Московский авиационный институт
(Национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Операционные системы»

Студент: Аминов Степан Сергеевич

Группа: М8О-208Б-19

Преподаватель: Е. С. Миронов

Дата:

Оценка:

Москва, 2020

Содержание

Постановка задачи

Необходимо написать 3-и программы. Далее будем обозначать эти программы A, B, C.
Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор, пока программа A не примет «сообщение о получение строки» от программы C, она не может отправлять следующую строку программе C.
Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно.

Метод решения

Способом организации межпроцессорного взаимодействия мной был выбраны каналы (pipe) из-за простоты их использования и удобства при использовании в работе простых процессов. Для удобства работы были опередедены функции str_read и str_length для получения длины вводимой строки, так как подсчёт количества символов является одним из заданий, поставленных перед программой.

Исходный код

A.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

size_t str_read(char **str_, int fd) {
    free(*str_);
    size_t str_size = 0;
    size_t cap = 4;
    char *str = (char*) malloc(sizeof(char) * cap);
    char c;

    while (read(fd, &c, sizeof(char)) == 1) {
        if (c == '\n') {
            break;
        }
        str[(str_size)++] = c;
        if (str_size == cap) {
            str = (char*) realloc(str, sizeof(char) * cap * 3 / 2);
            cap = cap * 3 / 2;
        }
    }
    str[str_size] = '\0';

    *str_ = str;
    return str_size;
}

size_t str_length(char *str) {
    size_t length = 0;
    for (int i = 0; str[i] != '\0'; ++i) {
        length++;
    }
    return length;
}

int main() {
    int fd_A_to_B[2];
    int fd_A_to_C[2];
    int fd_C_to_A[2];
    int fd_C_to_B[2];
```

```
pipe(fd_A_to_B);
pipe(fd_A_to_C);
pipe(fd_C_to_A);
pipe(fd_C_to_B);

int id = fork();

if (id < 0) {
    perror("Fork error");
    exit(1);
}
else if (id == 0) {
    close(fd_A_to_C[1]);
    close(fd_C_to_A[0]);
    close(fd_C_to_B[0]);
    close(fd_A_to_B[0]);
    close(fd_A_to_B[1]);

    char pac[3];
    sprintf(pac, "%d", fd_A_to_C[0]);

    char pca[3];
    sprintf(pca, "%d", fd_C_to_A[1]);

    char pcb[3];
    sprintf(pcb, "%d", fd_C_to_B[1]);

    execl("./c", "./c",  pac, pca, pcb, NULL);
}
else {
    int id2 = fork();
    if (id2 < 0) {
        perror("Fork error");
        exit(1);
    }
    else if (id2 == 0) {
        close(fd_A_to_C[0]);
        close(fd_A_to_C[1]);
        close(fd_C_to_A[0]);
        close(fd_C_to_A[1]);
        close(fd_C_to_B[1]);
        close(fd_A_to_B[1]);

        char pcb[2];
        sprintf(pcb, "%d", fd_C_to_A[0]);
```

```
            char pab[2];
            sprintf(pab, "%d", fd_C_to_B[0]);

            execl("./b", "./b",  pcb, pab, NULL);
        }
        else {
            close(fd_A_to_C[0]);
            close(fd_C_to_A[1]);
            close(fd_A_to_B[0]);
            close(fd_C_to_B[1]);
            close(fd_C_to_B[0]);

            char *str = NULL;
            while ((str_read(&str, 0)) > 0) {
                size_t size = str_length(str);
                write(fd_A_to_C[1], &size, sizeof(size_t));
                write(fd_A_to_C[1], str, size);
                write(fd_A_to_B[1], &size, sizeof(size_t));
                int ok;
                read(fd_C_to_A[0], &ok, sizeof(ok));
            }

            close(fd_C_to_A[0]);
            close(fd_A_to_C[1]);
            close(fd_A_to_B[1]);
        }
    }

    return 0;
}
```

B.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>



int main(int argc, char *argv[]) {
    int fromC = atoi(argv[1]);
    int fromA = atoi(argv[2]);

    size_t size;
```

```
    while (read(fromA, &size, sizeof(size_t)) > 0) {
        printf("B is alive:\n");
        printf("Number of symbols A sent: %zu\n", size);
        read(fromC, &size, sizeof(size_t));
        printf("Number of symbols C received: %zu\n", size);
    }
    close(fromC);
    close(fromA);


    printf("C\n");


    return 0;
}
```

C.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>


int main(int argc, char *argv[]){
    int fromA = atoi(argv[1]);
    int toA = atoi(argv[2]);
    int toB = atoi(argv[3]);
    size_t size;


    while (read(fromA, &size, sizeof(size_t)) > 0){
        char *str = (char*) malloc(size);
        read(fromA, str, size);
        printf("B is alive\n");
        printf("string received from A: %s\n", str);
        write(toB, &size, sizeof(size_t));
        int check = 1;
        write(toA, &check, sizeof(int));
        free(str);
    }
    close(fromA);
    close(toA);
    close(toB);


    return 0;
}
```

Пример работы программы

```
magic@magical:~/CLionProjects/kurs$ ./A
the programm is working(probably)
C is alive
string received from A: the programm is working(probably)
B is alive
Number of symbols A sent: 33
Number of symbols C received: 33
I cant believe its really happening
C is alive
string received from A: I cant believe its really happening
B is alive
Number of symbols A sent: 35
Number of symbols C received: 35
```

Вывод

Каналы pipe являются простым и удобным средством межроцессорного взаимодействия, особенно, как мне показалось, при работе с небольшими программами с простым функционалом. В данной работе я смог ещё раз попрактиваться в реализации межроцессорного взаимодействия.