**ASSIGNMENT -3**

1. **Explain the Difference Between Mutex and semaphore with an example code ?**

**Explain: A mutex is an object but semaphore is an integer variable.**

**Code:**

**Mutex,**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

pthread\_t tid[2];

int counter;

pthread\_mutex\_t lock;

void\* trythis(void\* arg)

{

pthread\_mutex\_lock(&lock);

unsigned long i = 0;

counter += 1;

printf("\n Job %d has started\n", counter);

for (i = 0; i < 100; i++)

;

printf("\n Job%d has finished\n", counter);

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main(void)

{

int i = 0;

int error;

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

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

return 1;

}

while (i < 2) {

error = pthread\_create(&(tid[i]),NULL,&trythis,NULL;

if (error != 0)

printf("\nThread can't be created :[%s]",

strerror(error));

i++;

}

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

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

pthread\_mutex\_destroy(&lock);

return 0;

}

Sem:

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t mutex;

void\* thread(void\* arg)

{

sem\_wait(&mutex);

printf("\nEntered..\n");

*//critical section*

sleep(4);

*//signal*

printf("\nJust Exiting...\n");

sem\_post(&mutex);

}

int main()

{

sem\_init(&mutex, 0, 1);

pthread\_t t1,t2;

pthread\_create(&t1,NULL,thread,NULL);

sleep(2);

pthread\_create(&t2,NULL,thread,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

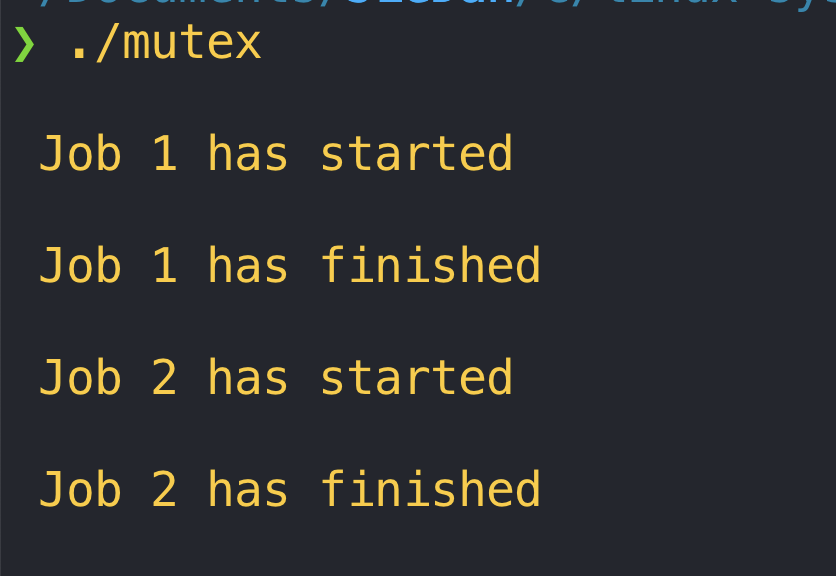
sem\_destroy(&mutex);

return 0;

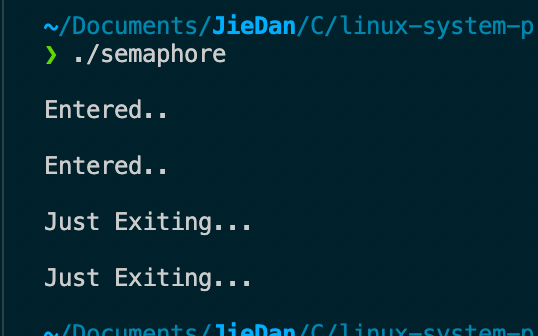
}

**Result:**

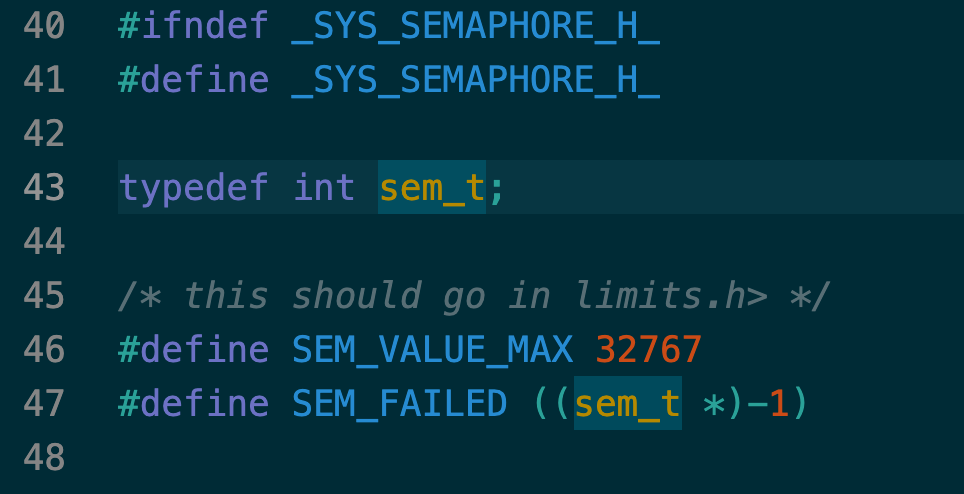
**Mutex,**

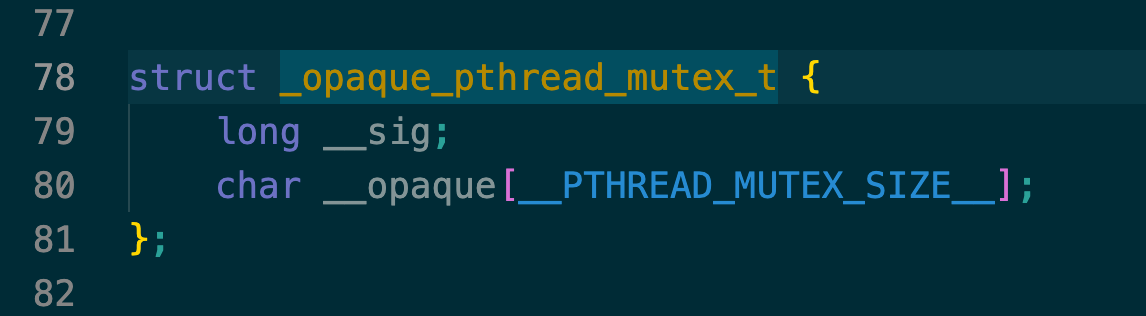
****

**Sem:**

****

**Sem is a typedef of int, while mutex is a struct.**

****

****

1. **Explain the Difference Between Process and Threads in Linux with an example code ?**

**Explain: A Process is mostly isolated, whereas Threads share memory, a process is more larger than a thread. Child process copy data from its parent, while thread share data.**

**Code:**

**Process,**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

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

{

pid\_t pid;

int data = 0x11;

printf("Original data is %x\n", data);

pid = fork();

*// this child*

if (pid == 0) {

*// modify data from child*

data = 0x22;

} else if (pid > 0) {

printf("The data from parent is:%x\n", data);

} else {

perror("fork");

exit(EXIT\_FAILURE);

}

*// wait for child*

wait(NULL);

}

**Thread:**

#include<stdio.h>

#include<string.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

pthread\_t tid[2];

void\* doSomeThing(void \*data)

{

\*(int \*)data = 0x11;

}

int main(void)

{

int i = 0;

int data = 0x22;

int err;

printf("Original data is: %x\n", data);

while(i < 2)

{

err = pthread\_create(&(tid[i]), NULL, &doSomeThing, &data);

if (err != 0)

printf("\ncan't create thread :[%s]", strerror(err));

else

printf("\n Thread created successfully\n");

i++;

}

printf("Thread modified data, the data is:%x\n", data);

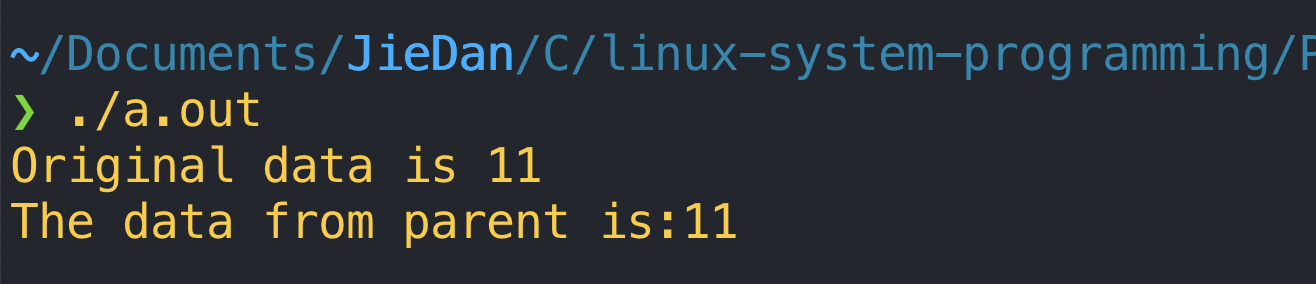
sleep(5);

return 0;

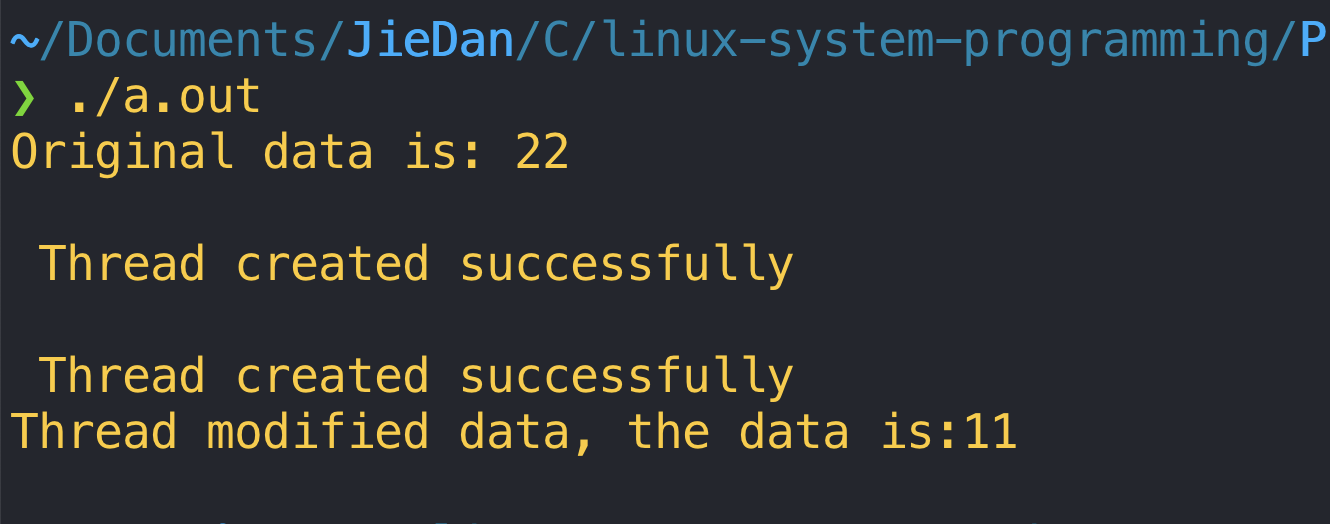
}

**Result:**

**Process communicate:**

****

**Thread communicate:**

****

1. **what are Different types of IPC mechanism involved in Linux Explain In detail?**
2. **Signal: Signals are one of the oldest inter-process communication methods used by Unix TM systems. They are used to signal asynchronous events to one or more processes.**
3. **Pipe: a pipe is implemented using two file data structures which both point at the same temporary VFS inode which itself points at a physical page within memory.**
4. **Socket: communicate over internet.**
5. **Explain and illustrate IPC Shared Memory using an example code ?**

**Code:**

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

void\* create\_shared\_memory(size\_t size) {

int protection = PROT\_READ | PROT\_WRITE;

int visibility = MAP\_SHARED| MAP\_ANONYMOUS;

return mmap(NULL, size, protection, visibility, -1, 0);

}

int main() {

char parent\_message[] = "hello"; *// parent process will write this message*

char child\_message[] = "goodbye"; *// child process will then write this one*

void\* shmem = create\_shared\_memory(128);

memcpy(shmem, parent\_message, sizeof(parent\_message));

int pid = fork();

if (pid == 0) {

printf("Child read: %s\n", shmem);

memcpy(shmem, child\_message, sizeof(child\_message));

printf("Child wrote: %s\n", shmem);

} else {

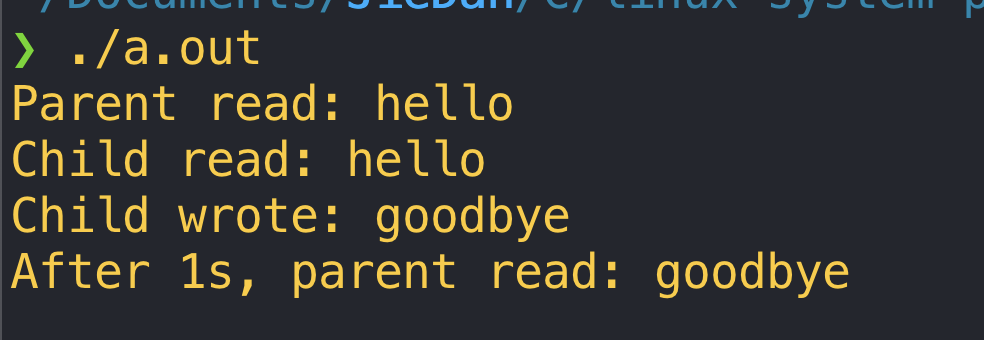
printf("Parent read: %s\n", shmem);

sleep(1);

printf("After 1s, parent read: %s\n", shmem);

}

}

**Result:**

1. **Explain and Illustrate the Difference between Unnamed Pipe and FIFO with and example code ?**

**FIFO also called named pipe, is another kind of pipe. Comparing to unnamed pipe, named pipe has a name.**

**Code:**

**Client.c**

*/\* Filename: fifoclient.c \*/*

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#define FIFO\_FILE "MYFIFO"

int main() {

int fd;

int end\_process;

int stringlen;

char readbuf[80];

char end\_str[5];

printf("FIFO\_CLIENT: Send messages, infinitely, to end enter \"end\"\n");

fd = open(FIFO\_FILE, O\_CREAT|O\_WRONLY);

strcpy(end\_str, "end");

while (1) {

printf("Enter string: ");

fgets(readbuf, sizeof(readbuf), stdin);

stringlen = strlen(readbuf);

readbuf[stringlen - 1] = '\0';

end\_process = strcmp(readbuf, end\_str);

*//printf("end\_process is %d\n", end\_process);*

if (end\_process != 0) {

write(fd, readbuf, strlen(readbuf));

printf("Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen(readbuf));

} else {

write(fd, readbuf, strlen(readbuf));

printf("Sent string: \"%s\" and string length is %d\n", readbuf, (int)strlen(readbuf));

close(fd);

break;

}

}

return 0;

}

**Server.c**

*/\* Filename: fifoserver.c \*/*

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#define FIFO\_FILE "MYFIFO"

int main() {

int fd;

char readbuf[80];

char end[10];

int to\_end;

int read\_bytes;

*/\* Create the FIFO if it does not exist \*/*

mknod(FIFO\_FILE, S\_IFIFO|0640, 0);

strcpy(end, "end");

while(1) {

fd = open(FIFO\_FILE, O\_RDONLY);

read\_bytes = read(fd, readbuf, sizeof(readbuf));

readbuf[read\_bytes] = '\0';

printf("Received string: \"%s\" and length is %d\n", readbuf, (int)strlen(readbuf));

to\_end = strcmp(readbuf, end);

if (to\_end == 0) {

close(fd);

break;

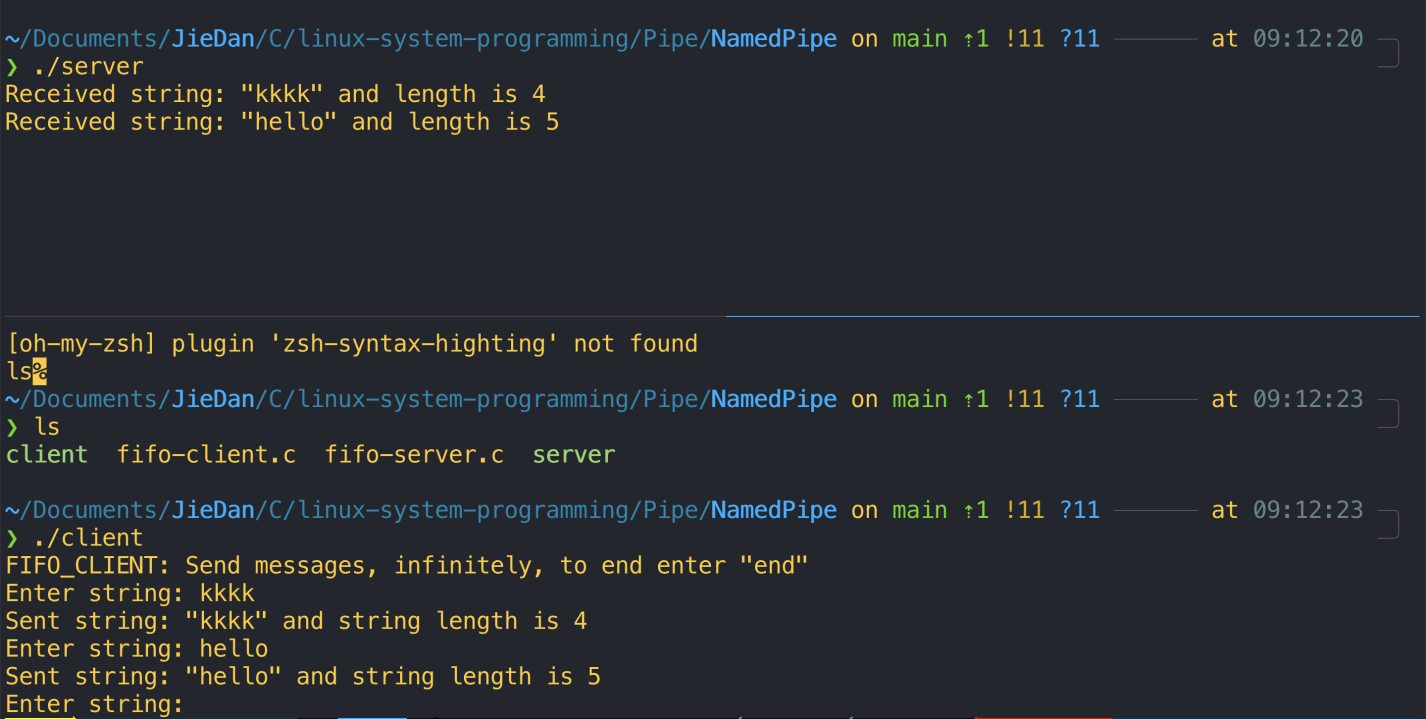
}

}

return 0;

}

**Result:**

****

1. **Explain and Illustrate IPC mechanism using sockets (Server and client Program ) with an example code ?**

**Explain:**

**A socket is one endpoint of a two-way communication link between two programs running on the network. Bind port and ip address to socket and use send and receive to send data and get data from socket.**

**Code:**

**Client:**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

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

{

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

char \*hello = "Hello from client";

char buffer[1024] = {0};

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

{

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

*// Convert IPv4 and IPv6 addresses from text to binary form*

if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("\nInvalid address/ Address not supported \n");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

{

printf("\nConnection Failed \n");

return -1;

}

send(sock , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

valread = read( sock , buffer, 1024);

printf("%s\n",buffer );

return 0;

}

**Server:**

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

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

{

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[1024] = {0};

char \*hello = "Hello from server";

*// Creating socket file descriptor*

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

*// Forcefully attaching socket to the port 8080*

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT,

&opt, sizeof(opt)))

{

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons( PORT );

*// Forcefully attaching socket to the port 8080*

if (bind(server\_fd, (struct sockaddr \*)&address,

sizeof(address))<0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0)

{

perror("listen");

exit(EXIT\_FAILURE);

}

if ((new\_socket =accept(server\_fd, (struct sockaddr \*)&address,

(socklen\_t\*)&addrlen))<0)

{

perror("accept");

exit(EXIT\_FAILURE);

}

valread = read( new\_socket , buffer, 1024);

printf("%s\n",buffer );

send(new\_socket , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

return 0;

}

**Result:**

****

1. **Explain and Illustrate IPC mechanism Message Queue with an example code?**

**Explain:**

**A message queue is a queue of messages sent between applications.**

**Code:**

**Reader.c**

#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#define PORT 8080

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

{

int sock = 0, valread;

struct sockaddr\_in serv\_addr;

char \*hello = "Hello from client";

char buffer[1024] = {0};

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

{

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

*// Convert IPv4 and IPv6 addresses from text to binary form*

if(inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr)<=0)

{

printf("\nInvalid address/ Address not supported \n");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0)

{

printf("\nConnection Failed \n");

return -1;

}

send(sock , hello , strlen(hello) , 0 );

printf("Hello message sent\n");

valread = read( sock , buffer, 1024);

printf("%s\n",buffer );

return 0;

}

**Writer.c**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX 10

*// structure for message queue*

struct mesg\_buffer {

long mesg\_type;

char mesg\_text[100];

} message;

int main()

{

key\_t key;

int msgid;

*// ftok to generate unique key*

key = ftok("progfile", 65);

*// msgget creates a message queue*

*// and returns identifier*

msgid = msgget(key, 0666 | IPC\_CREAT);

message.mesg\_type = 1;

printf("Write Data : ");

fgets(message.mesg\_text,MAX,stdin);

*// msgsnd to send message*

msgsnd(msgid, &message, sizeof(message),0);

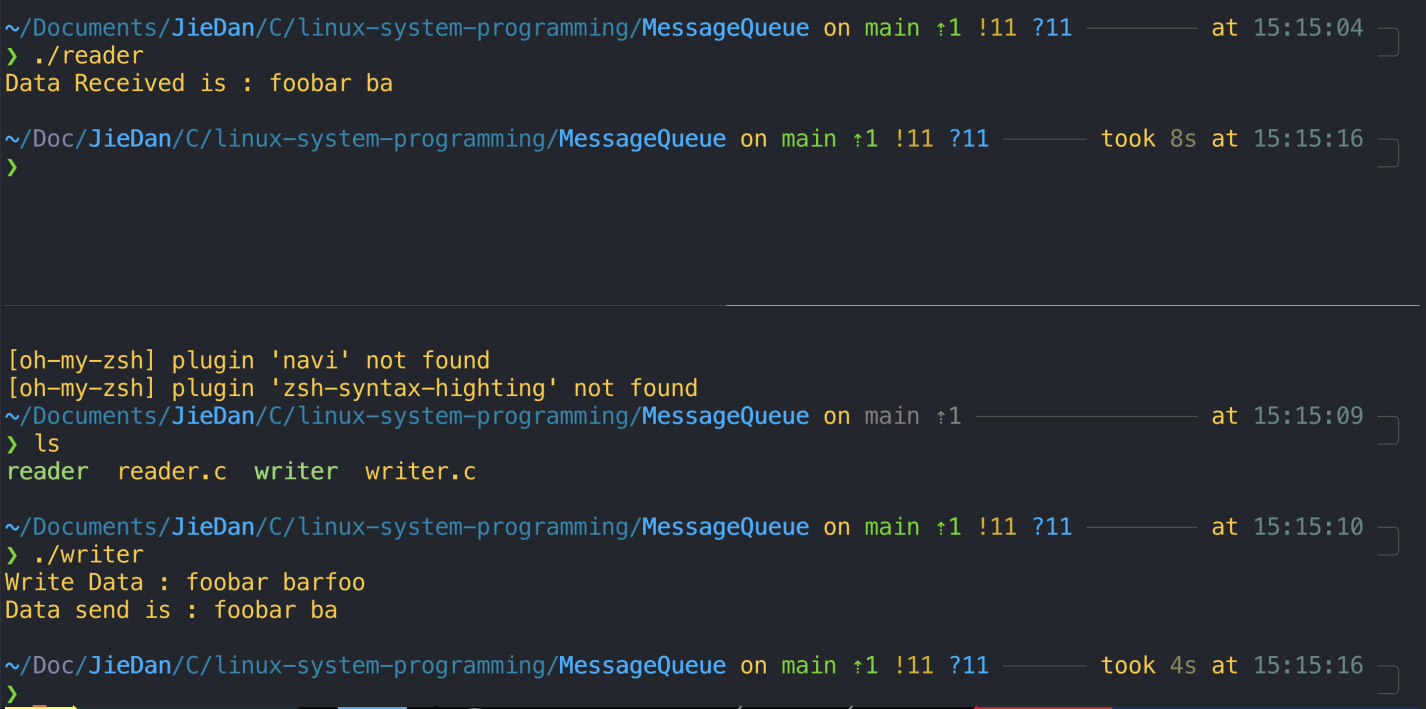
*// display the message*

printf("Data send is : %s \n", message.mesg\_text);

return 0;

}

**Result:**

****

1. **Explain and illustrate with how to create a Kernel Module with and example code?**

**Explain:**

**Explain:**

**Kernel use module\_init to initialize start function and use module\_exit to register function that will be called at the end of the kernel module.**

**Code:**

#include <linux/module.h> */\* Needed by all modules \*/*

#include <linux/kernel.h> */\* Needed for KERN\_INFO \*/*

#include <linux/init.h> */\* Needed for the macros \*/*

static int \_\_init hello\_start(void)

{

printk(KERN\_INFO "Loading hello module...\n");

printk(KERN\_INFO "Hello world\n");

return 0;

}

static void \_\_exit hello\_end(void)

{

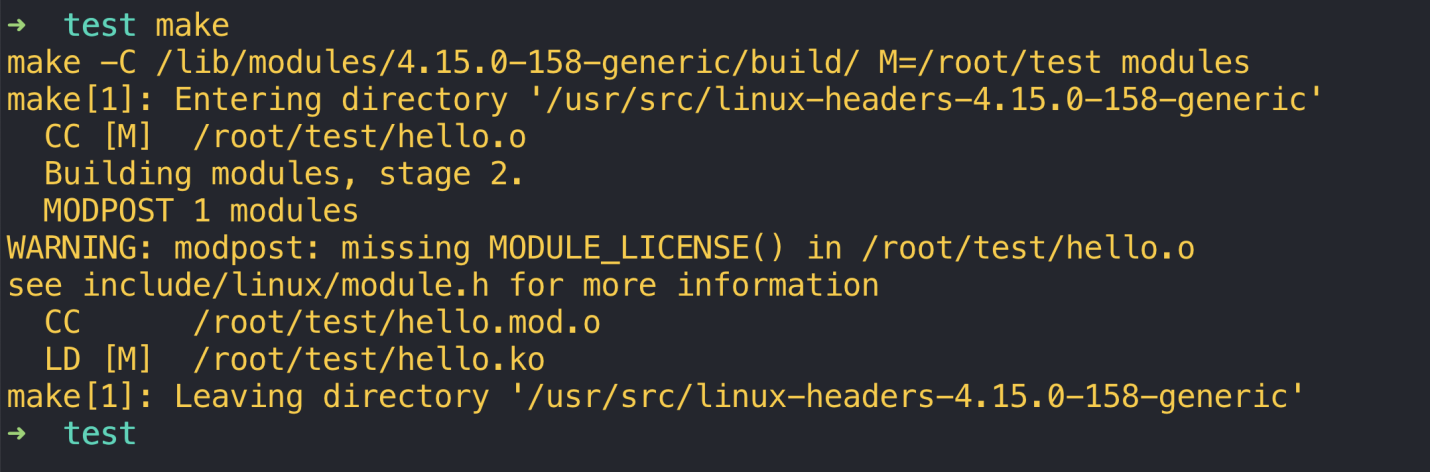
printk(KERN\_INFO "Goodbye Mr.\n");

}

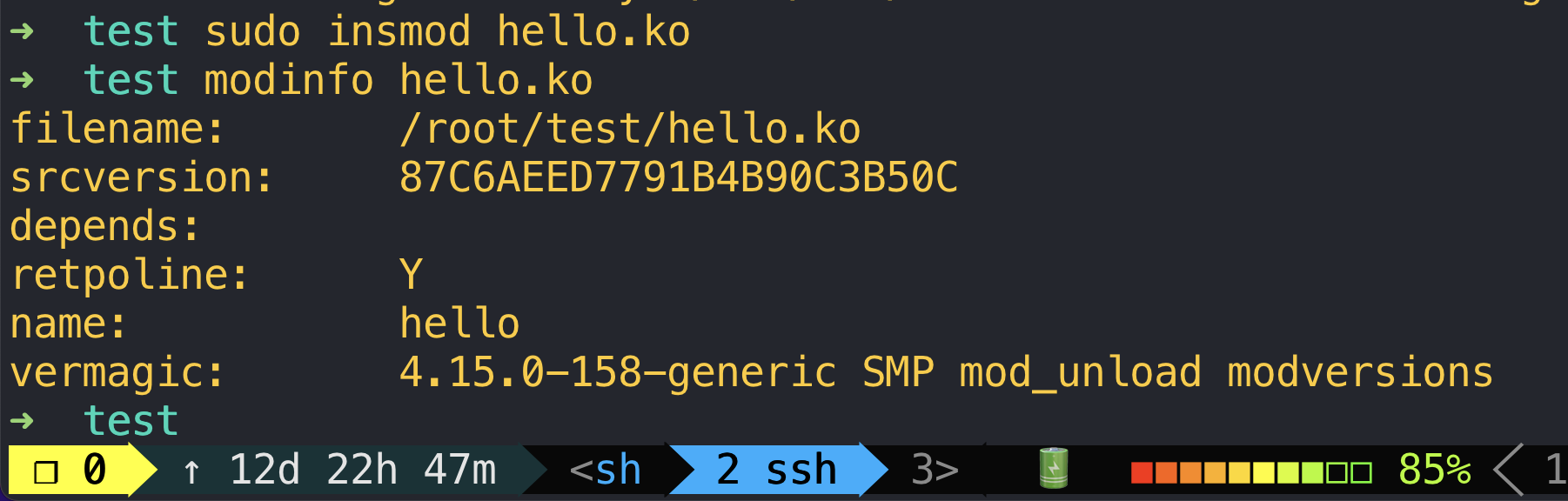
module\_init(hello\_start);

module\_exit(hello\_end);

**Compile kernel module:**

****

**Install module and see module information:**

****

1. **Explain and Illustrate How kernel threads are created with an example code ?**

**Explain:**

**Linux allows us to use kthread\_run to create a kernel thread.**

**Code:**

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/init.h>

#include <linux/kthread.h>

#include <linux/sched.h>

static int kthread\_func(void \*arg)

{

printk(KERN\_INFO "I am thread: %s[PID = %d]\n", current->comm, current->pid);

return 0;

}

static int \_\_init init\_func(void)

{

struct task\_struct \*ts1;

struct task\_struct \*ts2;

int err;

printk(KERN\_INFO "Starting 2 threads\n");

ts1 = kthread\_run(kthread\_func, NULL, "thread-1");

if (IS\_ERR(ts1)) {

printk(KERN\_INFO "ERROR: Cannot create thread ts1\n");

err = PTR\_ERR(ts1);

ts1 = NULL;

return err;

}

ts1 = kthread\_run(kthread\_func, NULL, "thread-1");

if (IS\_ERR(ts1)) {

printk(KERN\_INFO "ERROR: Cannot create thread ts1\n");

err = PTR\_ERR(ts1);

ts1 = NULL;

return err;

}

printk(KERN\_INFO "I am thread: %s[PID = %d]\n", current->comm, current->pid);

return 0;

}

static void \_\_exit exit\_func(void)

{

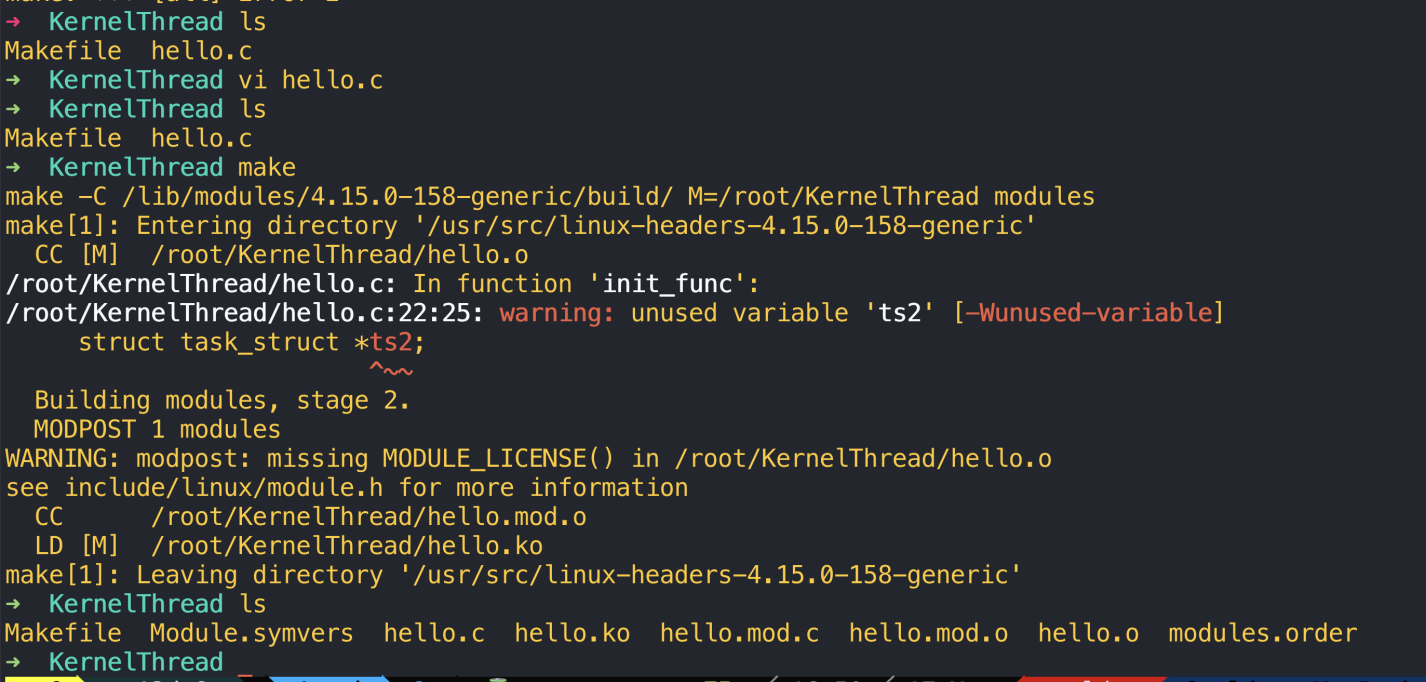
printk(KERN\_INFO "Exiting the module\n");

}

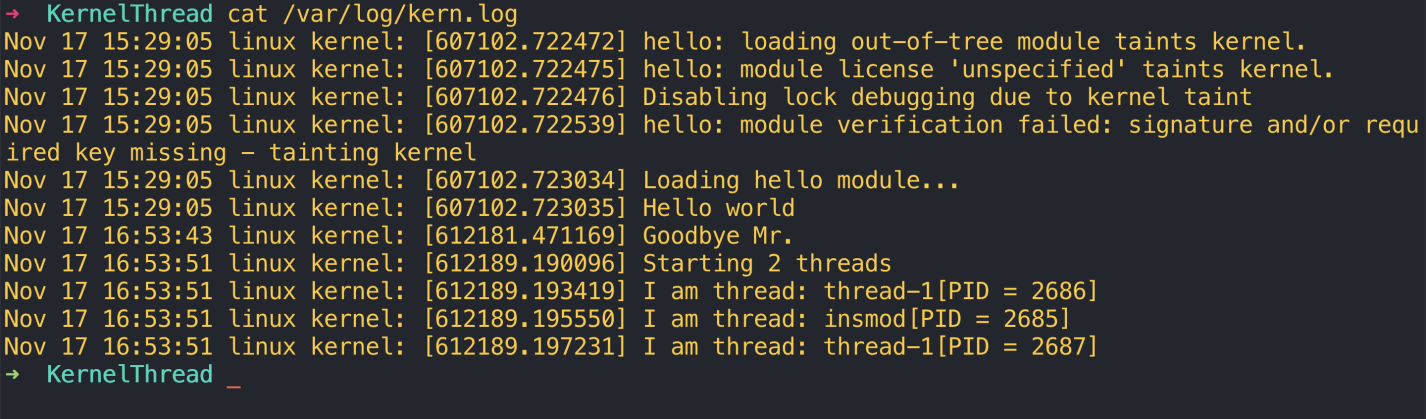
module\_init(init\_func);

module\_exit(exit\_func);

**Compile:**

****

**Result:**

****

1. **Write a Brief summary of the concepts that in involved in Linux system programming and your Understanding w.r.t to each Module.**

**After exercising, I’v leaned internal process communication between two process, such as socket, shared memory and message queue that Linux kernel provide. The API is easy to use, but it’s hard to understand its internal. I also learn that mutex is an object of struct, while semaphore is just a typedef of an integer. For makefile of linux kernel module, there are a lot on the web. Make sure that linux header is installed.**