**Name: Aliyan Ahmed Cheema**

**Reg# FA22-BCE-028(B)**

**Objectives**

* To perform message passing between related processes (parent and child) using an unnamed pipe.
* To perform message passing among a chain of processes using unnamed pipes.
* To perform message passing among fans of processes using unnamed pipes.
* To perform message passing between unrelated processes using a named pipe (Special File).

**Pre-Lab Theory**

**1. pipe() synopsis:**

int pipe(int fd[]) **Description:**

The inter-process communication channel is created and two descriptors are allocated and stored in the 2-element integer array provided as an argument in the pipe() function. **file descriptor fd[0]:**

Represents a handle for reading from the pipe. It is the index of the entry created in the process file table. **file descriptor fd[1]:**

Represents a handle for writing to the pipe. It is the index of the entry created in the process file table.

**Default Descriptors in the process file table with symbolic names:**

**Standard output:** STDOUT\_FILENO

**Standard input:** STDIN\_FILENO

**Standard Error:** STDERR\_FILENO

**return**: Negative value if the pipe(special file) is not created in case of an error **Named Pipe: Special File**

#include <sys/stat.h>

int mkfifo(const char \*path, mode\_t mode); Description:

The mkfifo() function shall create a new FIFO special file named by the pathname pointed to by the path. The file permission bits of the new FIFO shall be initialized from mode. The file permission bits of the mode argument shall be modified by the process file creation mask.

**Return**: 0 on successful creation of named pipe and -1 in case of an error Example Usage:

#include <sys/types.h>

#include <sys/stat.h>

int status;

...

status = mkfifo("/home/OS/Lab4/myfifo", S\_IWUSR | S\_IRUSR |

S\_IRGRP | S\_IROTH);

fd = Open(("/home/OS/Lab4/myfifo",O\_RDWR); read(fd,….,….) write(fd,…,….)

**In-Lab Tasks**

header file: unistd.h, stdio.h

System call: pipe(), fork() read(), write(), open(), close() Task1(a)

Include the header files and the following line of code in your program. Compile and run the code and describe its behavior.

|  |
| --- |
| Int fd[2];  char buffer\_p[] = “Welcome”;  char buffer\_c[10];  If (Pipe(fd) > 0)  Printf(“Pipe is successfully created\n”);  Else  Printf(“Error”); exit();  If ((pid = fork()) > 0)  Printf(“Child Process created\n”) write(fd[1], buffer, sizeof(buffer);  Else if(pid == 0) read(fd[0], buffer\_c, sizeof(buffer);  printf(“%d Received %s from Parent Process %d,getpid(), buffer\_c,getppid() ); |

Brief the working/output:

|  |
| --- |
| Pipe is created successfully  Child Process created  50024 received Welcome from Parent Process 1373 |

Task 1(b)

Include the header files, declare the required variables, extend the code in Task 1(a) to create a two way communication between parent and child process. The child process after receiving and displayed the “Welcome” message from the parent send the “Thanks” message to the parent. Parent displays the message received from the child process to its Standard Output.

Parent Output format:

<process\_id> received Thanks from <child\_process\_id> Child Output format:

<process\_id> received Welcome from <parent\_process\_id>

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  int main() {  int fd[2], fd2[2];  char buffer\_p[] = "Welcome";  char buffer\_c[10], buffer\_t[] = "Thanks";  pid\_t pid;  pipe(fd); // Parent to Child  pipe(fd2); // Child to Parent  if ((pid = fork()) > 0) {  // Parent process  write(fd[1], buffer\_p, sizeof(buffer\_p));  read(fd2[0], buffer\_c, sizeof(buffer\_c));  printf("%d received %s from %d\n", getpid(), buffer\_c, pid);  } else if (pid == 0) {  // Child process  read(fd[0], buffer\_c, sizeof(buffer\_c));  printf("%d received %s from %d\n", getpid(), buffer\_c, getppid());  write(fd2[1], buffer\_t, sizeof(buffer\_t));  }  return 0;  } |

Output:

57187 received Welcome from 57186

57186 received Thanks from 57187

Task 2(a)

Include the header files, declare the required variables, and extend the code in Task 1(a), and Task 1(b) to create a two-way message relay system between a chain of processes. Every child process after receiving and displaying the “Welcome” message from the parent replies with the “Thanks” message and forwards the “Welcome” message to the child process in the chain.

Parent Output format:

<process\_id> received Thanks from <child\_process\_id> Child Output format:

<process\_id> received Welcome from <parent\_process\_id>

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  int main() {  int fd1[2], fd2[2];  char buffer\_p[] = "Welcome";  char buffer\_c[10], buffer\_t[] = "Thanks";  pid\_t pid1, pid2;  pipe(fd1); // Pipe for parent -> child1  pipe(fd2); // Pipe for child1 -> child2  pid1 = fork(); // First child process  if (pid1 > 0) {  // Parent process  write(fd1[1], buffer\_p, sizeof(buffer\_p)); // Send "Welcome" to child1  wait(NULL); // Wait for child1  read(fd1[0], buffer\_c, sizeof(buffer\_c)); // Read "Thanks" from child1  printf("%d received %s from %d\n", getpid(), buffer\_c, pid1);  } else if (pid1 == 0) {  // Child1 process  read(fd1[0], buffer\_c, sizeof(buffer\_c)); // Read "Welcome" from parent  printf("%d received %s from %d\n", getpid(), buffer\_c, getppid());  write(fd2[1], buffer\_p, sizeof(buffer\_p)); // Send "Welcome" to child2  write(fd1[1], buffer\_t, sizeof(buffer\_t)); // Send "Thanks" to parent  pid2 = fork(); // Second child process (child2)  if (pid2 == 0) {  // Child2 process  read(fd2[0], buffer\_c, sizeof(buffer\_c)); // Read "Welcome" from child1  printf("%d received %s from %d\n", getpid(), buffer\_c, getppid());  write(fd2[1], buffer\_t, sizeof(buffer\_t)); // Send "Thanks" to child1  }  wait(NULL); // Wait for child2  }  return 0;  }  Output: 77542 received Welcome from 77539  77543 received Welcome from 77542  77539 received Thanks from 77542 |

Task 2(b)

Include the header files, declare the required variables, and extend the code in Task 1(a), and Task 1(b) to create a two-way message relay system between a Fan of processes. Every child process after receiving and displaying the “Welcome” message from the parent replies with the “Thanks” message. The parent process displays all the messages received with the sender process ID.

Parent Output Format:

<process\_id> received Thanks from <Child\_process\_id> Child Output Format:

<process\_id> received Welcome from <Parent\_Process\_Id>

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  int main() {  int fd[2], fd\_child[2], i;  char buffer\_p[] = "Welcome";  char buffer\_c[10], buffer\_t[] = "Thanks";  pid\_t pid[3];  pipe(fd); // Pipe for parent -> child  for (i = 0; i < 3; i++) {  pipe(fd\_child); // Pipe for each child -> parent  pid[i] = fork();  if (pid[i] == 0) {  // Child process  read(fd[0], buffer\_c, sizeof(buffer\_c)); // Read "Welcome" from parent  printf("%d received %s from %d\n", getpid(), buffer\_c, getppid());  write(fd\_child[1], buffer\_t, sizeof(buffer\_t)); // Send "Thanks" to parent  exit(0);  }  }  // Parent process  for (i = 0; i < 3; i++) {  write(fd[1], buffer\_p, sizeof(buffer\_p)); // Send "Welcome" to child[i]  read(fd\_child[0], buffer\_c, sizeof(buffer\_c)); // Read "Thanks" from child[i]  printf("%d received %s from %d\n", getpid(), buffer\_c, pid[i]);  }  return 0;  }  Output:  91644 received Welcome from 91641  91641 received Thanks from 91642  91643 received Welcome from 91641 |

Task 3

Create a two-way communication channel between two unrelated processes (Client and Server) using a named pipe (Fifo). The Client Process gets some text from standard input and sends it to the server process with its process Id. The server process displays the message received from the client and replies with a “Welcome” Message with its process id. The client displays the “Welcome” message received from the server and server process id. Both Processes use the same named pipe for message exchange.

**Server Side code:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

int main() {

int fd;

char buffer[100];

char \*fifo = "/tmp/myfifo";

mkfifo(fifo, 0666); // Create named pipe

while (1) {

fd = open(fifo, O\_RDONLY); // Open FIFO for reading

read(fd, buffer, sizeof(buffer)); // Read from client

printf("Server (PID %d) received: %s\n", getpid(), buffer);

close(fd);

fd = open(fifo, O\_WRONLY); // Open FIFO for writing

write(fd, "Welcome from server", sizeof("Welcome from server")); // Send message to client

close(fd);

}

return 0;

}

**Client Side Code:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

int main() {

int fd;

char buffer[100];

char \*fifo = "/tmp/myfifo";

while (1) {

fd = open(fifo, O\_WRONLY); // Open FIFO for writing

sprintf(buffer, "Hello from client (PID %d)", getpid());

write(fd, buffer, sizeof(buffer)); // Send message to server

close(fd);

fd = open(fifo, O\_RDONLY); // Open FIFO for reading

read(fd, buffer, sizeof(buffer)); // Read response from server

printf("Client (PID %d) received: %s\n", getpid(), buffer);

close(fd);

}

return 0;

}

Server Side Output:

Client Side output: