Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

Тема работы
“Изучение взаимодействий между процессами”

Студент: Прохоров Данила Михайлович
Группа: М8О-208Б-20
Вариант: 16
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/dmprokhorov/OS

Постановка задачи

Задача: реализовать программу, в которой родительский процесс создает один дочерний процесс. Родительский процесс принимает путь к файлу и строки, которые отправляются в тот дочерний процесс, там те из них, которые оканчиваются на символы ‘;’ или ‘.’, записываются в файл, если же строки не удовлетворяют этому правилу, то они возвращаются в родительский процесс. Далее в родительском процессе сначала выводятся строки из файла (если его удалось открыть и там есть хотя бы одна строка), а потом строки, вернувшиеся из родительского процесса в дочерний.

Общие сведения о программе

В программе используются следующие библиотеки:

- <iostream> - для вывода информации на консоль
- <fstream> - для записи текста в файл.
- <unsitd.h> - для системных вызовов read и write в Ubuntu.
- <sys/wait.h> - для функции waitpid, когда родительский процесс ждёт дочерний.

В задании используются такие команды и строки, как:

- **int fd[2]** - создание массива из 2 дескрипторов, 0 - чтение (read), 1 - передача (write). И
- **pipe(fd)** - конвейер, с помощью которого выход одной команды подается на вход другой (также “труба”). Это неименованный канал передачи данных между родственными процессами.

- **pid_t child = fork ()** - создание дочернего процесса, в переменной child будет храниться “специальный код” процесса (-1 - ошибка fork, 0 - дочерний процесс, >0 - родительский)
- **read(int fd, void* buf, size_t count) (здесь дан общий пример)** - команда, предназначенная для чтения данных, посланных из другого процесса, принимающая на вход три параметра: элемент массива дескрипторов с индексом 0 (в моей программе fd1[0], fd2[0]), указатель на память получаемого объекта (переменной, массива и т.д.), размер получаемого объекта (в байтах).
- **write(int fd, void* buf, size_t count) (здесь дан общий пример) -** команда, предназначенная для записи данных в другой процесс, принимающая на вход три параметра: элемент массива дескрипторов с индексом 1 (в моей программе fd1[1], fd2[1]), указатель на память посылаемого объекта (переменной, массива и т.д.), размер посылаемого объекта (в байтах).
- **close(int fd)** - команда, закрывающая файловый дескриптор.
- **int wstatus;**
- **waitpid(child, &wstatus, 0)** – команда ожидания завершения процесса с id, равным child, если она завершилась без ошибок, то в переменной wstatus будет лежать значение 0, и родительский процесс продолжит своё выполнение, в противном случае там будет лежать ненулевое значение, и родительский процесс аварийно завершится.
- **int fstat(int fd, struct stat *statbuf) –** команда, заполняющая структуру, на которую указывает специальная структура, информацией о файле, связанном с дескриптором файла, и принимающая 2 аргумента:

  А) Файловый дескриптор.

  Б) Указатель на специальную структуру типа struct stat.

В случае успеха функция вернёт значение 0, а в противном случае -1.

нулевое значение, и родительский процесс аварийно завершится.

Общий метод и алгоритм решения

Сначала в родительском процессе вводится путь к файлу, и строки, считываемы как одна, с символами переноса строки, и символом конца строки (‘\0’), одновременно делается попытка отобразить на ОЗУ переменную size типа int, характеризующую размер будущей изменённой строки, если этого сделать не удалось, то программа, почистив нужную память и удалив все отображаемые файлы, аварийно завершается.
Далее делается попытка отобразить на ОЗУ строку ptr заданного ранее размера size, если этого сделать не удалось, то программа, почистив нужную память и удалив все отображаемые файлы, аварийно завершается.
Затем делается попытка открыть файл по введённому пути, если этого сделать не удалось, то программа, почистив нужную память и удалив все отображаемые файлы, аварийно завершается.
После делается попытка отобразить на ОЗУ ещё одну строку f, на сей раз размера sizeof(char) и ассоциированного с файловым дескриптором.
Затем программа пытается создать дочерний процесс, если этого сделать не удалось, то программа, почистив нужную память и удалив все отображаемые файлы, аварийно завершается.
В дочернем процессе после прохода по строке и некоторых операций получаются 2 строки file_string и out_string.
Делается попытка увеличить размер файла, если этого сделать не удалось, то дочерний процесс, почистив нужную память, аварийно завершается.
После делается попытка изменить размер строки f, если этого сделать не удалось, то дочерний процесс, почистив нужную память, аварийно завершается. Иначе f теперь имеет размер file_string, и вся строка file_string записывается в f.

Потом делается попытка изменить размер строки ptr, если этого сделать не удалось, то дочерний процесс, почистив нужную память, аварийно завершается. Иначе f теперь имеет размер out_string, и вся строка out_string записывается в ptr.
Значение size делается равным значению длины строки out_string + 1.
Дочерний процесс успешно завершается.
Родительский процесс как только начался, ждёт завершения дочернего. Если получаемое значение 1, то программа завершается, предварительно очистив всю память и удалив все отображаемые файлы. Если же возвращаемое значение 0, то тогда программа пытается прочесть размер файла, если возвращаемое значение функции меньше 0, то программа, почистив нужную память и удалив все отображаемые файлы, аварийно завершается.
Далее печатаются строки f и ptr. После закрывается файл, и делается попытка удалить отображаемые файлы size, f и ptr. Если этого сделать не удалось, то программа завершается, предварительно очистив всю память.
Программа успешно завершается.

Собирается программа при помощи команды g++ task.cpp -o task, запускается при помощи команды ./task.

Исходный код

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <string.h>
#include <string>
int main()
{
    printf("You're in the parent process with id [%i]\n", getpid());
    char symbol, *in = (char*)malloc(sizeof(char)), *file_path = (char*)malloc(sizeof(char));
    int *size = (int *)mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, 0, 0), counter = 0;
    if (size == MAP_FAILED)
    {
            printf("Mapping failed in creation of integer value\n");
            exit(1);
    }
    *size = 1;
    printf("Input path to the file\n");
    while ((symbol = getchar()) != '\n')
    {
            file_path[counter++] = symbol;
            if (counter == *size)
            {
```

```
                *size *= 2;
                file_path = (char*)realloc(file_path, (*size) * sizeof(char));
        }
}
file_path = (char*)realloc(file_path, (counter + 1) * sizeof(char));
file_path[counter] = '\0';
counter = 0, *size = 1;
printf("Now input some strings. If you want to end input, press Ctrl+D\n");
while ((symbol = getchar()) != EOF)
{
        in[counter++] = symbol;
        if (counter == *size)
        {
                *size *= 2;
                in = (char*)realloc(in, (*size) * sizeof(char));
        }
}
*size = counter + 1;
in = (char*)realloc(in, (*size) * sizeof(char));
in[(*size) - 1] = '\0';
char* ptr = (char *)mmap(NULL, (*size)  * sizeof(char), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, 0, 0);
if(ptr == MAP_FAILED)
{
        printf("Mapping failed in creation of array of chars\n");
        free(in);
        free(file_path);
        int err = munmap(size, sizeof(int));
        if (err != 0)
        {
                printf("Unmapping failed\n");
        }
        exit(1);
}
strcpy(ptr, in);
int fd = open(file_path, O_RDWR | O_CREAT, S_IRWXU | S_IRWXG | S_IRWXO);
if (fd == -1)
{
        printf("Failed to open file\n");
        free(in);
        free(file_path);
        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(size, sizeof(int));
        if ((err1 != 0) || (err2 != 0))
        {
                printf("Unmapping failed\n");
        }
        exit(1);
}
char* f = (char *)mmap(NULL, sizeof(char), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (f == MAP_FAILED)
{
        printf("Failed to create string associated with file\n");
        free(in);
        free(file_path);
        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(size, sizeof(int));
        if ((err1 != 0) || (err2 != 0))
        {
                printf("Unmapping failed\n");
        }
        exit(1);
}
pid_t child_pid = fork();
if (child_pid == -1)
{
        printf("Failed to create child process\n");
        free(in);
        free(file_path);
        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(size, sizeof(int));
        if ((err1 != 0) || (err2 != 0))
        {
                printf("Unmapping failed\n");
        }
        exit(1);
}
else if (child_pid == 0)
{
```

```
//child
printf("You are in child process with id [%i]\n", getpid());
std::string string = std::string(), file_string = std::string(), out_string = std::string();
for (int i = 0; i < *size; i++)
{
        if (i != (*size) - 1)
        {
                string += ptr[i];
        }
        if ((ptr[i] == '\n') || (i == (*size) - 1))
        {
                if ((i > 0) && ((ptr[i - 1] == '.') || (ptr[i - 1] == ';')))
                {
                        file_string += string;
                }
                else
                {
                        out_string += string;
                }
                string = std::string();
        }
}
if ((file_string.length()) && (file_string[file_string.length() - 1] != '\n'))
{
        file_string += '\n';
}
if ((ftruncate(fd, std::max((int)file_string.length(), 1) * sizeof(char))) == -1)
{
        printf("Failed to truncate file\n");
        free(in);
        free(file_path);
        return 1;
}
if ((f = (char *)mremap(f, sizeof(char), (file_string.length() + 1) * sizeof(char), MREMAP_MAYMOVE)) == ((void *)-1))
{
        printf("Failed to resize memory for string associated with file\n");
        free(in);
        free(file_path);
        return 1;
}
sprintf(f, "%s", file_string.c_str());
if ((out_string.length()) && (out_string[out_string.length() - 1] != '\n'))
{
        out_string += '\n';
}
if ((ptr = (char *)mremap(ptr, (*size) * sizeof(char), out_string.length() + 1, MREMAP_MAYMOVE)) == ((void *)-1))
{
```

```
                printf("Failed to truncate file for string\n");
                free(in);
                free(file_path);
                return 1;
        }
        *size = out_string.length() + 1;
        sprintf(ptr, "%s", out_string.c_str());
}
else
{
        //parent
        int wstatus;
        waitpid (child_pid, &wstatus, 0);
        if (wstatus)
        {
                free(in);
                free(file_path);
                int err1 = munmap(ptr, (*size) * sizeof(char));
                int err2 = munmap(f, counter * sizeof(char));
                int err3 = munmap(size, sizeof(int));
                if ((err1 != 0) || (err2 != 0) || (err3 != 0))
                {
```

```
                        printf("Unmapping failed\n");
                }
                exit(1);
        }
        printf("You are back in parent process with id [%i]\n", getpid());
        struct stat statbuf;
        if (fstat (fd, &statbuf) < 0)
        {
                printf ("Problems with opening file %s\n", file_path);
                exit(1);
        }
        counter = statbuf.st_size;
        printf("These strings end in character '.' or ';':\n");
        if (statbuf.st_size > 1)
        {
                printf("%s", f);
        }
        printf("--------------------------------------------------------------------------------\n");
        printf("These strings don't end in character '.' or ';':\n");
        printf("%s", ptr);
        close(fd);
        int err1 = munmap(ptr, (*size) * sizeof(char));
        int err2 = munmap(f, counter * sizeof(char));
        int err3 = munmap(size, sizeof(int));
        if ((err1 != 0) || (err2 != 0) || (err3 != 0))
        {
                printf("Unmapping failed\n");
                free(in);
                free(file_path);
                exit(1);
        }
    }
    free(in);
    free(file_path);
    return 0;
}
```

Демонстрация работы программы

Тест 1

Корректное имя файла, вводится 4 строки, 2 из них оканчиваются на точку или точку с запятой, 2 нет.

```
danila@danila-VirtualBox:~/operation_systems/OS4$ ./task
You're in the parent process with id [7691]
Input path to the file
input.txt
Now input some strings. If you want to end input, press Ctrl+D
dflbvnfdlb;
sdvsev
dsvsdgvse;e
ddsvvdsd.
You are in child process with id [7696]
You are back in parent process with id [7691]
These strings end in character '.' or ';':
dflbvnfdlb;
ddsvvdsd.
--------------------------------------------------------------------------------
These strings don't end in character '.' or ';':
sdvsev
dsvsdgvse;e
danila@danila-VirtualBox:~/operation_systems/OS4$
```

Содержание файла input.txt

```
flbvnfdlb;
dsvvdsd.
```

Тест 2

Корректное имя файла, но все строки оканчиваются на точку или точку с запятой.

```
danila@danila-VirtualBox:~/operation_systems/OS4$ ./task
You're in the parent process with id [7855]
Input path to the file
input.txt
Now input some strings. If you want to end input, press Ctrl+D
f;gojme;
ervrev;
rvre.
vr r...
.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;r;
You are in child process with id [7859]
You are back in parent process with id [7855]
These strings end in character '.' or ';':
f;gojme;
ervrev;
rvre.
vr r...
.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;r;
--------------------------------------------------------------------------------
These strings don't end in character '.' or ';':
danila@danila-VirtualBox:~/operation_systems/OS4$
```

Содержание файла input.txt

```
f;gojme;
ervrev;
rvre.
vr r...
.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;r;
```

Тест 3

Корректное имя файла, но все строки не оканчиваются на точку или точку с запятой.

```
danila@danila-VirtualBox:~/operation_systems/OS4$ ./task
You're in the parent process with id [7982]
Input path to the file
input.txt
Now input some strings. If you want to end input, press Ctrl+D
rereger
revrevre.r
rgregr;t
You are in child process with id [7991]
You are back in parent process with id [7982]
These strings end in character '.' or ';':
--------------------------------------------------------------------------------
These strings don't end in character '.' or ';':
rereger
revrevre.r
rgregr;t
danila@danila-VirtualBox:~/operation_systems/OS4$ vim input.txt
```

Содержание файла input.txt

Тест 4

Некорректное имя файла (просто символ переноса строки)

```
danila@danila-VirtualBox:~/operation_systems/OS4$ ./task
You're in the parent process with id [8342]
Input path to the file

Now input some strings. If you want to end input, press Ctrl+D
fvnfdbk
fvdrb;
dfvfdfd.
dsscdvg
Failed to open file
danila@danila-VirtualBox:~/operation_systems/OS4$
```

Выводы

Эта лабораторная работа помогла узнать такую новую вещь для меня, как отображаемые файлы, а также поупражняться в их применении.