Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Павлов О.В.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 06.03.25

Москва, 2024

**Постановка задачи**

**Вариант 8.**

В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс;
* int shm\_open(const char \*name, int oflag, mode\_t mode) – создание или открытие объекта в разделенной памяти (в реальном времени);
* int ftruncate(int fd, off\_t length) - обрезают обычный файл, указанный по имени path или ссылке fd, до размера, указанного в length (в байтах);
* int shm\_unlink(const char \*name) – удаляет объект общей памяти (в реальном времени);
* void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset) – отображение файлов или устройств в память;
* int munmap(void \*addr, size\_t length);
* int execl(const char \*path, const char \*arg,…/\* (char \*) NULL \*/) – заменяет текущий процесс другим процессом;
* pid\_t wait(int \*stat\_loc) – блокирует процесс до тех пор, пока один из его дочерних процессов не выйдет или не будет получен сигнал.

Эта лабораторная работа демонстрирует создание и взаимодействие процессов в ОС Linux с использованием разделяемой памяти (shared memory).

Программа состоит из двух частей:

1. Родительский процесс (parent.c):

* Создаёт разделяемую память;
* Записывает в неё имя файла и команду;
* Создаёт дочерний процесс (fork()), который выполняет вычисления;
* Ожидает завершения работы дочернего процесса;
* Читает результат из памяти.

1. Дочерний процесс (child.c):

* Подключается к разделяемой памяти;
* Читает из неё имя файла и команду;
* Разбирает команду на числа и выполняет деление;
* Записывает результат в разделяемую память и файл;
* Если встречается деление на 0, процесс завершает работу.

**Код программы**

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/wait.h>

#define BUFFER\_SIZE 1024

#define SHM\_NAME "/my\_shared\_memory"

int main()

{

int shm\_fd;

char \*shm\_ptr;

shm\_fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);

if (shm\_fd == -1)

{

perror("Failed to create shared memory");

return 1;

}

if (ftruncate(shm\_fd, BUFFER\_SIZE) == -1)

{

perror("Failed to set the size of shared memory");

close(shm\_fd);

shm\_unlink(SHM\_NAME);

return 1;

}

shm\_ptr = mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shm\_ptr == MAP\_FAILED)

{

perror("Memory mapping failed");

close(shm\_fd);

shm\_unlink(SHM\_NAME);

return 1;

}

char fileName[BUFFER\_SIZE];

printf("Enter the filename: ");

scanf("%s", fileName);

char command[BUFFER\_SIZE];

printf("Enter the command (numbers separated by spaces): ");

getchar();

fgets(command, BUFFER\_SIZE, stdin);

snprintf(shm\_ptr, BUFFER\_SIZE, "%s\n%s", fileName, command);

pid\_t pid = fork();

if (pid == -1)

{

perror("Fork failed");

munmap(shm\_ptr, BUFFER\_SIZE);

close(shm\_fd);

shm\_unlink(SHM\_NAME);

return 1;

}

if (pid == 0)

{

execl("./child", "child", (char \*)NULL);

perror("execl failed");

return 1;

}

else

{

wait(NULL);

printf("Child process output: %s\n", shm\_ptr);

munmap(shm\_ptr, BUFFER\_SIZE);

close(shm\_fd);

shm\_unlink(SHM\_NAME);

}

return 0;

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#define BUFFER\_SIZE 1024

#define SHM\_NAME "/my\_shared\_memory"

int main()

{

int shm\_fd;

char \*shm\_ptr;

shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);

if (shm\_fd == -1)

{

perror("Failed to open shared memory");

return 1;

}

shm\_ptr = mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shm\_ptr == MAP\_FAILED)

{

perror("Memory mapping failed");

close(shm\_fd);

return 1;

}

char fileName[BUFFER\_SIZE];

char command[BUFFER\_SIZE];

sscanf(shm\_ptr, "%[^\n]\n%[^\n]", fileName, command);

FILE \*file = fopen(fileName, "w");

if (file == NULL)

{

perror("Failed to open file");

snprintf(shm\_ptr, BUFFER\_SIZE, "Error: Failed to open file.\n");

munmap(shm\_ptr, BUFFER\_SIZE);

close(shm\_fd);

return 1;

}

int numbers[BUFFER\_SIZE], num\_count = 0;

char \*token = strtok(command, " ");

while (token != NULL)

{

numbers[num\_count++] = atoi(token);

token = strtok(NULL, " ");

}

if (num\_count < 2)

{

snprintf(shm\_ptr, BUFFER\_SIZE, "Error: Not enough numbers in command.\n");

fprintf(file, "Error: Not enough numbers in command.\n");

fclose(file);

munmap(shm\_ptr, BUFFER\_SIZE);

close(shm\_fd);

return 1;

}

int result = numbers[0];

for (int i = 1; i < num\_count; i++)

{

if (numbers[i] == 0)

{

snprintf(shm\_ptr, BUFFER\_SIZE, "Error: Division by zero.\n");

fprintf(file, "Error: Division by zero.\n");

fclose(file);

munmap(shm\_ptr, BUFFER\_SIZE);

close(shm\_fd);

return 1;

}

result /= numbers[i];

}

fprintf(file, "Result: %d\n", result);

snprintf(shm\_ptr, BUFFER\_SIZE, "Result: %d\n", result);

fclose(file);

munmap(shm\_ptr, BUFFER\_SIZE);

close(shm\_fd);

return 0;

}

**Протокол работы программы**

**Тестирование:**

gcc parent.c -o parent

gcc child.c -o child

./parent

Enter the filename: output.txt

Enter the command (numbers separated by spaces): 100 2 5

Child process output: Result: 10

./parent

Enter the filename: output.txt

Enter the command (numbers separated by spaces): 100 2 0

Child process output: Error: Division by zero.

./parent

Enter the filename: ouput.txt

Enter the command (numbers separated by spaces): qwe

Child process output: Error: Not enough numbers in command.

./parent

Enter the filename: output.txt

Enter the command (numbers separated by spaces):

Child process output: Error: Not enough numbers in command.

**Strace:**

$ strace -f ./main

**Вывод**

Эта лабораторная работа отлично показывает, как процессы могут общаться между собой через разделяемую память. Я научился создавать процессы (fork()), управлять ими (wait()), передавать данные (mmap()) и загружать другой исполняемый файл (execl()).

Особенно важно, что программа обрабатывает ошибки, например, деление на ноль. Это не просто демонстрация IPC, а реальный пример работы с памятью и процессами в Linux.

В итоге получилось следующее: родительский процесс передаёт данные дочернему, тот выполняет расчёты и возвращает результат. Такой подход используется в реальных системах для оптимизации работы и разделения задач между процессами.