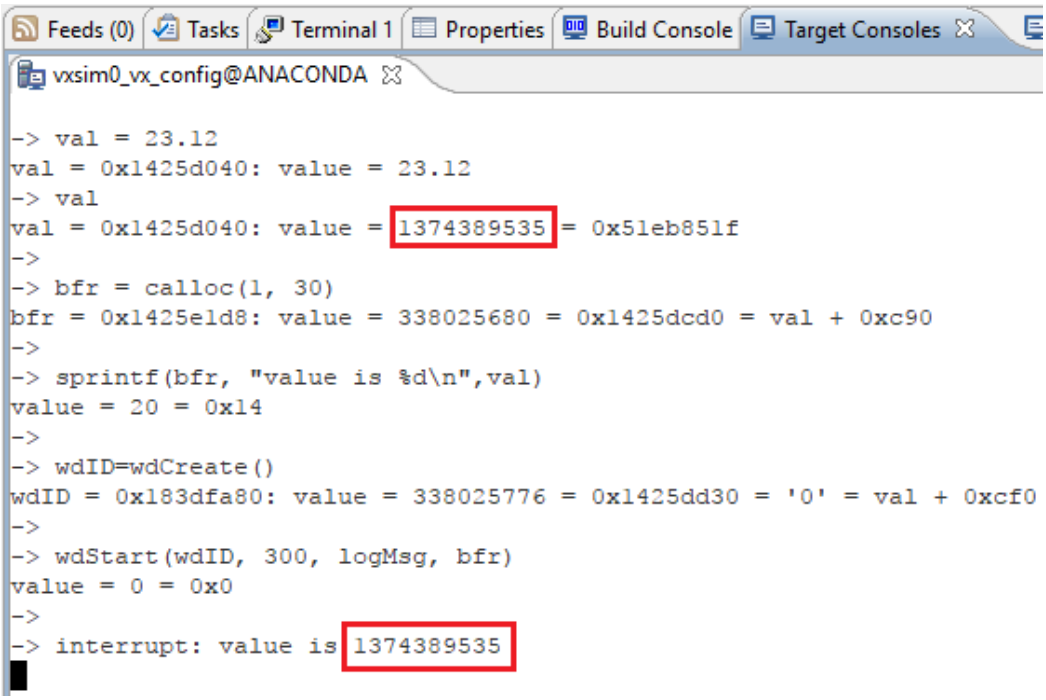
**Answer:** Clock resolution can be displayed by using the command sysClkRateGet(), default clock rate is 60. This can be changed by the command sysClkRateSet(). However, this should only be done at system startup. Along with these fucntions, we can also get and set the tick values to get the actual time taken by different tasks.

```
-> sysClkRateGet()
value = 70 = 0x46 = 'F' = _vx_offset_TRIGGER_status
-> sysClkRateSet(70)
value = 0 = 0x0
-> sysClkRateGet()
value = 70 = 0x46 = 'F' = _vx_offset_TRIGGER_status
-> tickGet()
value = 30182 = 0x75e6
-> tickSet(100)
value = 0 = 0x0
-> tickGet()
value = 301 = 0x12d = _vx_offset_PARTITION_curWordsAllocatedInternal + 0x19
```

**A5.** Describe the experiments showing the VxWorks shell command lines required to:
(a) add val = 23.12 to the symbol table,
(b) create buffer of 30 characters and fill it with string "value is XX.XX" (where XX.XX is the numerical value of variable val),
(c) start the watchdog to print the string with 5 seconds delay. Show your commands and results.

**Answer:** This can be achieved by the following set of commands…

```
val = 23.12                          //setting the value
bfr = calloc(1, 30)                  //creating a buffer of 30
sprintf(bfr, "value is %d\n",val)    //copying the exact value of val into the buffer
wdID=wdCreate()                      //creating a watch dog
wdStart(wdID, 300, logMsg, bfr)      //starting the watch dog to print the buffer after 5 seconds
```




**Part B:**

**B1.** Add program timing.c to the project. Explain the functionality of functions:
ifac(), function_to_time(), p_timing(), and v_timing().

**Answer:** **Function ifac** runs with the long parameter "i" and it is triggered from the function function_to_time. Based on the value of i, the function returns a value. If "i=0 or i=1" it returns value 1, else computes the value based on the formula "i*ifac(i-1)" and returns the value.

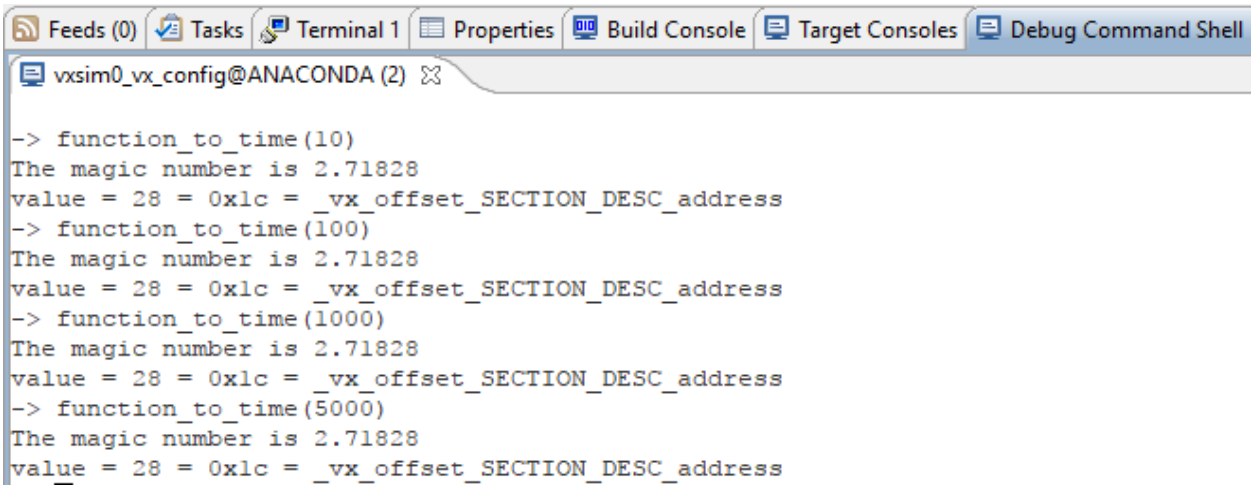
**Function function_to_time** has the integer parameter "howmany ", there are 3 integer variables i, j and k for 3 loops. First loop does nothing except wasting time, executing till the value is less than "howmany". 2nd loop calculates the value of sum, runs for 499 times. 3rd loop is inside the 2nd loop and runs 9 times calling ifac function with different values of j(1 to 9). And at the end, the function prints the value of sum.

**Function p_timing()** calculates the time taken by the function function_to_time and prints the execution time in Microseconds and Milliseconds.

**Function v_timing()** calculates the execution time using VxWorks timing utility "timex"

**B2.** Execute function_to_time() in WindSh with different arguments. For example, type function_to_time(10) to specify a value of 10 for the integer argument howmany. What shell command you must use? What are the results?

**Answer:** Executed the function with the arguments (10, 100, 1000, 5000), the results are shown on the host shell.
function_to_time(10)
function_to_time(100)
function_to_time(1000)
function_to_time(5000)




**B3:** Execute v_timing() and p_timing() functions with an argument of 200 five times. DO NOT INCLUDE RAW OUTPUT IN THE REPORT. Are the results the same? Where do you get the information on the timing? What are the units? What is the average?

**Answer:** The results are not same, execution time returned by both functions is almost the same. The results are shown on the target console. Units are in microseconds and the average is 28571.

Executing the p_timing function with argument 200, returns smooth results using the POSIX clock. v_timing function on the other hand, which uses VxWorks timing utility timex, is sometimes not able to calculate the time returning the message "…execution time too short to be measured meaningfully…"

**B4:** Repeat experiment B2 for different arguments (use e.g. 100 and 300) recording the execution time. Use both p_timing() and v_timing(). Build a table repeating each experiment ten times and computing the average and standard deviation for each of the timing method.

**Answer:** Running the experiment with argument 100 for both p_timing and v_timing. Function p_timing runs fine as usual whereas v_timing is not able to run with argument 100 returning message "execution time too short to be measured meaningfully in a single execution". The execution is so fast relative to the clock rate that the time is meaningless in case of VxWorks timex utility.

| argument (100) | Time Taken(p_timing) | Time Taken(v_timing) |
|---|---|---|
| Experiment 1 | 14285.714 | NA |
| Experiment 2 | 14285.714 | NA |
| Experiment 3 | 14285.714 | NA |
| Experiment 4 | 14285.714 | NA |
| Experiment 5 | 14285.714 | NA |

| | | |
|---|---|---|
| Experiment 6 | 28571.428 | NA |
| Experiment 7 | 14285.714 | NA |
| Experiment 8 | 14285.714 | NA |
| Experiment 9 | 14285.714 | NA |
| Experiment 10 | 14285.714 | NA |

| | | |
|---|---|---|
| **Average** | 15714.2854 | #DIV/0! |
| **SD** | 4517.539424 | #DIV/0! |

When argument 300 was used, the results are as below…

| **argument (300)** | **Time Taken(p_timing)** | **Time Taken(v_timing)** |
|---|---|---|
| Experiment 1 | 42857.142 | 28571 |
| Experiment 2 | 28571.428 | 28571 |
| Experiment 3 | 42857.142 | 28571 |
| Experiment 4 | 42857.142 | 42857 |
| Experiment 5 | 28571.428 | 42857 |
| Experiment 6 | 42857.142 | 42857 |
| Experiment 7 | 42857.142 | 42857 |
| Experiment 8 | 28571.428 | 42857 |
| Experiment 9 | 28571.428 | 42857 |
| Experiment 10 | 42857.142 | 28571 |

| | | |
|---|---|---|
| **Average** | 37142.8564 | 37142.6 |
| **SD** | 7377.110988 | 7377.258678 |

Average time taken and standard deviation for both functions is almost the same in this case.

**B5:** What are the clock resolutions observed using the POSIX and VxWorks routines? Confirm your answers based on the above experiments.

**Answer:** I have observed that POSIX clock returns the lowest possible execution time and VxWorks timing utility timex has a limitation to calculate time.

**B6:** Repeat experiment B2 with smaller arguments (try 20, 10, 5, 2) for both p_timing() and v_timing(). What happens? What do you need to modify in the v_timing() function to get the timing in the case when the message says: "...execution time too short to be measured meaningfully..."?

**Answer:** p_timing with argument 20 sometimes return the time, lower argument values are returning time 0 most of the times. v_timing returns the message saying "..the execution time is short.."

To overcome the issue with v_timing, we need to use timexN() routine instead of timex().

**Section 4: Observations, Comments, and Lessons Learned**

I have learned about watchdog timers, POSIX and VxWorks timex utility. I also have learned based on the experiments that VxWorks timex has some limitations whereas POSIX clock works fine for many cases.