NAME-KRISHNA KUMAR MAHTO
REG- 16BIT0453

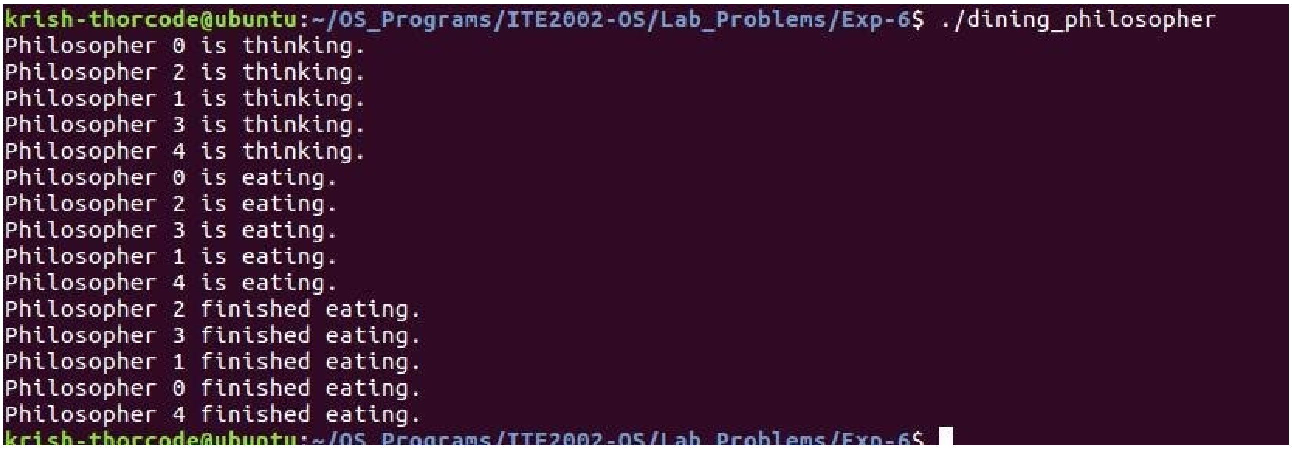
EXPERIMENT 6

a. DININING PHILOSOPHER

CODE:

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_mutex_t mutex_fork[5]; // 5 mutexes for each fork
pthread_key_t phil_num;

void *philosopher_func(void *arg)
{
//printf("%d\n",*((int*)arg));
pthread_setspecific(phil_num,arg);

printf("Philosopher %d is
thinking.\n",*(int*)pthread_getspecific(phil_num));
sleep(3);

/*philosopher number phi_no will now pick up two forks.*/
pthread_mutex_lock(&mutex_fork[*(int*)pthread_getspecific(phil_num
)]);
pthread_mutex_lock(&mutex_fork[(*(int*)pthread_getspecific(phil_nu
m)+1)%5]);

printf("Philosopher %d is
eating.\n",*(int*)pthread_getspecific(phil_num));
sleep(2);

pthread_mutex_unlock(&mutex_fork[*(int*)pthread_getspecific(phil_n
um)]);
pthread_mutex_unlock(&mutex_fork[(*(int*)pthread_getspecific(phil_
num)+1)%5]);

printf("Philosopher %d finished
eating.\n",*(int*)pthread_getspecific(phil_num));

return NULL;
}

int main(int argc,char **argv)
{
pthread_t thread_philosopher[5];
pthread_key_create(&phil_num,NULL);
int* i=(int*)malloc(sizeof(int));
int j;

for(j=0;j<5;j++)
pthread_mutex_init(&mutex_fork[j],NULL);

for(j=0,*i=j;j<5;i++,j++,*i=j)
{
//printf("%d\n",i);
pthread_create(&thread_philosopher[j],NULL,philosopher_func,
(void*)i);
}
for(j=0;j<5;j++)
pthread_mutex_destroy(&mutex_fork[j]);// freeing the space
occupied by mutex

for(j=0;j<5;j++)
pthread_join(thread_philosopher[j],NULL);

return 0;
}
```

OUTPUT:



```
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$ ./dining_philosopher
Philosopher 0 is thinking.
Philosopher 2 is thinking.
Philosopher 1 is thinking.
Philosopher 3 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is eating.
Philosopher 2 is eating.
Philosopher 3 is eating.
Philosopher 1 is eating.
Philosopher 4 is eating.
Philosopher 2 finished eating.
Philosopher 3 finished eating.
Philosopher 1 finished eating.
Philosopher 0 finished eating.
Philosopher 4 finished eating.
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$
```

b.) Socket Programming

Server:

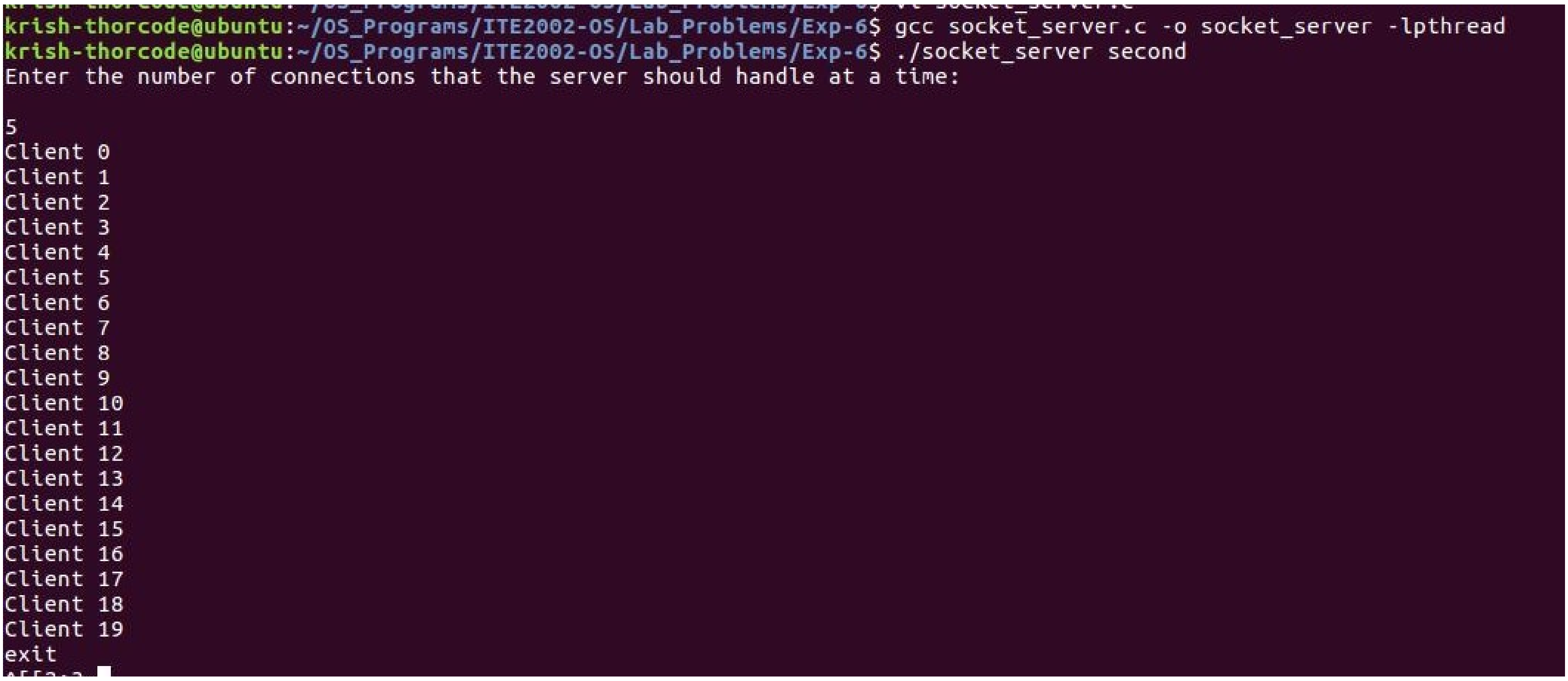
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<unistd.h>
#include<semaphore.h>

sem_t num_of_connections;

int server(int client_fd)
{
char buffer[20];
sleep(2);
read(client_fd,buffer,20);
printf("%s \n",buffer);
sem_post(&num_of_connections);
if(strcmp(buffer,"exit")==0)
return 1;

return 0;
}

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

const char* const socket_name=argv[1]; // const char* = value
pointed by the pointer is const. const socket_name = the pointer
variable name socket_name is also constant

int n=0,exit=0;
puts("Enter the number of connections that the server should
handle at a time: \n");
scanf("%d",&n);

sem_init(&num_of_connections,0,n);

int server_fd;
struct sockaddr_un name; // address structure holds the socket
address for binding

server_fd = socket(PF_LOCAL,SOCK_STREAM,0); //protocol parameter =
0 for local namespace, PF_LOCAL indicates that the server socket
will follow local namespace, this information is for the reference
of server socket itself

/* filling values into address structure of the server */
name.sun_family = AF_LOCAL; //indicates that the address structure
is of local namespace. this information is used by client when
client stores server's address to maintain that server follows
local namespace, this information is for the reference of client
socket
strcpy(name.sun_path,socket_name);

/* bind the server socket fd with the server socket address */

bind(server_fd,(struct sockaddr*)&name,SUN_LEN(&name));

/* server should now listen for connections from clients */

listen(server_fd,5); // the max number of requests that can stay
in waiting queue= 5

do
{
int client_fd;
struct sockaddr_un client_name;
socklen_t client_name_len;

sem_wait(&num_of_connections);
client_fd = accept(server_fd,(struct
sockaddr*)&client_name,&client_name_len);

/*handle the connection*/

server(client_fd);

/*close connection for client that returns */

exit=close(client_fd);
}while(exit!=1);

close(server_fd);
unlink(socket_name);

return 0;
}
```

CLIENT:

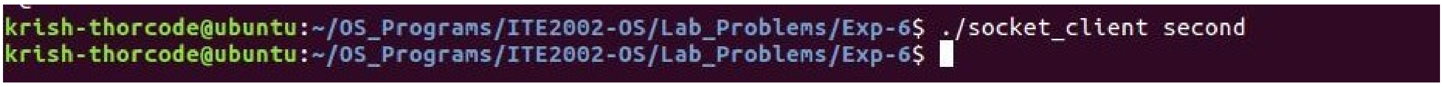
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<sys/socket.h>
```

```
#include<sys/un.h>
#include<unistd.h>
#define NUM_OF_CONNECTIONS 20

int main(int argc,char* const argv[])
{
int count=0;
const char* const socket_name = argv[1]; // same as the socket
name created by the server

do
{

struct sockaddr_un name;
int client_fd;

client_fd = socket(PF_LOCAL,SOCK_STREAM,0);

/* fill in the server details into client's name structure,ie,
client_name= serve_ name */
name.sun_family=AF_LOCAL;
strcpy(name.sun_path,socket_name);

connect(client_fd,(struct sockaddr*)&name,SUN_LEN(&name)); //check
symmetry with the accept() sys call in server code
char buffer[20];

if(count!=NUM_OF_CONNECTIONS)
{
sprintf(buffer,"Client %d",count++);
write(client_fd,buffer,sizeof(buffer));
}
else
{
sprintf(buffer,"exit");
count++;
write(client_fd,buffer,sizeof(buffer));
}
close(client_fd);
}while(count<=NUM_OF_CONNECTIONS);

return 0;

}
```

OUTPUT:



```
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$ vi socket_server.c
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$ gcc socket_server.c -o socket_server -lpthread
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$ ./socket_server second
Enter the number of connections that the server should handle at a time:

5
Client 0
Client 1
Client 2
Client 3
Client 4
Client 5
Client 6
Client 7
Client 8
Client 9
Client 10
Client 11
Client 12
Client 13
Client 14
Client 15
Client 16
Client 17
Client 18
Client 19
exit
^[[3;2
```



```
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$ ./socket_client second
krish-thorcode@ubuntu:~/OS_Programs/ITE2002-OS/Lab_Problems/Exp-6$
```