A Mini Project Report

On

***GUI Based Chat Application using Shared Memory***

By

**Pradyumna .N. Shirude (TI046)**

**Samiksha .G. Nehe (TI062)**

**Saket .S. Khopkar (TI060)**

Under the guidance of

**Prof. Bhavana Kanawade**



**Department of Information Technology**

**Hope Foundation's**

**International Institute of Information Technology,Pune**

**Savitribai Phule Pune University**

**2020-2021**

Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CERTIFICATE**

This is to certify that,

**Pradyumna .N. Shirude (TI046)**

**Samiksha .G. Nehe (TI062)**

**Saket .S. Khopkar (TI060)**

of class T.E IT; have successfully completed their mini project work on “**GUI Based Chat Application using Shared Memory**” at **International Institute of Information Technology , Pune** in the partial fulfillment of Graduate Degree course in T.E at department of **Information Technology**, in the academic year 2020-2021 Semester – V as prescribed by **Savitribai Phule Pune University**.

Prof. Bhavana Kanawade Prof. Sarang Saoji

Guide Head of Department

(Department of Information Technology)

**ACKNOWLEDGEMENTS**

Nothing concrete can be achieved without an optimal combination of inspiration and perspiration. No work can be accomplished without taking guidance of experts. It is only the views and advice from ingenious intellectual that helps in the transformation of product into a quality product.

We gratefully thank **Prof. Bhavana Kanawade** for their guidance and constant supervision as for providing necessary information regarding the project and also for their support in completing the project.

We would truly like to express our special gratitude towards our Principal **Dr. Vaishali Patil** and Head of Department **Prof. Sarang Saoji** for their kind cooperation and encouragement which helped us in completing this project.

We would also like to express gratitude towards our parents and friends for giving us constant motivation and their moral support throughout the project.

**Pradyumna .N. Shirude (TI046)**

**Samiksha .G. Nehe (TI062)**

**Saket .S. Khopkar (TI060)**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sr.No | Topic | Pg No. |
|  | **Acknowledgements** | **3** |
|  | **Table of Contents** | **4** |
|  | **Acronyms / Abbreviations** | **5** |
|  | **Abstract** | **6** |
| **Chapter - 1** | **Introduction** |  |
|  | * 1. Introduction | **7** |
|  | * 1. Problem Statement | **10** |
| **Chapter - 2** | **Software Requirement Specification** |  |
|  | 2.1) Hardware Requirements | **12** |
|  | 2.2) Software Requirements | **12** |
| **Chapter - 3** | **Assumptions / Performance Output** | **13** |
| **Chapter - 4** | **Design** | **15** |
| **Chapter - 5** | **Features** | **16** |
| **Chapter - 6** | **Future Improvement in the project** | **17** |
| **Chapter - 7** | **Conclusion** | **18** |
| **Chapter - 8** | **References** | **19** |

**Acronyms / Abbreviations**

1. OS :- Operating System
2. GUI :- Graphical User Interface
3. FIFO :- First In First Out
4. IPC :- Inter Process Communication

**Abstract**

The GUI Based Chat application designed in this project makes use of concept of shared memory in operating system. Typically, this project can be considered as replica of modern chat application, but designed in low level languages. The extensive use of C programming language and the concept of shared memory is what that makes this project statement look like.

The purpose of this project is to demonstrate how processes communicate with one another with the use of OS characteristic know as Shared Memory. In computer science, shared memory is memory that may be simultaneously accessed by multiple programs with intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs.

We have implemented Inter process communication using shared Memory. [Inter Process Communication](https://www.geeksforgeeks.org/inter-process-communication/) through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.

These are some of the main concepts used in the following project to retrieve final output.

**Chapter – 1 :- Introduction**

* 1. **Introduction**

The problem with pipes, FIFO and message queue – is that for two process to exchange information. The information has to go through the kernel.

* Server reads from the input file.
* The server writes this data in a message using either a pipe, FIFO or message queue.
* The client reads the data from the IPC channel, again requiring the data to be copied from kernel’s IPC buffer to the client’s buffer.
* Finally the data is copied from the client’s buffer.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

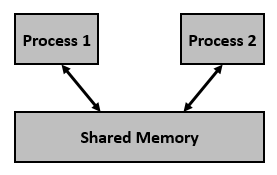


Figure: - Shared Memory

The following system calls are used during the implementation of shared memory:

* **int shmget(key\_t key, size\_t size, int shmflg)**

The above system call creates or allocates a System a shared memory segment. The arguments that need to be passed are as follows –

The **first argument, key,** recognizes the shared memory segment. The key can be either an arbitrary value or one that can be derived from the library function ftok().

The **second argument, size,** is the size of the shared memory segment rounded to multiple of PAGE\_SIZE.

The **third argument, shmflg,** specifies the required shared memory flag/s such as IPC\_CREAT (creating new segment) or IPC\_EXCL

This call would return a valid shared memory identifier (used for further calls of shared memory) on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

* **void \* shmat (int shmid , const void \*shmaddr, int shmflg)**

The above system call performs shared memory operation for System shared memory segment i.e., attaching a shared memory segment to the address space of the calling process. The arguments that need to be passed are as follows –

**The first argument, shmid,** is the identifier of the shared memory segment. This id is the shared memory identifier, which is the return value of shmget() system call.

**The second argument, shmaddr,** is to specify the attaching address. If shmaddr is NULL, the system by default chooses the suitable address to attach the segment. If shmaddr is not NULL and SHM\_RND is specified in shmflg, the attach is equal to the address of the nearest multiple of SHMLBA (Lower Boundary Address).

**The third argument, shmflg,** specifies the required shared memory flag/s such as SHM\_RND (rounding off address to SHMLBA) or SHM\_EXEC (allows the contents of segment to be executed) or SHM\_RDONLY (attaches the segment for read-only purpose, by default it is read-write) or SHM\_REMAP (replaces the existing mapping in the range specified by shmaddr and continuing till the end of segment).

This call would return the address of attached shared memory segment on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

* **int shmdt (const void \*shmaddr)**

The above system call performs shared memory operation for System V shared memory segment of detaching the shared memory segment from the address space of the calling process. The argument that needs to be passed is −

**The argument, shmaddr**, is the address of shared memory segment to be detached. The to-be-detached segment must be the address returned by the shmat() system call.

This call would return 0 on success and -1 in case of failure. To know the cause of failure, check with errno variable or perror() function.

* **int shmctl(int shmid, int cmd, struct shmid\_ds \*buf)**

The above system call performs control operation for a System shared memory segment. The following arguments needs to be passed −

**The first argument, shmid**, is the identifier of the shared memory segment. This id is the shared memory identifier, which is the return value of shmget() system call.

**The second argument, cmd**, is the command to perform the required control operation on the shared memory segment.

**The third argument, buf**, is a pointer to the shared memory **structure named struct shmid\_ds**. The values of this structure would be used for either set or get as per cmd.

* 1. **Problem Statement**

To create a GUI based Chat system using shared memory.

Thus in order to fulfill the requirement of above problem statement, we used the basic C programming language. The sole purpose of this project is to show the working of IPC (Inter Process Communication) using **Shared Memory concept** by demonstration of a chatting application.

The main function of this project is that it perfectly performs like modern chat platforms like WhatsApp, Telegram, etc. Talking about the internal working or abstractional point of view, the working of the project is hidden from the operator of this project so as not to confuse the end user in terms of operational complexities implemented in the project.

The project is basically implemented on CLI (Command Line Interface) and GUI is provided to make the project more interesting. It uses two separate terminals (One for Server and other for Client). The input will be given in the server terminal which it will be immediately reflected into the other client side terminal. The aim is to clearly display the messages the server sends to client and vice versa.

**Chapter – 2 :- Software Requirement Specification**

The Software Requirements Specification is produced at the culmination of the analysis task. SRS is a document that completely describes what the proposed software should do without describing how the software will do it.

The basic limitation for this is that the user need keeps changing as environment in which the system was to function as it changes with time. This leads to a request for requirement changes even after the requirement phase is done and the SRS is produced.

The function and performance allocated to software as part of System

Engineering are refined by:

* Establishing a complete information description of the System.
* A detailed functional description.
* A representation of System behavior.
* An indication of performance requirements and design constraints.
* Appropriate validation criteria, & other information pertinent to requirements.

**2.1) Hardware Requirements**

The System should be Graphical User Interface that is more user-friendly. Graphical user interface provide more enhanced performance.

|  |  |  |
| --- | --- | --- |
| **Description** | **Minimum** | **Recommended** |
| Processor | Intel Dual Core or  equivalent with Clock  Speed of 2000 MHZ | Intel Core 2 Duo  or equivalent of |
| Primary Memory (RAM) | 1 GB | 2 GB |
| Display | 1024x768, 32 bit  color SVGA | 1289x800, 32 bit  color SVGA |

**2.2) Software Requirements**

|  |  |  |
| --- | --- | --- |
| **Description** | **Minimum** | **Recommended** |
| Operating System | Linux Distributions | UNIX, Fedora |
| Browser | NA | NA |
| Operating Environment | Terminal | Terminal |

**Chapter – 3 :- Assumptions / Performance Output**

The schedules, estimates and costs herein are based on the assumptions identified below.  If any of the below assumption is false, then the team members will be fully responsible for the incorrect assumption and crash of the system for further operations.

## Methodology:

The project will be completed using tools and techniques as learnt by group members through various learning programs. With usage of RAD Model in Software Development, the end product was delivered in time and in efficient manner.

* **Performance:**

The traditional style of measuring performance, (i.e., in the 3270 world) where users enter data into a screen, press an enter key or a function key wait for a response and measure the time to complete the function is no longer applicable for client/server applications.  In a client/server GUI environment each user/machine interaction results in some processing at one or more software servers.  Therefore, the process of entering a GUI screen and working on a GUI screen will typically result in many system transactions, most of which are transparent and unknown to the user.  Each of these transactions require the user to wait for a response and therefore each system interaction has a different set of response expectations by the user – most of which have been fostered and groomed in the popular MS Windows environments.

* **Conversion of Data:**

Conversion of data is done from user input to a GUI decorated one. The Windsor pilot system will be replaced not converted. Also Windows library don’t support these actions, UNIX, Linux, Fedora, etc. are used.

* **Translation:**

The language primarily used in the software is English as it is better understood by many people across the premises. No bilingual documents will be produced.  Translation will begin only after acceptance of the English version of the document.

The sponsor translation includes the translation and the review and verification of the translated text prior to delivery to them for approval.  The Development team will review and revisions will be incorporated directly.

**Chapter – 4 :- Design**

The project was implemented on Ubuntu OS using Virtual Machine. To run this project, following is the process to execute in order to produce a proper output. Following figure shows a screenshot of the terminals in which the project is executed using the CLI (Command Line Interface).

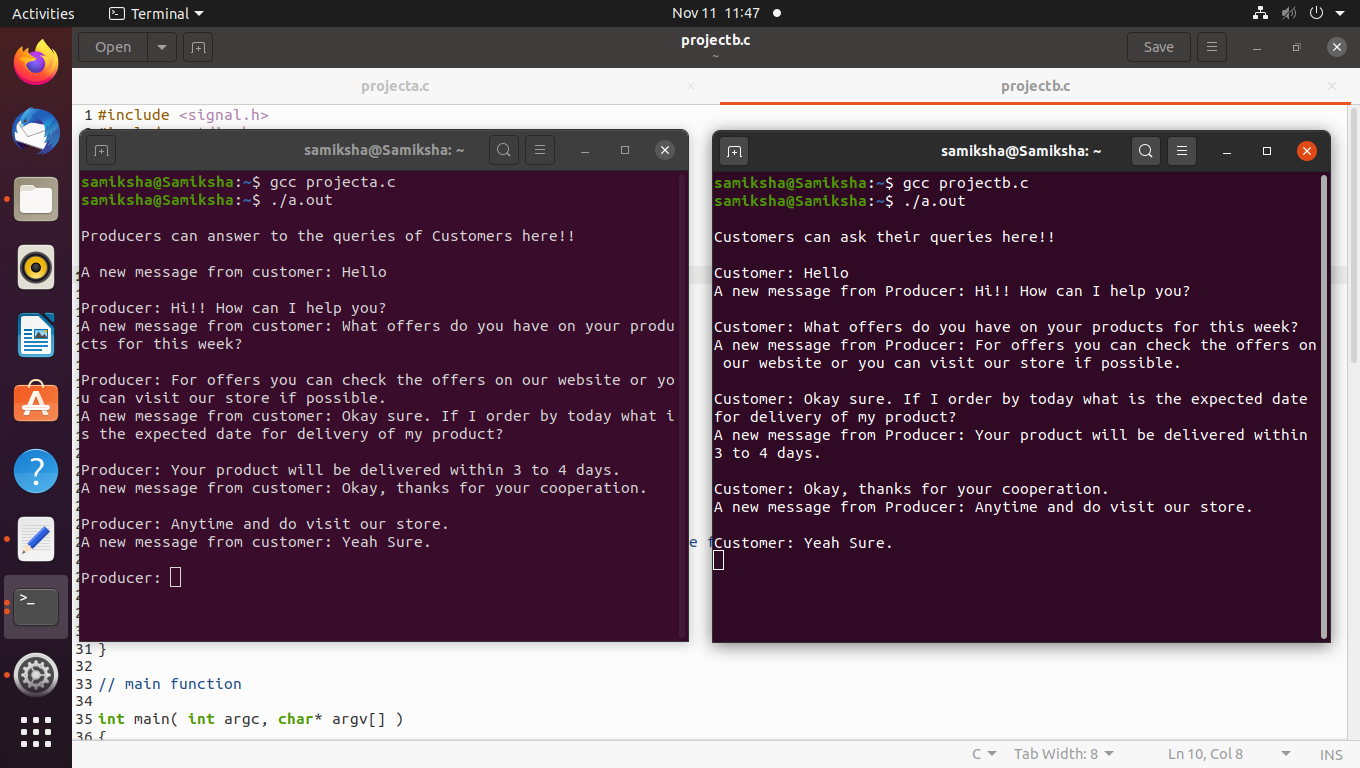
****

Figure: - Working of the project in CLI mode. **GUI will be added in further developments of this same project**.

**Chapter – 5 :- Features**

* Built using the basic C programming language with the help of Linux Distribution incorporating 1-1 communication.
* Fast Message Transfer.
* Easy to interact with the system.
* Compatible in all of the Linux Distributions like Fedora, UNIX, etc.

**Chapter – 6 :- Future Improvement in the project**

* As mentioned in the topic, we have **not added GUI** in the project. The GUI will be added in further improvements.
* Currently the system can handle **only 1 to 1** communication. We will add multi user communication in further updates.
* The project does not give proper output when we try to perform multiple terminals in order to perform **multi user communication**. We will try to rectify the problem in **future update**.

**Chapter – 7 :- Conclusion**

The report has discussed the development of a GUI Based Chat Application project (GUI to be added later). The main objective was to implement the concept of Shared Memory in the project in order to grasp its concept more clearly by the students. Also another concept of Inter Process communication was also used. Both of the objectives were met to the final as we, the students, understood the topic more thoroughly. By using the concept of Shared Memory, the application was able to perform as it was expected. Only thing we encountered in this project was GUI which we will be using the gtk library for implementing the GUI in the same project in later stages of development.

The lab introduced us to the concept of operating system and many other important concepts to be learned in the subject. In the lab, we learnt and implemented many Operating System concepts which are necessary to be learnt by every student in their tenure as a learner in their respective colleges. Besides the concept of the Shared Memory, it has many other applications in real life such as *e*X*treme*DB allows two or more processes in a multi-processing operating environment (for example, Linux and Windows platforms) to share a common database.

Thus we have clearly understood what we have implemented and also know the working of the application where we can understand the flow of the project and develop it further in near future time to come by.

**Chapter – 8 :- References**

1. <https://www.gtk.org/docs/>
2. Shared Memory Guide online :- <https://docs.oracle.com/cd/E19455-01/806-4750/chap7rt-67/index.html>
3. For detailed explanation of Shared Memory Concept :- <https://fast-dds.docs.eprosima.com/en/latest/fastdds/transport/shared_memory/shared_memory.html>
4. Inter-process Communication Guide :- https://opensource.com/article/19/4/interprocess-communication-linux-storage
5. <https://www.youtube.com/playlist?list=PLaybP4QvyRH1obigtMQwhB2DWhPKg32_T>