Assigment 7

Inter process communication in Linux using following.
A. FIFOS: Full duplex communication between two independent processes. First process accepts
sentences and writes on one pipe to be read by second process and second process counts number of
characters, number of words and number of lines in accepted sentences, writes this output in a text file
and writes the contents of the file on second pipe to be read by first process and displays onstandard
output.

7A_client.c

```
#include<stdio.h>       // Include standard input/output library
#include<stdlib.h>      // Include standard library for general functions
#include<sys/types.h>   // Include definitions for data types used in system calls
#include<sys/stat.h>    // Include definitions for file status
#include<unistd.h>      // Include standard symbolic constants and types
#include<fcntl.h>       // Include file control options
#include<string.h>      // Include string handling functions

int main() {
    puts("\n\tClient - Listening\n"); // Print a message indicating the client is listening

    // Create two named FIFOs (first-in-first-out special files) for communication
    int code6 = mkfifo("fifo6.txt", 0666); // FIFO for reading
    int code7 = mkfifo("fifo7.txt", 0666); // FIFO for writing

    char strMessage[5000]; // Buffer for messages

    // Check if FIFO creation was successful
    if(code6 == -1)
        perror("\n\tmkfifo6 returned an error - file may already exist\n"); // Print error if FIFO6 failed
    if(code7 == -1)
        perror("\n\tmkfifo7 returned an error - file may already exist\n"); // Print error if FIFO7 failed

    // Open the FIFOs for reading and writing
    int fd = open("fifo6.txt", O_RDONLY); // Open FIFO6 for reading
    int fd2 = open("fifo7.txt", O_WRONLY); // Open FIFO7 for writing

    // Check if the FIFO for reading was opened successfully
    if(fd == -1) {
        perror("Cannot open FIFO6 for read"); // Print error message
        return EXIT_FAILURE; // Exit with failure status
    }

    // Check if the FIFO for writing was opened successfully
    if(fd2 == -1) {
        perror("Cannot open FIFO7 for write"); // Print error message
        return EXIT_FAILURE; // Exit with failure status
    }

    puts("FIFO OPEN"); // Indicate that FIFOs are open

    // Buffer to read the incoming message
    char stringBuffer[5000];
    memset(stringBuffer, 0, 5000); // Initialize buffer to zero

    int res; // Variable for read results
    char Len; // Variable to hold the length of the message

    // Main loop for reading and processing messages
    {
        res = read(fd, &Len, 1); // Read the length of the message (1 byte)

        // Read the actual message into the buffer
        read(fd, stringBuffer, Len); // Read string characters

        stringBuffer[(int)Len] = 0; // Null-terminate the string
        printf("\nClient Received: %s\n", stringBuffer); // Print the received message

        int j = 0, w = 0, line = 0; // Counters for words, characters, and lines

        // Count words, characters, and lines in the received message
        while(stringBuffer[j] != '\0') {
            char ch = stringBuffer[j];
            if((ch == ' ') || (ch == '\n')) { // Check for spaces and newlines
                w++; // Increment word count
                if(ch == '\n') // If newline is found, increment line count
                    line++;
            }
            j++; // Move to the next character
        }

        // Prepare strings for output
        char LC = (char)strlen(strMessage); // Get length of the message
        char str1[256], str2[256], str3[256]; // Buffers for formatted output

        sprintf(str1, " No.of Words : %d:::", w); strcat(strMessage, str1); // Append word count to
message
        sprintf(str2, " No.of Characters: %d:::", (j - 1)); strcat(strMessage, str2); // Append character
count
        sprintf(str3, " No.of Lines: %d", line); strcat(strMessage, str3); // Append line count

        strcat(strMessage, "\0"); // Null-terminate the message
        printf("\n\tString: %s", strMessage); // Print the final message

        write(fd2, &LC, 1); // Write length of the message to FIFO7
        write(fd2, strMessage, strlen(strMessage)); // Write the message to FIFO7
        fflush(stdin); // Clear the input buffer (not necessary here)

        strMessage[0] = 0; // Reset the character array for the next message

        // Check for termination condition (commented out)
        // if(LC == 1)
        //     break;
    }

    printf("\n"); // Print a newline
    puts("CLIENT CLOSED"); // Indicate the client is closed
    puts("SERVER CLOSED"); // Indicate the server is closed
    close(fd); // Close FIFO6
    close(fd2); // Close FIFO7
    return 0; // Return success
}
```

7A_server.c

```
#include<stdio.h>       // Include standard input/output library for I/O functions
#include<stdlib.h>      // Include standard library for general functions like memory allocation
#include<unistd.h>      // Include standard symbolic constants and types for UNIX standard
functions
#include<sys/types.h>   // Include definitions for data types used in system calls
#include<fcntl.h>       // Include file control options for file handling
#include<string.h>      // Include string handling functions

int main() {
    int n; // Variable declaration (not used in this snippet)
    puts("Server"); // Print a message indicating that this is the server

    char strMessage[5000]; // Buffer for messages to be sent to the client

    // Open FIFO6 for writing (to send messages to the client)
    int fd = open("fifo6.txt", O_WRONLY);

    // Open FIFO7 for reading (to receive messages from the client)
    int fd2 = open("fifo7.txt", O_RDONLY);

    // Check if opening FIFO6 for writing was successful
    if(fd == -1) {
        perror("cannot open fifo6"); // Print error message if failed
        return EXIT_FAILURE; // Exit the program with failure status
    }

    // Check if opening FIFO7 for reading was successful
    if(fd2 == -1) {
        perror("cannot open fifo7"); // Print error message if failed
        return EXIT_FAILURE; // Exit the program with failure status
    }

    puts("FIFO OPEN"); // Indicate that the FIFOs are successfully open
    // Buffer for reading the incoming message
    char stringBuffer[5000];
    memset(stringBuffer, 0, 5000); // Initialize the buffer to zero

    int res; // Variable for read results (not used in this snippet)
    char Len; // Variable to hold the length of the message

    // Main loop for sending and receiving messages
    {
        // Prompt the user to enter a message
        printf("\n\n\t\tEnter the Message to be passed (hitting ENTER without any string will
terminate program): ");
        fgets(strMessage, 100, stdin); // Read user input into strMessage

        char L = (char)strlen(strMessage); // Get the length of the input message

        // Write the length of the message to FIFO6
        write(fd, &L, 1);
        // Write the actual message to FIFO6
        write(fd, strMessage, strlen(strMessage));

        fflush(stdin); // Clear the input buffer (not necessary for this use case)

        strMessage[0] = 0; // Reset the character array for the next message

        // Read the length of the response from the client
        int len2;
        res = read(fd2, &len2, 1);

        // Read the actual response message from the client
        read(fd2, stringBuffer, 5000); // Read string characters into the buffer

        // Print the message received from the client
        printf("\nServer Received: %s\n", stringBuffer);
        stringBuffer[(int)len2] = 0; // Null-terminate the received string (this should actually be done
before printing)
    };

    // Cleanup and exit logic (commented out)
    // printf("\n\nCLIENT CLOSED\n")
    // return 0;
}
```

Output:

```
pl-17@pl17-OptiPlex-3020:~/IT/07$ gcc 7A_client.c
pl-17@pl17-OptiPlex-3020:~/IT/07$ ./a.out

            Client – Listening
FIFO OPEN

Client Received: Hello my Friends
```

```
        String:  No.of Words : 3::: No.of Characters: 16::: No.of Lines: 1
CLIENT CLOSED
SERVER CLOSED

pl-17@pl17-OptiPlex-3020:~/IT/07$ gcc 7A_server.c
pl-17@pl17-OptiPlex-3020:~/IT/07$ ./a.out
Server
FIFO OPEN


                Enter the Message to be passed (hitting ENTER without any string will terminate
program): Hello my Friends

Server Received:  No.of Words : 3::: No.of Characters: 16::: No.of Lines: 1
```

B. Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

7B_server.c

```c
#include <stdlib.h>      // For exit()
#include <unistd.h>      // For sleep()
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

#define SHMSZ 27

int main()  // Explicitly define the return type
{
   char c;
   int shmid;
   key_t key;
   char *shm, *s;

   /*
    * We'll name our shared memory segment
    * "5678".
    */
   key = 5678;

   /*
    * Create the segment.
    */
   if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
       perror("shmget");
       exit(1);
   }

   /*
    * Now we attach the segment to our data space.
    */
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
       perror("shmat");
       exit(1);
   }

   /*
    * Now put some things into the memory for the
    * other process to read.
    */
   s = shm;

   for (c = 'a'; c <= 'z'; c++)
       *s++ = c;
   *s = '\0';  // Use '\0' instead of NULL

   /*
    * Finally, we wait until the other process
    * changes the first character of our memory
    * to '*', indicating that it has read what
    * we put there.
    */
   while (*shm != '*')
       sleep(1);

   exit(0);
}
```

7B_client.c

```
/*
 * shm-client - client program to demonstrate shared memory.
 */
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

#define SHMSZ     27

int main() // Change made here
{
    int shmid;
    key_t key;
    char *shm, *s;

    /*
     * We need to get the segment named
     * "5678", created by the server.
     */
    key = 5678;

    /*
     * Locate the segment.
     */
    if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    /*
     * Now we attach the segment to our data space.
     */
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    /*
     * Now read what the server put in the memory.
     */
    for (s = shm; *s != '\0'; s++) // Change made here
        putchar(*s);
    putchar('\n');

    /*
     * Finally, change the first character of the
     * segment to '*', indicating we have read
     * the segment.
     */
    *shm = '*';

    exit(0);
}
```

Output:

```
pl-17@pl17-OptiPlex-3020:~/IT/07$ gcc 7B_server.c
pl-17@pl17-OptiPlex-3020:~/IT/07$ ./a.out

pl-17@pl17-OptiPlex-3020:~/IT/07$ gcc 7B_client.c
pl-17@pl17-OptiPlex-3020:~/IT/07$ ./a.out
abcdefghijklmnopqrstuvwxyz
```