CMPE 207 Homework #3
Due: May 21 at 11:59 p.m.
Name: Aakash Shah
SJSU ID: 010767354
Class ID: 64

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

Answer:

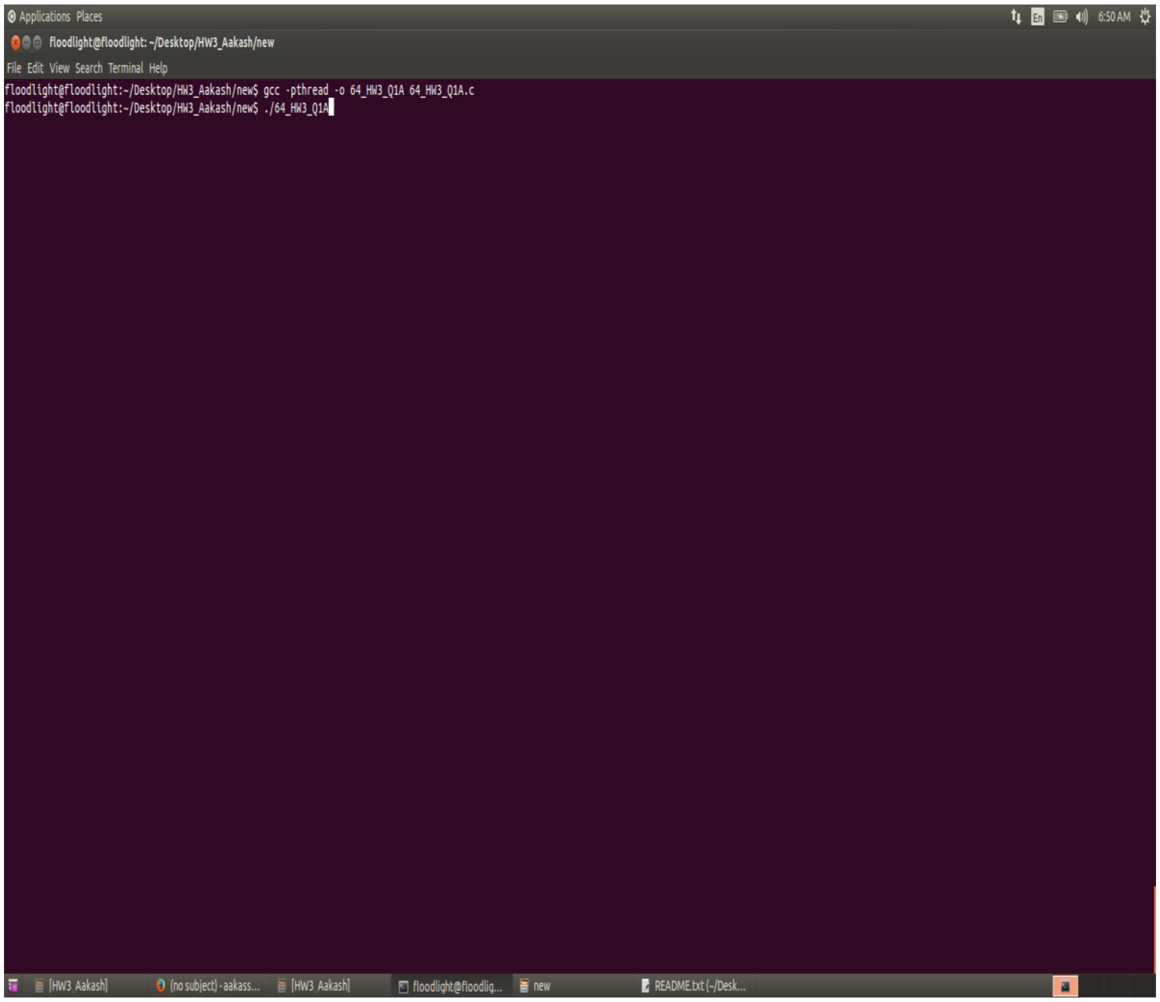
1. Race condition occurs when multiple threads are able to access the shared data and try to change the data at the same time.

   Below is the c program source code and screenshot which demonstrates the race condition. The sequence of the program follows like this:
   - First, two threads are created.
   - One thread generates the numbers and writes to the buffer, other thread reads the data from the buffer at the same time.
   - This results in race condition which ultimately changes the values. This is known as race condition.

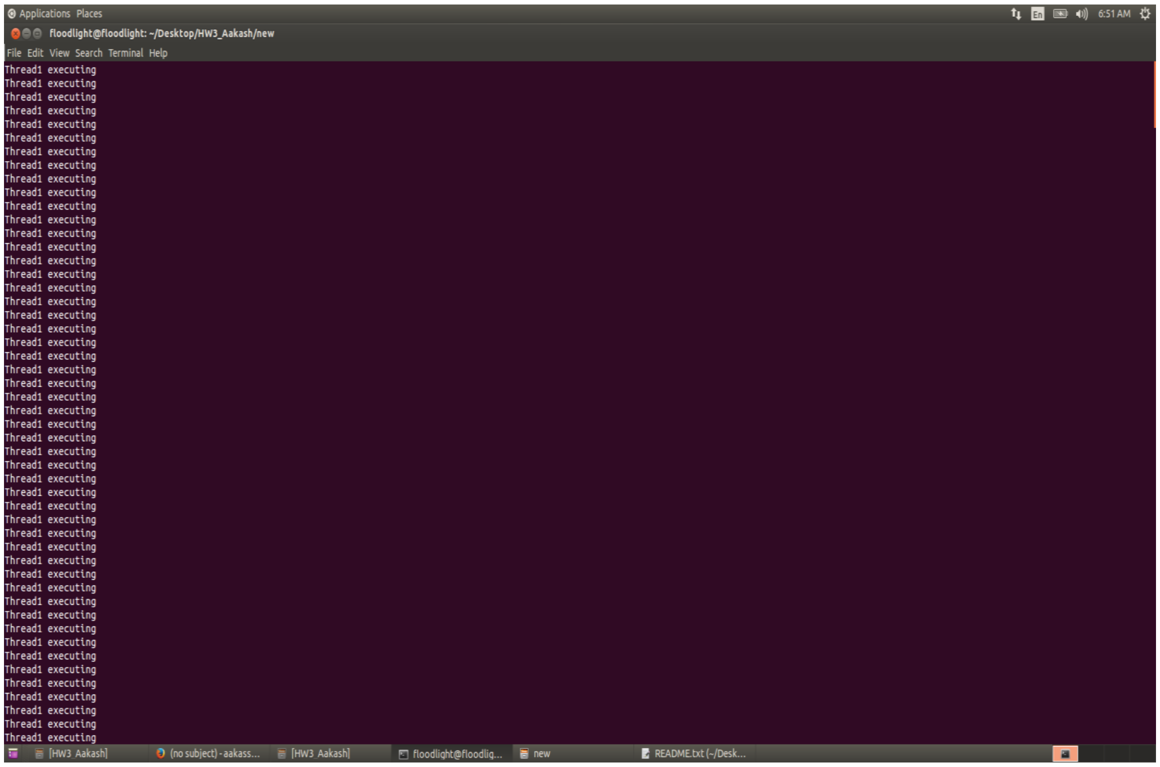
```
/* Code with Race Condition problem - q1_race_condition.c */

#include <pthread.h>
#include <stdio.h>
#define SIZE 4
#define ITEMS 5
#define THREADS 2

typedef struct
{
  char buf[SIZE];
  int occupied;
  int in,out;
} buffer_t;
```

```
buffer_t buffer;

void * thread1(void *);
void * thread2(void *);
pthread_t tid[THREADS];

int main( int argc, char *argv[] )
{
        int i;
        pthread_create(&tid[1], NULL, thread2, NULL);
        pthread_create(&tid[0], NULL, thread1, NULL);

        for ( i = 0; i < THREADS; i++)
                pthread_join(tid[i], NULL);
        return 0;
}

void * thread1(void * parm)
{
        char item[ITEMS]="AKASH";
        int i;
        printf("Thread1 is started\n");

        for(i=0;i<ITEMS;i++)
        {
                if (item[i] == '\0')
                        break;
                if (buffer.occupied >= SIZE)
                        printf("Thread1 waiting\n");
                while (buffer.occupied >= SIZE)
                        printf("Thread1 executing\n");

                buffer.buf[buffer.in++] = item[i];
                buffer.in %= SIZE;
                buffer.occupied++;
        }
        printf("Thread1 exiting\n");
        pthread_exit(0);
}

void * thread2(void * parm)
{
        char item;
        int i;
        printf("Thread2 is started\n");
        for(i=0;i<ITEMS;i++)
        {
                if (buffer.occupied <= 0)
                        printf("Thread2 waiting\n");
                while(buffer.occupied <= 0)
                        printf("Thread2 executing\n");
                item = buffer.buf[buffer.out++];
                printf("%c\n",item);
                buffer.out %= SIZE;
                buffer.occupied--;
        }
        printf("Thread2 exiting\n");
        pthread_exit(0);
}
```

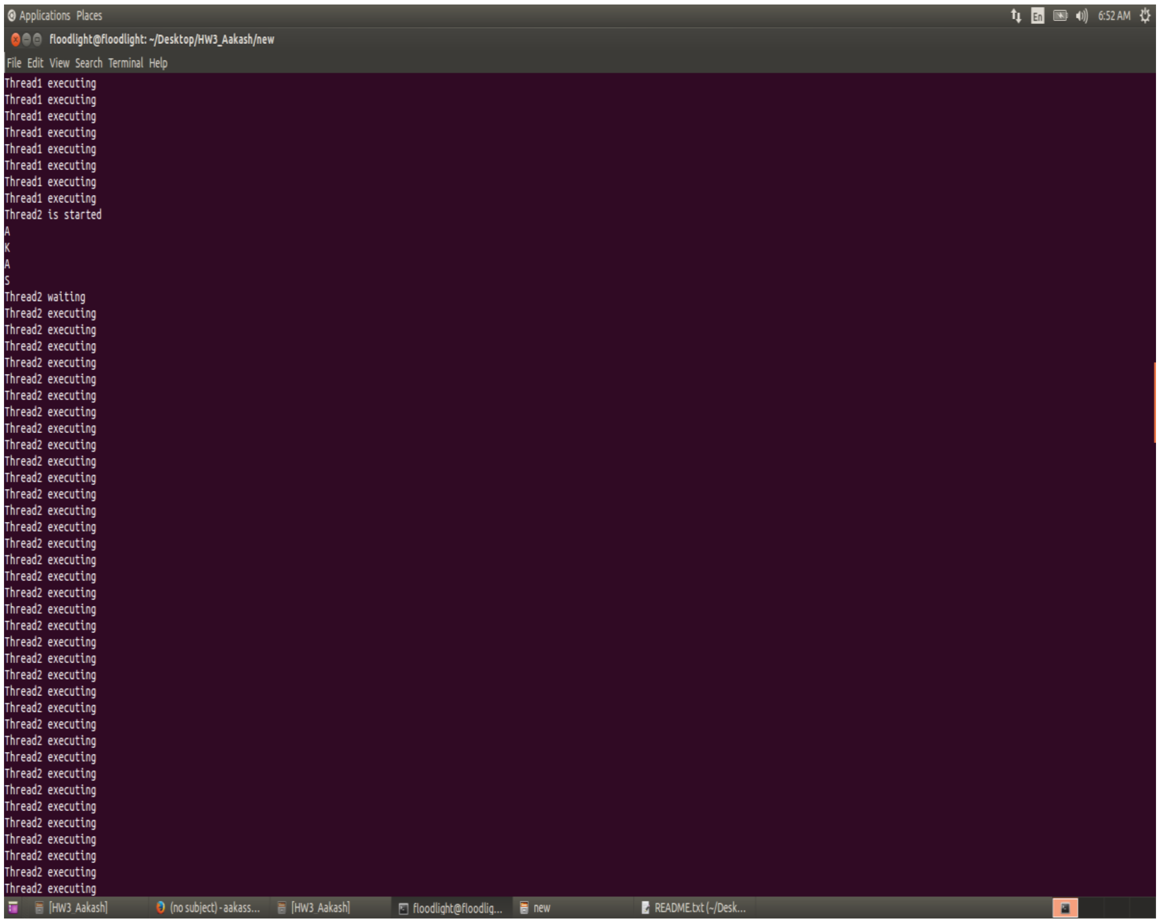
Screenshots:


Applications Places
floodlight@floodlight: ~/Desktop/HW3_Aakash/new
File Edit View Search Terminal Help

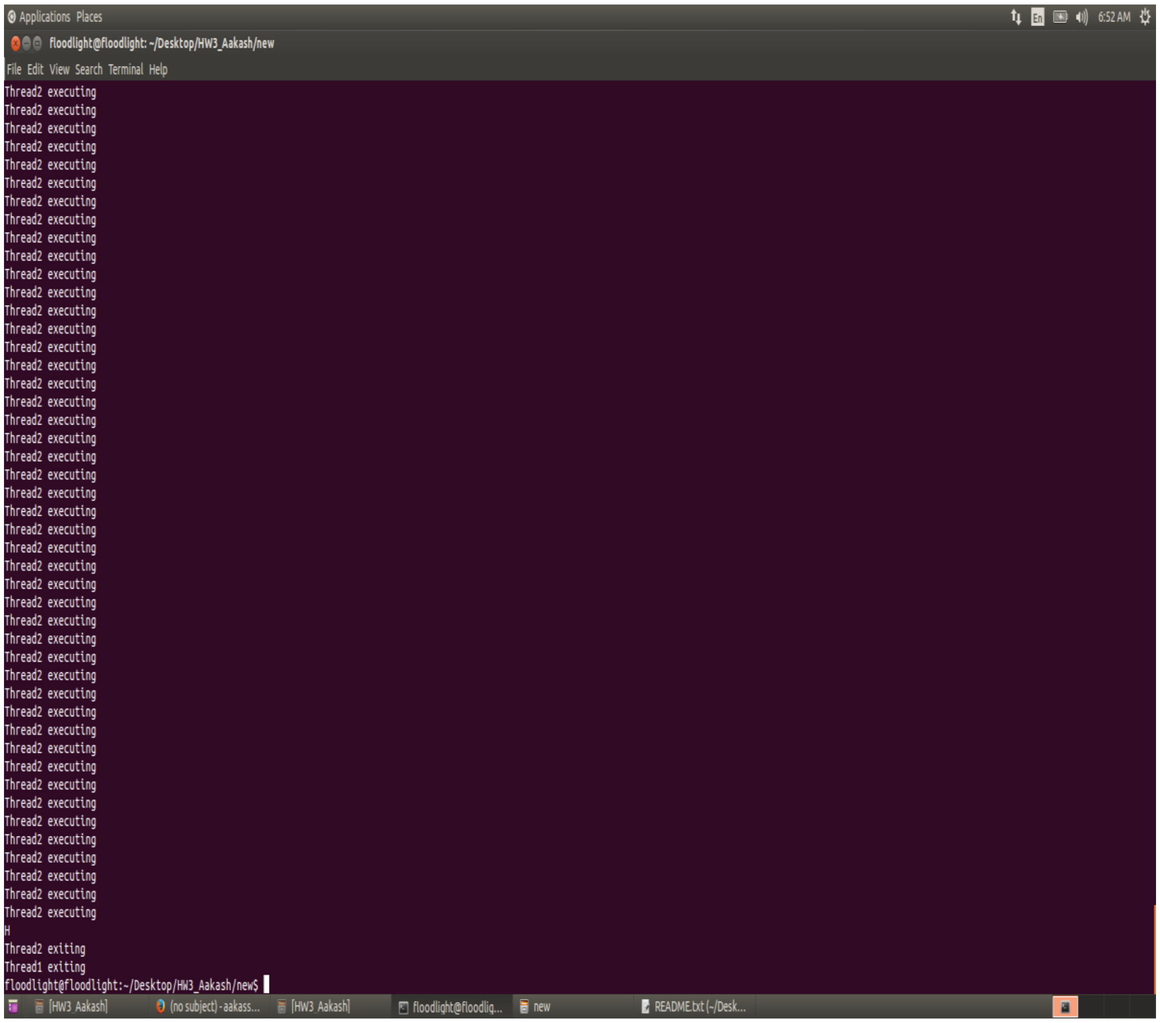
```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -pthread -o 64_HW3_Q1A 64_HW3_Q1A.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q1A
```



```
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
```



```
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread2 is started
A
K
A
S
Thread2 waiting
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
```



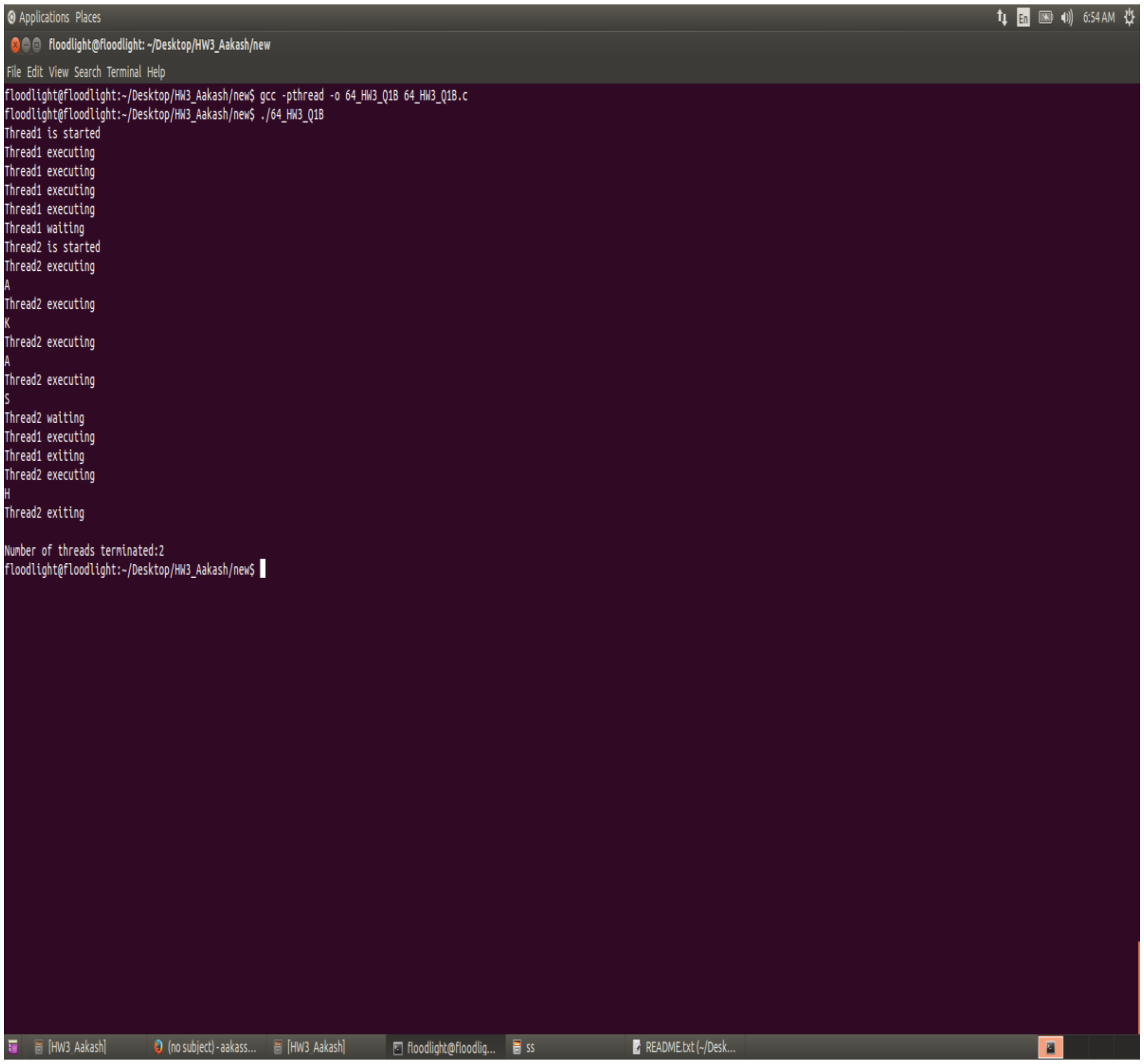
```
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
H
Thread2 exiting
Thread1 exiting
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ 
```

2. Below is the fixed code to solve race condition. Solution for the race condition is to implement **condition variable and mutex.**

   Conditional Variable: Condition variable is one form of synchronizing primitives which combines a lock with a "signaling" mechanism. It is used when threads need to wait for a resource to become available. A thread can "wait" on a Conditional Variable and then the resource producer can "signal" that variable, in which case the threads that wait for the Conditional Variable get notified and can continue execution.

   Mutex: A mutex is combined with Conditional Variable to avoid the race condition where a thread starts to wait on a Conditional variable at the same time another thread wants to signal it; then it is not controllable whether the signal is delivered or gets lost.
   The source code and screenshot for implementation of condition variable will be:

```
/* Code with Race Condition problem solved using mutex and condition variables –
q1_solved.c */

#include <pthread.h>
#include <stdio.h>
#define SIZE 4
#define ITEMS 5
#define THREADS 2

typedef struct {
        char buf[SIZE];
        int occupied;
        int in,out;

        pthread_mutex_t mutex;  //Used mutex
        pthread_cond_t m;  //Used Conditional variable
        pthread_cond_t l;  //Used Conditional variable
        } buffer_t;

buffer_t buffer;
void * thread1(void *);
void * thread2(void *);
pthread_t tid[THREADS];

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

        //Initialized Condition variable and Mutex
        pthread_cond_init(&(buffer.m), NULL);
        pthread_cond_init(&(buffer.l), NULL);

        pthread_create(&tid[1], NULL, thread2, NULL);
        pthread_create(&tid[0], NULL, thread1, NULL);
        for ( i = 0; i < THREADS; i++)
                pthread_join(tid[i], NULL);
        printf("\nNumber of threads terminated:%d\n", i);
}
void * thread1(void * parm)
{
        char item[ITEMS]="AKASH";
        int i;
        printf("Thread1 is started\n");
        for(i=0;i<ITEMS;i++)
        {
                if (item[i] == '\0')
                break;
                //Lock the mutex
                pthread_mutex_lock(&(buffer.mutex));
                if (buffer.occupied >= SIZE)
                        printf("Thread1 waiting\n");
                while (buffer.occupied >= SIZE)

                        //Condition variable wait
                        pthread_cond_wait(&(buffer.l), &(buffer.mutex) );

                printf("Thread1 executing\n");
                buffer.buf[buffer.in++] = item[i];
                buffer.in %= SIZE;
                buffer.occupied++;

                //Condition variable signal
                pthread_cond_signal(&(buffer.m));

                //Unlock the mutex
                pthread_mutex_unlock(&(buffer.mutex));
        }
        printf("Thread1 exiting\n");
        pthread_exit(0);
}
void * thread2(void * parm)
{
        char item;
        int i;
        printf("Thread2 is started\n");
        for(i=0;i<ITEMS;i++)
        {

                //Lock the mutex
                pthread_mutex_lock(&(buffer.mutex) );

                if (buffer.occupied <= 0)
                        printf("Thread2 waiting\n");
                while(buffer.occupied <= 0)

                //Condition variable wait
                pthread_cond_wait(&(buffer.m), &(buffer.mutex) );

                printf("Thread2 executing\n");
                item = buffer.buf[buffer.out++];
                printf("%c\n",item);
                buffer.out %= SIZE;
                buffer.occupied--;

                //Condition variable signal
                pthread_cond_signal(&(buffer.l));

                //Unlock the mutex
                pthread_mutex_unlock(&(buffer.mutex));
        }
        printf("Thread2 exiting\n");
        pthread_exit(0);
}
```



```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -pthread -o 64_HW3_Q1B 64_HW3_Q1B.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q1B
Thread1 is started
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 waiting
Thread2 is started
Thread2 executing
A
Thread2 executing
K
Thread2 executing
A
Thread2 executing
S
Thread2 waiting
Thread1 executing
Thread1 exiting
Thread2 executing
H
Thread2 exiting

Number of threads terminated:2
floodlight@floodlight:~/Desktop/HW3_Aakash/new$
```

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

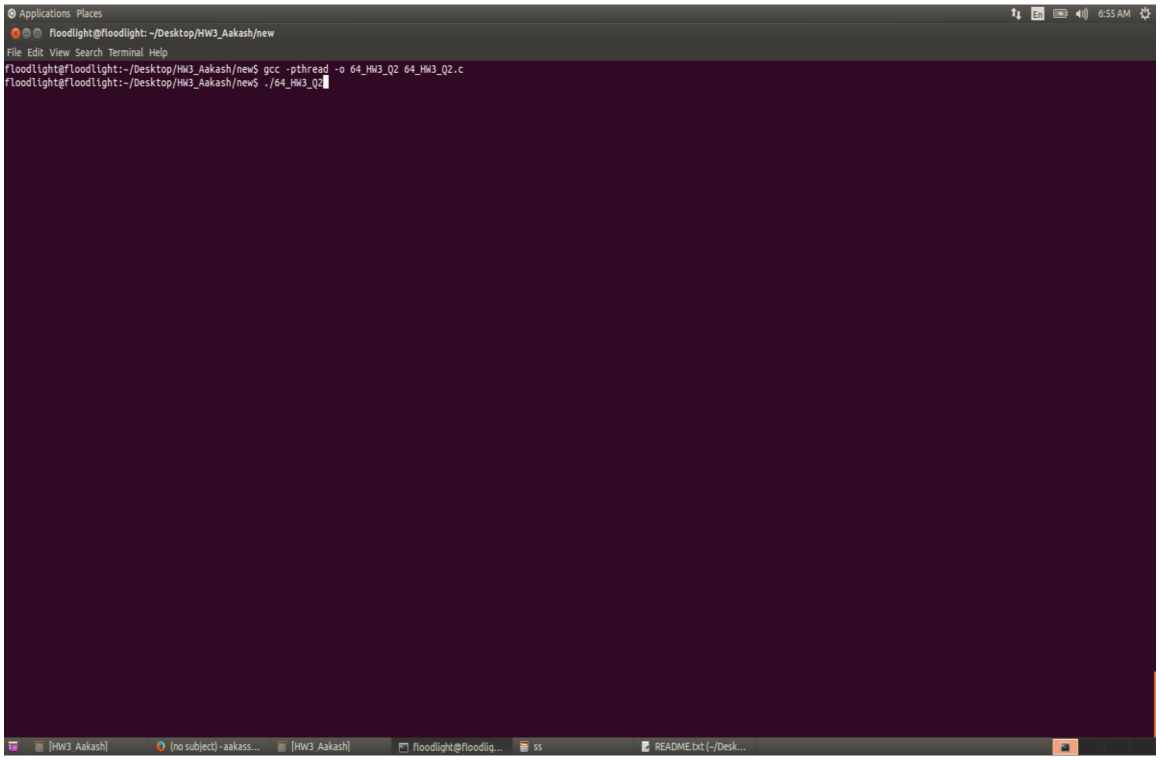
Answer:

If the problem of race condition is solved by semaphore, then the source code and the screenshot for the same would be:

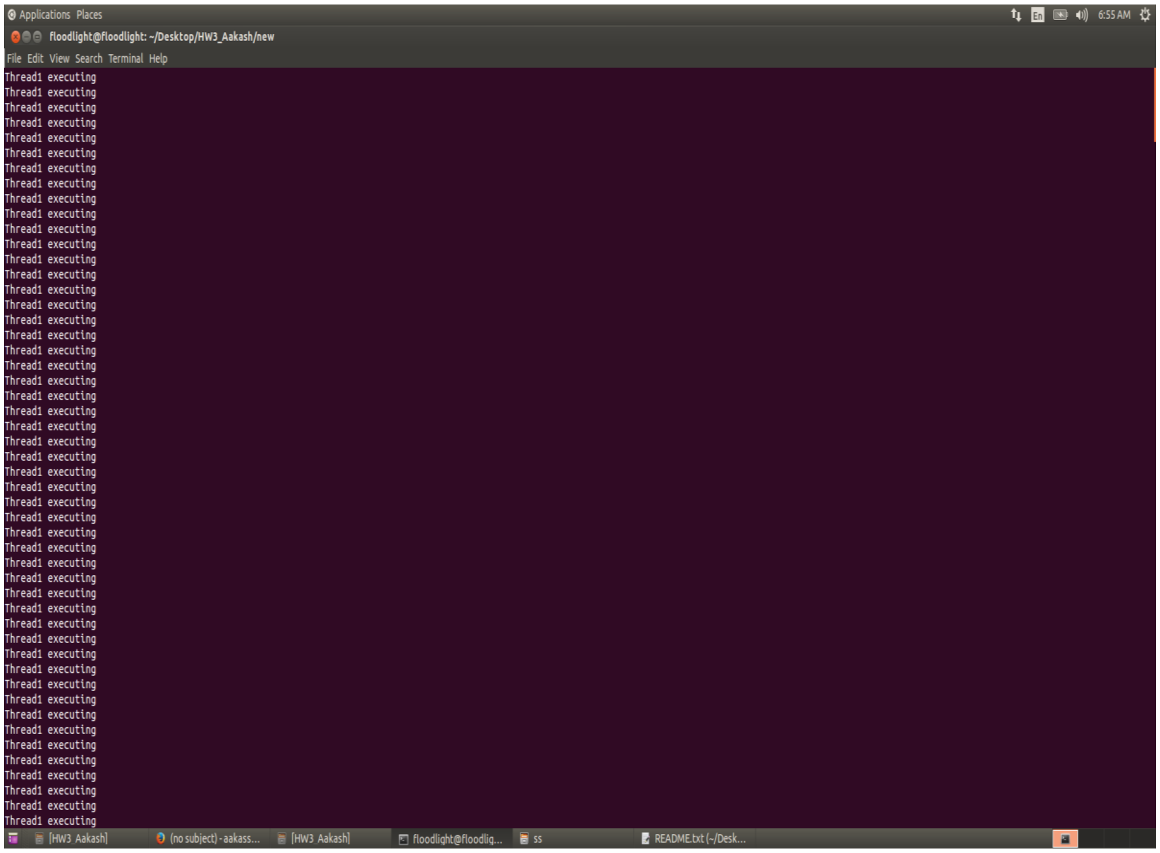
Source Code:

```
/* Code with Race Condition problem solved using semaphore */

#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#define SIZE 4
#define ITEMS 5
#define THREADS 2

typedef struct {
        char buf[SIZE];
        int occupied;
        int in,out;

} buffer_t;

buffer_t buffer;

void * thread1(void *);
void * thread2(void *);

static sem_t sema;


pthread_t tid[THREADS];

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

        //Initialized semaphore
        sem_init(&sema, 0, 2);

        pthread_create(&tid[1], NULL, thread2, NULL);
        pthread_create(&tid[0], NULL, thread1, NULL);
        for ( i = 0; i < THREADS; i++)
                pthread_join(tid[i], NULL);

        printf("\nNumber of threads terminated:%d\n", i);

        //Destroying the semaphore
        sem_destroy(&sema);
}

void * thread1(void * parm)
{
        char item[ITEMS]="AKASH";
        int i;

        printf("Thread1 is started\n");

        for(i=0;i<ITEMS;i++)
        {
                if (item[i] == '\0')
                break;

                //Waiting of semaphore and decreasing the value by 1
                sem_wait(&sema);

                if (buffer.occupied >= SIZE)
                        printf("Thread1 waiting\n");

                while (buffer.occupied >= SIZE)

                printf("Thread1 executing\n");
                buffer.buf[buffer.in++] = item[i];
                buffer.in %= SIZE;
                buffer.occupied++;


                //Posting of the semaphore and increasing the value by 1
                sem_post(&sema);
        }
        printf("Thread1 exiting\n");
        pthread_exit(0);
}

void * thread2(void * parm)
{
        char item;
        int i;
        printf("Thread2 is started\n");

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

                //Waiting of semaphore and decreasing the value by 1
                sem_wait(&sema);

                if (buffer.occupied <= 0)
                        printf("Thread2 waiting\n");

                while(buffer.occupied <= 0)
                        printf("Thread2 executing\n");

                item = buffer.buf[buffer.out++];
                printf("%c\n",item);
                buffer.out %= SIZE;
                buffer.occupied--;

                //unlock the semaphore and increases the value by 1
                sem_post(&sema);
        }
        printf("Thread2 exiting\n");
        pthread_exit(0);
}
```

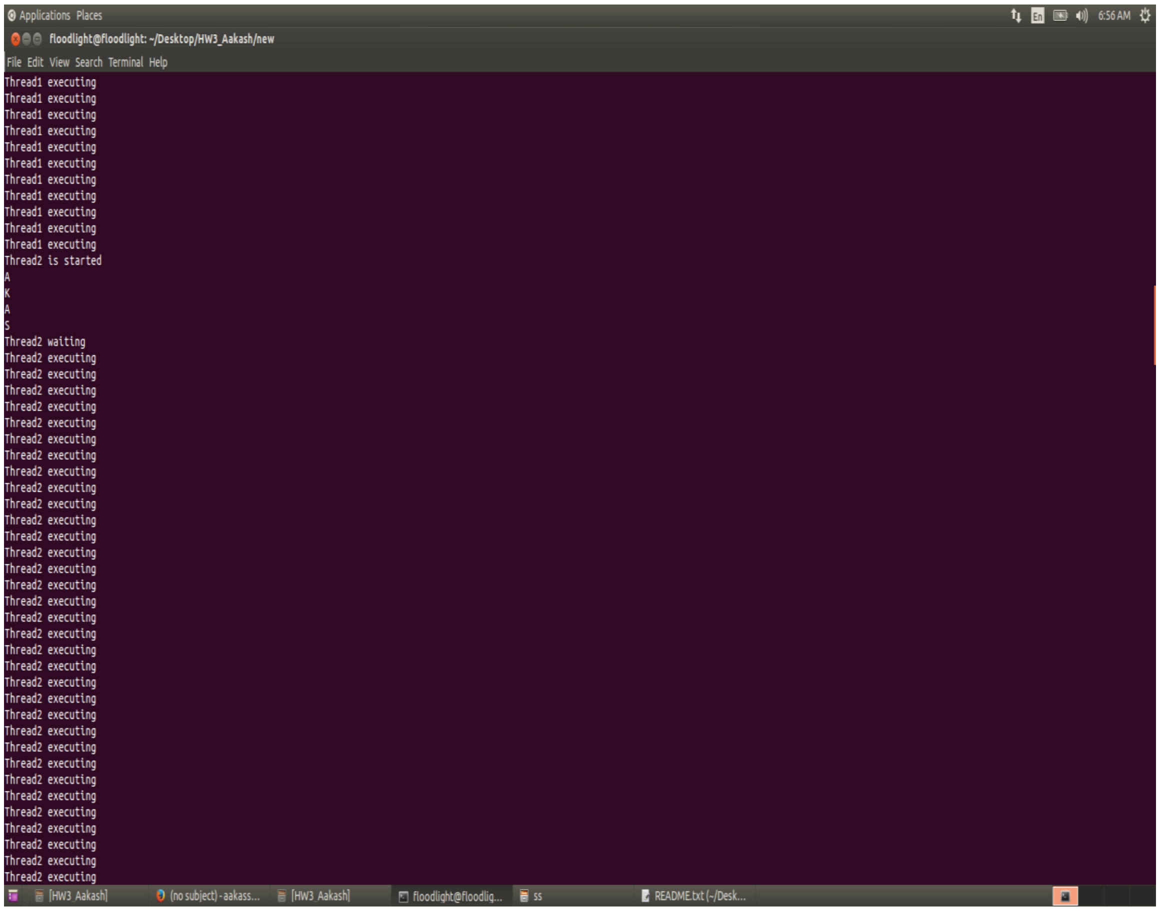
Screen shot:


Applications Places
floodlight@floodlight: ~/Desktop/HW3_Aakash/new
File Edit View Search Terminal Help
6:55 AM

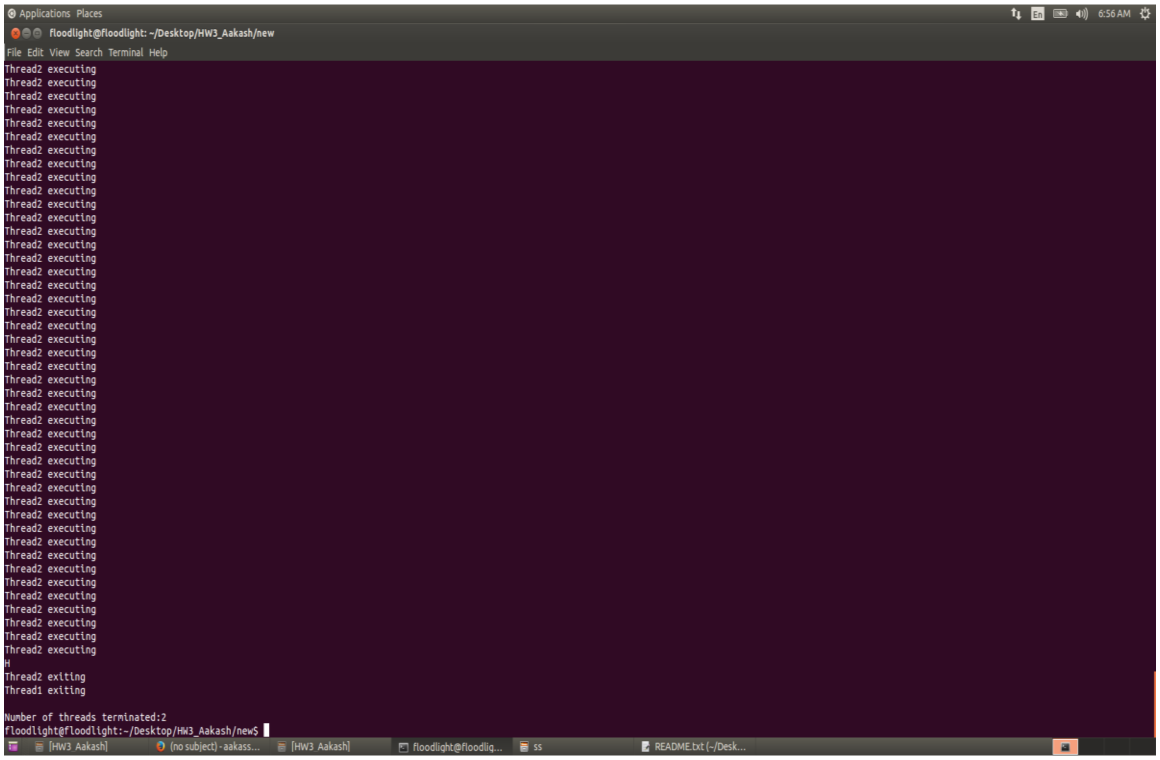
```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -pthread -o 64_HW3_Q2 64_HW3_Q2.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q2
```


Applications Places
floodlight@floodlight: ~/Desktop/HW3_Aakash/new
File Edit View Search Terminal Help
6:55 AM

```
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
```


Applications Places
floodlight@floodlight: ~/Desktop/HW3_Aakash/new
File Edit View Search Terminal Help
6:56 AM

```
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread1 executing
Thread2 is started
A
K
A
S
Thread2 waiting
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
```


Applications Places
floodlight@floodlight: ~/Desktop/HW3_Aakash/new
File Edit View Search Terminal Help
6:56 AM

```
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
Thread2 executing
H
Thread2 exiting
Thread1 exiting

Number of threads terminated:2
floodlight@floodlight:~/Desktop/HW3_Aakash/new$
```

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.

Answer:

- getsockoption() function usage is to get the values of options set for a particular socket

getsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);

- level indicates the level of the socket. Some of the levels are –

SOL_SOCKET, IPPROTO_IP, IPPROTO_IPV6, IPPROTO_ICMPV6, IPPROTO_TCP
Below is the source code to demonstrate the getsockopt() function to display the default values of receive buffer –

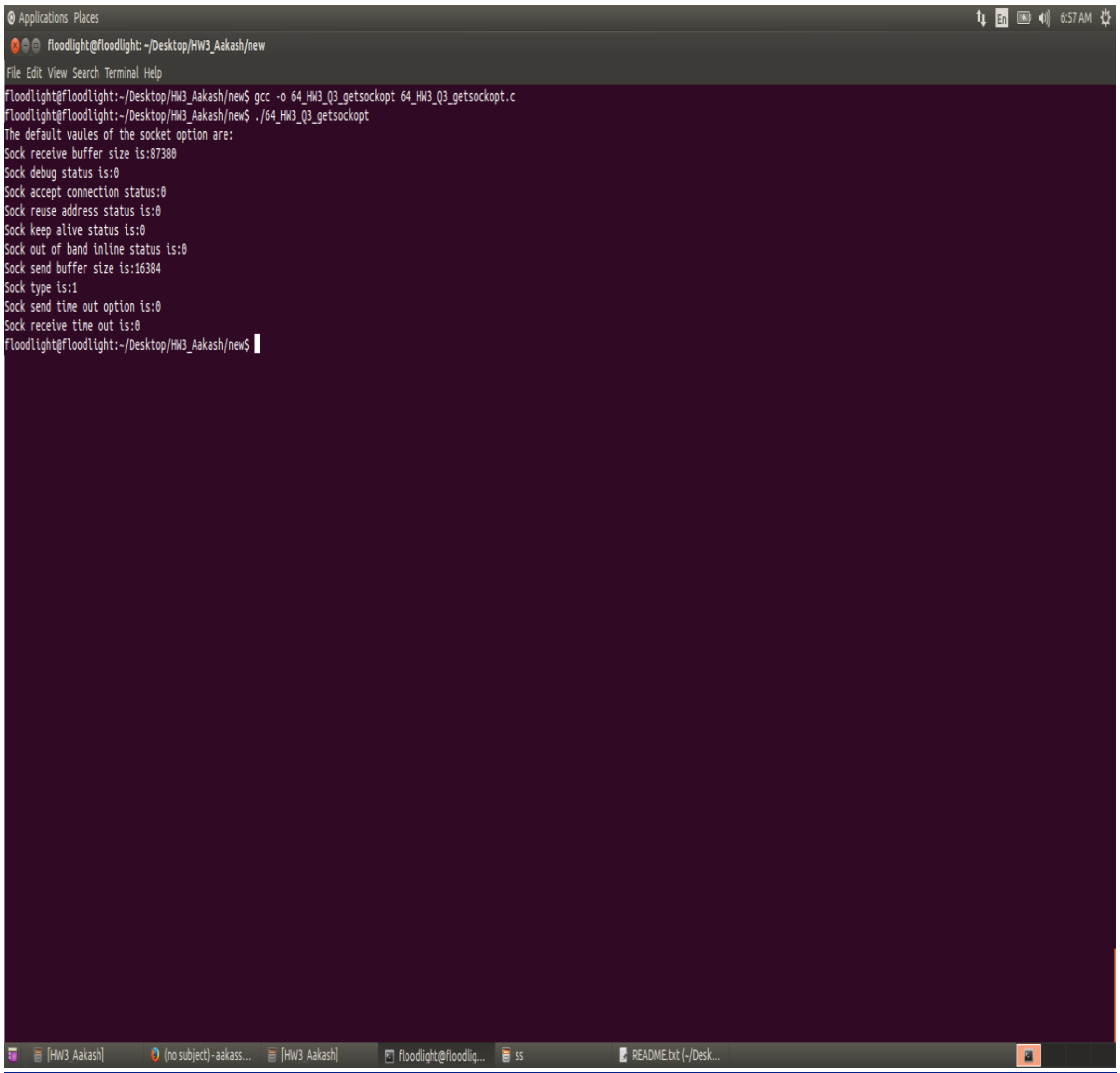
/*
Source code: to get default values of receive buffer using getsockopt() – getsockopt.c
*/

```
  s1 = sizeof(rcvbuf);
  s2 = sizeof(debug);
  s3 = sizeof(acceptconn);
  s4 = sizeof(reusaddr);
  s5 = sizeof(keepalive);
  s6 = sizeof(oobinline);
  s7 = sizeof(sndbuf);
  s8 = sizeof(type);
  s9 = sizeof(sndtimeo);
  s10 = sizeof(rcvtimeo);

  printf("The default vaules of the socket option are:\n");

  getsockopt(sock, SOL_SOCKET, SO_RCVBUF, &rcvbuf, &s1);
  printf ("Sock receive buffer size is:%d\n", rcvbuf);

  getsockopt(sock, SOL_SOCKET, SO_DEBUG, &debug, &s2);
  printf ("Sock debug status is:%d\n", debug);

  getsockopt(sock, SOL_SOCKET, SO_ACCEPTCONN, &acceptconn, &s3);
  printf ("Sock accept connection status:%d\n", acceptconn);
  getsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &reusaddr, &s4);
  printf ("Sock reuse address status is:%d\n", reusaddr);

  getsockopt(sock, SOL_SOCKET, SO_KEEPALIVE, &keepalive, &s5);
  printf ("Sock keep alive status is:%d\n", keepalive);

  getsockopt(sock, SOL_SOCKET, SO_OOBINLINE, &oobinline, &s6);
  printf ("Sock out of band inline status is:%d\n", oobinline);

  getsockopt(sock, SOL_SOCKET, SO_SNDBUF, &sndbuf, &s7);
  printf ("Sock send buffer size is:%d\n", sndbuf);

  getsockopt(sock, SOL_SOCKET, SO_TYPE, &type, &s8);
  printf ("Sock type is:%d\n", type);

  getsockopt(sock, SOL_SOCKET, SO_SNDTIMEO, &sndtimeo, &s9);
  printf ("Sock send time out option is:%d\n", sndtimeo);

  getsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &rcvtimeo, &s10);
  printf ("Sock receive time out is:%d\n", rcvtimeo);
}
```

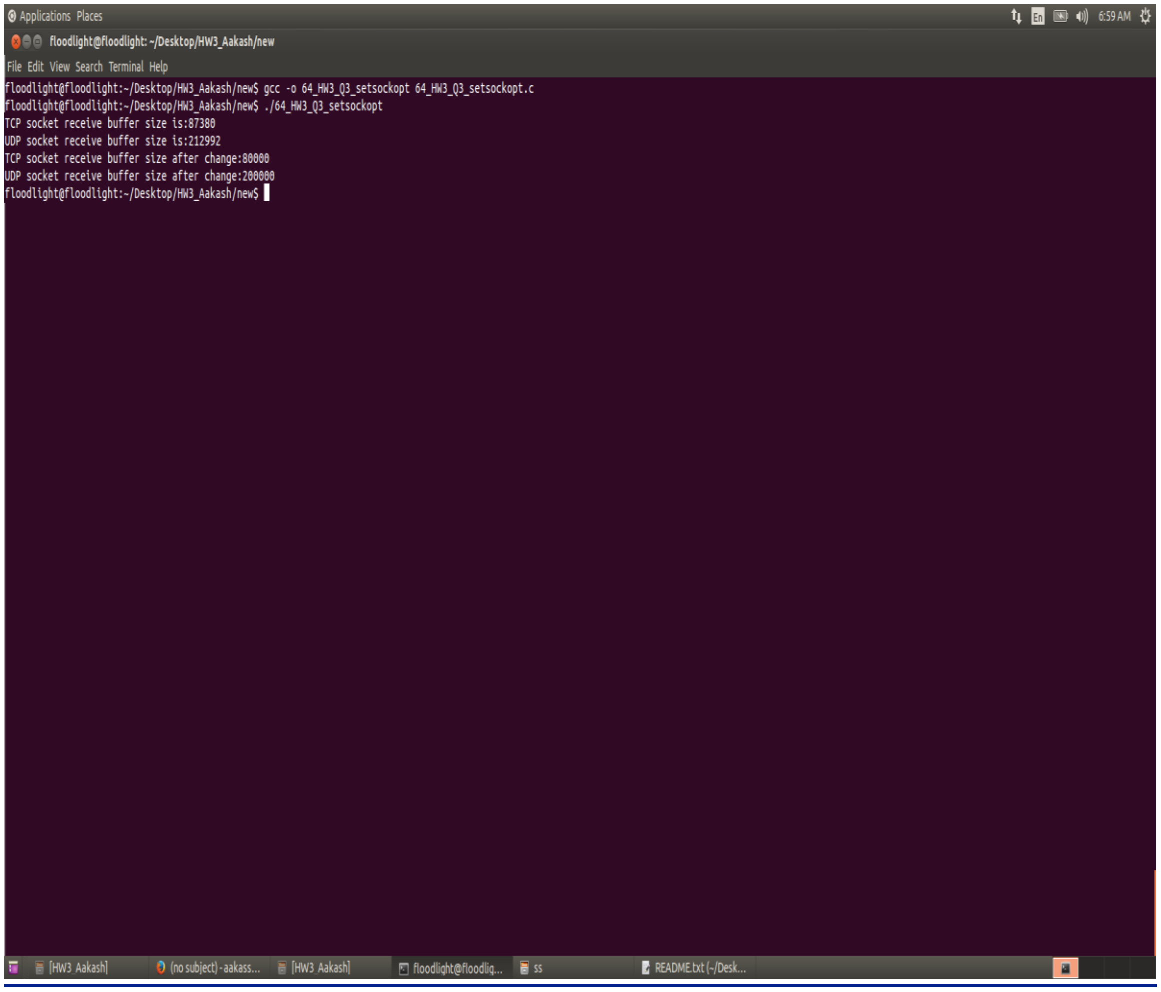
/* Code to demonstrate setsockopt() – q3_getsockopt.c */

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>

void main()
{
  int tcpsock,udpsock;
  tcpsock = socket(AF_INET, SOCK_STREAM,0);
  udpsock = socket(AF_INET, SOCK_DGRAM, 0);
  int tcprcvbufsize, udprcvbufsize;
  int size;

  // using getsockopt to get default value of receive buffer for tcp and udp
  size = sizeof(tcprcvbufsize);
  getsockopt(tcpsock, SOL_SOCKET, SO_RCVBUF, &tcprcvbufsize, &size);
  printf ("TCP socket receive buffer size is:%d\n", tcprcvbufsize);

  size = sizeof(udprcvbufsize);
  getsockopt(udpsock, SOL_SOCKET, SO_RCVBUF, &udprcvbufsize, &size);
  printf ("UDP socket receive buffer size is:%d\n", udprcvbufsize);

  //setsockopt() to change tcp receive buffer and udp receive buffer
  tcprcvbufsize = 80000;
  udprcvbufsize = 200000;

  size = sizeof(tcprcvbufsize);
  setsockopt(tcpsock, SOL_SOCKET, SO_RCVBUF, &tcprcvbufsize, size);
  printf ("TCP socket receive buffer size after change:%d\n", tcprcvbufsize);

  size = sizeof(udprcvbufsize);
  setsockopt(udpsock, SOL_SOCKET, SO_RCVBUF, &udprcvbufsize, size);
  printf ("UDP socket receive buffer size after change:%d\n", udprcvbufsize);
}
```

Screenshot:



```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -o 64_HW3_Q3_getsockopt 64_HW3_Q3_getsockopt.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q3_getsockopt
The default vaules of the socket option are:
Sock receive buffer size is:87380
Sock debug status is:0
Sock accept connection status:0
Sock reuse address status is:0
Sock keep alive status is:0
Sock out of band inline status is:0
Sock send buffer size is:16384
Sock type is:1
Sock send time out option is:0
Sock receive time out is:0
floodlight@floodlight:~/Desktop/HW3_Aakash/new$
```



```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -o 64_HW3_Q3_setsockopt 64_HW3_Q3_setsockopt.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q3_setsockopt
TCP socket receive buffer size is:87380
UDP socket receive buffer size is:212992
TCP socket receive buffer size after change:80000
UDP socket receive buffer size after change:200000
floodlight@floodlight:~/Desktop/HW3_Aakash/new$
```

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.

Answer:

Below is the source code to demonstrate the TCPdaytime program to work with either IPV4 or IPV6:

<u>Source Code: </u>q4_server.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

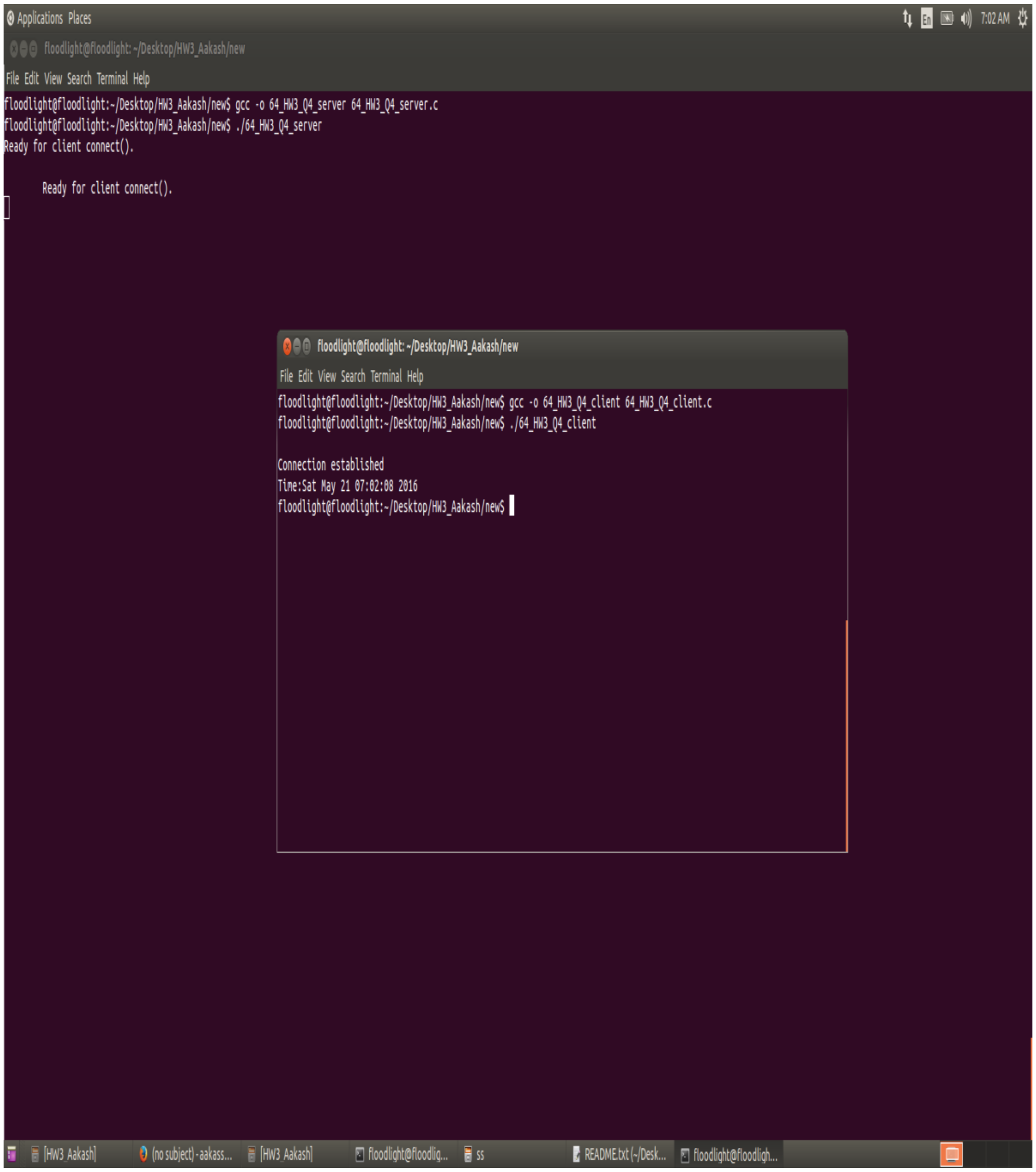
```
#include<string.h>
#include <stdlib.h>
#include<unistd.h>

#define SERVER_PORT     1212
#define BUFFER_LENGTH    250
#define FALSE              0

void TCPdaytimed(int fd);

void main()
{
   int servfd=-1, clientfd=-1;
   int rc, on=1, rcdsize=BUFFER_LENGTH;

   struct sockaddr_in6 servaddr={0}, clientaddr={0};
   int addrlen=sizeof(clientaddr);
   char str[INET6_ADDRSTRLEN];


     if ((servfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
      {
         perror("socket() failed");
         exit(0);
      }


      if (setsockopt(servfd, SOL_SOCKET, SO_REUSEADDR,(char *)&on,sizeof(on)) <
0)
      {
         perror("setsockopt(SO_REUSEADDR) failed");
         exit(0);
      }

      servaddr.sin6_family = AF_INET6;
      servaddr.sin6_port   = htons(SERVER_PORT);

      servaddr.sin6_addr   = in6addr_any;


      if (bind(servfd,
               (struct sockaddr *)&servaddr,
               sizeof(servaddr)) < 0)
      {
         perror("bind() failed");
         exit(0);
```

```
    }


    if (listen(servfd, 10) < 0)
    {
      perror("listen() failed");
      exit(0);
    }
   do
   {
    printf("Ready for client connect().\n");


    if ((clientfd = accept(servfd, NULL, NULL)) < 0)
    {
      perror("accept() failed");
      break;
    }
    else
    {

      getpeername(clientfd, (struct sockaddr *)&clientaddr, &addrlen);
      if(inet_ntop(AF_INET6, &clientaddr.sin6_addr, str, sizeof(str)))
      {
        printf("\n \t");
      }

TCPdaytimed(clientfd);
    }

  } while (1);

  if (servfd != -1)
    close(servfd);
}

void TCPdaytimed(int fd)
{
        char *pts;
        time_t now;
        char *ctime();
        (void)time(&now);
        pts=ctime(&now);
        (void)write(fd, pts, strlen(pts));
close(fd);
}
```

<u>Source code: </u>q4_client.c

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdlib.h>


#define SERVER_NAME "localhost"
#define LINELEN 128

void main(int argc, char *argv[])
{
  int    clientfd=-1, rc, n=0;
  char   buffer[LINELEN+1];
  char   servernode[100];
  char   servport[] = "1212";
  struct in6_addr servaddr;
  struct addrinfo hints, *res=NULL;

  if (argc > 1)
      strcpy(servernode, argv[1]);
  else
      strcpy(servernode, SERVER_NAME);

    memset(&hints, 0x00, sizeof(hints));
    hints.ai_flags    = AI_NUMERICSERV;
    //can be both ipv4 or ipv6 addresses
    hints.ai_family   = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;

    //To convert the text form of the address to binary
    rc = inet_pton(AF_INET, servernode, &servaddr);
    if (rc == 1)    /* valid IPv4 text address? */
    {
      hints.ai_family = AF_INET;
      hints.ai_flags |= AI_NUMERICHOST;
    }
    else
    {
      rc = inet_pton(AF_INET6, servernode, &servaddr);
      if (rc == 1) /* valid IPv6 text address? */
      {

        hints.ai_family = AF_INET6;
        hints.ai_flags |= AI_NUMERICHOST;
      }
    }

    //getaddrinfo() combines the getservbyname() and gethostbyname() functionalities
into 1 function
    //getaddrinfo() is reentrant and allows programs to eliminate IPv4-versus-IPv6
dependencies

    rc = getaddrinfo(servernode, servport, &hints, &res);
    if (rc != 0)
    {
      printf("Host not found:%s\n", gai_strerror(rc));
      if (rc == EAI_SYSTEM)
        printf("getaddrinfo() failed");
      exit(0);
    }


    clientfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (clientfd < 0)
    {
      printf("socket() failed");
      exit(0);
    }
       rc = connect(clientfd, res->ai_addr, res->ai_addrlen);
    if (rc < 0)
    {

      printf("connect() failed");
      exit(0);
    }

        printf("\nConnection established\nTime:");


                while((n=recv(clientfd, buffer, LINELEN, 0))>0)
                {
                        buffer[n]='\0';
                        (void)fputs(buffer, stdout);
                        n=0;
                }
    close(clientfd);

  if (res != NULL)
    freeaddrinfo(res);
}
```

Screen Shot:



```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -o 64_HW3_Q4_server 64_HW3_Q4_server.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q4_server
Ready for client connect().

        Ready for client connect().
```

```
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ gcc -o 64_HW3_Q4_client 64_HW3_Q4_client.c
floodlight@floodlight:~/Desktop/HW3_Aakash/new$ ./64_HW3_Q4_client

Connection established
Time:Sat May 21 07:02:08 2016
floodlight@floodlight:~/Desktop/HW3_Aakash/new$
```