Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Стрыгин Д.Д.

Группа: М8О–306Б–19

Вариант: на “удовлетворительно”

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021.

## Цель работы

* Приобретение практических навыков в использовании знаний, полученных в течении курса
* Проведение исследование в выбранной предметной области

## Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

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 С. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе А сообщение о том, что строка получена. До тех, пор пока программа А не примет «сообщение о получение строки» от программы С, она не может отправлять следующую строку программе С. Программа B пишет в стандартный вывод количество отправленных символов программой А и количество принятых символов программой С. Данную информацию программа B получает от программ A и C соответственно. Способ организация межпроцессорного взаимодействия выбирает студент.

**Общие сведения о программе**

Программа компилируется из файлов А.с, В.с, С.с. Также используется заголовочные файлы: <stdio.h>, <stdlib.h>, <unistd.h>, <fcntl.h>, <ctype.h>. В программе используются следующие системные вызовы:

1. **execl** – переданная в качестве аргумента программа загружается в память вместо старой, которая вызвала exec. Старой программе больше не доступны сегменты памяти, которые перезаписаны новой программой.  Аргументы командной строки передаются в форме списка arg0, arg1.... argn, NULL. Эту форму используют, если количество аргументов известно.
2. **fork** – создает копию текущего процесса, который является дочерним процессом для текущего процесса.
3. **pipe** – создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами.
4. **read** – чтение из файла в буфер.
5. **write** – запись из буфера в файл.
6. **perror –** вывод сообщения об ошибке.
7. **exit –** завершает работу программы.
8. **atoi** – конвертирует строку, на которую указывает параметр str, в величину типа int. Строка должна содержать корректную запись целого числа. В противном случае возвращается 0.
9. **sprintf** – идентична printf(), за исключением того, что вывод производится в массив, указанный аргументом buf. Возвращаемая величина равна количеству символов, действительно занесенных в массив.

**Основные файлы программы**

**progA.c:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <ctype.h>

#define MIN\_CAP 4

#define STDIN 0

size\_t read\_string(char \*\*str\_, int fd) {

    free (\*str\_);

    size\_t str\_size = 0;

    size\_t cap = MIN\_CAP;

    char \*str = (char\*) malloc (sizeof(char) \* cap);

    if (str == NULL) {

        perror ("Malloc error");

        exit (-1);

    }

    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;

            if (str == NULL) {

                perror("Realloc error");

                exit (-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 ab[2];

    int ac[2];

    int ca[2];

    int cb[2];

    pipe(ab);

    pipe(ac);

    pipe(ca);

    pipe(cb);

    int id1 = fork();

    if (id1 < 0) {

        perror("Fork error");

        exit(1);

    }

    else if (id1 == 0) {

        close(ac[1]);

        close(ca[0]);

        close(cb[0]);

        close(ab[0]);

        close(ab[1]);

        char pac[3];

        sprintf(pac, "%d", ac[0]);

        char pca[3];

        sprintf(pca, "%d", ca[1]);

        char pcb[3];

        sprintf(pcb, "%d", cb[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(ac[0]);

            close(ac[1]);

            close(ca[0]);

            close(ca[1]);

            close(cb[1]);

            close(ab[1]);

            char pcb[2];

            sprintf (pcb, "%d", ca[0]);

            char pab[2];

            sprintf (pab, "%d", cb[0]);

            execl("./B", "./B", pcb, pab, NULL);

        }

        else {

            close(ac[0]);

            close(ca[1]);

            close(ab[0]);

            close(cb[0]);

            close(cb[1]);

            char \*str = NULL;

            while (read\_string(&str, STDIN) > 0) {

                size\_t size = str\_length(str);

                write(ac[1], &size, sizeof(size\_t));

                write(ac[1], str, size);

                write(ab[1], &size, sizeof(size\_t));

                int ok;

                read(ca[0], &ok, sizeof(ok));

            }

            close(ca[0]);

            close(ac[1]);

            close(ab[1]);

        }

    }

    return 0;

}

**progB.c:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <ctype.h>

int main(int argc, char \*argv[]) {

    int pcb = atoi(argv[1]);

    int pab = atoi(argv[2]);

    size\_t size;

    while (read(pab, &size, sizeof(size\_t)) > 0) {

        printf("B - From a: %zu\n", size);

        read(pcb, &size, sizeof(size\_t));

        printf("B - from c: %zu\n", size);

    }

    close(pcb);

    close(pab);

    printf("C\n");

    return 0;

}

**progC.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <ctype.h>

int main(int argc, char \*argv[]) {

    int pac = atoi(argv[1]);

    int pca = atoi(argv[2]);

    int pcb = atoi(argv[3]);

    size\_t size;

    while (read(pac, &size, sizeof(size\_t)) > 0){

        char \*str = (char\*) malloc(size);

        if  (str == NULL) printf("MALLOC from C\n");

        read (pac, str, size);

        printf("C - from a: %s\n", str);

        write(pcb, &size, sizeof(size\_t));

        int ok = 1;

        write(pca, &ok, sizeof(int));

        free(str);

    }

    close(pac);

    close(pca);

    close(pcb);

    printf("B\n");

    return 0;

}

**Пример работы**

den@DESKTOP-1B5EV3F:/mnt/c/U

ers/danst/OneDrive/Документы/GitHub/OC/KP$ ./A

Hello

C - from a: Hello

B - From a: 5

B - from c: 5

World

C - from a: World

B - From a: 5

B - from c: 5

**Вывод**

При организации межпроцессорной связи довольно просто использовать fork-pipe в сочетании с exec(l, v, и тд), но необходимо проверять данные вызовы на ошибки, и уметь завершать процессы, программировать их завершение в нужной для задачи очереди.