Politechnika Wrocławska

Programowanie Systemowe

LABORATORIUM

Temat: **Kolejka FIFO, named pipe() i argumenty funkcji main()**

Hanna Kieszek – 283797

23.11.2025

1. Cel ćwiczenia

Celem ćwiczenia było zmodyfikowanie pliku z poprzednich zajęć tak aby możliwa była komunikacja dwukierunkowa za pomocą nazwanych potoków oraz wprowadzenie argumentów funkcji main()

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                          Makefile
OUTDIR=build
SRCDIR=projectv2
CC=gcc
TARGET=$(OUTDIR)/plik
SRC=$(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 plików z poprzedniego laboratorium

Na potrzeby tego laboratorium zostały zmodyfikowane pliki z poprzednich zajęć. Plik main.c zawierający funkcję main(), zmienne oraz instrukcję warunkową obsługującą proces rodzica, dziecka i błędy. Plik server.c i odpowiadający mu server.h który zawierał funkcję server, która odczytywała tekst wprowadzony przez użytkownika, wysyłała go do procesu dziecka a później go od niego odbierała. Plik worker.c i odpowiadający mu worker.h, zawierający funkcję worker która odbiera tekst od rodzica, modyfikuje go i odsyła. Poniżej znajdują się obrazy wszystkich plików z poprzednich zajęć przed modyfikacją:

```
 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 2: Plik main.c z laboratorium 6

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

Obraz 3: Plik server.h z laboratorium 6

```
 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 4: Plik server.c z laboratorium 6

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

Obraz 5: Plik worker.h z laboratorium 6

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

#include "worker.h"

void worker(int fd[2], int fd2[2])
{
        char text2[20];

        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);
        }
}
```

Obraz 6: Plik worker.c z laboratorium 6

Te pliki zostały zmodyfikowany w taki sposób aby zamiast fork() i pipe(), komunikacja dwukierunkowa między workerem a serwerem odbywa się za pośrednictwem named pipes czyli nazwanych potoków FIFO. Każdy taki potok jest odpowiedzialny za komunikację w jedną stronę tak jak zwykła funkcja pipe(), dlatego aby komunikacja w dwie strony była możliwa zostały stworzone dwie takie funkcje.

Oprócz tego zostały dodane argumenty funkcji main. Gdy jako argument zostanie wpisane słowo ‘server”, program uruchomi funkcję server, a gdy zostanie wpisane słowo “worker” uruchomi się funkcja worker. Jedna nie będzie działać poprawnie. W momencie uruchomienia tylko jednej, w terminalu da się wpisywać różne rzeczy ale nic się nie wydarzy. Dopiero gdy w drugim oknie zostanie uruchomiony program z argumentem odpowiadającym tej drugiej funkcji, program zacznie działać poprawnie.

Poniżej zostały zamieszczone zrzuty ekranu zawierające zmodyfikowane programy.

```
ubuntu@Ubuntu:~$ make
gcc -c projectv2/main.c -o build/main.o
gcc build/main.o build/server.o build/worker.o -o build/plik
```

Obraz 7: Kompilacja za pomocą Makefile

```
 GNU nano 7.2                                    projectv2/main.c *
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include "server.h"
#include "worker.h"

int main(int argc, char *argv[])
{
        char * server2worker = "/tmp/server2worker";
        char * worker2server = "/tmp/worker2server";

        if(argc != 2)
        {
                fprintf(stderr, "Niewalasciwa liczba argumentow\n");
                return 1;
        }

        if(strcmp(argv[1], "server") == 0)
        {

                mkfifo(worker2server, 0666);
                mkfifo(server2worker, 0666);

                int fdwrite = open(server2worker, O_WRONLY);
                int fdread = open(worker2server, O_RDONLY);

                server(fdwrite, fdread);

                close(fdread);
                close(fdwrite);

                unlink(worker2server);
                unlink(server2worker);

        }
```

Obraz 8: Plik main.c cz.1

```
                unlink(worker2server);
                unlink(server2worker);

        }
        else if(strcmp(argv[1], "worker") ==0)
        {
                int fdread = open(server2worker, O_RDONLY);
                int fdwrite = open(worker2server, O_WRONLY);

                worker(fdread, fdwrite);

                close(fdread);
                close(fdwrite);

        }
        else
        {
                fprintf(stderr, "Nieznany argument: %s\n", argv[1]);
                return 1;
        }


        return 0;
}
```

Obraz 9: Plik main.c cz.2

```
  GNU nano 7.2                                    projectv2/worker.c
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include "worker.h"

void worker(int fdread, int fdwrite)
{
        char text2[20];

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

Obraz 10: Plik worker.c

```
 GNU nano 7.2                                projectv2/server.c
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>

#include "server.h"

void server(int fdwrite, int fdread)
{

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

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

                read(fdread, text_m, 20);
                printf("out: %s\n", text_m);
        }
}
```

Obraz 11: Plik server.c

```
 GNU nano 7.2                        projectv2/server.h
void server(int fdwrite, int fdread);
```

Obraz 12: Plik server.h

```
 GNU nano 7.2                         projectv2/worker.h
void worker(int fdread, int fdwrite);
```

Obraz 13: Plik worker.h

4. Omówienie fragmentów kodu i prezentacja działania

4.1. Plik main.c

Funkcja main() przyjmuje dwa argumenty: int argc który oznacza liczbę wprowadzonych argumentów oraz char *argv[] który jest tablicą przechowującą te argumenty. Poniżej znajdują się dwie deklaracje zmiennych które zawierają ścieżki do plików FIFO.

```
int main(int argc, char *argv[])
{
        char * server2worker = "/tmp/server2worker";
        char * worker2server = "/tmp/worker2server";
```

Następnie znajduje się instrukcja warunkowa która obsługuje błąd gdy zostanie wpisana niewłaściwa liczba argumentów

```
        if(argc != 2)
        {
                fprintf(stderr, "Niewalasciwa liczba argumentow\n");
                return 1;
        }
```

W kolejnej części znajduje się instrukcja warunkowa dotycząca przypadku w którym jako argument zostanie podany server. Jeżeli tak się stanie, za pomocą komendy mkfifo tworzone są potoki nazwane worker2server i server2worker. Argument 0666 jest odpowiedzialny za definiowanie praw dostępu do pliku w tym przypadku odczyt i zapis dla wszystkich użytkowników.

Następnie zostały otwarte potoki: server2worker - strona do pisania oraz worker2server-strona do czytania. Ich deskryptory są przechowywane w odpowiadającym im zmiennym fdwrite i fdread.

Poniżej została wywołana funkcja server() z deskryptorami jak argumenty. Pod nią znajduje się zamknięcie wcześniej otworzonych deskryptorów oraz usunięcie wcześniej utworzonych plików FIFO, aby przy następnym uruchamianiu programu nie został zwrócony błąd, że takie pliki już istnieją.

```
if(strcmp(argv[1], "server") == 0)
{

        mkfifo(worker2server, 0666);
        mkfifo(server2worker, 0666);

        int fdwrite = open(server2worker, O_WRONLY);
        int fdread = open(worker2server, O_RDONLY);

        server(fdwrite, fdread);

        close(fdread);
        close(fdwrite);

        unlink(worker2server);
        unlink(server2worker);

}
```

Następnie znajduje się część instrukcji warunkowej dotyczącej przypadku w którym jako argument zostanie wpisane worker. Zanim program przejdzie do wykonania tej funkcji, zostaną otwarte dwa potoki: server2worker - końcówka do czytania oraz worker2server - końcówka do pisania, z takimi samymi uprawnieniami jak zostały podane wcześniej. Ich deskryptory przechowywane są w zmiennych fdread i fdwrite. Następnie zostaje wywołana funkcja worker, a gdy ona się zakończy, deskryptory zostają zamknięte za pomocą funkcji close().

```
else if(strcmp(argv[1], "worker") ==0)
{
        int fdread = open(server2worker, O_RDONLY);
        int fdwrite = open(worker2server, O_WRONLY);

        worker(fdread, fdwrite);

        close(fdread);
        close(fdwrite);

}
```

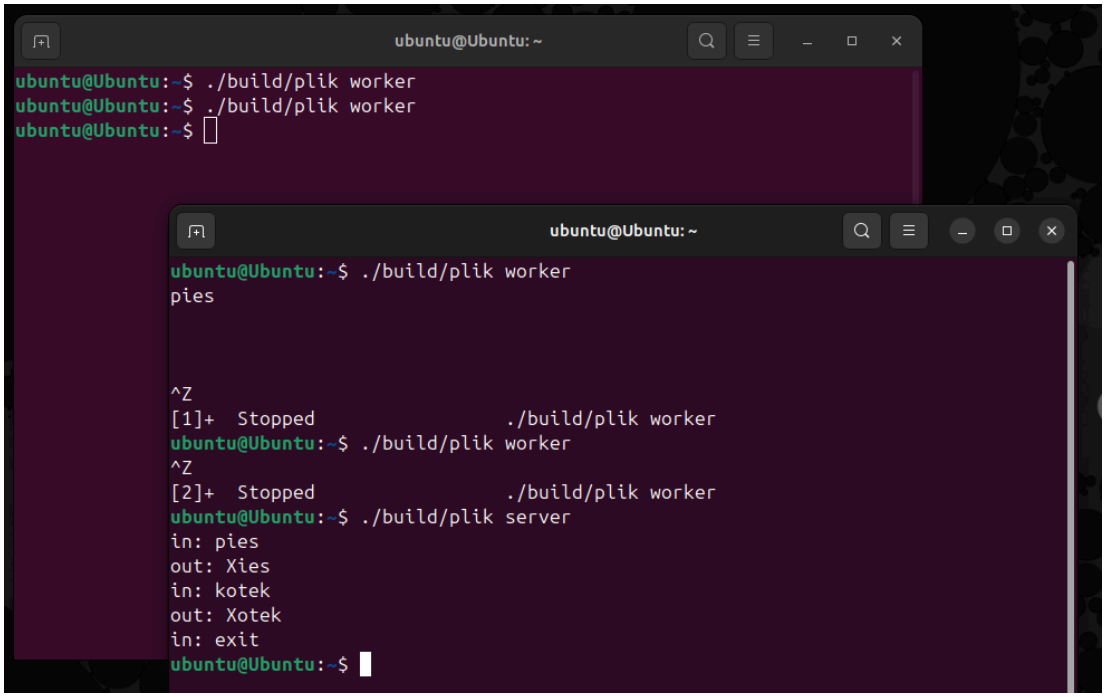
Poniżej znajduje się ostatnia część instrukcji warunkowej w której obsługiwany jest błąd niewłaściwego argumentu. Jeżeli zostanie podany nieznany argument, program to zakomunikuje i zwróci błąd.

```
else
{
        fprintf(stderr, "Nieznany argument: %s\n", argv[1]);
        return 1;
}
```

W plikach server.c, server.h, worker.c i worker.h nie zaszły znaczące zmiany.

4.2. Działanie programu

Poniżej znajdują się wyniki działania programu. W jednym oknie program został uruchomiony z argumentem worker, a w drugim z argumentem server.



W poniższym zrzucie ekranu została zaprezentowana obsługa błędów. Najpierw program został uruchomiony bez żadnego argumentu, a następnie został uruchomiony z błędnym.