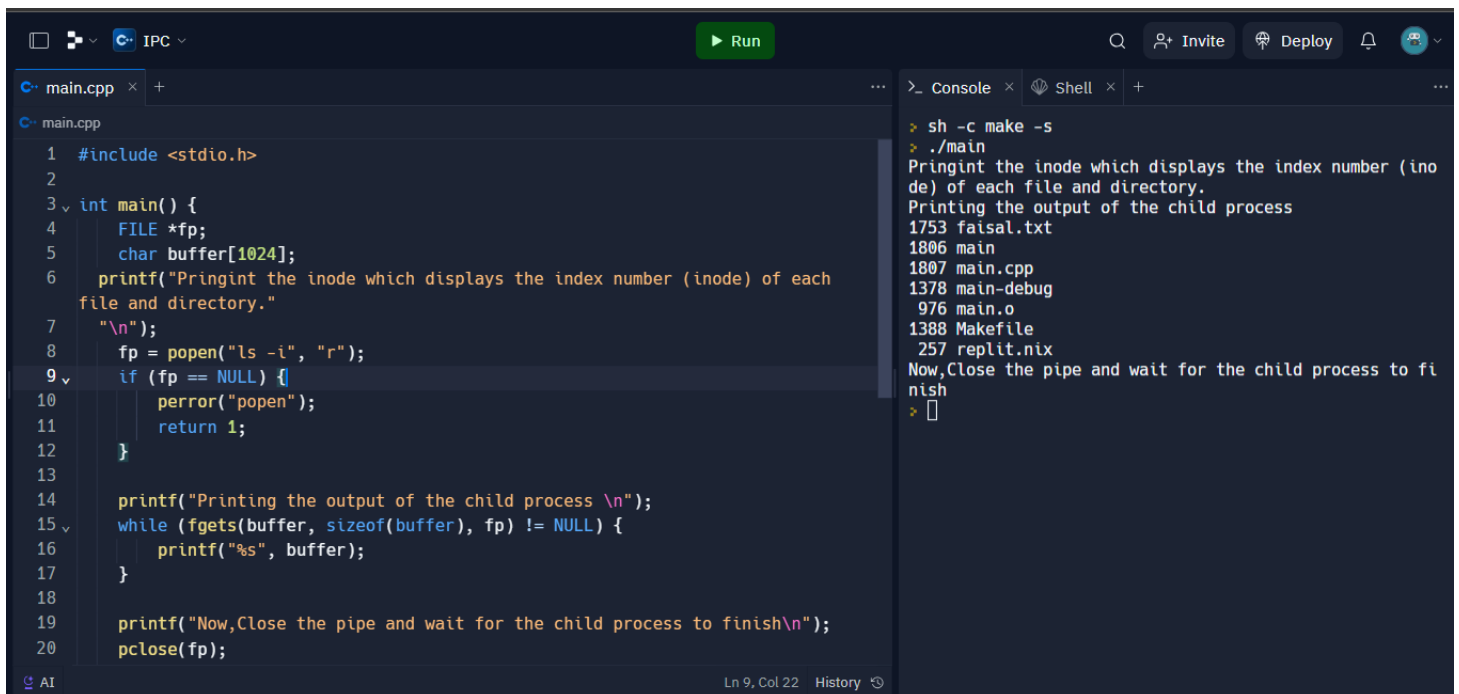


Assignment 07

Mohammad Faisal Sayed

2023PCS0034

1. popen() and pclose()
 - a. Scenario: You have a parent process that needs to execute a child process to perform a specific task
 - b. Explanation: This program uses **popen()** to execute the "**ls -i**" command in a child process and reads the output of the child process line by line, printing it to the console.
 - c. -i known as inode which displays the index number (inode) of each file and directory.

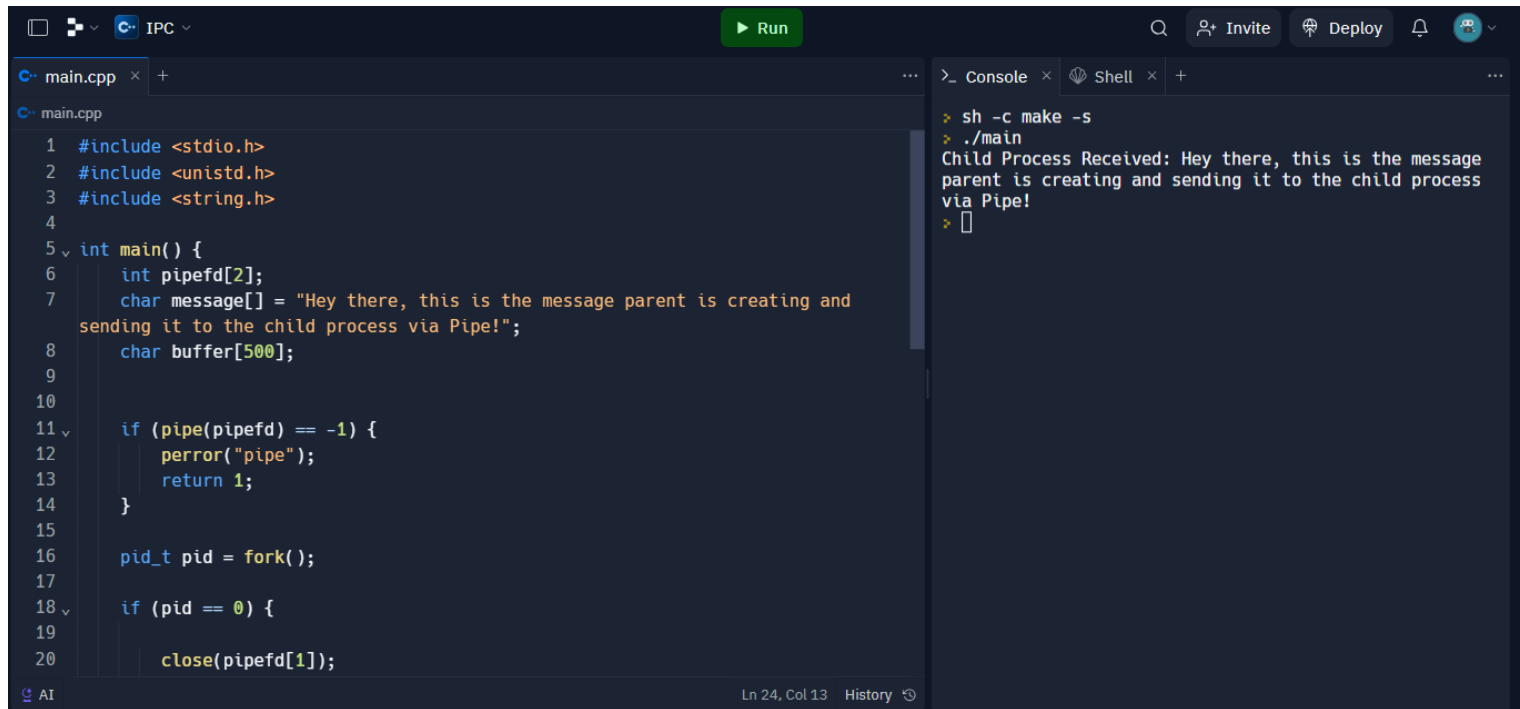


```
1 #include <stdio.h>
2
3 int main() {
4     FILE *fp;
5     char buffer[1024];
6     printf("Print the inode which displays the index number (inode) of each
7     file and directory.\n");
8     fp = popen("ls -i", "r");
9     if (fp == NULL) {
10         perror("popen");
11         return 1;
12     }
13
14     printf("Printing the output of the child process \n");
15     while (fgets(buffer, sizeof(buffer), fp) != NULL) {
16         printf("%s", buffer);
17     }
18
19     printf("Now, Close the pipe and wait for the child process to finish\n");
20     pclose(fp);
}
```

```
> sh -c make -s
> ./main
Print the inode which displays the index number (inode) of each file and directory.
Printing the output of the child process
1753 faisal.txt
1806 main
1807 main.cpp
1378 main-debug
976 main.o
1388 Makefile
257 replit.nix
Now, Close the pipe and wait for the child process to finish
>
```

2. pipe()

- Scenario: You want to create a simple pipe to send a message from one process to another.
- Explanation:
- This program creates a pipe using `pipe()` and forks into a parent and child process. The parent process writes the message "Hello, Pipe!" to the pipe, and the child process reads from the pipe and prints the received message.



```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <string.h>
4
5 int main() {
6     int pipefd[2];
7     char message[] = "Hey there, this is the message parent is creating and
    sending it to the child process via Pipe!";
8     char buffer[500];
9
10
11     if (pipe(pipefd) == -1) {
12         perror("pipe");
13         return 1;
14     }
15
16     pid_t pid = fork();
17
18     if (pid == 0) {
19         close(pipefd[1]);
20
```

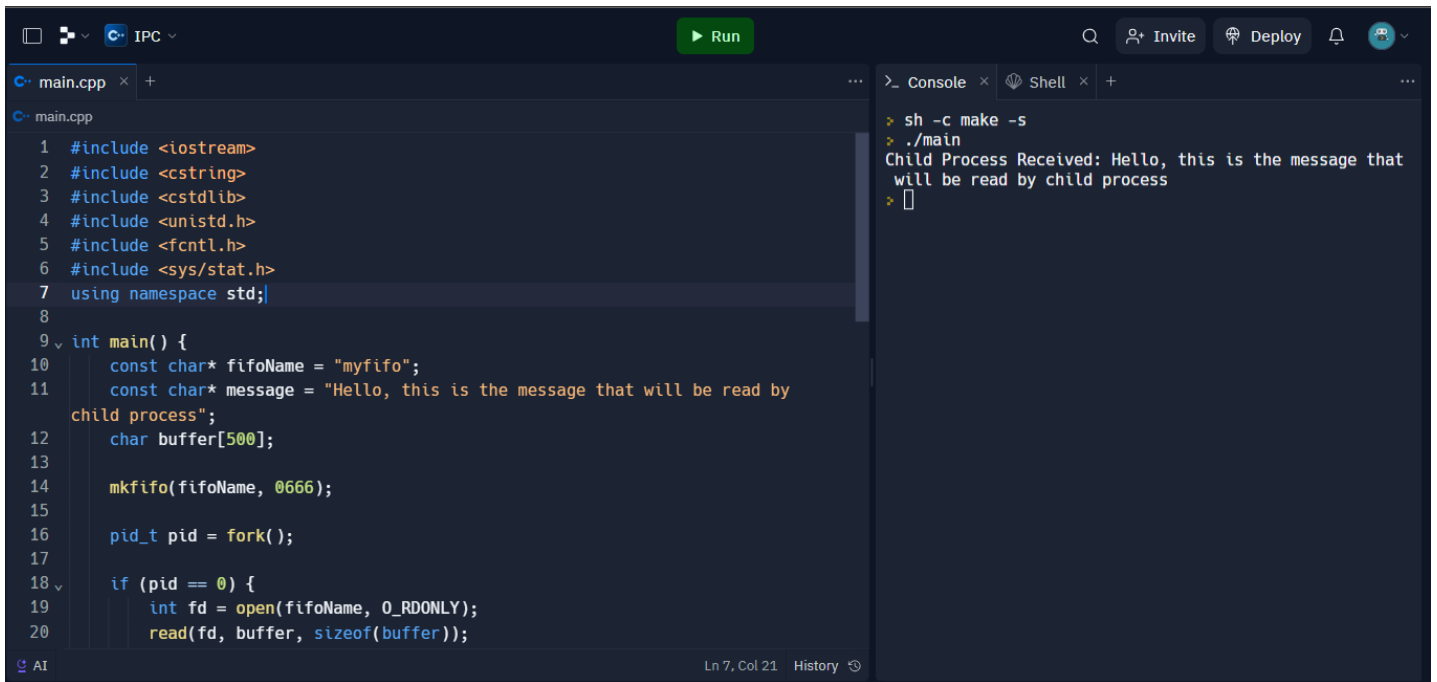
Console

```
> sh -c make -s
> ./main
Child Process Received: Hey there, this is the message
parent is creating and sending it to the child process
via Pipe!
> 
```

Ln 24, Col 13 History

3. mkfifo()

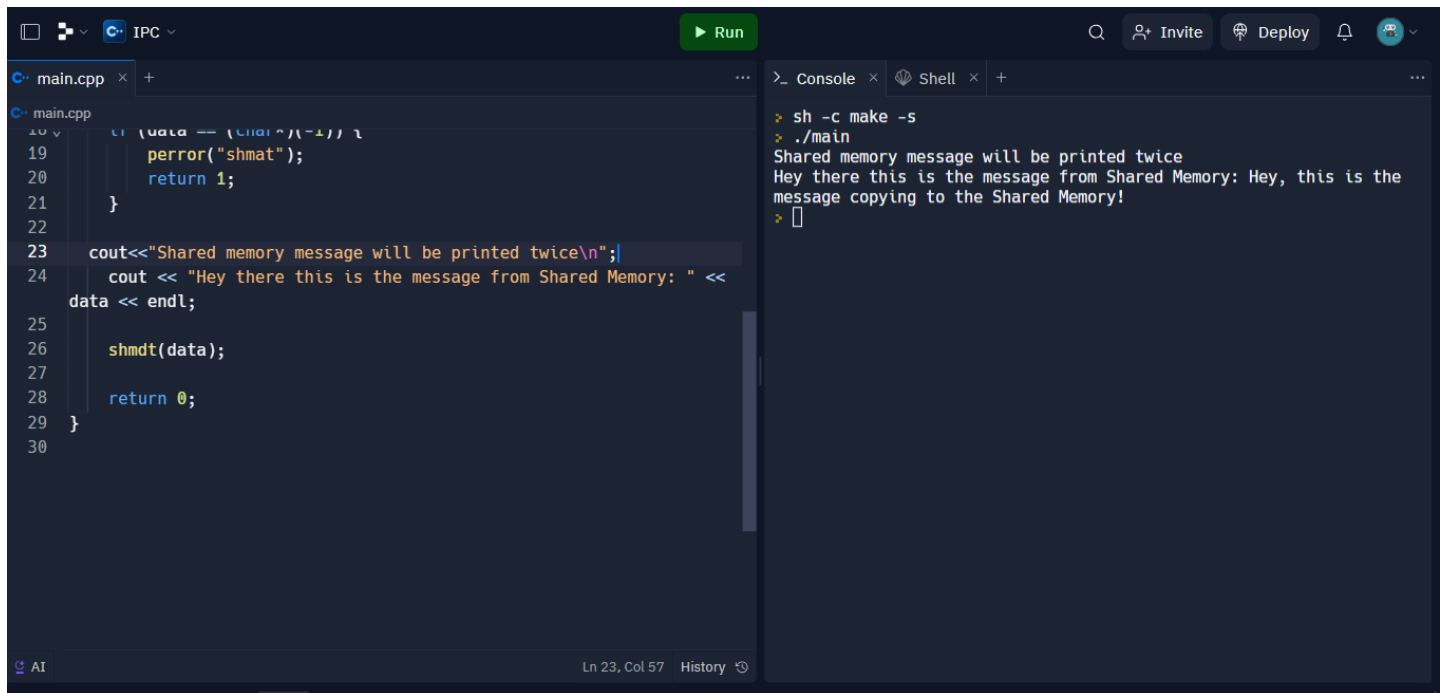
- a. It creates a named pipe which can be used exactly like a file. So, if you know how to read/write in a file this is a convenient method for IPC
- b. Note: The FIFO pipe works in blocked mode(by default) i.e., the writing process must be present on one end while the reading process must be present on the other side at the same time else the communication will not happen. Operating the FIFO special file in non-blocking mode is also possible.



```
1 #include <iostream>
2 #include <cstring>
3 #include <cstdlib>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <sys/stat.h>
7 using namespace std;
8
9 int main() {
10     const char* fifoName = "myfifo";
11     const char* message = "Hello, this is the message that will be read by
child process";
12     char buffer[500];
13
14     mkfifo(fifoName, 0666);
15
16     pid_t pid = fork();
17
18     if (pid == 0) {
19         int fd = open(fifoName, O_RDONLY);
20         read(fd, buffer, sizeof(buffer));
```

```
> sh -c make -s
> ./main
Child Process Received: Hello, this is the message that
will be read by child process
> []
```

4. Shared memory

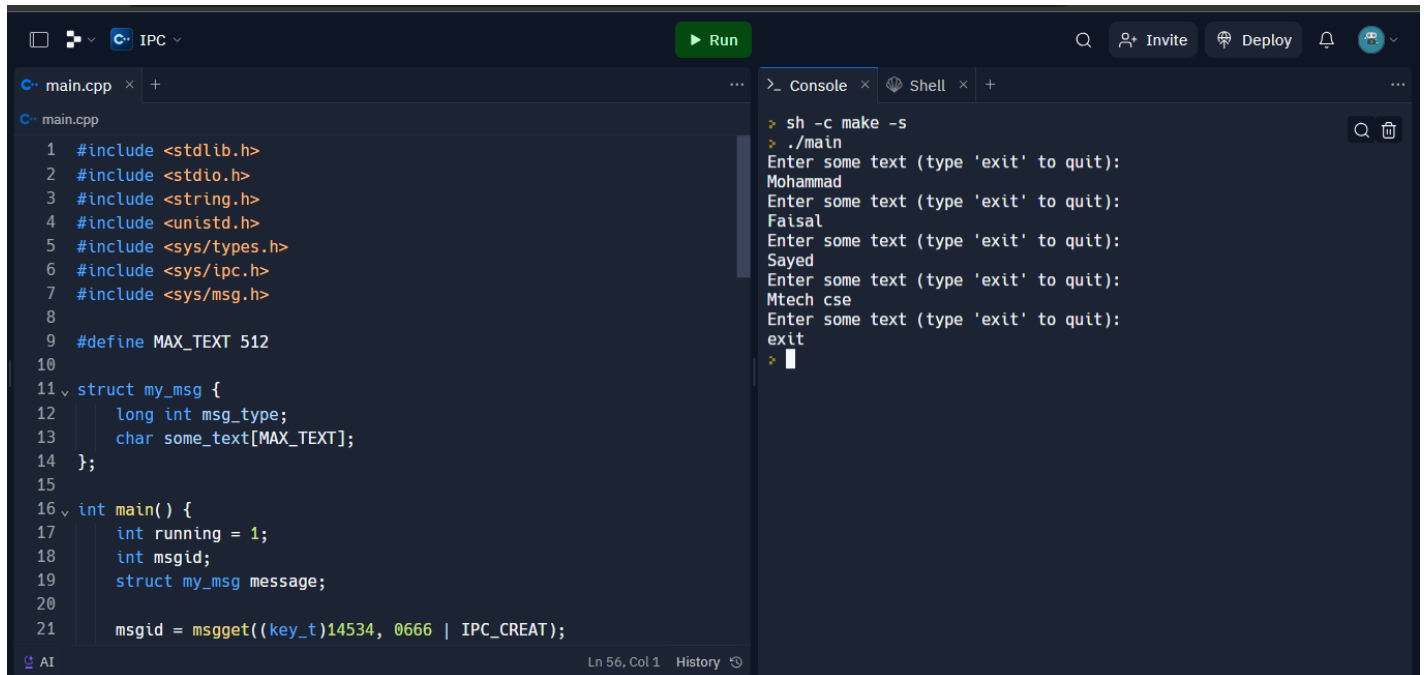


```
main.cpp x +
main.cpp
18  if (data == (char *)(-1)) {
19      perror("shmat");
20      return 1;
21  }
22
23  cout<<"Shared memory message will be printed twice\n";
24  cout << "Hey there this is the message from Shared Memory: " <<
data << endl;
25
26  shmdt(data);
27
28  return 0;
29 }
30
```

```
Console x Shell x +
> sh -c make -s
> ./main
Shared memory message will be printed twice
Hey there this is the message from Shared Memory: Hey, this is the
message copying to the Shared Memory!
>
```

Ln 23, Col 57 History

5. Message passing



The image shows a code editor with a C++ file named `main.cpp` and a terminal window showing the execution of the program.

main.cpp

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/ipc.h>
7 #include <sys/msg.h>
8
9 #define MAX_TEXT 512
10
11 struct my_msg {
12     long int msg_type;
13     char some_text[MAX_TEXT];
14 };
15
16 int main() {
17     int running = 1;
18     int msgid;
19     struct my_msg message;
20
21     msgid = msgget((key_t)14534, 0666 | IPC_CREAT);
```

Console Output

```
> sh -c make -s
> ./main
Enter some text (type 'exit' to quit):
Mohammad
Enter some text (type 'exit' to quit):
Faisal
Enter some text (type 'exit' to quit):
Sayed
Enter some text (type 'exit' to quit):
Mtech cse
Enter some text (type 'exit' to quit):
exit
>
```