Report

Assignment-2

Vivek Kumar(B18092)

Bhumanyu Goyal(B18012)

Harish Jaglan(B18115)

**Q1. Custom Linux Kernel**

*Steps that we followed:*

1. Download Ubuntu 20.04 iso file.
2. Install Virtualbox.
3. Create a virtual machine with Ubuntu diastro using the iso file in step 1. I provided ~50GB disk space, 2048 MB of RAM, three logical cores out of the four of my i3 processor.

Now, Moving to the virtual machine:

1. Downloaded the linux kernel 5.10.1.
2. Ran a few commands to prepare the system for the compilation process:

* apt-get update
* Apt-get install ncurses-devel bison flex elfutils-libelf-devel openssl-devel

(read about some of them: bison begin a parser, flex for lexical analysis, libelf because kernel didn’t compile without it in my case)

1. Used **make menuconfig** for creating the .config file.

I went with the default configuration but used INSTALL\_MOD\_STRIP=1 to get rid of the debugging symbols and slightly reduce the initrd image.

*Command for compilation:*

**sudo make -j 3 && sudo make INSTALL\_MOD\_STRIP=1 modules\_install -j 3 && sudo make install -j 3**

1. Somehow managed to compile the kernel. Rebooted with 5.10.1 this time.

Created a kernel module that writes something to the kernel log. It had a C file and a makefile. Tried to introduce some errors in the file like:

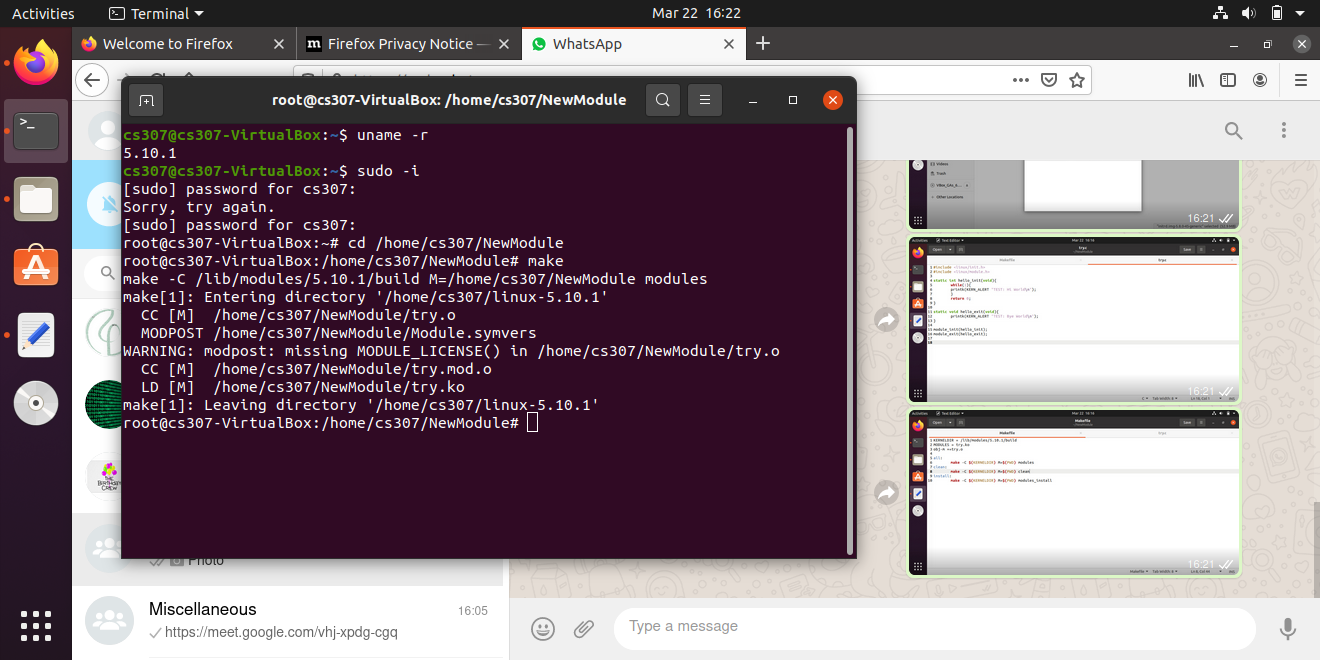
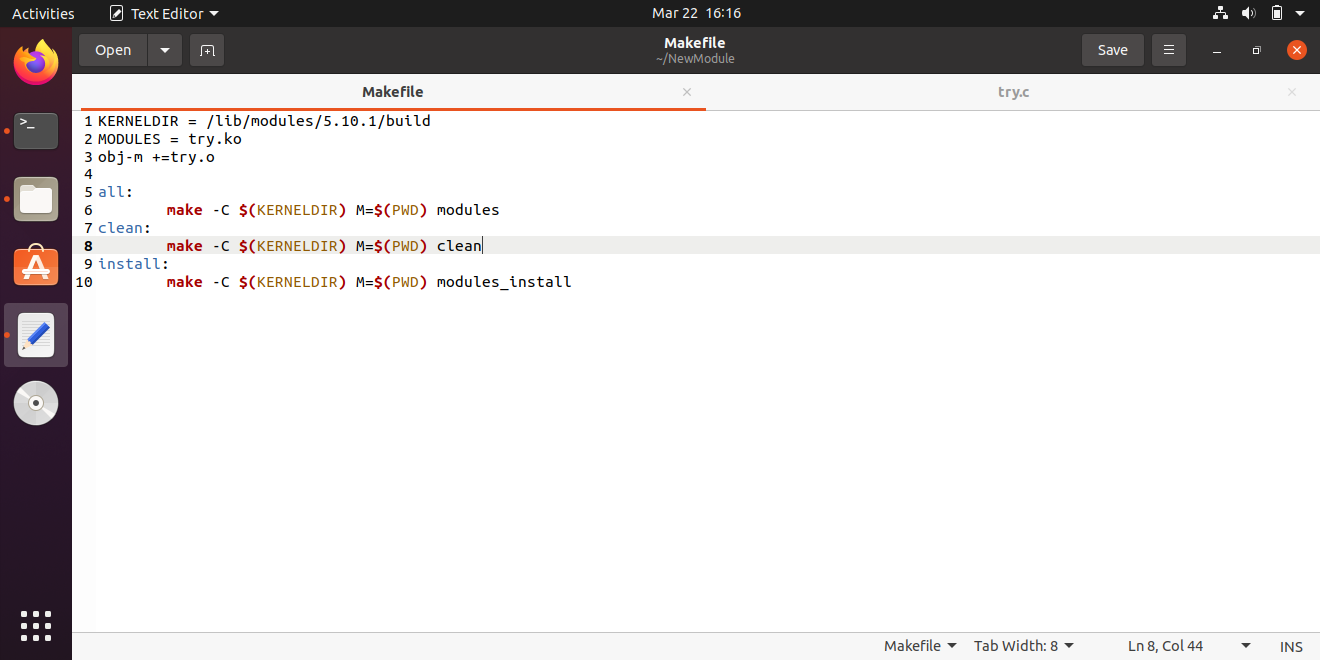
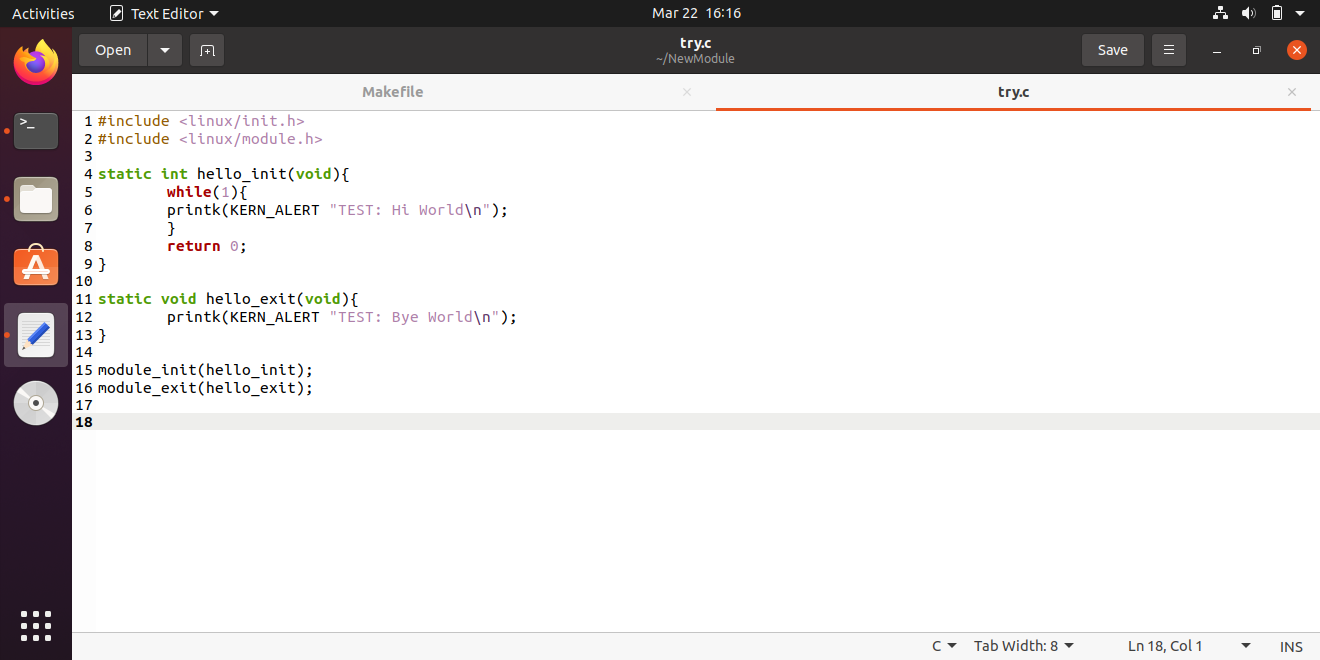
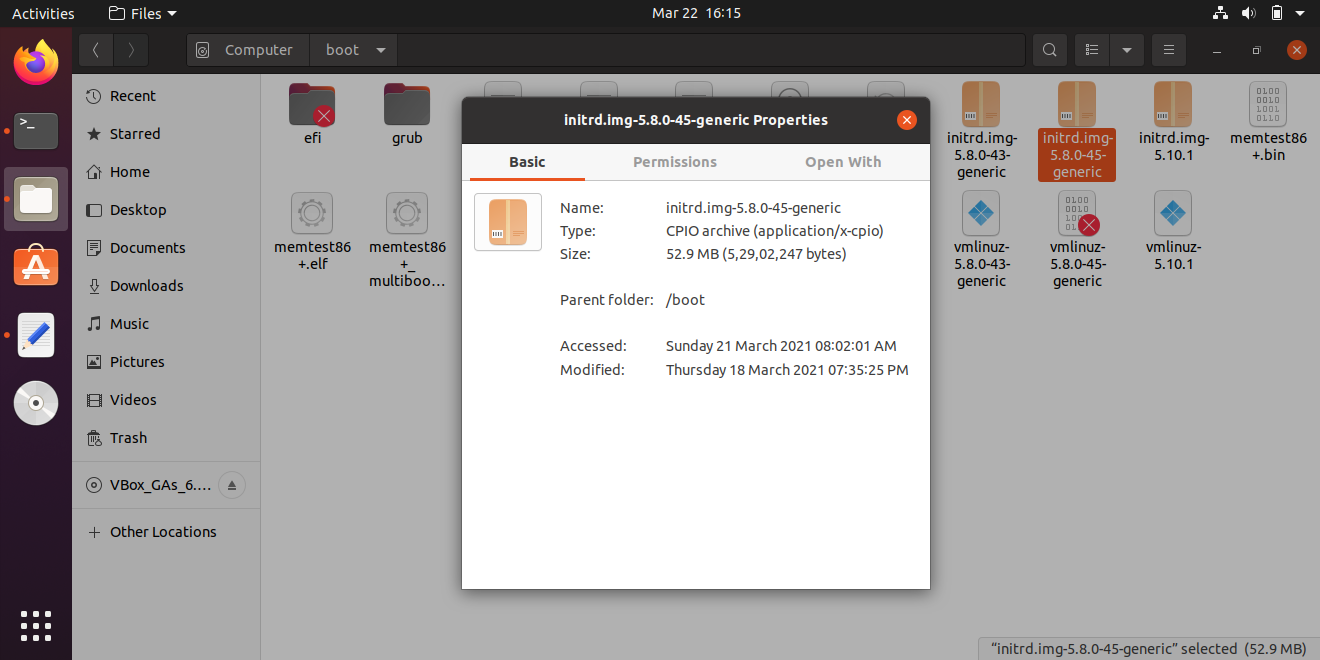
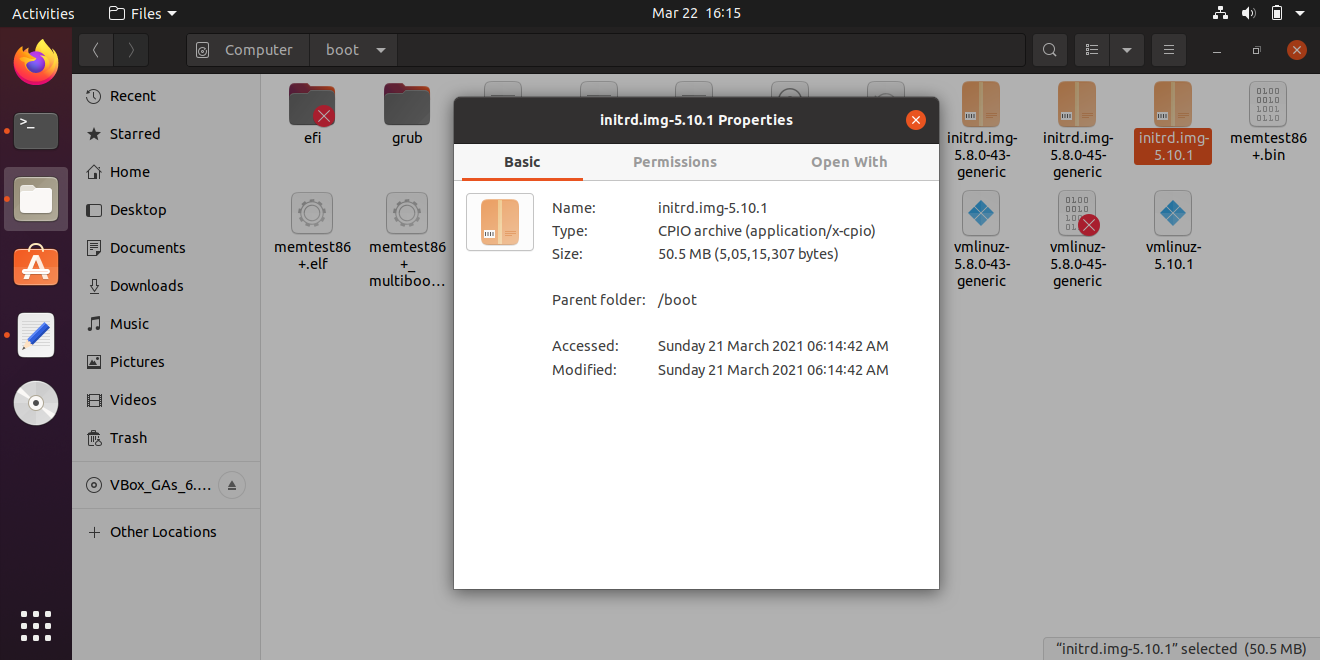
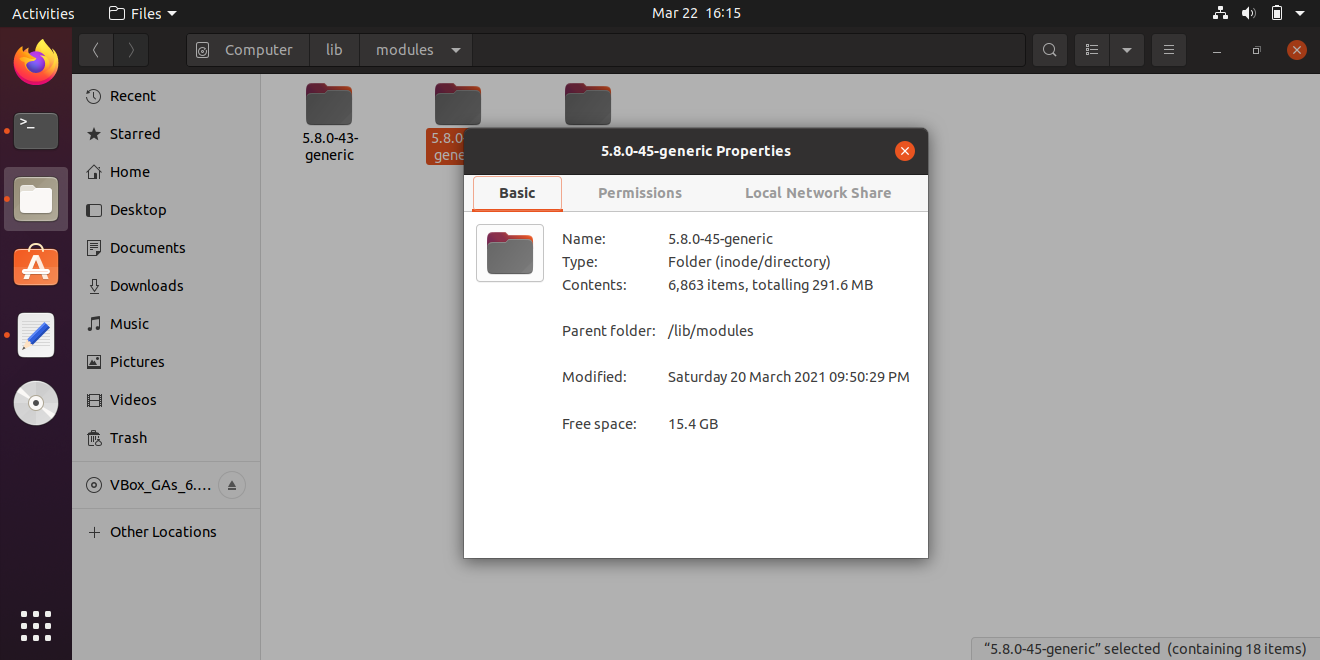
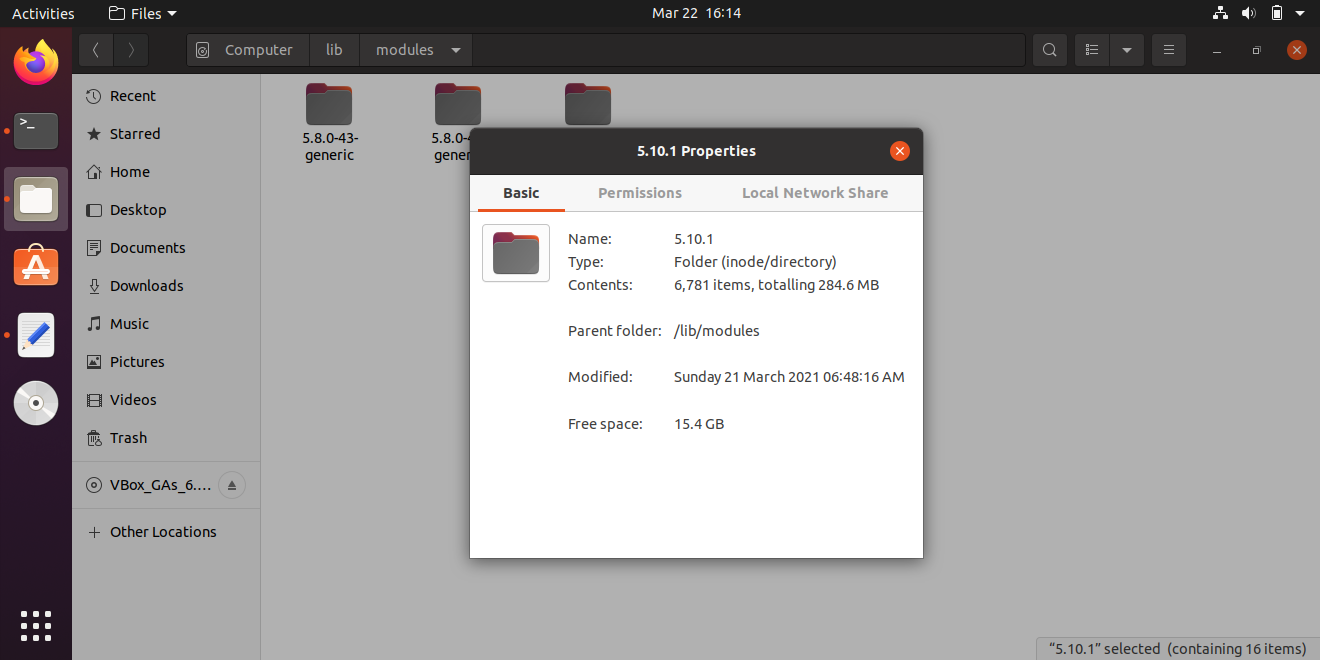
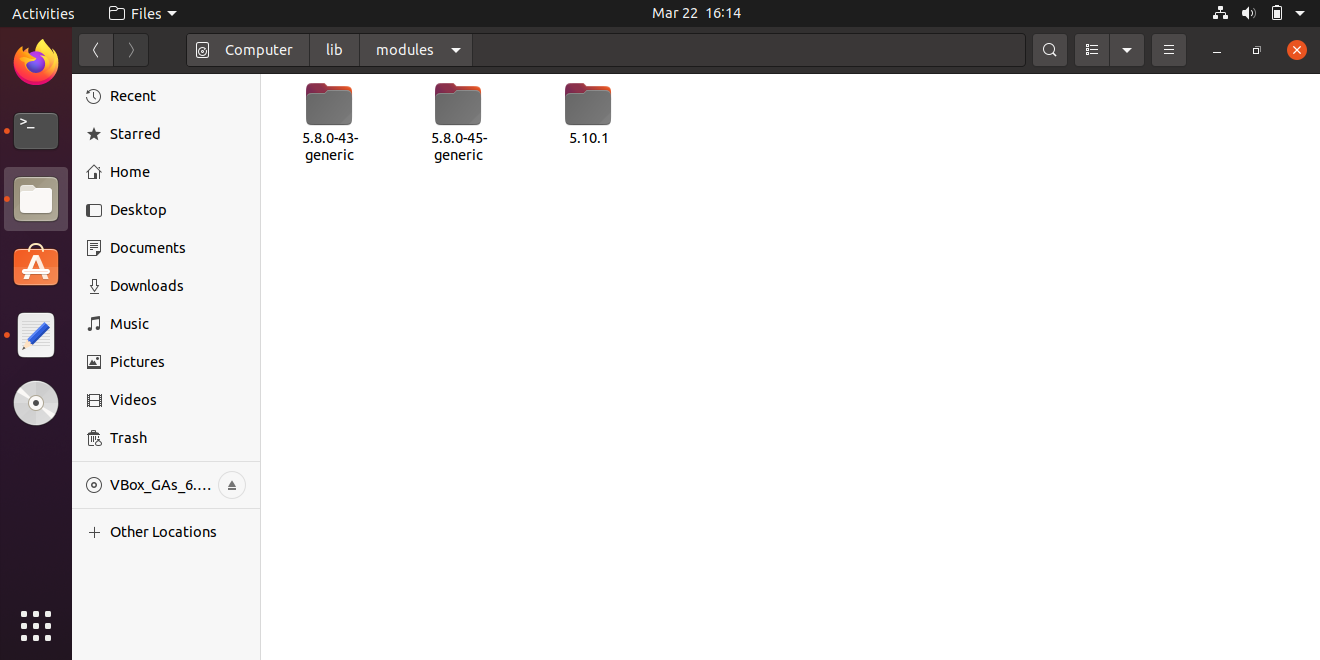
* Return 5/0 in the module init function which gave a segmentation fault.
* while(1){

printk(KERN ALERT “HI”);

}

This on running hangs the system.

*Some screenshots of the system:*



**Q3 Multithreaded In-place merge sort**

**Idea**

Convert the threads to the nearest no. of threads suitable to form a perfect tree, then use the normal function of merge sort with following changes:-

1. If the level in perfect tree(mergeSort tree) is greater than or equal to last level of perfect tree use mergeSort function without any threads
2. Else form 2 threads, for [left, mid] and [mid+1, hi] mergesort
3. Use *pthread\_join* to wait for both subarrays to get sorted
4. Merge the sorted subarrays in the same thread and return

**Note**:-

1. For measuring time of execution of mergeSort with threading use Wall clock time not CPU clock time.
2. Benefit of multithreading is observed for arrays of very large size, most likely because of overhead in thread creation

**Graphs**

X-axis → #threads, Y-axis → time

Fig1. Array size=1e2, thread\_step= +1

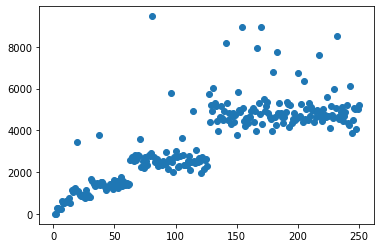


Fig2. Array size=1e4, thread\_step= +1

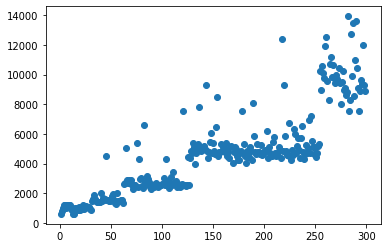


Fig3. Array size=1e7, thread\_step= +1

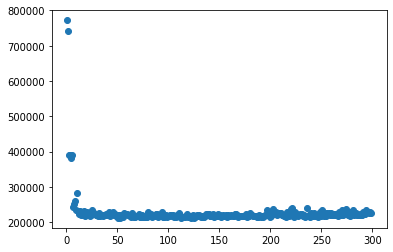
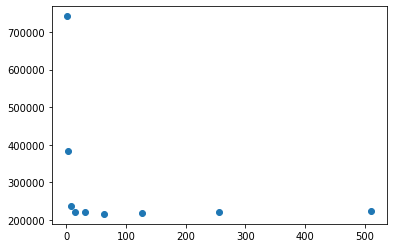


Fig4. Array size=1e7, thread\_step= x2



**Conclusion**

As observed from graphs, multithreading decreases execution time for arrays of size 1e7(or large array) while for others multithreading has more time than 1 thread execution.