Część 1

Wersja programu w jednym pilku, w której prowadziłam dwukierunkową komunikację między procesem rodzica a procesem potomnym przy użyciu dwóch pipe.

fd - służy do przesyłania danych rodzic -> dziecko

fd2 – dziecko -> rodzic

main.c:

deklarujemy zmienne:

str – widomość wpisana przez użytkownika

str2 - wiadomość otrzymana od dziecka

fd – pipe od rodzica do dziecka

fd2 - od dziecka do rodzica

tworzymy dwa pipe: fd[0] - do czytania, fd[1] - do pisania

scanf("%s", str) - wczytujemy dane od użytkownika

pid_t pid = fork() - tworzymy nowy proces potomny

proces dziecka:

zamykamy niepotrzebne końce pipe:

fd[1] – dziecko nie pisze do pipe rodzic -> dziecko

fd2[0] – dziecko nie czyta z pipe dziecko -> rodzic

read(fd[0], str2, sizeof(str2)) – dziecko odczytuje wiadomość od rodzica

str2[0] = 'X' – zamienia pierwszą litere na ‘X’

write(fd2[1], str2, strlen(str2)+1) – dziecko odsyła zmodyfikowaną wiadomość do rodzica

proces rodzica:

zamykamy niepotrzebne końce pipe:

fd[0] – rodzic nie czyta z pipe rodzic -> dziecko

fd2[1] – rodzic nie pisze do pipe dziecko -> rodzic

write(fd[1], str, strlen(str)+1) – wysyłamy wiadomość do dziecka

read(fd2[0], str2, sizeof(str2)) – odbieramy odpowiedź od dziecka

wyświetlamy zmodyfikowaną wiadomość

zamykamy deskryptory

Wynik działania:

```
karolina@karolina:~/ProgSys/C/lab5/part1$ make
mkdir -p build
gcc -c src/main.c -o build/main.o
gcc build/main.o -o build/hello
karolina@karolina:~/ProgSys/C/lab5/part1$ ./build/hello
Wpisz tekst: hello
Od dziecka: Xello
```

Część 2

W drugiej częśći kod został podzielony na osobne pliki:

- server.c + server.h - funkcje odpowiedzialne za część server
- worker.c + worker.h - funkcje odpowiedzialne za część worker
- main.c - tworzy proces pototmny i wywołuje odpowiednie funkcje z plików nagłówkowych

main.c:

tworzy proces potomny i uruchamia odpowiednie funkcje(worker_loop, server_loop)

wait(NULL) - rodzic czeka na zakończenie dziecka

server.c:

scanf - wczytuje dane od użytkownika

write(write_fd, str, strlen(str) + 1) - wysyła do workera

jeśli wpisano “exit”, kończy działanie i zamyka pipe

read(read_fd, str2, sizeof(str2)) - odbiera odpowiedzi

worker.c:

read(read_fd, str, sizeof(str)) - czyta dane od server

sprawdza, czy wiadomość to “exit” i jeśli tak, kończy pętlę

str[0] = 'X' - zamienia pierwszą litere na ‘X’

write(write_fd, str, strlen(str) + 1) - wysyła z powrotem do server

Wynik działania:

```
karolina@karolina:~/ProgSys/C/lab5/part2/src$ gcc main.c server.c worker.c -o program
karolina@karolina:~/ProgSys/C/lab5/part2/src$ ./program
Wpisz tekst: hello
SERVER: Odebarane od worker: Xello
Wpisz tekst: crypto
SERVER: Odebarane od worker: Xrypto
Wpisz tekst: exit
karolina@karolina:~/ProgSys/C/lab5/part2/src$
```