Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект
по курсу "Операционные системы"
3 семестр

Студент	Синюков А.С.
Группа	М8О-206Б-21
Преподаватель	Миронов Е.С.
Дата	07.01.2023
Оценка	
Подпись	

Москва, 2023

Оглавление

Задание:

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. Способ организация межпроцессорного взаимодействия выбирает студент.

Идея, метод, алгоритм:

Межпроцессорное взаимодействие я осуществил с помощью memory map, то есть через файлы, отображаемые в память.

При запуске программа A делает fork 2 раза для создания дочерних процессов под B и C.

Программа A записывает полученную строку в массив ptrAC, где на 0 месте расположен флаг, сигнализирующий о работе с этим участком памяти (0 — в массив идет запись, 1 — из массива читаются данные, 2 — работа с массивом завершена), когда флаг сменяется на 1 программа C начинает читать данные, после прочтения ставит флаг 0. Аналогично работает взаимодействие между A и B, C и B.

При вводе на стандартных ввод Ctrl+D программа A ставит флаги 2, сигнализирующие о завершении программам B и C, и сама завершается.

При завершении закрываются все mmap и файловые дескрипторы.

Для удобства буфер для строк использовался статический, на 1024 символа типа char.

Работа программы:

```
root@Anton-Sinyukov:/mnt/c/Users/sinyu/Desktop/os_kp_3# ./a
Programm A started
Programm B started
Programm C started
hello
C:hello
B: from A: 5
B: from C: 5
my
C:my
B: from A: 2
B: from C: 2
name
C:name
B: from A: 4
B: from C: 4
is
B: from A: 2
C:is
B: from C: 2
Gustavo
B: from A: 7
C:Gustavo
B: from C: 7
but
B: from A: 3
C:but
B: from C: 3
you
C:you
B: from A: 3
B: from C: 3
can
B: from A: 3
C:can
B: from C: 3
call
C:call
B: from A: 4
B: from C: 4
me
C:me
B: from A: 2
B: from C: 2
SUS
B: from A: 3
C:SUS
B: from C: 3
Programm stopped
root@Anton-Sinyukov:/mnt/c/Users/sinyu/Desktop/os_kp_3#
```

Непостоянство положений вывода результатов работы программ B и C обусловлено их параллельной работой в разных процессах, иногда B успевает вывести данные раньше чем C.

Листинг:

a.c:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>

#define True 1

int fdAB, fdAC;
/*  A ---string---> C
    A ---number---> B  */

int main(){

    printf("Programm A started\n");

    fdAB = open("tempAB", O_RDWR | O_CREAT | O_TRUNC, 0777);
    fdAC = open("tempAC", O_RDWR | O_CREAT | O_TRUNC, 0777);

    if (fdAB < 0 || fdAC < 0){
                        printf("Error while opening a file\n");
                        return 2;
```

```
        }

    ftruncate(fdAB, 2 * sizeof(int));
    ftruncate(fdAC, 1025 * sizeof(char));

    int *ptrAB;
    char *ptrAC;
    ptrAB = mmap(NULL, 2 * sizeof(int), PROT_WRITE | PROT_READ,
MAP_SHARED, fdAB, 0);
    ptrAC = mmap(NULL, 1025 * sizeof(char), PROT_WRITE | PROT_READ,
MAP_SHARED, fdAC, 0);
    /*
    ptr[0] = 0 - готово к записи
    ptr[0] = 1 - запись произошла, можно читать
    ptr[0] = 2 - запись окончена, a.k.a. данных больше не будет
    */
    if (ptrAB == MAP_FAILED || ptrAC == MAP_FAILED){
        printf("Error while memory map created\n");
        return 3;
        }

    ptrAB[0] = 0;
    ptrAC[0] = 0;

    int id1 = fork();

    if (id1 < 0){
        printf("Fork raise error\n");
```

```
    return 4;
} else if (id1 == 0){

    if (execl("b.out", "b.out", NULL, (char *)NULL) == -1){
        printf("Error in executing b.c\n");
        return 5;
    }

} else {

    int id2 = fork();

    if (id2 < 0){
        printf("Fork raise error\n");
        return 4;
    } else if (id2 == 0){

        if (execl("c.out", "c.out", NULL, (char *)NULL) == -1){
            printf("Error in executing c.c\n");
            return 6;
        }

    } else {

        char c;
        int count = 0, reading;

        while (True){
```

```
    while (ptrAC[0] == 1){
        // активное ожидание
    }
    while (ptrAB[0] == 1){
        // активное ожидание
    }
    reading = scanf("%c", &c);
    if (reading > 0){
        if (c != '\n'){
            ptrAC[count + 1] = c;
            count++;
        } else {
            ptrAC[count + 1] = c;
            ptrAC[0] = 1;

            ptrAB[1] = count;
            ptrAB[0] = 1;

            count = 0;
        }
    } else {
        ptrAB[0] = 2;
        ptrAC[0] = 2;
        break;
    }
}

if (close(fdAB) == -1 || close(fdAC) == -1){
```

```
                printf("Error while closing a files\n");
                return 17;
        }

        munmap(ptrAB, 2 * sizeof(int));
        munmap(ptrAC, 1025 * sizeof(char));

        printf("Programm stopped\n");

      }

   }

   return 0;

}
```

b.c:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>

#define True 1
```

```
int fdAB, fdCB;
/*  A ---number---> B
    C ---number---> B  */

int main(){

    fdAB = open("tempAB", O_RDWR);
    fdCB = open("tempCB", O_RDWR);

    if (fdAB < 0 || fdCB < 0){
                        printf("Error while opening a file\n");
                        return 2;
                }

    ftruncate(fdAB, 2 * sizeof(int));
    ftruncate(fdCB, 2 * sizeof(int));

    int *ptrAB;
    int *ptrCB;
    ptrAB = mmap(NULL, 2 * sizeof(int), PROT_WRITE | PROT_READ,
MAP_SHARED, fdAB, 0);
    ptrCB = mmap(NULL, 2 * sizeof(int), PROT_WRITE | PROT_READ,
MAP_SHARED, fdCB, 0);
    /*
    ptr[0] = 0 - готово к записи
    ptr[0] = 1 - запись произошла, можно читать
    ptr[0] = 2 - запись окончена, a.k.a. данных больше не будет
    */
```

```
if (ptrCB == MAP_FAILED || ptrAB == MAP_FAILED){
    printf("Error while memory map created\n");
    return 3;
            }

printf("Programm B started\n");

while (True){
    while (ptrAB[0] == 0){
        // активное ожидание
    }
    if (ptrAB[0] == 1){
        printf("B: from A: %d\n", ptrAB[1]);
        ptrAB[0] = 0;
    }
    while (ptrCB[0] == 0){
        // активное ожидание
    }
    if (ptrCB[0] == 1){
        printf("B: from C: %d\n", ptrCB[1]);
        ptrCB[0] = 0;
    }
    if (ptrAB[0] == 2 && ptrCB[0] == 2){
        break;
    }
}

if (close(fdCB) == -1 || close(fdAB) == -1){
```

```
                printf("Error while closing a files\n");
                return 17;
    }

    munmap(ptrCB, 2 * sizeof(int));
    munmap(ptrAB, 2 * sizeof(int));

    return 0;
}
```

c.c:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/wait.h>

#define True 1

int fdAC, fdCB;
/*  A ---string---> C
    C ---number---> B  */

int main(){

    fdAC = open("tempAC", O_RDWR);
```

```
    fdCB = open("tempCB", O_RDWR | O_CREAT | O_TRUNC, 0777);

    if (fdAC < 0 || fdCB < 0){
                        printf("Error while opening a file\n");
                        return 0;
            }

    ftruncate(fdCB, 2 * sizeof(int));

    char *ptrAC;
    int *ptrCB;
    ptrAC = mmap(NULL, 1025 * sizeof(char), PROT_WRITE | PROT_READ,
MAP_SHARED, fdAC, 0);
    ptrCB = mmap(NULL, 2 * sizeof(int), PROT_WRITE | PROT_READ,
MAP_SHARED, fdCB, 0);
    /*
    ptr[0] = 0 - готово к записи
    ptr[0] = 1 - запись произошла, можно читать
    ptr[0] = 2 - запись окончена, a.k.a. данных больше не будет
    */

    if (ptrCB == MAP_FAILED || ptrAC == MAP_FAILED){
       printf("Error while memory map created\n");
       return 0;
            }

    ptrCB[0] = 0;
    printf("Programm C started\n");
```

```
char string[1024];
int count = 0;
while (True){
    while (ptrAC[0] == 0){
        // активное ожидание
    }
    if (ptrAC[0] == 1){
        while(ptrAC[count + 1] != '\n'){
            string[count] = ptrAC[count + 1];
            count++;
        }
        ptrAC[0] = 0;
        printf("C:");
        for (int i = 0; i < count; i++){
            printf("%c", string[i]);
        }
        printf("\n");
        ptrCB[1] = count;
        ptrCB[0] = 1;
        count = 0;
    } else if (ptrAC[0] == 2){
        break;
    }
}

if (close(fdCB) == -1 || close(fdAC) == -1){
                        printf("Error while closing a files\n");
```

```
                return 0;
    }

    munmap(ptrCB, 2 * sizeof(int));
    munmap(ptrAC, 1025 * sizeof(char));

    return 0;
}
```

Вывод:

Данный курсовой проект подводит некоторую финальную линию в изучении курса «Операционных систем», заставляя вспомнить многие методы и технологии, которые мы использовали в лабораторных работах (данный курсач также можно было сделать с помощью «пайпов» или очередей сообщений).

Мною были выбраны memory map потому что они, как по мне, наиболее удобны для написания простых программ для межпроцессорного взаимодействия, хоть и для более крупных систем явно потребуются очереди сообщений или другие более комплексные архитектуры.

При написании курсового проекта я почувствовал, что применяю знания, накопленные за этот семестр на практике, лабораторные работы казались более типовыми. И, после написания лаб и курсового проекта, начинаешь ценить то, насколько удобнее стали языки программирования, появившиеся после языка Си.