|  |  |
| --- | --- |
|  | **Practical 10** |
| **Aim:** | Implement inter process communication (IPC) using PIPEs and FIFOs |
| **Code:** | **Using Pipes:**  #include<stdio.h>  #include<unistd.h>  int main() {     int pipefds[2];     int returnstatus;     char writemessages[2][20]={"Hi", "Hello"};     char readmessage[20];     returnstatus = pipe(pipefds);       if (returnstatus == -1) {        printf("Unable to create pipe\n");        return 1;     }       printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);     write(pipefds[1], writemessages[0], sizeof(writemessages[0]));     read(pipefds[0], readmessage, sizeof(readmessage));     printf("Reading from pipe – Message 1 is %s\n", readmessage);     printf("Writing to pipe - Message 2 is %s\n", writemessages[0]);     write(pipefds[1], writemessages[1], sizeof(writemessages[0]));     read(pipefds[0], readmessage, sizeof(readmessage));     printf("Reading from pipe – Message 2 is %s\n", readmessage);     return 0;  } |
| **Output:** |  |
|  | **Using FIFO**  #include <iostream>  #include <cstdlib>  #include <cstring>  #include <unistd.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  int main() {  *const* char\* fifo\_path = "myfifo";    *// Create the FIFO if it doesn't exist*      if (mkfifo(fifo\_path, 0666) == -1) {          perror("mkfifo");          exit(EXIT\_FAILURE);      }        pid\_t pid = fork();        if (pid == -1) {          perror("fork");          exit(EXIT\_FAILURE);      }        if (pid == 0) { *// Child process*          int fifo\_read = open(fifo\_path, O\_RDONLY);          if (fifo\_read == -1) {              perror("open");              exit(EXIT\_FAILURE);          }          char buffer[100];          ssize\_t bytes\_read = read(fifo\_read, buffer, sizeof(buffer));          if (bytes\_read == -1) {              perror("read");              exit(EXIT\_FAILURE);          }          std::cout << "Child received: " << std::string(buffer, bytes\_read) << std::endl;          close(fifo\_read);      } else { *// Parent process*          int fifo\_write = open(fifo\_path, O\_WRONLY);          if (fifo\_write == -1) {              perror("open");              exit(EXIT\_FAILURE);          }          std::string message = "Hello from parent";          ssize\_t bytes\_written = write(fifo\_write, message.c\_str(), message.size());          if (bytes\_written == -1) {              perror("write");              exit(EXIT\_FAILURE);          }          close(fifo\_write);      }    *// Remove the FIFO file*      if (unlink(fifo\_path) == -1) {          perror("unlink");          exit(EXIT\_FAILURE);      }        return 0;  } |
| **Output:** |  |