**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES-FAST**

**KARACHI CAMPUS**



**COURSE INSTRUCTOR:**  
**Dr Ghuffran Ahmed & Mr Zuhair Waheed**

**CS-2006:OPERATING SYSTEMS**

**TITLE: Comparison between Process And Threads**

**SECTION: BS-SE-5A**

**GROUP MEMBERS:**

**MAQBOOL AHMED (20K-1610)**

**HAMZA JAFRI (20K-1669)**

**UMER VOHRA (20K-1677)**

# OBJECTIVE:

Main Purpose of our project is to compare threads with processes in terms of performance and time taken to complete a task. We will use different sorting algorithms to conclude our results.

.

# INTRODUCTION:

We will show the comparison by implementing sorting algorithm i.e**. Merge Sort** .We will compare the performance of this sorting algorithm with respect to time, number of inputs and speed . The time taken at each step is recorded and displayed. At the end, the 4 sub arrays combine into one array and displayed. After executing, the program also shows the time complexity taken for a particular algorithm and IPC method.

# PROGRAMMING PLATFORM USED:

* The programming platform used is C Language
* The operating system that we worked on is Ubuntu.

# MODULES USED:

* Bubble sort
* Insertion sort
* Merge sort
* Quick sort
* Heap sort

# PROBLEM FACED:

When the processes break e.g., when the array breaks into 4 parts to be assigned to each core the process makes a copy of the array only, so when the processes end itself the copy of array is also deleted, although the array is sorted but it vanishes.

# SOLUTION:

solve this problem, we used to inter process communication so that that the sorted array could be send to the master process so that the array is merged from there. Due to this problem, we Also used comments in the C++ program to use it for debugging.

# IMPLEMENTATION:-

# POSIX THREADS:

The array was divided into 4 parts, and 4 threads were created alongside with each thread taking each part of the array. Threads sort each part of the array individually and finally combine them together to display the final sorted array.

# PROCESSES:

It was implemented using 2 fork calls and then child and parent processes were executed through different if else conditions to make them work accordingly.

# SHARED MEMORY:

We have to use shared memory for implementing Processes so that the sorting done on different sub arrays in different processes should reflect in parent process also.

# HEADER FILES:

#include <stdio.h>: for basic function

#include <unistd.h>: common library for basic stuff

#include <stdlib.h>: common library for basic stuff

#include <time.h>: we need to seed the random function with time(0)

#include <sys/wait.h>: wait(NULL) support from this library

#include <sys/shm.h>: shared memory support from this library

#include <sys/types.h>: common library for basic stuff

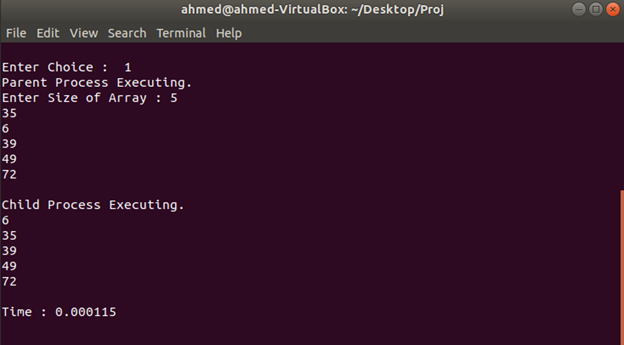
#include <sys/ipc.h>: INTER PROCESS COMUNICATION ipc used because we needed to make a channel through which parent and child could communicate

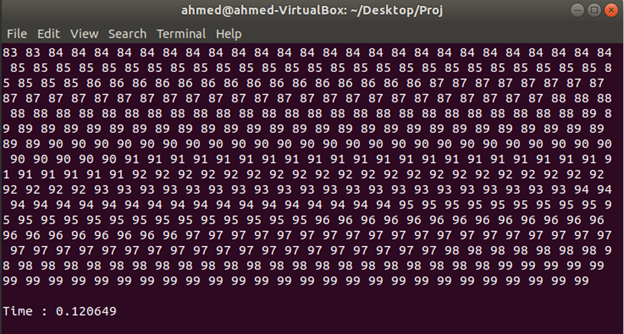
# CONCLUSION:

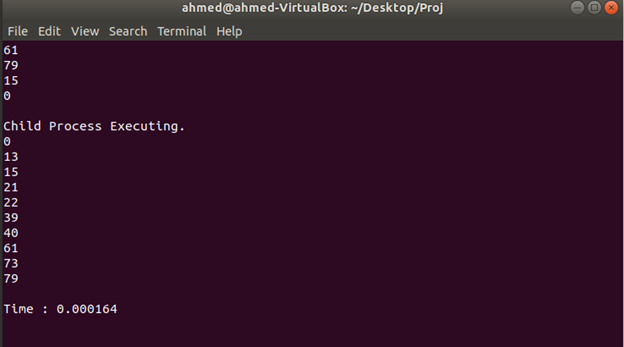
Through the displayed time complexity for each algorithm execution through IPC help us to analyze what algorithm runs faster in each mode. Consequently, we can select best method for sorting on basis of the results when handling large amount of data.

# SCREENSHOTS:

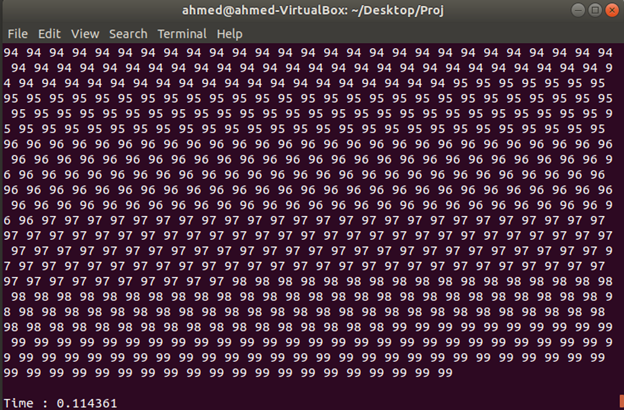
* **BUBBLE SORT PROCESS :**



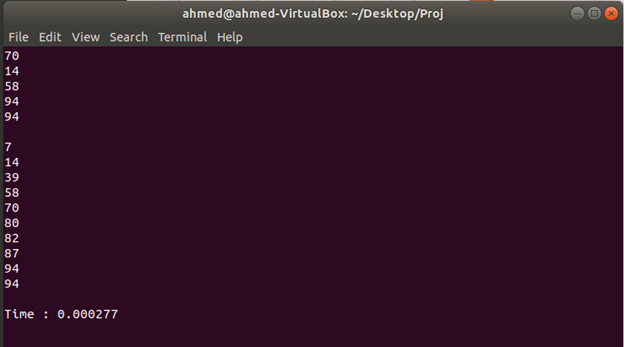
* **BUBBLE SORT THREAD :**
* **INSERTION SORT PROCESS :**



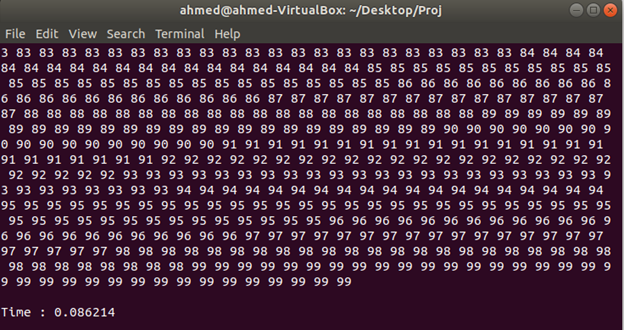
* **INSERTION SORT THREAD :**



* **MERGE SORT PROCESS :**

****

* **MERGE SORT THREAD :**

****

* **QUICK SORT PROCESS :**

**Text

Description automatically generated**

* **QUICK SORT THREAD :**

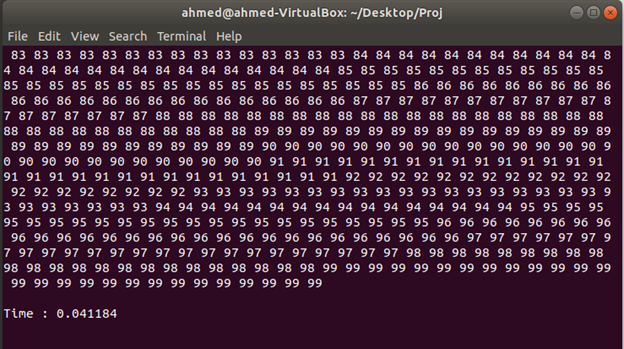
****

* **HEAP SORT PROCESS :**

**Text

Description automatically generated**

* **HEAP SORT THREAD :**

****

* **EXIT :**

**Text

Description automatically generated**

# THANK YOU …