**CMPE 207 Homework #3**

Q1. [30 points] Study **conditional variable locks**. First, **find one example code** that may have a race condition problem in the multi-threading environment through textbooks or reference books or website. Don’t use the same example codes that I gave already. Second, **fix the found example code with the conditional variable locks.** Explain the identified problem in the example code and explain your solution in the code. Please understand the logical concept of the conditional variable locks carefully.

1. Your code for race conditions with explanation.

Ans. Following code in c represent race condition.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

int mulFactor;

void \*PrintMulTable()

{

int i;

for(i=0; i<11;i++){

printf("%d",i\*mulFactor);

}

printf("\n");

}

void \*FetMulFac()

{

int i;

for(i=0; i<100000;i++);

printf("please enter multiplication Factor \n");

scanf("%d", &mulFactor);

}

int main (int argc, char \*argv[])

{

pthread\_t threads[2];

pthread\_create(&threads[0], NULL, PrintMulTable, NULL);

pthread\_create(&threads[1], NULL, FetMulFac, NULL);

pthread\_join(threads[0], NULL);

pthread\_join(threads[1], NULL);

printf ("Multiplication Table Printed \n");

pthread\_exit(NULL);

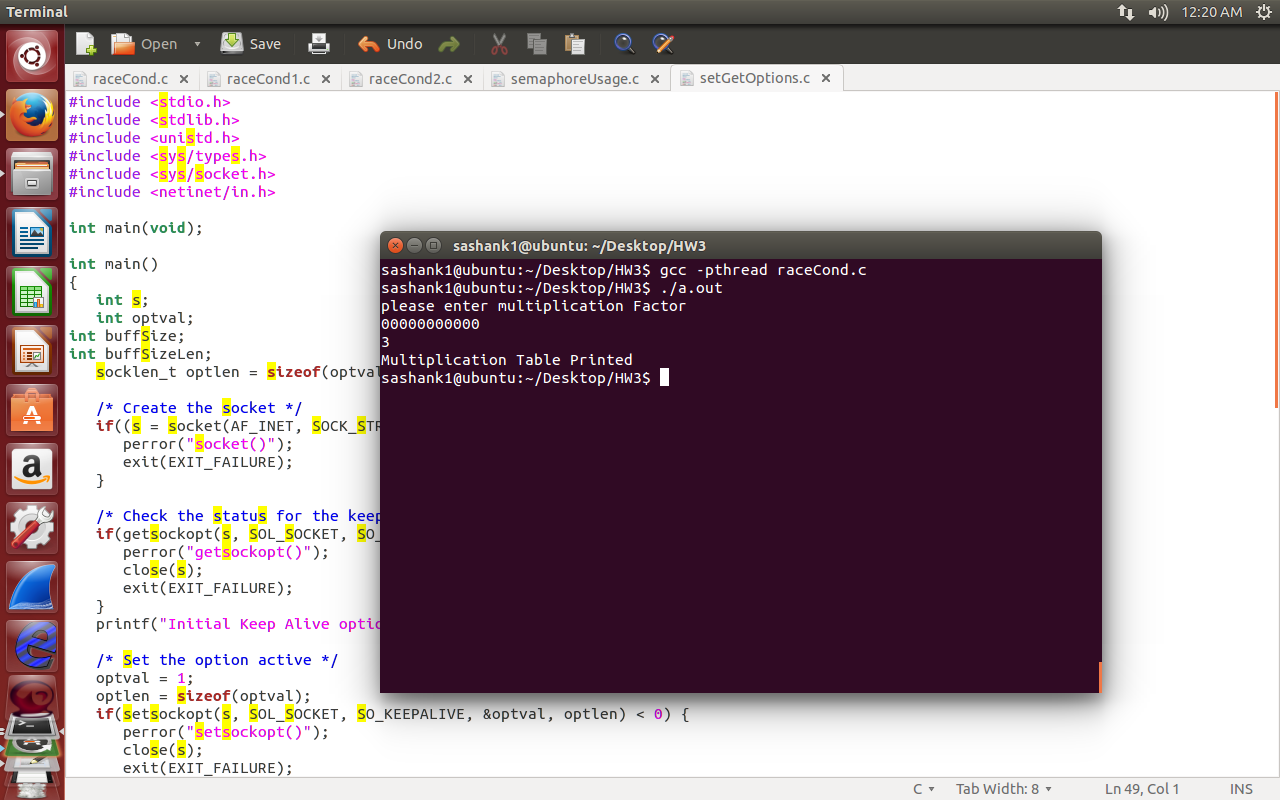
return 0;

}

In the above code multiplication factor is considered to be shared resource. The integer “**mulFactor**” in the above code is accessed by both **FetMulFac** and **PrintMulTable** threads.

As per logic, **PrintMulTable** should wait till the **mulFactor** is fetched from the user through **FetMulFac** thread. Based on the mulfactor corresponding multiplication table is displayed. But due to no proper synchronization, both threads run at same time and **PrintMulTable** prints all zeros before fetching multiplication factor from user. Following screen shot depicts the scenario.

Screen Shot for Race Condition



1. Your fixed code to solve race condition with explanation (how to solve).

**Ans.** As a solution to above race condition, I have used condition variable to properly synchronize between threads. The conditional variable makes **PrintMulTable**  to wait till multiplication factor variable to be greater than zero. It is assumed that if **mulFactor** is greater than zero then it is fetched from the user.

Following code shows how above scenario is solved using conditional variable.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

int mulFactor;

pthread\_mutex\_t lock;

pthread\_cond\_t lockCond;

void \*PrintMulTable()

{

pthread\_mutex\_lock(&lock);

while(mulFactor==0)

pthread\_cond\_wait(&lockCond,&lock);

int i;

for(i=0; i<11;i++){

printf("%d",i\*mulFactor);

}

printf("\n");

pthread\_mutex\_unlock(&lock);

}

void \*FetMulFac()

{

pthread\_mutex\_lock(&lock);

int i;

printf("please enter multiplication Factor \n");

int enteredVal;

scanf("%d", &enteredVal);

printf("enterd multiplication Factor is %d \n",enteredVal);

mulFactor=enteredVal;

pthread\_cond\_signal(&lockCond);

pthread\_mutex\_unlock(&lock);

}

int main (int argc, char \*argv[])

{

pthread\_t threads[2];

if (pthread\_mutex\_init(&lock, NULL) != 0)

{

printf("\n mutex init failed\n");

return 1;

}

if (pthread\_cond\_init(&lockCond, NULL) != 0)

{

printf("\n conditional variable init failed\n");

return 1;

}

pthread\_create(&threads[0], NULL, PrintMulTable, NULL);

pthread\_create(&threads[1], NULL, FetMulFac, NULL);

pthread\_join(threads[0], NULL);

pthread\_join(threads[1], NULL);

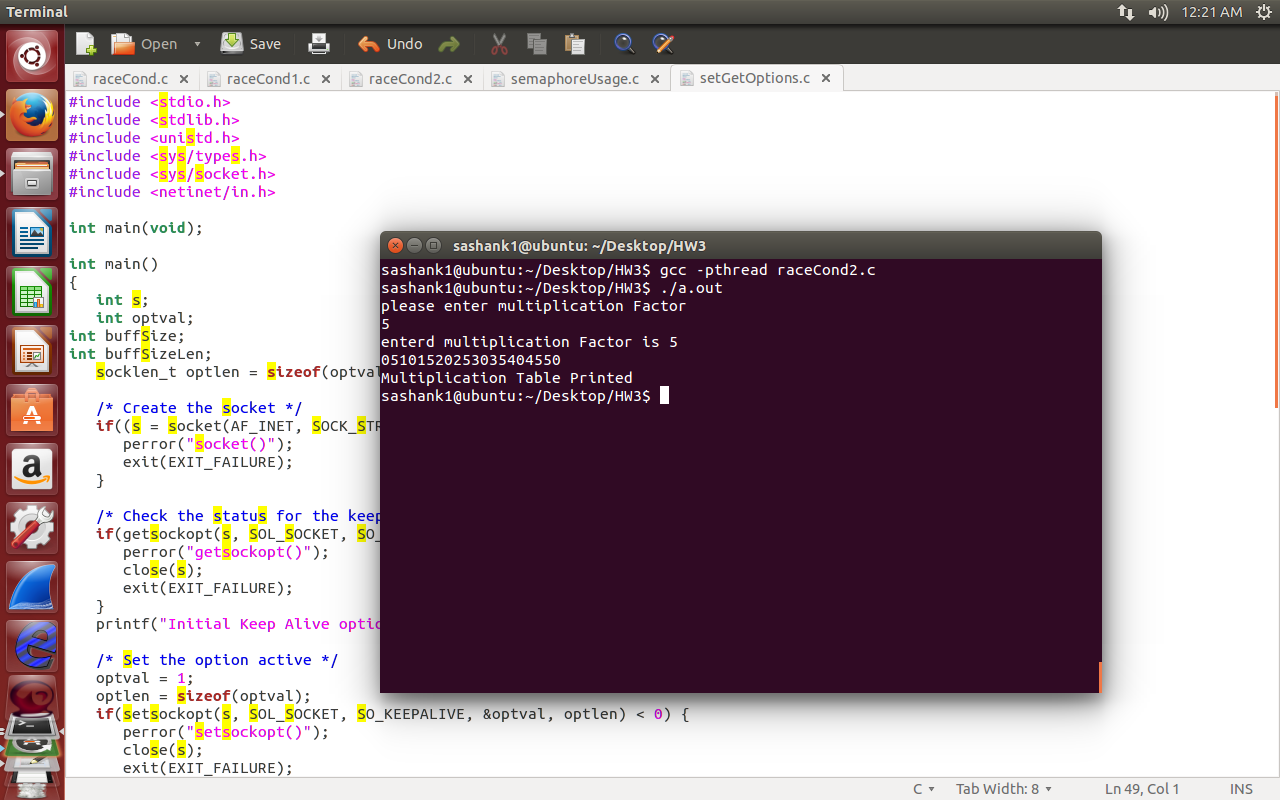
printf ("Multiplication Table Printed \n");

pthread\_exit(NULL);

return 0;

}

**Following screen shot depicts solving using conditional variable.**



Q2. [20 points] Please solve your race condition problem in Q1 by using a **semaphore.**

**Ans.**  The race condition in the described above can also be solved using semaphores. We can make **PrintMulTable**  to wait till we get input for mulFactor using sem\_wait. Once we get the input from user, we send signal to print multiplication table by using sem\_post().

Following code depicts the code utilized to solve the race condition\

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

int mulFactor;

sem\_t testSem;

void \*PrintMulTable()

{

sem\_wait(&testSem);

int i;

for(i=0; i<11;i++){

printf("%d",i\*mulFactor);

}

printf("\n");

}

void \*FetMulFac()

{

int i;

printf("please enter multiplication Factor \n");

int enteredVal;

scanf("%d", &enteredVal);

printf("enterd multiplication Factor is %d \n",enteredVal);

mulFactor=enteredVal;

sem\_post(&testSem);

}

int main (int argc, char \*argv[])

{

pthread\_t threads[2];

if (sem\_init(&testSem,0,0) != 0)

{

printf("\n mutex init failed\n");

return 1;

}

pthread\_create(&threads[0], NULL, PrintMulTable, NULL);

pthread\_create(&threads[1], NULL, FetMulFac, NULL);

pthread\_join(threads[0], NULL);

pthread\_join(threads[1], NULL);

printf ("Multiplication Table Printed \n");

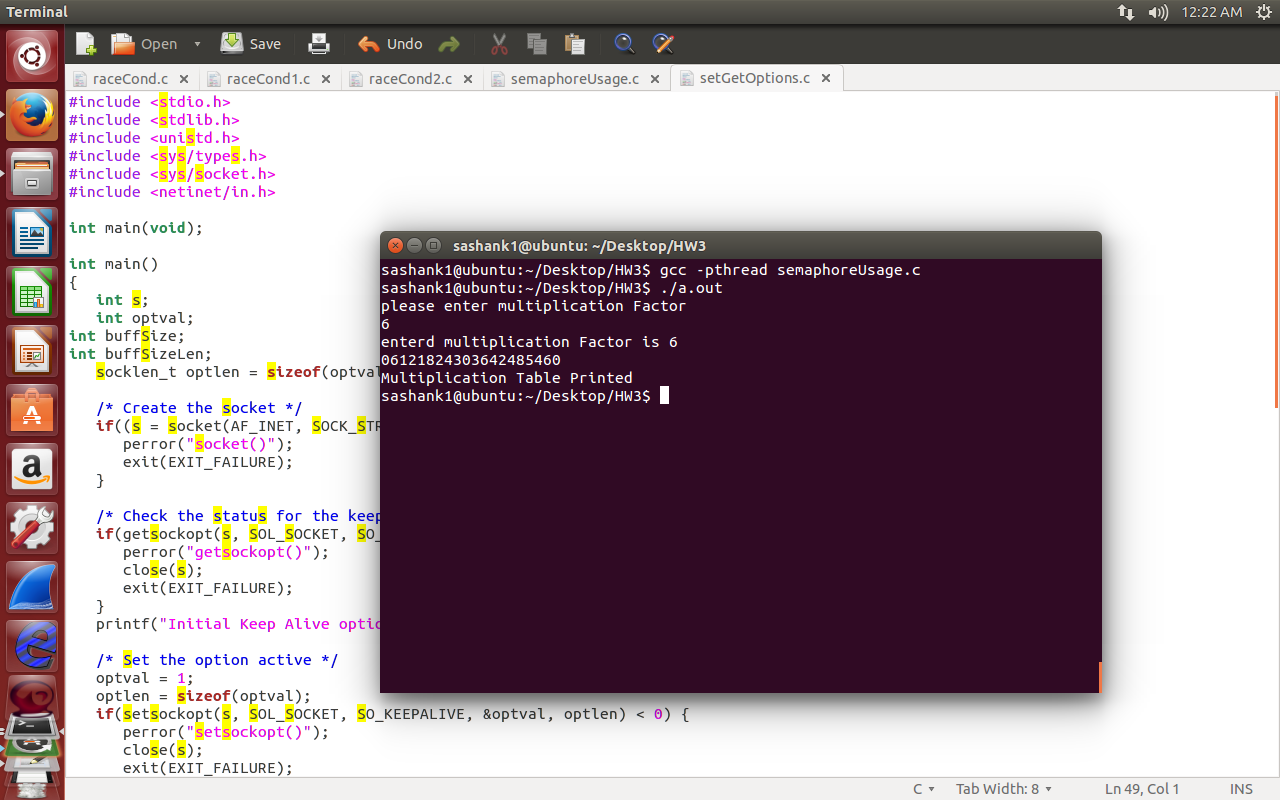
sem\_destroy(&testSem);

pthread\_exit(NULL);

return 0;

}

Following screen shot depicts the multiplication table after solving race condition using semaphores.



Q3. [30 points] By using “getsockoption()”, describe your current socket options by default in your local system. After identifying your default socket options, modify two socket options by using “setsockoption()”. For example, write a program that prints the default TCP and UDP send and receive buffer sizes and run it on the systems to which you have access. You can choose any other options available in your local system. Note that some options cannot be changed due to restrictions in Kernel.

Ans. Following is the code used to fetch and change socket level options of my current machine.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

int main(void);

int main()

{

int s;

int optval;

int buffSize;

int buffSizeLen;

socklen\_t optlen = sizeof(optval);

/\* Create the socket \*/

if((s = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket()");

exit(EXIT\_FAILURE);

}

/\* Check the status for the keepalive option \*/

if(getsockopt(s, SOL\_SOCKET, SO\_KEEPALIVE, &optval, &optlen) < 0) {

perror("getsockopt()");

close(s);

exit(EXIT\_FAILURE);

}

printf("Initial Keep Alive option %s\n", (optval ? "ON" : "OFF"));

/\* Set the option active \*/

optval = 1;

optlen = sizeof(optval);

if(setsockopt(s, SOL\_SOCKET, SO\_KEEPALIVE, &optval, optlen) < 0) {

perror("setsockopt()");

close(s);

exit(EXIT\_FAILURE);

}

printf("Keep alive option is set\n");

/\* Check the status again \*/

if(getsockopt(s, SOL\_SOCKET, SO\_KEEPALIVE, &optval, &optlen) < 0) {

perror("getsockopt()");

close(s);

exit(EXIT\_FAILURE);

}

printf("Keep Alive option after setting %s\n", (optval ? "ON" : "OFF"));

// check value of send buffer value

if(getsockopt(s, SOL\_SOCKET, SO\_SNDBUF, &buffSize, &buffSizeLen) < 0) {

perror("getsockopt()");

close(s);

exit(EXIT\_FAILURE);

}

printf(" Initial buffer size %d\n", buffSize);

/\* Set the option active \*/

buffSize=2048;

buffSizeLen=sizeof(buffSize);

if(setsockopt(s, SOL\_SOCKET, SO\_SNDBUF, &buffSize, buffSizeLen) < 0) {

perror("setsockopt()");

close(s);

exit(EXIT\_FAILURE);

}

printf("Buffer value is set to 2048 \n");

/\* Check the status again \*/

if(getsockopt(s, SOL\_SOCKET, SO\_SNDBUF, &buffSize, &buffSizeLen) < 0) {

perror("getsockopt()");

close(s);

exit(EXIT\_FAILURE);

}

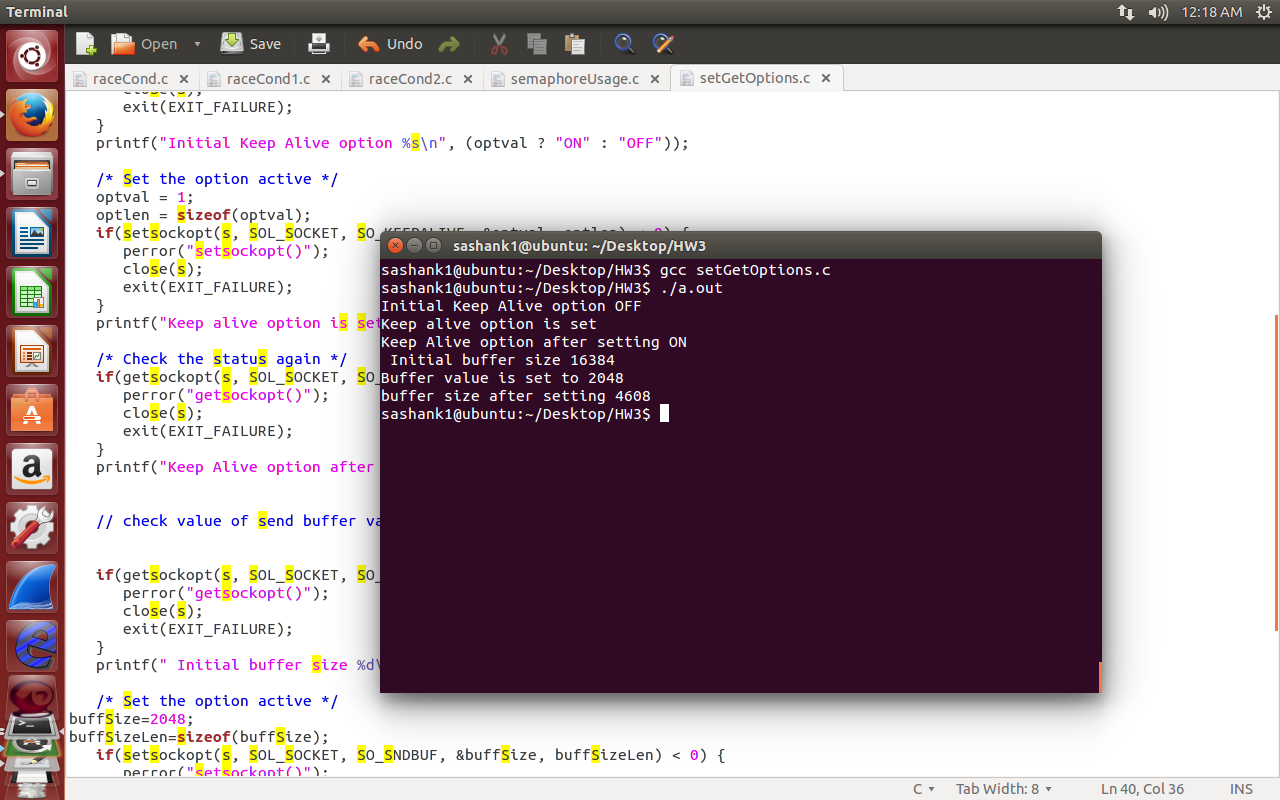
printf("buffer size after setting %d\n", buffSize);

close(s);

exit(EXIT\_SUCCESS);

}

Using above code, I have modified SO\_SNDBUF and SO\_KEEPALIVE options on socket level. For fetching the socket value I have used getsockopt and for setting I have used setsockopt subroutines. Following screen shot depicts fetching and modifying options.



As I have set send buffer to 2048, by default it will be doubled and assigned to value of 4608.

Q4. [20 points] Please modify your TCPdaytime program to work with either IPv4 or IPv6 after choosing one of the Daytime programs in your homework assignments. Or, you can modify the example of textbook or the reference book (Figure 11.4) to work with IPv4 or IPv6.

**Ans.** Following code is supported by both IP4 and IP6 connections.

#include <stdio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <sys/types.h>

#include <time.h>

void error(char \*err){

printf("%s", err);

exit(1);

}

int main(int argc, char \* argv[]){

if(argc<2){

error("Please pass port number.\n");

}

int socketfd,newSocketfd,portNo,cliLen;

char buffer[255];

struct sockaddr\_in6 serv\_addr, cli\_addr;

int n;

socketfd = socket(AF\_INET6, SOCK\_STREAM,0);

if(socketfd<0){

error("Error in creating a socket.\n");

}

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

//portNo=atoi("8000");

serv\_addr.sin6\_family= AF\_INET6;;

serv\_addr.sin6\_addr = in6addr\_any;;

serv\_addr.sin6\_port =htons(atoi(argv[1]));

if(bind(socketfd,(struct sockaddr\*)&serv\_addr,sizeof(serv\_addr)) <0){

error("Error on binding socket.\n");

}

listen(socketfd,5);

printf("Server is on and started listing to client.\n");

cliLen = sizeof(cli\_addr);

newSocketfd = accept(socketfd,(struct sockaddr \*) &cli\_addr, (socklen\_t\*) &cliLen );

if(newSocketfd<0){

error("Error on accepting to client.");

}

bzero(buffer,255);

time\_t t = time(NULL);

struct tm \*tm = localtime(&t);

char s[64];

strftime(s, sizeof(s), "%c", tm);

n = write(newSocketfd, s,64);

if(n<0){

error("Error on writing to the client.");

}

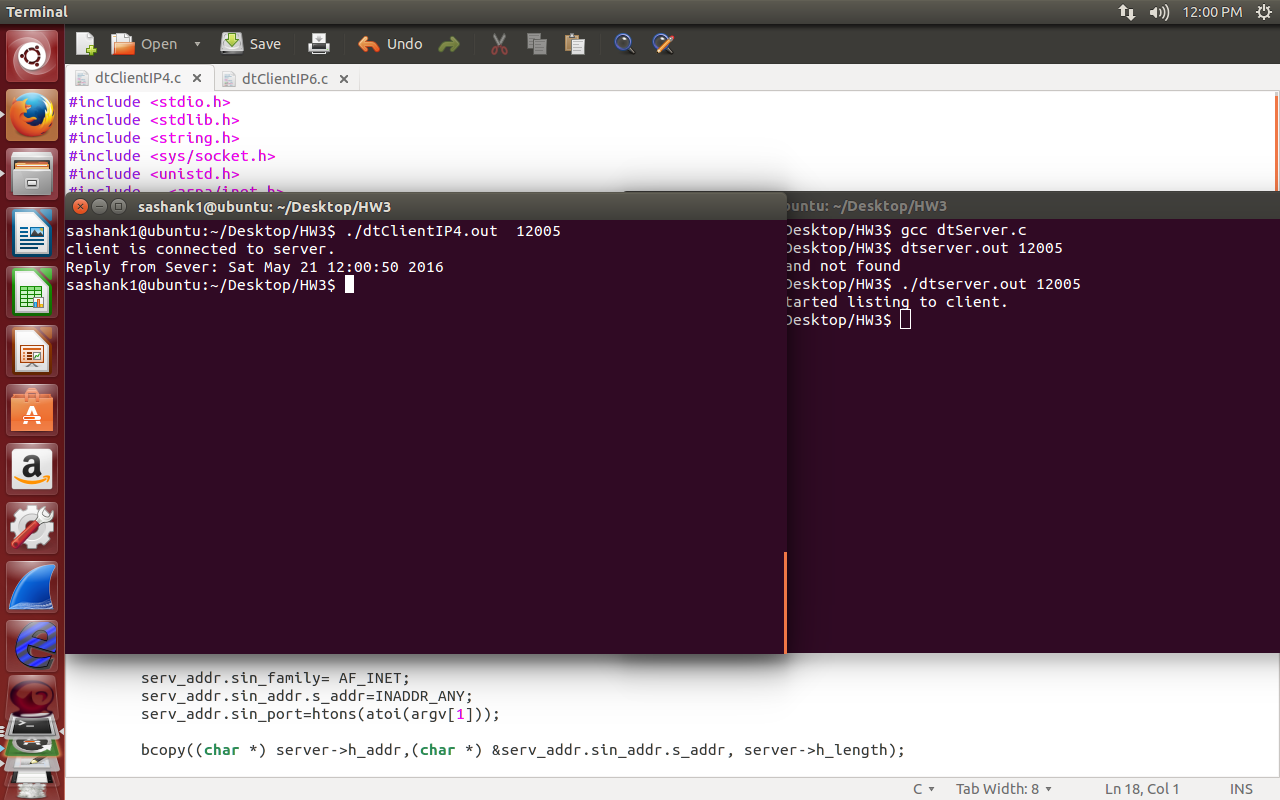
return 0;

}

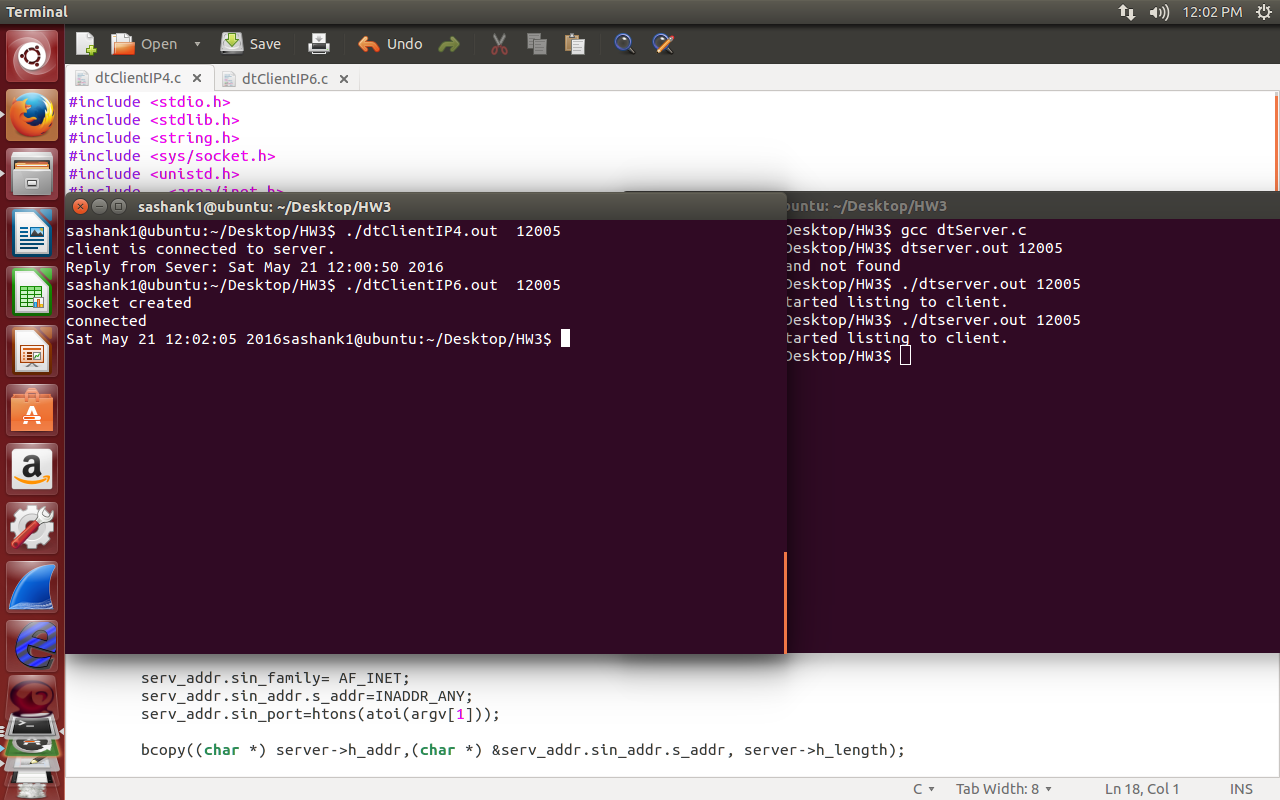
From the above code,

in6addr\_any states that the server accepts both ipv4 and ipv6

Following screen shots depicts both scenarios



Above image is for IPV4



Above image is for IPV6