Explanation of Problem and Solution

Academican: Muhammet Gökhan CİNSİDİKİCİ

CopyRight(C): Muhammet Gökhan CİNSİDİKİCİ

Name-Surname(Author): Berk KOCAMANOĞLU

1.Abstract

1.1 Definition of Problem:

Problem requires a server in order to checking updates of applications and giving them URL to update themselfs. Coursly solution is demo to understand interproccess communication (IPC) so applications (clients) was codded by myself.

1.2 Summary of Solution:

When server and client is executed, client sends a information message about itself to message queue(MQ) then server takes message and compares message between taken message and server`s array which includes client`s id,version and name. If client is up to date ,server sends received message. However,If client is not up to date ,server create a shared memory (SHM) then put a URL to update a new version then sends a message about new version of program and shared memory`s id. Client takes message and changes its version and prints URL.

2.Introduction

Codes are wroted in pseudocode because of ignorance in c programming. However, problem is solved in theoretical IPC and algorithms. Otherwise, programs have java

functions in order to arrange string types.Also programs uses message queue and shared memory.

2.1 Java Functions:

Coursly,there is an inspiration of java programming. We was enchanted by java`s basic and understandable default string functions so we took one by one from java.

- **“string”.substring(int FirstIndex ,int LastIndex)** : This function used after string types.The function returns between FirstIndex value and LastIndex value of string. However,the function counts FirstIndex value in string. For example, “MerhabaHocam”.substring(3,8) returns “habaH”.You can see that index of 3 is counted but index of 8 is not counted. Also,If function gets one integer value, function returns string of integer value to end of the string.
- **“string”.indexOf(char CHAR)** :This function scans string and find CHAR`s index which closest of string`s **left** side then returns index of CHAR. For example, “wondering sockets”.indexOf(‘ ’) returns integer and it is 9.
- **“string”.lastIndexOf(char CHAR)** : This function scans string and find CHAR`s index which closest of string`s **right** side then returns index of CHAR. For example, “wondering sockets”.indexOf(‘e’) returns integer and it is 14.
- **“string”.length** : It returns of number of character in string.
- **string.length** :If we assume that string is an array.Length value returns number of array`s elements.

2.2 Message Queue:

A message queue is a form of IPC that allows different processes or components of a system to communicate with each other by exchanging messages. It provides a way for applications to send and receive messages asynchronously, meaning that the sender and receiver do not need to be actively communicating at the same time. This decoupling of components can enhance the flexibility, reliability, and scalability of a system.

- **Creating a Message Queue (‘msgget’)**: To create a message queue, a process uses the ‘**msgget**’ system call. This call returns a message queue identifier (msgid), which is used to access the message queue later. If the message queue already exists, ‘**msgget**’ can be used to obtain its identifier.
- **Sending a Message (‘msgsnd’)**: Once a message queue is created, processes can send messages to it using the ‘**msgsnd**’ system call. The sending process specifies the

message queue identifier, a pointer to the message buffer, the size of the message, and a message type.

- **Receiving a Message (‘msgrcv’)**: Processes that want to receive messages from the queue use the ‘**msgrcv**’ system call. The receiving process specifies the message queue identifier, a pointer to the message buffer where the received message will be stored, the size of the buffer, and the desired message type. Messages in the queue are typically retrieved in a first-in, first-out (FIFO) order.

2.3 Shared Memory:

Shared memory is a mechanism for interprocess communication (IPC) in which multiple processes share a region of memory, allowing them to exchange data more efficiently. Instead of using message passing or other forms of communication, processes can read from and write to the shared memory, enabling faster communication and data sharing. Shared memory is often used when processes need to collaborate closely and exchange data frequently.

3. My Approach

3.1 Client Side:

```
1
2   #libraries
3
4   func main{
5
6       string prgrmID = "0000001 V2.3 CinsCalculator"
7
8       msgbuf msg
9       msg.mtype = 0
10      msg.mtext = programID
11
12      msgbuf receivedMsg
```

First of all,program have libraries , main function , programID (defined as string) and messages.prgrmID includes program`s id number ,version and name. msg and receivedMsg defined as msgbuf but there is not any structure. It came from libraries. msg.mtype means like message`s id and msg.mtext is text of msg. Program define receivedMsg but there is no need to define mtype and mtext because there will be filled msgrcv function.

```
13
14      key_t key = ftok("/home",'M')
15      int msgid = msgget(key,0666)
16      if(msgid == 1){
17          print("Message queue id err")
18          exit()
19      }
```

Program creates key. Program uses that key in order to create message queue and message queue id.msgget() function returns message queue id. If there is a message queue with that key, msgget() function returns message id which created before.

```
21      if(msgsnd(msgid,&msg,msg.mtext.length)==-1) {
22          print("Message queue send err")
23          exit()
24      }else{
```

Msgsnd() function sends message to message queue whose id is defined before.If message cannot be sended it returns –1 integer number then if block would be ran.Prints to screen “Message queue send err” then procces would be dead. After succesful send,server receives message and procces message.It would send message to message queue in order to specific client receives right message to understand up to date.Details are in **2.2 Server Side.**

```
24      }else{
25
26
27          if(msgrcv(msgid , &receivedMsg , prgrmID.substring(0,prgrmID.indexOf(' ')))==-1){
28              exit()
29          }
```

After client does not stick if block,it runs from else block.It gets message from in line 27.If there is no message in message queue it would wait until message comes. msgrcv() function needs 3 thing msgid , defined msgbuf`s adress and which message program should take from message queue.The working princible is basic. Firstly, function finds message queue which belongs msgid (defined before in this project).Secondly,function scans every msgbuf structure which belongs and defined in specific message queue then finds specific message whose id is same with defined id in function(In this project this side is **prgrmID.substring(0,prgrmID.indexOf(' '))==-1** .Also it is program`s id number.It is 0000001).Finally,defined receivedMsg`s adress would be found message adress so program take message and defines receivedMsg.The message received from message queue would be gone from message queue.

```
31          if(receivedMsg.mtext == msg.mtext){
32              print("Program is up to date")
33              print(receivedMsg.mtext)
34
35
36          }else{
```

It compares between received message and its prgrmID.If bot of them are same , program is up to date and prints “Program is up to date” and received message.

```
36      }else{
37
38
39          int shmid = (int)receivedMsg.mtext.substring(receivedMsg.lastIndexOf(' ')+1)
40          if(shmid==-1)(
41              exit()
42          )
43          string *shmURL = (string *)shmat(shmid)
44
```

After client does not stick if block,it runs from else block.We realize received message and prgrmID is not same so there is a new version of client.Program takes shared memory`s id from received message(received message`s text preparation is in **3.2 Server Side**).If there is a problem about shared memory program exits then program gets URL from shared memory.

```
46              print("URL is "+shmURL)
47              print("Program is updated")
48
49
50              shmctl(shmid , IPC_RMID)
51
52
53              prgrmID =  receivedMsg.mtext.substring(0,receivedMsg.lastIndexOf(' '))
54              print("New prgrmID "+prgrmID)
55
56
57          }
58      }
59
60      //Program contiunes
61
62
63
64  }
65
```

Program prints URL taken from shared memory and “Program is updated” then deletes shared memory.Finally, program changes its prgrmID and prints new prgrmID.

3.2 Server Side

```
1   #libraries
2
3
4   func main{
5
6
7       string newVersions[6]
8       string newURLs[6]
9       int serverID = 0
10
11      msgbuf receivedMsg
12      msgbuf sendedMsg
```

Program stores new versions of programs and their URL. Coursly ,server`s id is 0 and server defines msgbuf to send and receive message from message queue(Server communicate with clients but gets messages from message queue).

```
14      key_t msgKey = ftok("/home",'M')
15      int msgid = msgget(msgKey,0666)
16      if(msgid == 1){
17          print("Message queue id err")
18          exit()
19      }
20
21
22      key_t shmKey = ftok("/temp",'C')
23      int shmID
```

Server creates key same as clients then creates message queue with this key (If there is a message queue with this key msgget returns created message queue`s id).Server controls message queue is created or not then creates key about shared memory and defines shmID which would use after.

```
25      while(1){
26
27          if(msgrcv(msgid ,&receivedMsg,serverID) == -1){
28              print("Server err")
29              exit()
30          }else{
```

Server shouldn`t be dead because of that while loop runs until user`s demand. Firstly ,server gets message from message queue.If there is a error about that process would dies.

```
30          }else{
31              for(int i = 0 ; i<newVersions.length ; i++){
32
33                  if(receivedMsg.mtext.substring(0,prgrmID.indexOf(' ')) == newVersions[i].substring(0,newVersions[i].indexOf(' '))){
34
35                      if(receivedMsg.mtext == newVersions[i]){
```

After client does not stick if block,it runs from else block. In for block,program scans every elements of array then first if block (line 33) compares between program id number in received message and newVersions`s array elements` program id number. If both of them are same,second if block (line 35) checks program id number , versions and names.

```
35                      if(receivedMsg.mtext == newVersions[i]){
36
37                          sendedMsg.mtext = newVersions[i]
38                          sendedMsg.mtype = newVersions[i].substring(0,newVersions[i].indexOf(' '))
39                          msgsnd(msgid , &sendedMsg , sendedMsg.mtext.length)
40
41                      }else{
```

If received message and newVersions`s text are same,Server sends message to message queue in order to client can receive message to run.Message includes same with client`s sended message but there is a diffrence. Message only can be received by client which sends message to message queue.

```
41                      }else{
42                          shmID = shmget(shmKey , 30)
43                          string *shared_data = (string *)shmat(shmID)
44                          *shared_data = newURLs[i];
```

If versions are not same,else block would be active. Server creates a shared memory then attaches shared memory`s adress to shared data.Lastly ,server puts client`s update URL in shared memory.

```
46                          sendedMsg.mtext = newVersions[i] + " " + shmID
47                          sendedMsg.mtype = newVersions[i].substring(0,newVersions[i].indexOf(' '))
48                          msgsnd(msgid , &sendedMsg , sendedMsg.mtext.length)
49
50                      }
51                      break
```

Finally,server modifes sendedMsg.Server puts new version of program and shared memory`s id then changes message type.Message type is client`s id number so client can understand which messages belongs client.Lastly,server sends message to message queue then runs break command because there is no need to scan rest.

4.Discussion

First of all,code is pseudo code. This code can be written in c programming lanugages with this logic and runs well.

4.1 Fixes and Improvements:

- New versions and URL`s can be stored in database.Also,server can call client`s data with APIs.It would be much safer.
- Datas can be stored binary search tree in order to maximize server`s speed.
- If there are lots of clients ,semaphores can be used in clients with multiple servers in order to maximize speed

5.Conclusion

As a result,report represents server-clients communication and information about message queue and shared memory with interprocces communication.Also report represents information about basic string functions in java. Finally,problem is solved with theoretical interprocess communication and pseudo codes.