|  |
| --- |
| Hanoi university of science and technology |
| Online Auction Report |
| Group: Nguyễn Hoàng Sơn & Tăng Thế Cường |
|  |
| **Instructor: Trần Nguyên Ngọc** |
| **12/5/2014** |

Table of Contents

[I. Requirement 3](#_Toc388369789)

[II. Program Design 3](#_Toc388369790)

[1. Sign up & Sign in 3](#_Toc388369791)

[1.1 Sign up 3](#_Toc388369792)

[1.2 Sign in 4](#_Toc388369793)

[2. Auction process on server 5](#_Toc388369794)

[3. Join auction & Bid on client 5](#_Toc388369795)

[4. Bidding History 6](#_Toc388369796)

[5. Deposit money to user account 7](#_Toc388369797)

[III. Technical Application 8](#_Toc388369798)

[1. General implementation 8](#_Toc388369799)

[2. Sign up & Sign in 8](#_Toc388369800)

[2.1 Sign up 8](#_Toc388369801)

[2.2 Sign in 10](#_Toc388369802)

[3. Auction process on server 11](#_Toc388369803)

[4. Join auction & Bid 13](#_Toc388369804)

[5. Bidding History 15](#_Toc388369805)

[6. Deposit money to user account 16](#_Toc388369806)

[IV. Screenshot 18](#_Toc388369807)

[V. Work Assignment 18](#_Toc388369808)

[VI. Reference 18](#_Toc388369809)

# Requirement

The online auction program must simulate all the required steps of a real auction except for the method of bidding which people does not have to actually join in person but to send the bidding money to server over network.

Server selects a goods for auction and start a session by sending goods info to all of the clients which are connecting to server.

There is 60-second countdown for the first user to place a bidding money that is higher than initial price plus minimum increment.

User can register an account in auction system.

User can log in and join an available auction.

User can join the auction and bid with money in his account.

User who joins the auction gets updated each time a new highest bidding price is made.

User can deposit more money to his account.

User can see his history of winning goods.

After a period of time, if the bidding price stands, owner of the bid is declared as a winner to everyone.

# Program Design

## Sign up & Sign in

### Sign up

User enters user name, password and password confirmation. If password and password confirmation do not match, an error message will appear.

User name and password are sent to server. If user name is already registered by another user, an error message will be returned to client interface.

Once user is registered successfully, he gets logged in automatically and can proceed with the next menu.



Fig.1: Signup

* Exception: Instead of CMD\_SIGNUP, server may respond:

1. SIG\_EXCEPTION: This username has already existed.

### Sign in

User enters user name and password and send to server.

If the user or password is not correct, an error message will be returned to client interface. If user is already logged in by another system, an error message will also be returned.

Once user logs in successfully, he can proceed with the next menu.



Fig.2: Signin

* Exception: Instead of CMD\_SIGNIN, server may respond:

1. SIG\_EXCEPTION: Incorrect username and password
2. SIG\_ALI: This user has already signed in.

## Auction process on server

Select a single goods for auction by entering its ordinal number in the selection list. Auction is started with 60-second countdown waiting for the first bidder to join the auction. After this 60 seconds, if no one joins in, the auction ends without any winner. Otherwise, another 60-second countdown is triggered to wait for the first bid to be made.

Once a legal bid is made, current price is broad-casted to all bidder and the 1st 20-second countdown starts. After 20 seconds, a message that announces 1st 20 seconds has gone by is sent to everyone joining the auction and 2nd 20-second countdown follows right after that. The same manner is applied after this next 20 seconds and the auction comes to the possible last 20 seconds. If the current bidding price stands until the end, the winner will be declared, account balance gets subtracted, current auction ends and the server is ready for a new session of auction.

During 3 phase of 20-second countdown, if any bid that breaks the record is made, the process is reseted and the auction continues with the 1st 20-second countdown.

During auction time, if user who has the highest bid quit the auction by any reason, the highest bid is then updated by the closest previous bid of online user.

## Join auction & Bid on client

When user is logged in, the information of currently auctioned goods is automatically updated and each time a new auction session is started, the corresponding information of the goods is displayed for user's awareness.

User chooses an option in the menu to join the auction. After confirming that he will join despite not being able to quit during the auction, he will instantly get an update of current bidding information of the goods and from this moment, user can make a bid for the goods.



Fig.3: Join auction

* Exception: Instead of CMD\_JOIN, server may respond:

1. CMD\_REJECT: The auction phase hasn't started yet. Unable to join.

If the bidding money is lower than the sum of initial price and minimum increment, user will get an error message. Otherwise, the bidding money is sent to server and soon get declared to all bidder as the highest bidding price at the time.



Fig.4: Bidding

* Exception: Instead of CMD\_BID, server may respond:

1. SIG\_NEM: User doesn't have enough money for that bid.
2. SIG\_LOWBID: User bid lower than expected amount.

Also, if user received 5 exceptions in a row, he/she would be considered as spamming and will be forced to quit the auction.

The time countdown of the auction is updated each 10 seconds in client side so user is always aware of the clock.

At the end of the auction, the winner of goods and the price are announced in one final message before user is back to menu.

## Bidding History

User chooses an option in the menu to display his bidding history. Then name of the goods he won, winning price and date&time of the victory are all shown in the screen.



Fig.5: History

At server side, admin can also view results of all auction had been hold.

## Deposit money to user account

User chooses an option in menu to begin depositing money. User enters serial id and send to the server.

If serial id is not found, an error message is returned to client. If the deposit is successful, a message that contains the amount of deposited money is displayed on client side and new account balance are updated and shown right after.



Fig.6: Deposit

* Exception: Instead of SIG\_SUCCESS, server may respond:

1. SIG\_FAIL: Incorrect serial.

# Technical Application

## General implementation

To handle the communication with many client, server is implemented as concurrent single server by using select() function.

A list of file descriptor of type fd\_set (allfds) is initially set with server file descriptor(server\_sockfd). File descriptor list (readfs) is assigned to allfds and passed to select() function to act as a list connection that the server will listen to.

|  |
| --- |
| FD\_ZERO(&readfds);  FD\_ZERO(&allfds);  FD\_SET(server\_sockfd,&allfds);  select(FD\_SETSIZE,&readfds,NULL,NULL,&timeout); |

Each time there is a new connection from client, an activity is triggered on server file descriptor. Then the connection is accepted by accept() function and the newly created client file descriptor is set into allfds for further listening.

|  |
| --- |
| client\_sockfd = accept(server\_sockfd, NULL, NULL);  FD\_SET(client\_sockfd,&allfds); |

From that point, any communication of the client will trigger an activity on its corresponding file descriptor. Ioctl with parameter FIONREAD will look into the buffer of connection that triggered activity to see if any data is transferred.

|  |
| --- |
| ioctl(sockfd,FIONREAD,&byte\_count); |

If there is no byte on the buffer, the connection is closed and server will respond accordingly to the status of client. Otherwise, a proper logical process will continue.

## Sign up & Sign in

### Sign up

User name, password and password confirmation are entered by user. Client program check if user name is empty. Password and password confirmation are compared to make sure they are matched to be able to send to server.

|  |
| --- |
| **while**(**1**){  printf("New Username:"); scanf("%s",temp.name);  **if**(strcmp(temp.name,"")==**0**){  printf("Username must not be empty**\n**");  } **else** {  **break**;  }  }  **while**(**1**){  printf("Password:"); scanf("%s",temp.password);  printf("Password Confirmation:"); scanf("%s",password);  **if**(strcmp(temp.password,password)!=**0**){  printf("Password and confirmation do not match.**\n**");  } **else** {  **break**;  }  } |

Firstly, a command called CMD\_REGISTER is sent to the server to tell it that a request to register is coming up. Then user name and password are sent to server.

|  |
| --- |
| command = CMD\_REGISTER;  write(sockfd,&command,**sizeof**(**int**)); |

Server reads the user name and password that client sends. Open user.txt to check if user name already existed. If user name is legal, a new line of user name and password is appended to the end of user.txt.

|  |
| --- |
| **int** **registerUser**(User\* user){  **FILE** \*file;  **if**(getUserByName(user->name) != NULL){  **return** **0**;  }  **if**((file = fopen("users.txt","a"))==NULL) {  printf("Error opening file!**\n**");  **return** -**1**;  }  fprintf(file,"%s**\t**%s**\t**%d**\n**",user->name,user->password,user->balance);  fclose(file);  **return** **1**;  } |

Then CMD\_REGISTER command is sent back to client as a signal of registering successfully. Otherwise, SIG\_EXCEPTION is sent as a mark of error happened.

Once the user is registered, he is also get logged in by changing the status of corresponding element in users array which has client file descriptor as index to STT\_ONLINE.

|  |
| --- |
| users[i].status = STT\_ONLINE;  command = CMD\_REGISTER;  write(i,&command,**sizeof**(**int**)); |

After that, user's balance and information of current goods are sent to client.

|  |
| --- |
| write(i,&users[i].balance,**sizeof**(**int**));  write(i,getGoodsinfo(),**sizeof**(**char**)\***100**); |

### Sign in

After user enters user name and password, a command called CMD\_SIGNIN is sent to server as a signal of upcoming log in information. Then user name and password are sent over to server to authenticate.

|  |
| --- |
| command = CMD\_SIGNIN;  write(sockfd,&command,**sizeof**(**int**));  write(sockfd,&user,**sizeof**(User)); |

User name and password are checked in user.txt for their validity.

|  |
| --- |
| **int** **authenticate**(User \*user){  User\* temp;  temp = getUserByName(user->name);  **if**(temp==NULL) **return** **0**;  **if**(strcmp(user->password,temp->password)==**0**){  user->balance = temp->balance;  **return** **1**;  }  **return** **0**;  } |

If they are correct, another check will be launched by looking at the status of every online user to make sure that this user is not currently logged in.

|  |
| --- |
| **int** **isLoggedIn**(**char**\* name){  **int** i;  **for**(i = **0**; i<FD\_SETSIZE; i++){  **if**(strcmp(name,users[i].name)==**0**)  **return** **1**;  }  **return** **0**;  } |

If user name and password pass these two checks, user's status will be set as STT\_ONLINE in users array.

|  |
| --- |
| users[i].status = STT\_ONLINE; |

In case of successful log in, CMD\_SIGNIN is sent back to client. Then balance and goods info are also get sent.

|  |
| --- |
| command = CMD\_SIGNIN;  write(i,&command,**sizeof**(**int**));  write(i,&users[i].balance,**sizeof**(**int**));  write(i,getGoodsinfo(),**sizeof**(**char**)\***100**); |

In case of incorrect user name or password, SIG\_EXCEPTION is sent to client.

|  |
| --- |
| command = SIG\_EXCEPTION; |

In case of already logged in user, SIG\_ALI is sent to client.

|  |
| --- |
| command = SIG\_ALI; |

## Auction process on server

To simulate countdown, timeout period of select() function is set to 1 second.

|  |
| --- |
| timeout.tv\_sec = **1**;  timeout.tv\_usec = **1000**; |

Each time select() timeout, time is minus by 1.

|  |
| --- |
| **if**(retval == **0**){ //Signal timeout  **if**(countdown != **0**){  countdown--;  }  } |

State of the auction is stored in auction\_state variable.

|  |
| --- |
| auction\_state = **0** //default  auction\_state = **1** //After the first user join auction  auction\_state = **2** //During 1st 20-second of bid  auction\_state = **3** //During 2nd 20-second of bid  auction-state = **4** //During 3rd 20-second of bid |

At first, there are 60 seconds countdown for the first user to join in. When server receives the first CMD\_JOIN command, 60 second for first bid begins.

|  |
| --- |
| **if**(auction\_state == **0** && user\_bidding == **1**){  auction\_state = **1**;  printf("Start 60s auction:**\n**");  countdown = **60**;  } |

Each time a legal bid is made, server receives CMD\_BID command and 1st 20-second countdown is started.

|  |
| --- |
| auction\_state = **2**;  countdown = **20**;  broadcast(line); |

At the end of 1st 20-second, time is reseted to a new 20-second phase.

|  |
| --- |
| countdown = **20**;  auction\_state = **3**;  broadcast(line); |

At the end of 2nd 20-second, time is reseted to a new 20-second phase.

|  |
| --- |
| countdown = **20**;  auction\_state = **4**;  broadcast(line); |

Finally, if the price stands, winner is declared, user balance is updated and log history is written.

|  |
| --- |
| auction\_state = **0**;  broadcast(line);  updateUser(winner);  writeLog(winner.name,goods.name,goods.curbroadcastEnd(logging[log\_number-**1**].client);\_price,replace(ctime(&now),' ','.'));  broadcastEnd(logging[log\_number-**1**].client); |

## Join auction & Bid

Client user select() function to receive data from server and user at the same time.

|  |
| --- |
| FD\_ZERO(&readfds);  FD\_ZERO(&allfds);  FD\_SET(sockfd,&allfds);  FD\_SET(STDIN,&allfds);  select(FD\_SETSIZE,&readfds,NULL,NULL,NULL); |

CMD\_JOIN command is sent to server as a signal to join auction.

|  |
| --- |
| command = CMD\_JOIN;  write(sockfd,&command,**sizeof**(**int**)); |

Then client read returned signal, current price and goods information from server

|  |
| --- |
| read(sockfd,&command,**sizeof**(**int**));  read(sockfd,&price,**sizeof**(**int**));  read(sockfd,line,**sizeof**(line)); |

If client receives data from server, data is displayed or the auction is end.

|  |
| --- |
| read(sockfd,line,**sizeof**(**char**)\***100**);  **if**(strlen(line) > **1**){  printf("%s",line);  } **else**{  command = atoi(line);  **if**(command == CMD\_END){  printf("End of auction**\n**");  **break**;  } **else** **if**(command == CMD\_WINNER){  read(sockfd,&user.balance,**sizeof**(**int**));  printf("End of auction**\n**");  **break**;  }  } |

If user enters bidding price, it is checked if it is bigger than current price and smaller than balance.

|  |
| --- |
| **if**(bid\_money > users[i].balance){  command = SIG\_NEM;  // Account not having enough money to bid  write(i,&command,**sizeof**(**int**));  } **else** **if** ((bid\_money>= (goods.cur\_price+goods.min\_incr))||((bid\_money >= goods.init\_price)&&(log\_number == **0**))){  **char** line[**100**];  //Set current price, logging and auction state  goods.cur\_price = bid\_money;  logging[log\_number].client = i;  logging[log\_number++].bid\_money = bid\_money;  auction\_state = **2**;  sprintf(line,"**\r**%s had bid %d USD for %s**\n\n\n**",users[i].name,bid\_money,goods.name);  printf("%s",line);  command = CMD\_BID;  write(i,&command,**sizeof**(**int**));  broadcast(line);  //Set 1st countdown  countdown = **21**;  } **else** {  command = SIG\_LOWBID;  // Bid less than current price + increment  write(i,&command,**sizeof**(**int**));  } |

## Bidding History

After user chooses History option on the menu, CMD\_HISTORY command is sent to server.

|  |
| --- |
| command = CMD\_HISTORY;  write(sockfd,&command,**sizeof**(**int**)); |

Server receives the command then open log.txt, read the file line by line and check if each line belongs to the requesting user. All the log history of the user will be constructed as an string for returning to the client.

|  |
| --- |
| **char**\* **getHistory**(**char**\* user\_name){  **if**((file = fopen("log.txt","r"))==NULL) {  printf("Error opening file!**\n**");  **return** NULL;  }  sprintf(history,"Goods **\t**Price **\t**Date&Time**\n**");  **while**(fscanf(file,"%s %s %s %s",name,goods\_name,price,datetime)!= EOF){  **if**(strcmp(name,user\_name)==**0**){  datetime = replace(datetime,'.',' ');  strcat(price," USD");  sprintf(line,"%-30s**\t**%-10s**\t**%s**\n**",goods\_name,price,datetime);  strcat(history,line);  }  } |

Client receives the result string and displays on screen.

When admin want to view the history of all auctions has been held, he can choose at the main menu. From then, the process is much the same as above, with a little different at presenting:

|  |
| --- |
| **void** **viewHistory**()  {  **FILE** \*fi;  **char** line[**100**];  **if**((fi = fopen("log.txt","r"))==NULL) {  printf("Error opening auction history file!**\n**");  **return**;  }  printf("**\n\n**");  printf("\*\*\*\* Auction History \*\*\*\*");  printf("**\n\n**");  **while** (fgets(line,**40**,fi)>**0**) {  printf("%s",line);  }  printf("**\n\n**");  fclose(fi);  } |

## Deposit money to user account

User chooses Deposit Money option on the menu and then enter serial.

CMD\_DEPOSIT command and serial are respectively sent to server.

|  |
| --- |
| command = CMD\_DEPOSIT;  write(sockfd,&command,**sizeof**(**int**));  write(sockfd,serial,**sizeof**(**char**)\***100**); |

File serial.txt is read line by line to check if the serial is matched. Account balance of user will be added with an amount corresponding to the serial.

|  |
| --- |
| **int** **deposit**(User user,**char**\* serial){  **if**((file = fopen("serial.txt","r"))==NULL) {  printf("Error opening file!**\n**");  **return** **0**;  }  **while**(fscanf(file,"%s %d",str,&money)!= EOF){  **if**(strcmp(serial,str)==**0**){  flag = **1**;  **break**;  }  }  fclose(file);  **if**(!flag){  **return** **0**;  }  user.balance += money;  updateUser(user);  **return** money;  } |

In case of successful deposit, SIG\_SUCCESS is returned to client. Otherwise, SIG\_FAIL is sent back.

|  |
| --- |
| **if**(money>**0**){  command = SIG\_SUCCESS;  users[i].balance += money;  write(i,&command,**sizeof**(**int**));  write(i,&money,**sizeof**(**int**));  } **else** {  command = SIG\_FAIL;  write(i,&command,**sizeof**(**int**));  } |

On client side, a suitable message is displayed according to the returned signal.

|  |
| --- |
| **if**(command == SIG\_SUCCESS){  read(sockfd,&money,**sizeof**(**int**));  printf("%dUSD has been deposited**\n**",money);  user.balance += money;  } **else** **if**(command == SIG\_FAIL){  printf("Your serial ID is not correct**\n**");  } |

# Screenshot

# Work Assignment

|  |  |
| --- | --- |
| **Nguyễn Hoàng Sơn** | **Tăng Thế Cường** |
| * Build core of program | * Design program's workflow |
| * Register/Login Coding | * Implement admin's utilities |
| * Join/Bidding Coding | * Handle program exceptions |
| * Log bidding history Coding | * Fix various bugs |
| * Deposit money Coding | * Tuning interface |
| * Report writing | * Report writing |

# Reference

[1] Unix Network Programming volum 1 – W. Reachard Stevens , Bin Fenner , Andrew M. Redoff