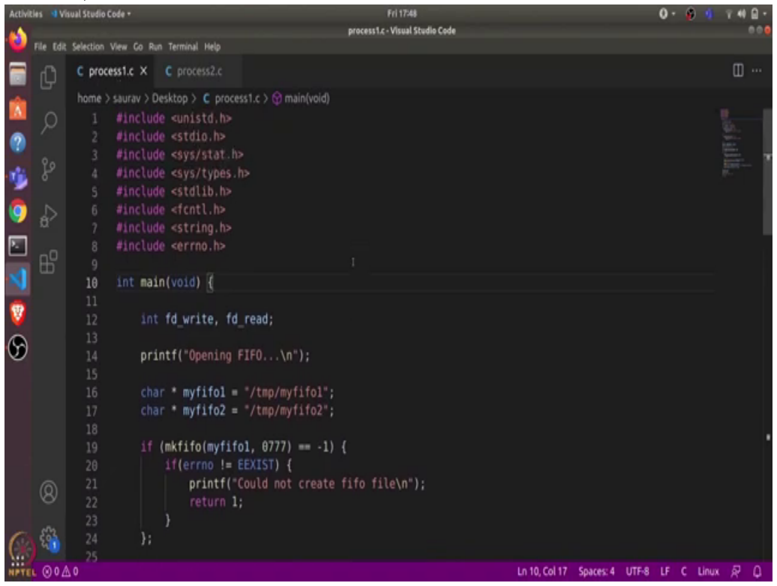
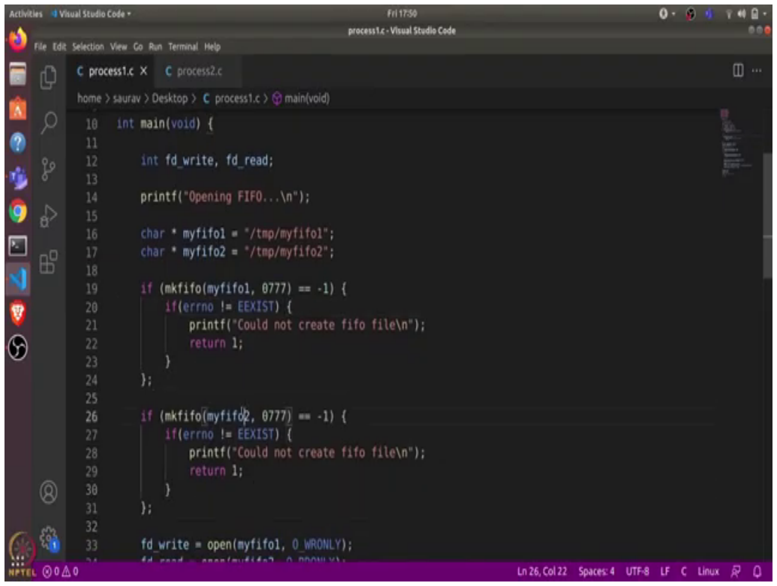
Design and Engineering of Computer Systems
Professor Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 44 (Week 6, Tutorial 2)
Inter Process Communication using fifo

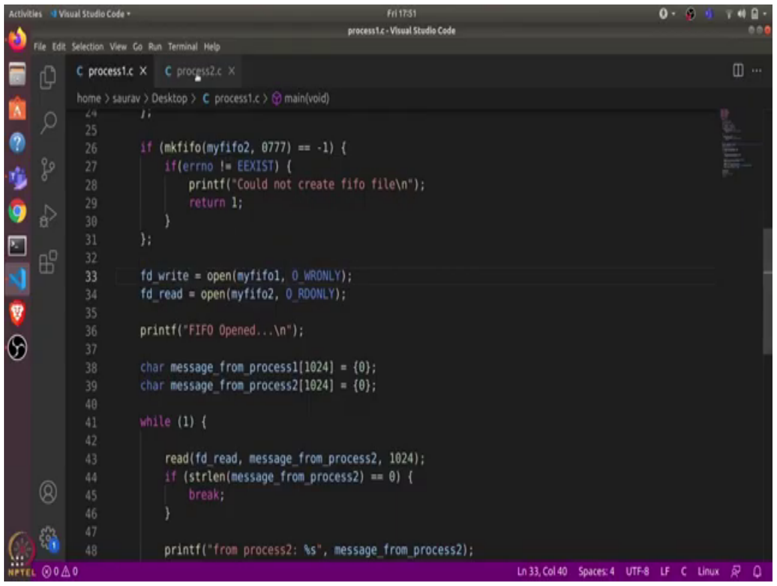
(Refer Slide Time: 00:19)


process1.c — Visual Studio Code
process1.c
process2.c
home > saurav > Desktop > process1.c > main(void)
Ln 10, Col 17 Spaces: 4 UTF-8 LF C Linux

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>

int main(void) {

    int fd_write, fd_read;

    printf("Opening FIFO...\n");

    char * myfifo1 = "/tmp/myfifo1";
    char * myfifo2 = "/tmp/myfifo2";

    if (mkfifo(myfifo1, 0777) == -1) {
        if(errno != EEXIST) {
            printf("Could not create fifo file\n");
            return 1;
        }
    };
```


process1.c — Visual Studio Code
process1.c
process2.c
home > saurav > Desktop > process1.c > main(void)
Ln 26, Col 22 Spaces: 4 UTF-8 LF C Linux

```c
int main(void) {

    int fd_write, fd_read;

    printf("Opening FIFO...\n");

    char * myfifo1 = "/tmp/myfifo1";
    char * myfifo2 = "/tmp/myfifo2";

    if (mkfifo(myfifo1, 0777) == -1) {
        if(errno != EEXIST) {
            printf("Could not create fifo file\n");
            return 1;
        }
    };

    if (mkfifo(myfifo2, 0777) == -1) {
        if(errno != EEXIST) {
            printf("Could not create fifo file\n");
            return 1;
        }
    };

    fd_write = open(myfifo1, O_WRONLY);
```



```
    };

    if (mkfifo(myfifo2, 0777) == -1) {
        if(errno != EEXIST) {
            printf("Could not create fifo file\n");
            return 1;
        }
    };

    fd_write = open(myfifo1, O_WRONLY);
    fd_read = open(myfifo2, O_RDONLY);

    printf("FIFO Opened...\n");

    char message_from_process1[1024] = {0};
    char message_from_process2[1024] = {0};

    while (1) {

        read(fd_read, message_from_process2, 1024);
        if (strlen(message_from_process2) == 0) {
            break;
        }

        printf("from process2: %s", message_from_process2);
```

Hi everyone, in this video we will learn about inter process communication. So, there are various mechanisms using which two processes can interact with each other. But what we will use here is the named pipes. So, we will run one process, process 1, and we will run another process called process 2. And we will send some messages from process 1 to process 2 using a named pipe. And then we will have another pipe using which will send messages back from process 2 to process 1.

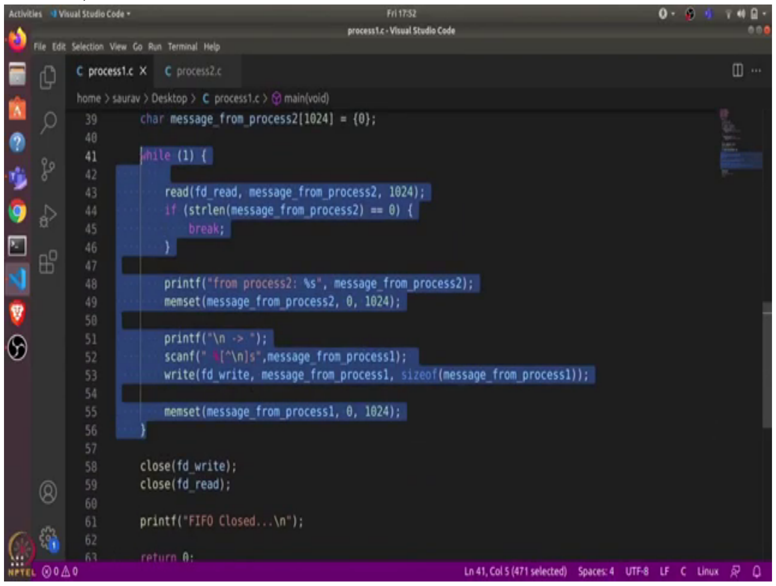
So, let us look at the code. So, this is code for process 1. So, we will open two pipes between process 1 and process 2, one is for writing and one is for reading. So, process 1 will write in one pipe from which process 2 will read. And there will be another pipe in which process 1 will write and process 2 will read. So, we have two file descriptors, one is for writing one is for reading, we print this opening fifo.

A named pipe is also called fifo and Linux. So, this fifo is just like a normal file, and we can see that in file explorer, but it acts like a named pipe. So, we first define just two paths, this is the path for our fifo file. So, this is inside a folder, they are myfifo1 and myfifo2, we use this mkfifo function to create the fifo file, its first argument is the path to the file and second is the permission. So, if we check if the return value is equal to -1 that means there were some error and here we are checking if the error number is not EEXIT.

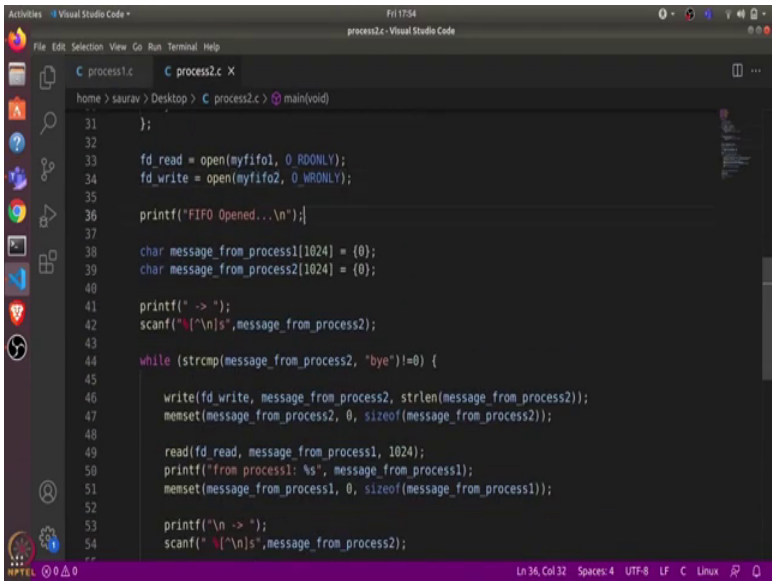
So, if it is EEXIT, so that would mean that this fifo was already there. And we do not worry because we just want that there is a fifo at this particular path which we can use. We similarly create the another fifo file. And then we open both these files. So, in process 1, we open FIFO1 for writing, and we open fifo 2 for reading.

So, we give the respective file descriptor to these variables. And if we look at process 2's code, we have similar code till this point where we define these myfifo1 myfifo2, we make these both fifos. But here, we use myfifo1 for reading, and myfifo2 for writing. So, let us continue. So, we print that fifo was open. And then we have two buffers. One is for process 1 and messages and another is for process 2 messages.

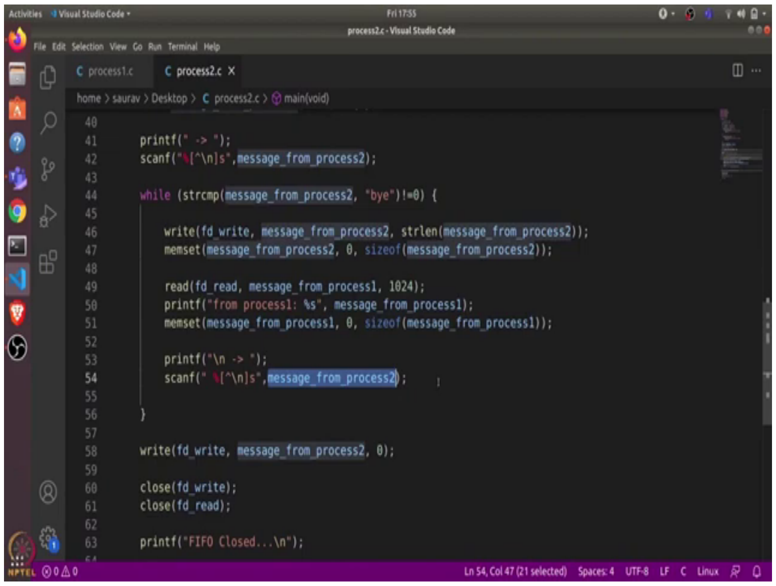
(Refer Slide Time: 02:46)


Activities
Visual Studio Code
Fri 17:52
process1.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
process1.c
process2.c
home > saurav > Desktop > process1.c > main(void)
Ln 41, Col 5 (471 selected) Spaces: 4 UTF-8 LF C Linux

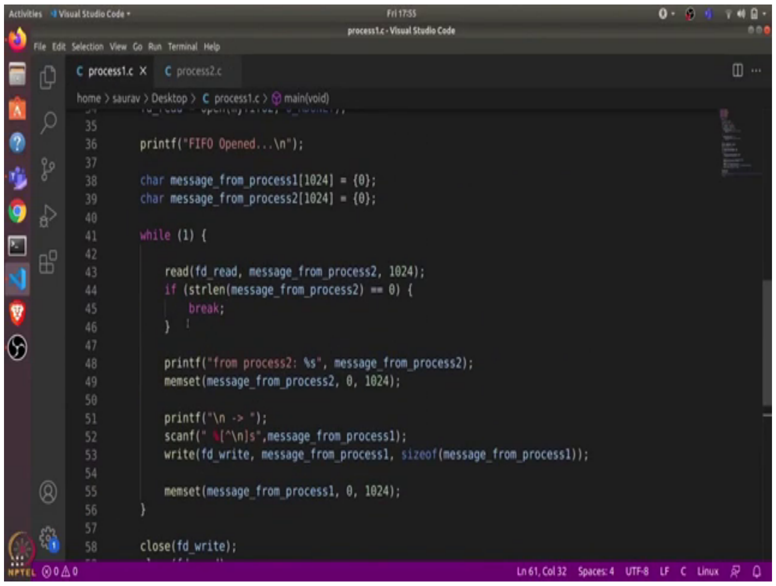
```c
    char message_from_process2[1024] = {0};

    while (1) {

        read(fd_read, message_from_process2, 1024);
        if (strlen(message_from_process2) == 0) {
            break;
        }

        printf("from process2: %s", message_from_process2);
        memset(message_from_process2, 0, 1024);

        printf("\n -> ");
        scanf(" %[^\n]s",message_from_process1);
        write(fd_write, message_from_process1, sizeof(message_from_process1));

        memset(message_from_process1, 0, 1024);
    }

    close(fd_write);
    close(fd_read);

    printf("FIFO Closed...\n");

    return 0;
```


Activities
Visual Studio Code
Fri 17:54
process2.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
process1.c
process2.c
home > saurav > Desktop > process2.c > main(void)
Ln 36, Col 32 Spaces: 4 UTF-8 LF C Linux

```c
    };

    fd_read = open(myfifo1, O_RDONLY);
    fd_write = open(myfifo2, O_WRONLY);

    printf("FIFO Opened...\n");

    char message_from_process1[1024] = {0};
    char message_from_process2[1024] = {0};

    printf(" -> ");
    scanf("%[^\n]s",message_from_process2);

    while (strcmp(message_from_process2, "bye")!=0) {

        write(fd_write, message_from_process2, strlen(message_from_process2));
        memset(message_from_process2, 0, sizeof(message_from_process2));

        read(fd_read, message_from_process1, 1024);
        printf("from process1: %s", message_from_process1);
        memset(message_from_process1, 0, sizeof(message_from_process1));

        printf("\n -> ");
        scanf(" %[^\n]s",message_from_process2);
```



```c
    printf(" -> ");
    scanf("%[^\n]s",message_from_process2);

    while (strcmp(message_from_process2, "bye")!=0) {

        write(fd_write, message_from_process2, strlen(message_from_process2));
        memset(message_from_process2, 0, sizeof(message_from_process2));

        read(fd_read, message_from_process1, 1024);
        printf("from process1: %s", message_from_process1);
        memset(message_from_process1, 0, sizeof(message_from_process1));

        printf("\n -> ");
        scanf(" %[^\n]s",message_from_process2);

    }

    write(fd_write, message_from_process2, 0);

    close(fd_write);
    close(fd_read);

    printf("FIFO Closed...\n");
```



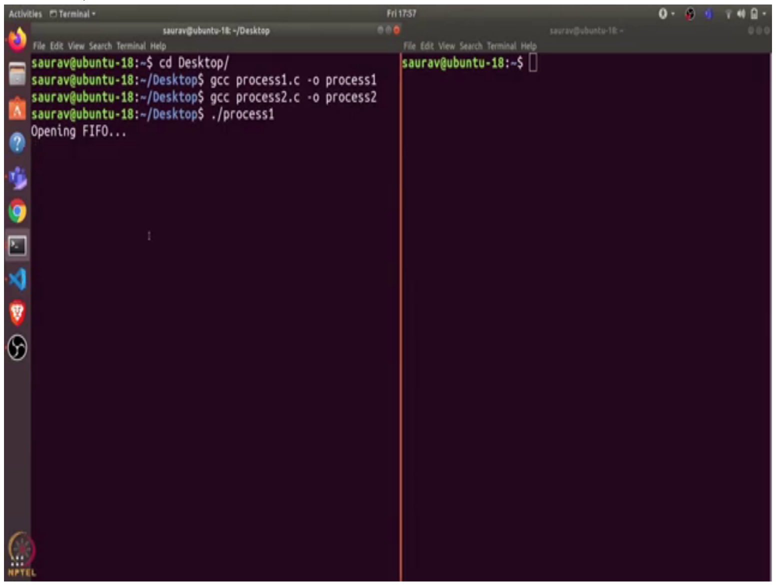
```c
    printf("FIFO Opened...\n");

    char message_from_process1[1024] = {0};
    char message_from_process2[1024] = {0};

    while (1) {

        read(fd_read, message_from_process2, 1024);
        if (strlen(message_from_process2) == 0) {
            break;
        }

        printf("from process2: %s", message_from_process2);
        memset(message_from_process2, 0, 1024);

        printf("\n -> ");
        scanf(" %[^\n]s",message_from_process1);
        write(fd_write, message_from_process1, sizeof(message_from_process1));

        memset(message_from_process1, 0, 1024);
    }

    close(fd_write);
```

Now, we execute this while loop. And we first read the message from fd_read, which means we want first message from process 2 and we read it in this buffer, we check if the message length is not 0. Then we continue and we print that this was the message received from process 2 we reset this message from process 2 buffer and then we take in input some new message from the terminal processor 1 we write it to the fd_write file descriptor.

So, here we are reading a message from the first pipe and here we are writing to the second pipe using the write function, then we reset this message from process 1 buffer. So, this will execute in a while loop. And in the end, we close both the file descriptors and we print before closed and return 0 and in process 2's code we have a similar code till this point. But here first we take an input some message from the user and then we write it to fd_write.

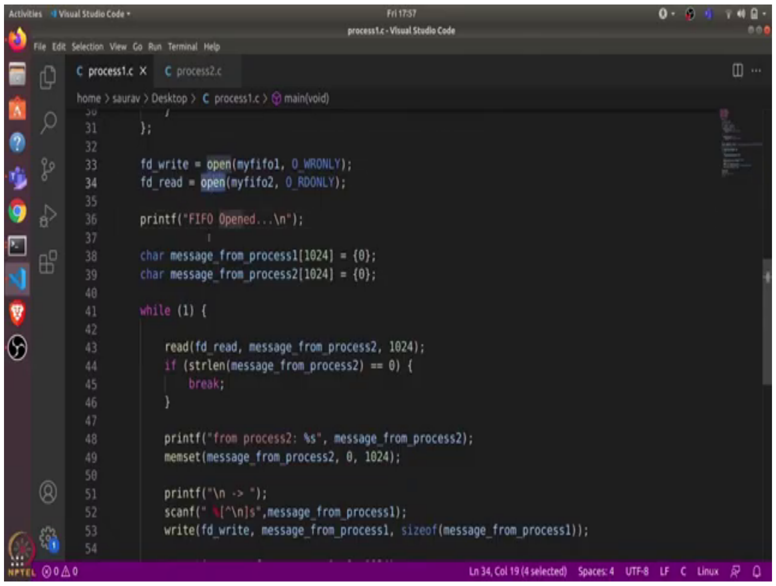
So, we send some message from pipe 1 to process 1 and process 2 reads that message and then we read a reply from process 1 using this read function and we print out that message and then we again take an input this message from process 2. And this we continue till we get by in process 2. And finally, we write an empty message to fd_write so that in process 1, this while loop can exit. So, the length of message from process 2 will be 0 and it will break and then we close both these file descriptors and we print a fifo closed.

(Refer Slide Time: 04:29)



```
saurav@ubuntu-18:~$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ gcc process1.c -o process1
saurav@ubuntu-18:~/Desktop$ gcc process2.c -o process2
saurav@ubuntu-18:~/Desktop$ ./process1
Opening FIFO...
```

```
saurav@ubuntu-18:~$
```

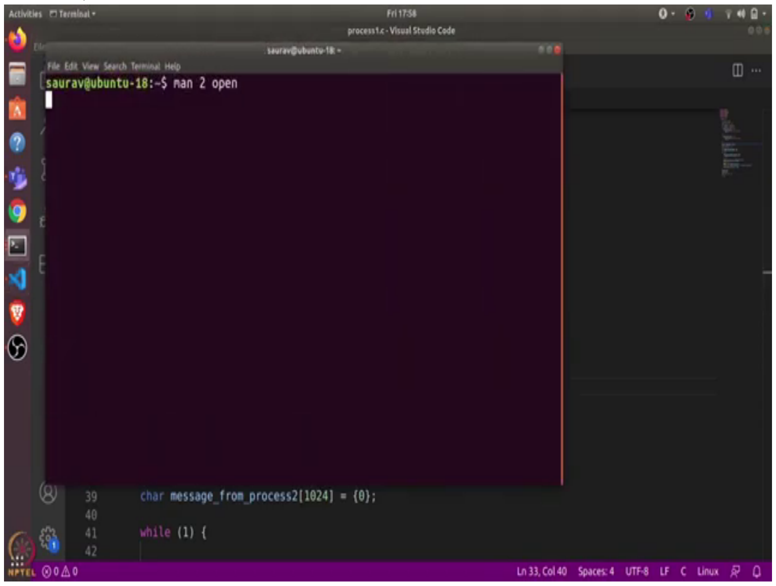


```c
    };

    fd_write = open(myfifo1, O_WRONLY);
    fd_read = open(myfifo2, O_RDONLY);

    printf("FIFO Opened...\n");

    char message_from_process1[1024] = {0};
    char message_from_process2[1024] = {0};

    while (1) {

        read(fd_read, message_from_process2, 1024);
        if (strlen(message_from_process2) == 0) {
            break;
        }

        printf("from process2: %s", message_from_process2);
        memset(message_from_process2, 0, 1024);

        printf("\n -> ");
        scanf(" %[^\n]s",message_from_process1);
        write(fd_write, message_from_process1, sizeof(message_from_process1));
```

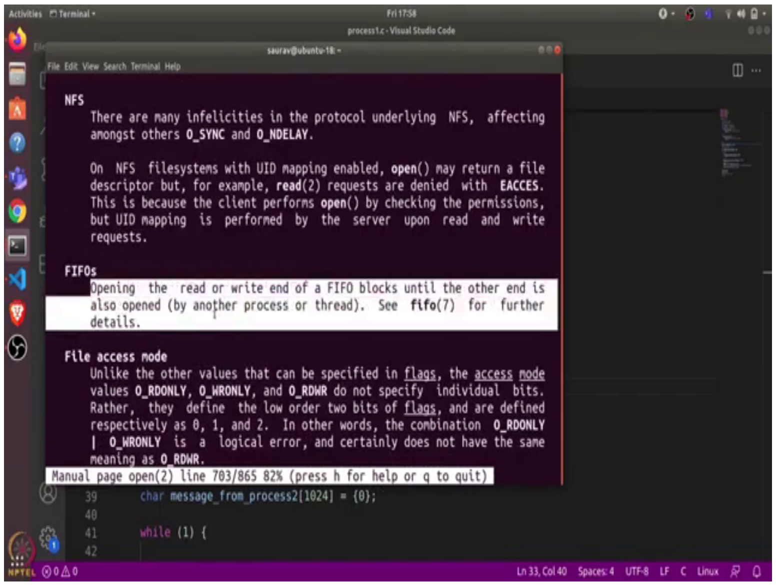
So, let us compile both of these programs and see the output. Let us open two terminals. I will compile process 1 and name the executable as process1. I will similarly compile, process2.c. And now I run process 1. So, here if we see this process 1 is stuck at opening fifo. Let us try to understand why it is stuck after opening fifo. Let us have a look at the code again.

So, what we are doing in process 1 is we are creating two files, we then open both these fifo files, one for writing another for reading, and then we print fifo open. But it did not print before open, which means that it is stuck somewhere here. So, it is stuck because in case of fifo the open calls blocks until there is another open for reading. So, it is stuck after this point.

(Refer Slide Time: 05:44)

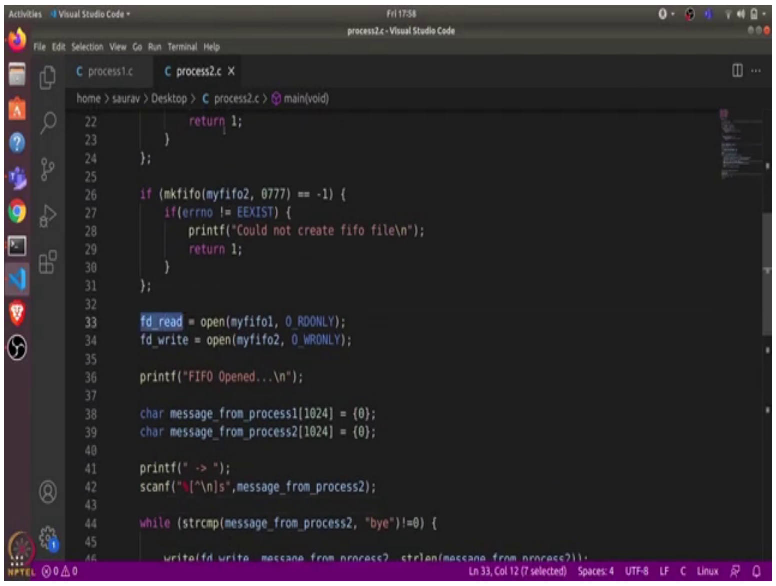


```
saurav@ubuntu-18:~$ man 2 open
```

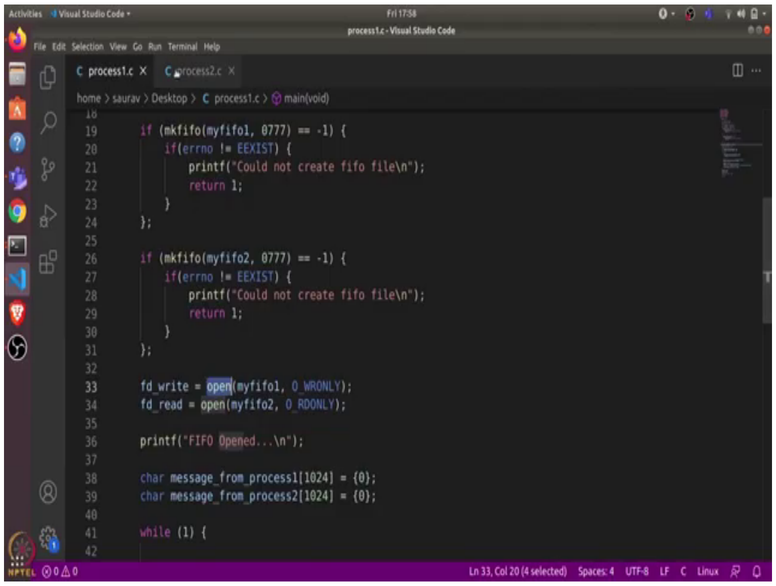


```
NFS
    There are many infelicities in the protocol underlying  NFS,  affecting
    amongst others O_SYNC and O_NDELAY.

    On  NFS  filesystems with UID mapping enabled, open() may return a file
    descriptor but, for example, read(2) requests are denied  with  EACCES.
    This is because the client performs open() by checking the permissions,
    but UID mapping  is  performed  by  the  server  upon  read  and  write
    requests.

FIFOs
    Opening  the  read or write end of a FIFO blocks until the other end is
    also opened (by another process or thread).  See  fifo(7)  for  further
    details.

File access mode
    Unlike the other values that can be specified in flags, the access mode
    values O_RDONLY, O_WRONLY, and O_RDWR do not specify  individual  bits.
    Rather,  they  define  the low order two bits of flags, and are defined
    respectively as 0, 1, and 2.  In other words, the combination  O_RDONLY
    |  O_WRONLY  is  a  logical error, and certainly does not have the same
    meaning as O_RDWR.
Manual page open(2) line 703/865 82% (press h for help or q to quit)
```

```c
    char message_from_process2[1024] = {0};

    while (1) {
```



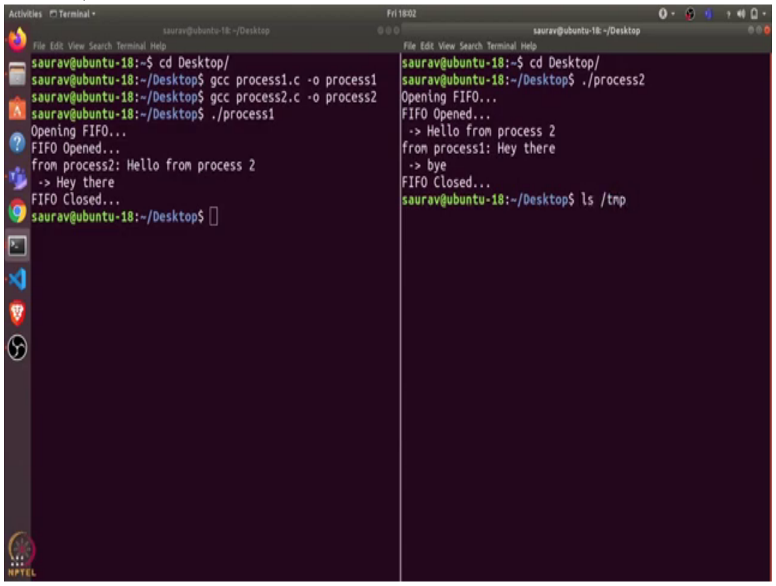
```c
                return 1;
            }
        };

        if (mkfifo(myfifo2, 0777) == -1) {
            if(errno != EEXIST) {
                printf("Could not create fifo file\n");
                return 1;
            }
        };

        fd_read = open(myfifo1, O_RDONLY);
        fd_write = open(myfifo2, O_WRONLY);

        printf("FIFO Opened...\n");

        char message_from_process1[1024] = {0};
        char message_from_process2[1024] = {0};

        printf(" -> ");
        scanf("%[^\n]s",message_from_process2);

        while (strcmp(message_from_process2, "bye")!=0) {

            write(fd_write, message_from_process2, strlen(message_from_process2));
```



```c
    if (mkfifo(myfifo1, 0777) == -1) {
        if(errno != EEXIST) {
            printf("Could not create fifo file\n");
            return 1;
        }
    };

    if (mkfifo(myfifo2, 0777) == -1) {
        if(errno != EEXIST) {
            printf("Could not create fifo file\n");
            return 1;
        }
    };

    fd_write = open(myfifo1, O_WRONLY);
    fd_read = open(myfifo2, O_RDONLY);

    printf("FIFO Opened...\n");

    char message_from_process1[1024] = {0};
    char message_from_process2[1024] = {0};

    while (1) {
```

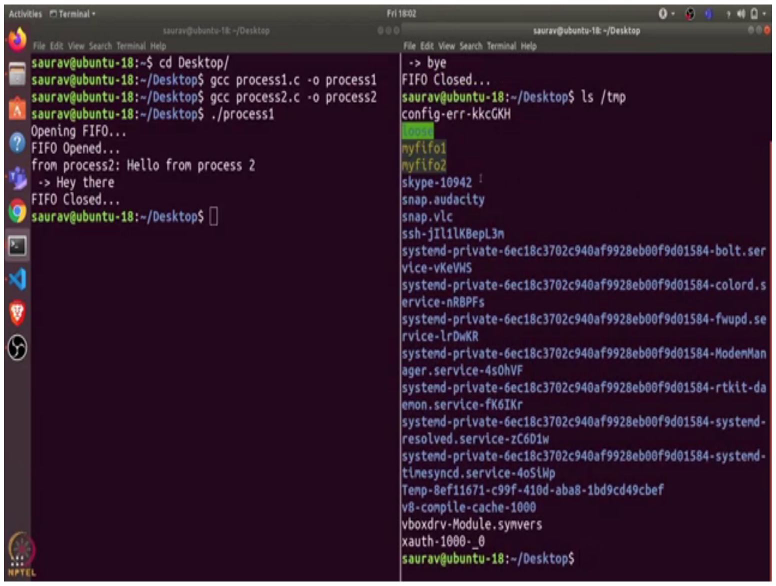
So, if we look at the man page of open, man2 open, and if we scroll down, so here it is mentioned that opening the read or write end of a fifo blocks until the other end is also opened. So, here this open call is blocking till we also open the reading end of this pipe in some other process. So, now if I execute process 2 only then can process 1, go ahead.

(Refer Slide Time: 06:12)



```
saurav@ubuntu-18:~$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ gcc process1.c -o process1
saurav@ubuntu-18:~/Desktop$ gcc process2.c -o process2
saurav@ubuntu-18:~/Desktop$ ./process1
Opening FIFO...
FIFO Opened...
from process2: Hello from process 2
 -> Hey there
FIFO Closed...
saurav@ubuntu-18:~/Desktop$
```

```
saurav@ubuntu-18:~$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ ./process2
Opening FIFO...
FIFO Opened...
 -> Hello from process 2
from process1: Hey there
 -> bye
FIFO Closed...
saurav@ubuntu-18:~/Desktop$ ls /tmp
```



```
saurav@ubuntu-18:~$ cd Desktop/
saurav@ubuntu-18:~/Desktop$ gcc process1.c -o process1
saurav@ubuntu-18:~/Desktop$ gcc process2.c -o process2
saurav@ubuntu-18:~/Desktop$ ./process1
Opening FIFO...
FIFO Opened...
from process2: Hello from process 2
 -> Hey there
FIFO Closed...
saurav@ubuntu-18:~/Desktop$
```

```
 -> bye
FIFO Closed...
saurav@ubuntu-18:~/Desktop$ ls /tmp
config-err-kkcGKH
loose
myfifo1
myfifo2
skype-10942
snap.audacity
snap.vlc
ssh-jIl1lKBepL3m
systemd-private-6ec18c3702c940af9928eb00f9d01584-bolt.ser
vice-vKeVWS
systemd-private-6ec18c3702c940af9928eb00f9d01584-colord.s
ervice-nRBPFs
systemd-private-6ec18c3702c940af9928eb00f9d01584-fwupd.se
rvice-lrDwKR
systemd-private-6ec18c3702c940af9928eb00f9d01584-ModemMan
ager.service-4sOhVF
systemd-private-6ec18c3702c940af9928eb00f9d01584-rtkit-da
emon.service-fK6IKr
systemd-private-6ec18c3702c940af9928eb00f9d01584-systemd-
resolved.service-zC6D1w
systemd-private-6ec18c3702c940af9928eb00f9d01584-systemd-
timesyncd.service-4oSiWp
Temp-8ef11671-c99f-410d-aba8-1bd9cd49cbef
v8-compile-cache-1000
vboxdrv-Module.symvers
xauth-1000-_0
saurav@ubuntu-18:~/Desktop$
```

So, let us again open these terminals. So, here I will execute process 2. And once I press enter, then you will see that it will print out opened fifo. So, we have both the fifo opened in both the processes. And process 2 is showing us this cursor for input. So, let us send some message hello from process 2. And we receive that message here in process 1.

So, here we can send messages back and forth between process 1 and process 2, using two named pipes. We can in fact see these fifo files in the tmp folder. So, let us execute this ls tmp. So, here you can see these two files, myfifo1 and myfifo2. So, these are those two fifo files. So, that is it for this video. Thanks, and have a nice day.