ZADANIE 1

```c
int main() {
        char str[50];
        int fd[2];
        char str2[50];
        int fd2[2];

        pipe(fd);
        pipe(fd2);
        printf("Enter the text: ");
        scanf("%s", str);

        pid_t pid = fork();

        if (pid == 0) {
                close(fd[1]);
                close(fd2[0]);
                read(fd[0], str2, sizeof(str2));
                close(fd[0]);
                str2[0] = 'X';
                write(fd2[1], str2, strlen(str2) + 1);
                close(fd2[1]);
        }
```

fd[0] to koniec do czytania, fd[1] to koniec do pisania.

fd to **pierwszy pipe** do komunikacji, czyli od rodzica do dziecka.

fd2 – **drugi pipe**, tym razem do powrotu danych od dziecka do rodzica.

pipe(fd);

pipe(fd2);

Tworzę dwa pipe — jedna do wysyłania od rodzica → dziecka, druga od dziecka → rodzica.

pid_t pid = fork();

Proces rodzica dostanie w pid wartość większą od zera.

Proces dziecka dostanie w pid wartość 0.

Dzięki temu wiemy, który fragment kodu należy do rodzica, a który do dziecka.

close(fd[1]);

Dziecko nie będzie nic pisało do rodzica, więc zamykamy niepotrzebny koniec pierwszego pipe (zapis)

close(fd2[0]);

Dziecko nie będzie czytać z 2 pipe, więc zamykamy niepotrzebny koniec drugiego pipe (odczyt)

read(fd[0], str2, sizeof(str2));

Dziecko czeka, aż rodzic wyśle mu tekst przez pierwszy pipe, i zapisuje ten tekst do `str2`

close(fd[0]);

Zamyka pipe po otrzymaniu i zapisywaniu teksta.

str2[0] = 'X';

Zmienia pierwszy znak tekstu na X.

write(fd2[1], str2, strlen(str2) + 1);

Dziecko wysyła zmieniony tekst do rodzica.

close(fd2[1]);

Zamyka rurę i kończy wysyłanie.

```
    else {
            close(fd[0]);
            close(fd2[1]);
            write(fd[1], str, strlen(str) + 1);
            close(fd[1]);

            read(fd2[0], str2, sizeof(str2));
            close(fd2[0]);
            printf("From child: %s\n", str2);
    }
    return 0;
```

Proces rodzica (pid != 0)

close(fd[0]);

Rodzic zamyka fd do czytania — będzie tylko pisać.

close(fd2[1]);

Zamyka końcówkę zapisu w drugiej rurze, bo rodzic nie bedzię pisać do fd2

write(fd[1], str, strlen(str) + 1);

Rodzic wysyła tekst, który wpisał użytkownik do dziecka

close(fd[1]);

Kończy wysyłanie.

read(fd2[0], str2, sizeof(str2));

Odczytuje tekst, który odesłało dziecko.

close(fd2[0]);

Zamyka pipe, komunikacja zakończona.

Wynik:

```
karyna@karyna-VirtualBox:~/lab1/build$ ./hello
Enter the text: text
From child: Xext
```

ZADANIE 2

server.h

#ifndef SERVER_H

sprawdza, czy stała SERVER_H nie została jeszcze zdefiniowana. Jeśli nie — to kod poniżej zostanie wykonany.

#define SERVER_H

Tutaj ta stała zostaje zdefiniowana, aby przy kolejnym włączeniu pliku kompilator wiedział, że ma go pominąć.

void server_loop(int write_fd, int read_fd);

funkcja przyjmuje dwa parametry typu int – write_fd i read_fd, i nic nie zwraca

worker.h

#ifndef WORKER_H

Sprawdza, czy ten nagłówek nie był jeszcze wczytany.

void worker_loop(int read_fd, int write_fd);

deklaracja funkcji

worker.c

#include <stdio.h>

#include "worker.h"

Wczytuje deklarację funkcji worker_loop

char str[50];

Tworzy bufor na tekst do 50 znaków

while(1)

Nieskończona pętla.

read(read_fd, str, sizeof(str));

Odczytuje dane do bufora.

if (strcmp(str, "exit") == 0) return;

Jeśli worker dostał „exit”, kończy działanie.

str[0] = 'X';

Zamienia pierwszy znak w napisie na ‘X’.

write(write_fd, str, strlen(str) + 1);

Wysyła zmieniony tekst z powrotem.

server.c

char str[50], str2[50];

Dwa bufory na tekst.

scanf("%s", str);

Odczytuje napis od użytkownika.

write(write_fd, str, strlen(str) + 1);

Wysyła tekst do workera.

read(read_fd, str2, sizeof(str2));

Odczytuje odpowiedź od workera.

main.c

int fd[2], fd2[2];

Dwie pary potoków: jeden dla komunikacji serwer→worker, drugi worker→serwer.

pipe(fd); pipe(fd2);

Tworzy dwa potoki.

pid_t pid = fork();

Tworzy nowy proces.

worker_loop(fd[0], fd2[1]);

Worker czyta z jednego potoku i pisze do drugiego.

server_loop(fd[1], fd2[0]);

Serwer pisze do pierwszego potoku i czyta z drugiego.

wait(NULL);

Czeka, aż proces dziecka zakończy działanie.

Wynik:

```
karyna@karyna-VirtualBox:~/lab/src$ gcc main.c server.c worker.c -o program
karyna@karyna-VirtualBox:~/lab/src$ ./program
Enter text: text
SERVER: received from  worker: Xext
Enter text: program
SERVER: received from  worker: Xrogram
Enter text: exit
```