RPC Based Online Test Module

Tushar B. Kute
Assistant Professor
Department of Information Technology
Sandip Institute of Technology and Research Centre, Nashik, India

Tushar J. Surwade
Assistant Professor
Department of Information Technology
Sandip Institute of Technology and Research Centre, Nashik, India

Abstract

In our day-to-day life we have to appear for various types of online examination and while appearing for examination, candidates are able to switch between many different programs. One such program can be a Web Browser or an Informative Document and may lead to malpractice. To avoid such malpractices, we must ensure that other programs are inaccessible to the candidate. So, we present a system about an online test module which works on the principle of RPC (Remote Procedure Call) developed in C-language and is currently implemented to work on Linux Operating System. The reason behind choosing Linux as a platform is that it has many advantages, one of them being that it avoids Windows malwares up to 80% to 90%. In addition, Linux is an open source operating system and it provides any user to deeply explore its features. The system we are about to present executes on terminal-only mode or CLI (Command Line Interface). It is based on Client-Server architecture in which there is a server and number of clients. The system also uses Multithreading to handle various clients simultaneously, where each client is allotted to an individual candidate.

Keywords: Client, Command Line Interface, Exam, Linux, RPC, Server, Terminal

I. INTRODUCTION

In today's world managers need faster assessment to bring out best talents out of the crowd. To find such talents various examinations are held in different ways, like written tests, aptitudes, online tests. As all the specified exams can take lot of time, except the online tests, to deliver results which may delay the decision-making process. These online examinations are the best way to show results faster. Generally, online examinations are not secure because there is a possibility that the candidate may switch over to another program and can search for solutions. So there is a need of a secured online examination system which prevents application switching and thus restricts the candidate from malpractice.

To fulfill this need of secured online examination system, we have created an online test module that does not allow application switching and is totally based on Command Line Interface in Linux operating system. Linux operating system provides a Terminal-Only mode which does not support application switching. In case the candidate knows how to close the terminal and access another application then the examination process running in the terminal ends there. Since, this online test module is created in such a way that once the Console is switched then the execution of test process terminates and the score is declared, which denies any chance of resuming the test.

Our module consists of one server and multiple clients. The server implements multithreading in such a way that there is a thread that caters to a client connected to it. Each thread provides response to the client's request. Every client has two threads running in parallel, one for keeping track of test timing and one for communicating with the server. All these threads utilize RPC (Remote Procedure Call) for communication between them.

II. CLIENT SERVER ARCHITECTURE

Client Server Architecture is a model in which a client sends a request to the server and the server provides a response that fulfills the request [3]. This architecture can be implemented on a single machine but it can be more efficiently used in a network by interconnecting programs that are distributed across different locations. Usually, in client server model, one server is activated and waits for the request from the client. Multiple client programs can avail the services of the server program concurrently. The client provides a user interface to the user through which the user requests for a service. This service is responded by the server according to the availability of resources.

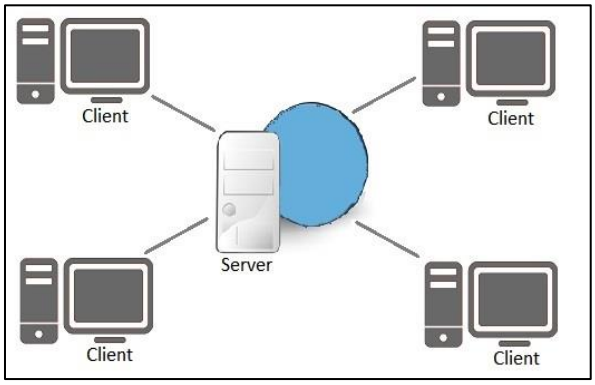

Fig. 1: Client-Server Architecture

Some of the advantages and disadvantages of using the Client Server Architecture are as follows:

1) Centralized Resources: Redundant and inconsistent data is prevented because the resources are managed centrally by the server.
2) Scalable Network: We can modify the network by adding or removing clients without affecting the operation of the network.
3) Overloaded Servers: A centralized architecture makes the server overloaded, because of the simultaneous request of the different individual clients on the network.
4) Server Failure: A failure in a client will not affect the working of the network but a server crash will definitely bring down the network.

III. REMOTE PROCEDURE CALL

A remote procedure call (RPC) is when the procedure being called and the calling procedure are in different processes. Procedure calls are used for transferring control and data inside a program running on one computer. This same mechanism is extended for transferring control and data over a communication network, known as RPC[7]. When the Remote Procedure is called, the calling program is paused and the parameters for the procedure are passed over the network to the program where the procedure will execute, and then the remote procedure would execute. When the remote procedure finishes execution, its output or result is passed back to the calling function, where the execution resumes in the calling program.[1]

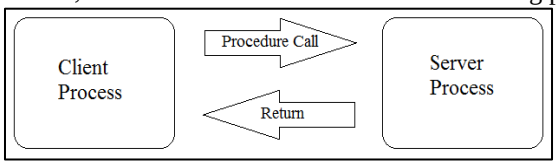

Fig. 2: Remote Procedure Call

Suppose a thread T1 wants to communicate with thread T2, then T1 needs to call a procedure of T2 which would require passing of necessary parameters to T2 for calling T2's procedure. Thread T1 gets blocked until either procedure of T2 returns some value to T1 or procedure of T2 finishes execution. The thread T2 executes the procedure called by T1. T1 gets unblocked after the execution of procedure of T2 and continues its execution.

To process this communication, RPC needs five modules viz., the user, the user-stub, the RPC package, the server-stub, and the server. The RPC package contains supporting instances for communication. Client machine runs three modules user, user-stub, and an instance of RPC package which is required for client-side communication. Server machine also runs three modules server, server-stubs and instance of RPC package which is also required for server-side communication. User module requests for the particular service from the user stubs. The user-stub accumulates the required data and passes it to instance of Client's RPC package. The client instance of RPC package passes this information to server's instance of RPC package. The Server instance passes all data packets to server-stub and paused Client's process completely. The server stub invokes exactly the required procedure in the server for the service required. After the response is provided by the server, it is passed through server stub and result of which client process was awaiting is passed to them and they are released.

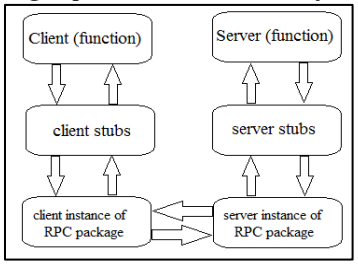

Fig. 3: RPC Communication Structure

A. Advantages of RPC [2]:

1) You don't have to worry about getting a unique port id. The server can bind to any port and register the port with its RPC name server. The client will contact this name server and request the port number that corresponds to the program it needs.
2) The system is transport independent. This makes code more portable to environments that may have different transport providers in use.
3) Applications on the client only need to know one transport address.
4) The function-call model can be used instead of the send/receive (read/write) interface provided by sockets.

IV. MULTI-THREADING USING POSIX THREADS

Multiple strands of execution in a single program are called threads. An alternate explanation is that a thread is a sequence of control within a process [1]. A process can have multiple threads executing in it. Multi-threading is a type of execution model that allows multiple threads to exist within a process such that they execute independently but share the process resources. Threads within the same process share the same address space. This allows threads to read from and write to the same data structures and variables, and also facilitates communication between threads. A common example of multi-threading is that you can have a word processor that prints a document using a background thread, but at the same time another thread is running that accepts user input, so that you can type up a new document. If we were dealing with an application that uses only one thread, then the application would only be able to do one thing at a time – so printing and responding to user input at the same time would not be possible in a single threaded application.

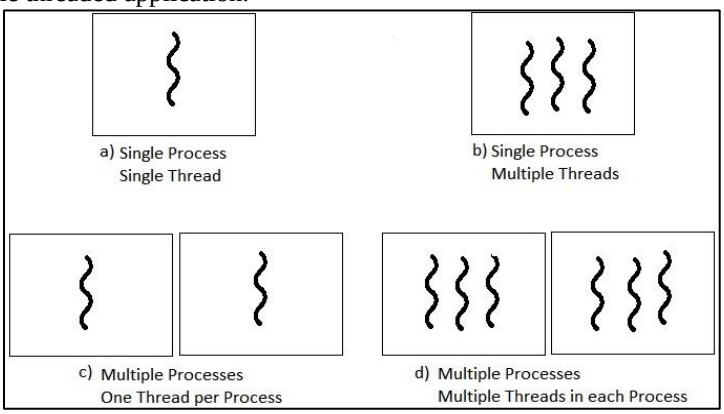

Fig. 4: Multi-Threading

A. Resource Sharing Between Threads [4]:

Threads are sometimes called lightweight processes, since a thread is "lighter weight" than a process. That is, thread creation can be 10-100 times faster than process creation. All threads within a process share the same global memory. This makes the sharing of information easy between the threads, but along with this simplicity comes the problem of synchronization**.** But more than just the global variables are shared.

1) All threads within a process share: process instructions, most data, open files (e.g., descriptors), signal handlers and signal dispositions, current working directory, and user and group IDs.
2) Each thread has its own: thread ID, set of registers, including program counter and stack pointer, stack (for local variables and return addresses), errno, signal mask, and priority.

B. Essential Thread Functions [4]:

To perform some operations, threads between processes has to communicate:

1) pthread_create:

Whenever any process is created, it is initiated by a main thread or initial thread and it is created automatically without manual intervention. For creating new threads to support the process, we need to call pthread_create. For creating new thread pthread_create is used in the following syntax:

```
#include <pthread.h>

int pthread_create(pthread_t *tid, const pthread_attr_t *attr
                   void * (*func) (void *) , void *arg) ;
```

The function pthread_create returns 0 if successful creation of thread is done else it will return an error_no. of unsuccessful creation of thread.

2) pthread_join:

After the creation of thread, it must be specified to wait for a thread to complete its execution. The pthread_join function specifies the thread to wait until the specified thread has completed execution. This function is used to join a new thread with a terminated thread. The syntax for writing pthread_join function is as follows:

```
#include <pthread.h>

int pthread-join(pthread-t tid, void **status);
```

After a successful call to *pthread_join*() the caller is guaranteed that target thread has completed. The function pthread_join returns 0 if successful and returns *error_no* if unsuccessful in completely execution in threads.

3) pthread_exit:

The function pthread_exit is used to terminate the thread. This function terminates the calling thread and returns a value via a parameter that is available to another thread in the same process that calls pthread_join.

```
#include <pthread.h>

void pthread-exit (void *status) ;
```

This function does not return any value to the calling thread and is always successful.

V. ARCHITECTURE OF OUR SYSTEM

The overall working of the system is as described in the following diagram:

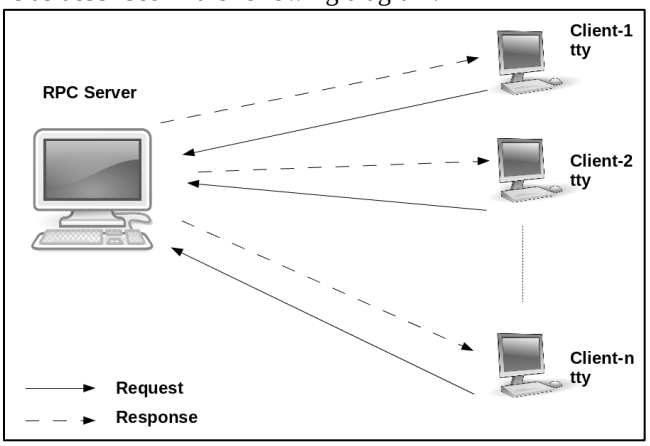

Fig. 5: Architecture of RPC based Online Test Module

Let us go step by step and know the working of whole system.

1) The architecture's base model is built upon Client-Server architecture.
2) All the questions, choices and correct answers will be stored in server machine.
3) Client will request for question from server and server would provide a random question with their choices from the database as response.
4) The client will select his option of answer and will be sent to server, and server will check for correct answer.
5) In addition, clients will also have a timer thread running in the background for the time limit of examination. The server does not keep track of timers of every client.
6) Candidates won't be able to switch between the programs because of terminal-only mode.

7) Question request and response done during this conversation is with the help of remote procedure call where required data are passed as parameters for these procedures within the network.

VI. ADVANTAGES

- Fully secure system that won't allow students to try any type of malpractice.
- Any number of nodes can be added or removed as per the requirement.
- Any candidate trying to switch between the programs will terminate the examination.
- All the question and answers are stored on the server's system, so there is no issue of security.

VII. DRAWBACKS

- Terminal based user interface is not much user-friendly.
- Any network traffic would lead to undesirable high latency network.
- Server failure will take down the whole examination system.

VIII. CONCLUSION

By proposing this system, we can be assured that it fulfills the needs of an online test module in day-to-day life. Because of using RPC we provide a better level of integrity and security to the data. The use of console mode restricts the candidate from switching to other applications and using them for malicious purpose. A timer thread executing in the background of the client ensures that the examination ends in stipulated time. Overall it satisfies almost every need, with some additional feature which makes our system better and different than the existing ones.

REFERENCES

[1] UNIX Network Programming, 2nd ed., vol. 2, W. Richard Stevens.
[2] Neil Mathew, Richard Stones, "Beginning Linux Programming", Wrox Publications, 4th Edition, Wiley India Edition, ISBN-978-81-265-1571-4
[3] Andrew Tanenbaum, Albert Woodhull, "The MINIX Book Operating System Design and Implementation", Eastern Economy Edition, Prentice Hall of India, Third Edition, ISBN-978-81-203-2955-3
[4] Achyut Godbole, "Operating Systems", Second Edition, The McGraw Hill Publishing Company, ISBN-13-978-0-07-059113-4
[5] Andrew D. Birrell, Bruce Jay Nelson, "Implementing Remote Procedure Calls", Xerox Palo Alto Research Center, ACM Transactions on Computer Systems, Vol. 2, No. 1, February 1984, Pages 39-59
[6] Andrew S. Tanenbaum, Robbert van Renesse, "A Critique of the Remote Procedure Call Paradigm", Dept. of Mathematics and Computer Science, Vrije Universiteit
[7] Andrew D. Birrell, "Secure Communication Using Remote Procedure Calls", ACM Transactions on Computer Systems, Vol. 3, No. 1, February 1985, Pages 1-14.