Politechnika Wrocławska

Programowanie Systemowe

LABORATORIUM

Temat: **Komunikacja dwukierunkowa za pomocą pipe()**

Hanna Kieszek – 283797

23.11.2025

1. Cel ćwiczenia

Celem ćwiczenia było zmodyfikowanie pliku z poprzednich zajęć tak aby możliwa była komunikacja dwukierunkowa.

2. Wykorzystane narzędzia

1. Ubuntu
2. Terminal

3. Przebieg laboratorium

3.1. Zmiana danych wejściowych w Makefile

Plik Makefile posłuży do skompilowania plików które zostały utworzone podczas tego laboratorium. Poniżej znajduje się zmodyfikowany Makefile:

```
  GNU nano 7.2                                                    M
OUTDIR=build
SRCDIR=src
CC=gcc
TARGET=$(OUTDIR)/text
SRC=$(wildcard $(SRCDIR)/*.c)
OBJ=$(patsubst $(SRCDIR)/%.c, $(OUTDIR)/%.o, $(SRC))

all: $(TARGET)

clean:
        rm -rf $(OUTTDIR)

$(TARGET): $(OBJ)
        $(CC) $^ -o $@

$(OUTDIR)/%.o: $(SRCDIR)/%.c | $(OUTDIR)
        $(CC) -c $< -o $@

$(OUTDIR):
        mkdir $@
```

Obraz 1: Wygląd pliku Makefile

3.2. Modyfikacja pliku z poprzedniego laboratorium

Plik z poprzednich zajęć dzielił się na dwa procesy: rodzica i dziecko. Zadaniem rodzica było pobranie tekstu z klawiatury wpisanego przez użytkownika i przesłanie do do pipe, a zadaniem dziecka było pobranie tekstu z pipe, zmodyfikowanie go i wyświetlenie.

```
 GNU nano 7.2                                                    src/all.c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        char text[20];
        int fd[2];
        char text2[20];

        pipe(fd);
        pid_t p = fork();

        if(p<0)
        {
                perror("fork fail");
                exit(1);
        }
        else if ( p>0)
        {
                scanf("%s", text);
                write(fd[1], text, strlen(text)+1);
        }

        else
        {
                read(fd[0], text2, 20);
                printf("%s", text2);
                text2[0]='X';
                printf("%s", text2);
        }
        return 0;
}
```

Obraz 2: Plik z laboratorium 5

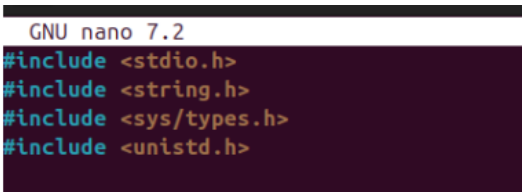
Na początku laboratorium powyższy plik został zmodyfikowany w taki sposób aby wdrożyć komunikację dwustronną między dzieckiem a rodzicem. Od tej pory rodzic miał pobierać z klawiatury i wysyłać do pipe, dziecko miało odczytywać z pipe, modyfikować i odsyłać z powrotem do rodzica, a na końcu rodzic miał wyświetlać to co odebrał od dziecka. Cały program miał działać w nieskończonej pętli, dopiero po wpisaniu “exit” program miał się zamykać. Poniżej został przedstawiony końcowy kod na tym etapie laboratorium a następnie został on wyjaśniony.

ubuntu@Ubuntu: ~/src2

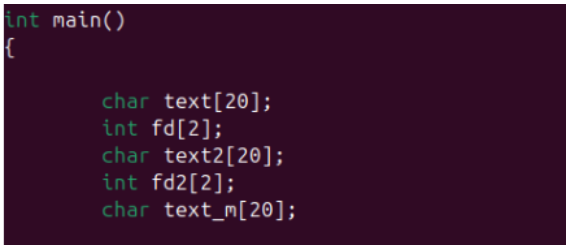
GNU nano 7.2 text.c

```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{

        char text[20];
        int fd[2];
        char text2[20];
        int fd2[2];
        char text_m[20];

        pipe(fd);
        pipe(fd2);

        pid_t p = fork();
        if(p>0)
        {
                close(fd[0]);
                close(fd2[1]);
                while(1){
                        printf("in: ");
                        scanf("%s", text);
                        write(fd[1], text, strlen(text)+1);
                        if (strcmp(text, "exit") == 0)
                        {
                                wait(NULL);
                                break;
                        }

                        read(fd2[0], text_m, 20);
                        printf("out: %s\n", text_m);
                }
        }

        else if(p==0)

        {
                close(fd[1]);
                close(fd2[0]);
                while(1){
                        read(fd[0], text2, 20);
                        if (strcmp(text2, "exit") == 0)
                        {
                                break;
                        }
                        text2[0]='X';
                        write(fd2[1], text2, strlen(text2)+1);
                }
        }

        return 0;
}
```

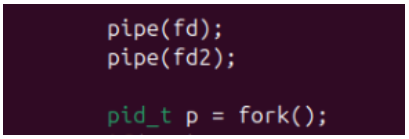
Obraz 3: Zmodyfikowany plik z laboratorium 5



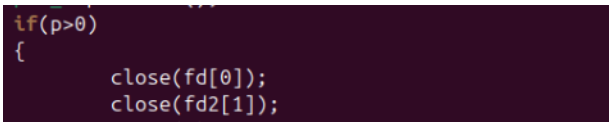
Niezbędne nagłówki aby program działał prawidłowo.



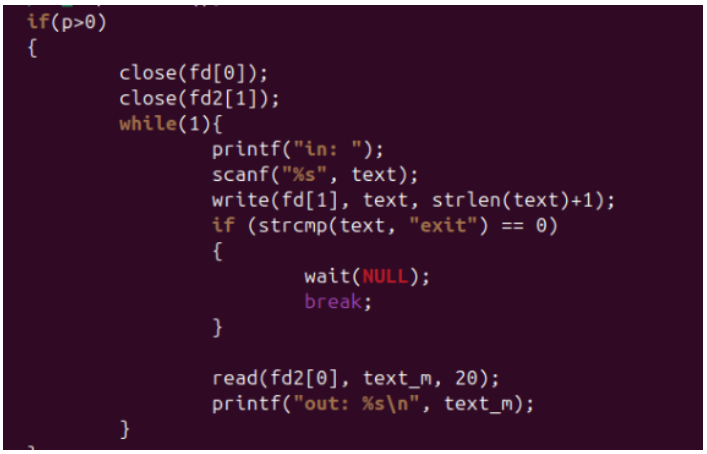
Zostały dodane dwie zmienne fd2 - zmienna dla drugiego pipe() (z dziecka do rodzica), text_m - zapisywanie zmodyfikowanego tekstu odczytanego w rodzicu.



Wywołanie pipe(fd) i pipe(fd2). Dzięki temu jest możliwa komunikacja w dwie strony. Poniżej zmienna p przechowująca id procesu funkcji fork().



Początek instrukcji warunkowej if odnoszącej się do procesu rodzica. Poniżej znajduje się funkcja close która najpierw zamyka końcówki do czytania i pisania dziecka.



W następnej linijce rozpoczyna się nieskończona pętla while. Później proces rodzica wczytuje tekst z klawiatury i wysyła go przez pipe do dziecka. Następnie sprawdza czy wpisany tekst był o treści “exit”. Jeśli tak, czeka aż zakończy się proces dziecka, a następnie sam się kończy. Jeżeli tekst nie

był sygnałem do wyjścia, rodzic odczytuje z drugiego pipe treść jaką wysłało dziecko i wyświetla ją. Dzięki funkcji wait(NULL) nie utworzą się procesy zombie ani osierocone ponieważ rodzic zaczeka na zakończenie dziecka, więc rodzic nie zakończy się przed nim, a następnie go wyczyści.

```
else if(p==0)

{
        close(fd[1]);
        close(fd2[0]);
        while(1){
                read(fd[0], text2, 20);
                if (strcmp(text2, "exit") == 0)
                {
                        break;
                }
                text2[0]='X';
                write(fd2[1], text2, strlen(text2)+1);
```

Druga część instrukcji warunkowej dotyczy procesu dziecka (czyli wtedy gdy id procesu jest równe 0). Na początku zostały zamknięte końcówki do pisania i czytania rodzica. Następnie rozpoczyna się pętla while w której proces dziecka czyta to co wysłał rodzic. Jeżeli odczytał słowo “exit”, proces dziecka się zamknie. W przeciwnym razie tekst zostanie zmodyfikowany i odesłany z powrotem do rodzica.

3.3. Rozbicie programu na osobne pliki

W drugiej części laboratorium plik text.c zostanie rozdzielony na trzy pliki. Pierwszy main.c w którym znajdować się będzie funkcja main(), niektóre zmienne oraz instrukcja warunkowa if w której zostaną wywołane funkcje server() i worker(), stworzone w kolejnych plikach.

Drugi z nich -server.c będzie zawierał definicję funkcji server(). Ta funkcja będzie odpowiadała za to co ma robić proces rodzica, czyli pobranie z klawiatury, wysłanie do dziecka, sprawdzenie czy nie zostało wpisane słowo “exit” (jeśli tak, wyjście z programu jak skończy się proces dziecka) oraz wyświetlanie zmodyfikowanego tekstu.

Trzeci plik o nazwie worker.c będzie zawierał definicję funkcji worker(). Ta funkcja będzie wywoływana w procesie dziecka. Będzie ona polegała na tym, że dziecko odczyta tekst który przesłał rodzic, sprawdzi czy nie zostało wpisane słowo “exit” (jeśli tak proces zostanie zakończony), zmodyfikuje tekst i wyśle do rodzica.

Oprócz tego zostały stworzone pliki server.h i worker.h odpowiadające funkcjom o tej samej nazwie.

Funkcje server() i worker() przyjmują dwa argumenty fd i fd2 które są tablicami reprezentującymi dwie funkcje pipe()

Wszystkie pliki zostały zapisane folderze project.

```
ubuntu@Ubuntu:~$ mkdir project
ubuntu@Ubuntu:~$ touch project/main.c
ubuntu@Ubuntu:~$ touch project/server.c
ubuntu@Ubuntu:~$ touch project/server.h
ubuntu@Ubuntu:~$ touch project/worker.h
ubuntu@Ubuntu:~$ touch project/worker.c
```

Obraz 4: Tworzenie katalogu oraz plików

```
 GNU nano 7.2                          project/main.c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

#include "server.h"
#include "worker.h"

int main()
{
        int fd[2];
        int fd2[2];

        pipe(fd);
        pipe(fd2);

        pid_t p = fork();
        if(p>0)
        {
                server(fd, fd2);

        }
        else if(p==0)
        {
                worker(fd, fd2);
        }

        return 0;
}
```

Obraz 5: Zawartość pliku main.c

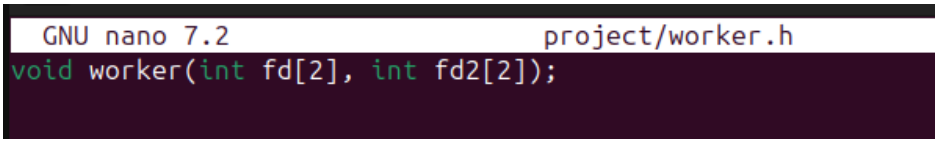
```
 GNU nano 7.2                          project/server.c
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>

#include "server.h"

void server(int fd[2], int fd2[2])
{

        char text[20];
        char text_m[20];

        close(fd[0]);
        close(fd2[1]);

        while(1)
        {
                printf("in: ");
                scanf("%s", text);
                write(fd[1], text, strlen(text)+1);
                if (strcmp(text, "exit") == 0)
                {
                        wait(NULL);
                        break;
                }

                read(fd2[0], text_m, 20);
                printf("out: %s\n", text_m);
        }
}
```

Obraz 6: Zawartość pliku server.c

```
 GNU nano 7.2                          project/server.h
void server(int fd[2], int fd2[2]);
```

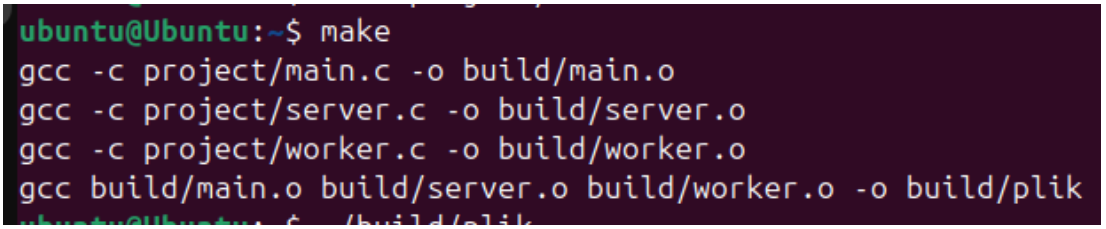
Obraz 7: Zawartość pliku server.h

Obraz 8: Zawartość pliku worker.c

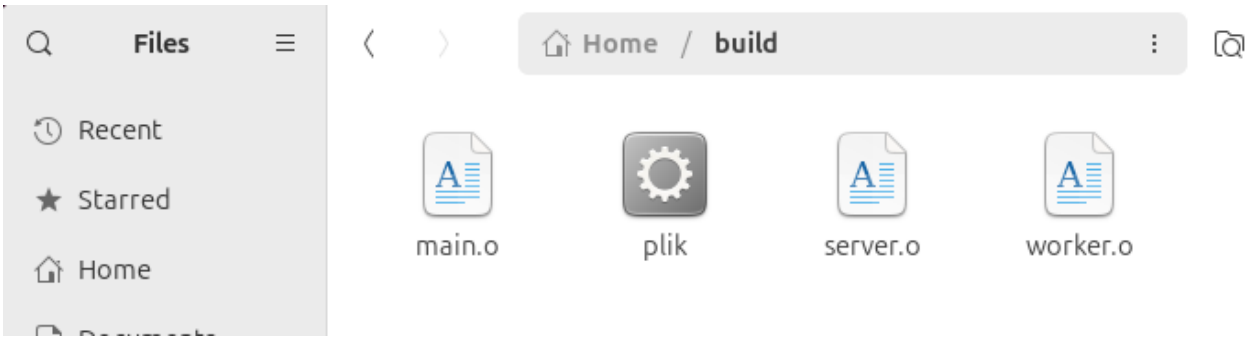


Obraz 9: Zawartość pliku worker.h

W wyniku kompilacji plików z katalogu project powstał plik wykonywalny o nazwie plik.

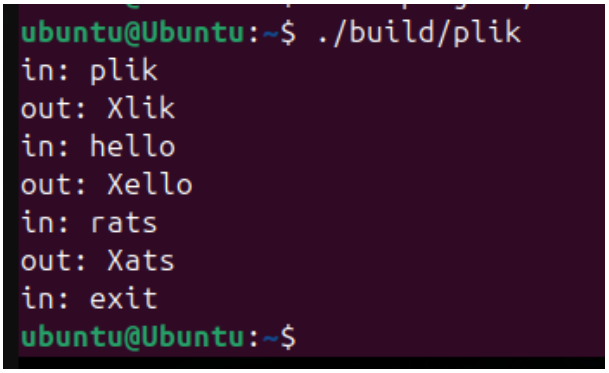


Obraz 10: Output po kompilowaniu plików z folderu project za pomocą make



Obraz 11: Zawartość katalogu build

Po uruchomieniu pliku wykonywalnego program poprosi o wpisanie tekstu. Po kliknięciu enter wyświetli się zmodyfikowany tekst. Po wpisaniu “exit” program się zakończy.



Obraz 12: Przykładowe działanie programu