Politechnika Wrocławska

Programowanie Systemowe

LABORATORIUM

Temat: **Funkcje fork() i pipe()**

Hanna Kieszek – 283797

23.11.2025

1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z działaniem funkcji fork() i pipe() oraz wykorzystanie ich w prostym programie w języku c.

2. Wykorzystane narzędzia

1. Debian
2. Terminal

3. Przebieg laboratorium

3.1. Utworzenie pliku i zmiana danych wejściowych w Makefile

Na początku został utworzony plik src/all.c w którym znajdzie się kod oraz w pliku Makefile zostały zmodyfikowane dane wejściowe tak aby pasowały do zadania z laboratoriów. Plik Makefile z poprzednich laboratoriów został wykorzystany w tym do kompilowania pliku all.c w wyniku czego zostanie utworzony plik text. 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. Napisanie programu i wykorzystanie pipe()

Docelowy program będzie polegał na tym, że program będzie rozdzielał się na dwa procesy, rodzica i dziecko, za pomocą funkcji fork(). Rodzic będzie odpowiadał za wczytanie tekstu z klawiatury który zostanie wprowadzony przez użytkownika oraz za przesyłanie tego teksty do pipe() za pomocą funkcji write(). Dziecko będzie odpowiadało za odczytanie tekstu z pipe() za pomocą funkcji read() oraz za zmodyfikowanie tego tekstu tak aby jego pierwszą literą było “X” oraz wyświetlenie zmodyfikowanego tekstu. Skończony kod w języku c został przedstawiony poniżej:

GNU nano 7.2 src/all.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];

        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: Kod pliku all.c

Funkcja fork() rozdziela program na dwa procesy rodzica i dziecko. Po wywołaniu tej funkcji w przypadku rodzica, program zwróci pid o wartości większej niż 0, natomiast w przypadku dziecka, pid będzie równy 0. Gdy wystąpi błąd pid będzie mniejszy od 0. W momencie wywołania funkcji fork(), proces rodzica tworzy swoją kopię która jest jego procesem potomnym.

Funkcja pipe() tworzy kanał porozumienia między procesami. Pozwala jednemu z nich przesyłać informacje aby drugi proces mógł je odebrać. Aby pipe() działała poprawnie potrzebuje tablicy dwuelementowej, której pierwszy argument x[0] będzie odpowiadał za koniec do odczytywania a x[1] za koniec do przesyłania informacji.

3.3. Wyjaśnienie kodu i działanie programu

Na początku zostały umieszczone odpowiednie nagłówki aby program działał prawidłowo. Następnie w int main() zostały zadeklarowane tablice. Pierwsza z nich to text[20] która będzie przechowywała tekst wprowadzony przez użytkownika, kolejna fd[2] to tablica funkcji pipe() oraz text2[20] to tablica do której zostaną odebrane dane z pipe().

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

W kolejnym kroku została wywołana funkcja pipe(), która posiada tablicę fd jako argument. Następnie w wierszu niżej została zadeklarowana zmienna p o typie danych pid_t, który jest używany do przechowywania id procesów. Do zmiennej p została przypisana funkcja fork(), dzięki temu p będzie przechowywać id wartość zwróconą przez fork().

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

Następnie w instrukcji warunkowej if program sprawdza id procesu i w zależności od wyniku wykonuje odpowiednie instrukcje. Jeżeli p<0 to program zwróci błąd i zakończy się. W przypadku kiedy p>0, czyli będzie to proces rodzica, program odczyta tekst wprowadzony przez użytkownika ze zmiennej text oraz za pomocą write() prześle do pipe(). Konstrukcja strlen(text)+1 określa długość danych przesyłanych przez pipe(). Została dodana 1 aby wysłać również znak końca stringa. W przeciwnym razie mogłyby pojawić się błędy przy przesyłaniu

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

Na końcu programu w else, kiedy p=0 (czyli gdy proces jest dzieckiem) program odczytuje dane z pipe() za pomocą funkcji read(). Wartość tablicy fd[0] oznacza odczyt z pipe(), text2 zapis tego do tej zmiennej a 20 oznacza długość ciągu znaków. Następnie program wyświetla wartość text2. W następnym wierszy pierwsza litera tekstu ze zmiennej text2 zostaje zamieniona na “X”. Na końcu proces dziecka wyświetla zmodyfikowany tekst.

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

Po skompilowaniu programu za pomocą napisanego wcześniej Makefile i uruchomieniu w terminalu ./build/text pojawi się linijka na wpisanie tekstu. Następnie wyświetli się wpisany tekst oraz ten zmodyfikowany.

```
ubuntu@Ubuntu:~$ ./build/text
hello
helloXelloubuntu@Ubuntu:~$
```