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-213Б-23

Студент: Колесник Д.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 17.11.24

Москва, 2024

Постановка задачи

Вариант 15.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Вместо каналов используется shared_memory.

Общий метод и алгоритм решения

Использованные системные вызовы:

- shmget – создание новой области разделяемой памяти
- shmat – привязка области разделяемой памяти к адресному пространству процесса
- read – чтение данных из стандартного ввода
- write – запись данных в стандартных вывод
- shmdt – отсоединение области разделяемой памяти от адресного пространства процесса
- shmctl – управление областью разделяемой памяти, удаление ее

Сервер создает область разделяемой памяти, которая будет использоваться для обмена данными между клиентом и сервером. Сервер запрашивает у пользователя имя файла и строки, которые будут записаны в разделяемую память. Для подключения клиента используется идентификатор разделяемой памяти. Клиент проверяет корректность строк и записывает их в файл, указанный в сервере, либо выводит сообщение об ошибке.
При запуске программы сервер создает новую область разделяемой памяти с помощью shmget. Он выделяет память для хранения данных, а также для имени файла. Затем с помощью shmat область разделяемой памяти привязывается к адресному пространству процесса, что позволяет серверу работать с ней как с обычным массивом.
Сервер запрашивает у пользователя ввод имени файла, который будет храниться в области разделяемой памяти. Имя файла записывается в память, начиная с определенного смещения, чтобы оставить место для других данных.

Сервер формирует сообщение, которое включает идентификатор созданной области разделяемой памяти, и выводит его на стандартный вывод. Этот идентификатор может быть полезен для клиентов, которые хотят подключиться к этой области.

Сервер входит в бесконечный цикл, в котором он запрашивает ввод строки от пользователя.

Введенные строки записываются в область разделяемой памяти. Если пользователь вводит пустую строку или завершает ввод, сервер завершает цикл.

Код программы

server.c

```
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <string.h>

static char CLIENT_PROGRAM_NAME[] = "client";

int main(int argc, char **argv) {
	if (argc == 1) {
		const char msg[] = "error: no filename provided\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_SUCCESS);
	}

	char progpath[1024];
	{
		ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
		if (len == -1) {
			const char msg[] = "error: failed to read full program path\n";
			write(STDERR_FILENO, msg, sizeof(msg));
			exit(EXIT_FAILURE);
		}

		while (progpath[len] != '/')
			--len;

		progpath[len] = '\0';
	}

	int shm_fd = shm_open("/my_shared_memory", O_CREAT | O_RDWR, 0666);
	if (shm_fd == -1) {
		const char msg[] = "error: failed to create shared memory\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	}

	ftruncate(shm_fd, 4096); // Размер 4096 байт

	void *shared_memory = mmap(0, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
```

```
	if (shared_memory == MAP_FAILED) {
		const char msg[] = "error: failed to map shared memory\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	}

	const pid_t child = fork();

	switch (child) {
	case -1: {
		const char msg[] = "error: failed to spawn new process\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	} break;

	case 0: {
		pid_t pid = getpid();

		const char msg[64];
		const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a child\n",
pid);
		write(STDOUT_FILENO, msg, length);

		{
			char path[1024];
			snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
CLIENT_PROGRAM_NAME);

			char *const args[] = {CLIENT_PROGRAM_NAME, argv[1], NULL};

			int32_t status = execv(path, args);

			if (status == -1) {
				const char msg[] = "error: failed to exec into new
executable image\n";
				write(STDERR_FILENO, msg, sizeof(msg));
				exit(EXIT_FAILURE);
			}
		}
	} break;

	default: {
		pid_t pid = getpid();

		{
			char msg[64];
			const int32_t length = snprintf(msg, sizeof(msg), "%d: I'm a
parent, my child has PID %d\n", pid, child);
			write(STDOUT_FILENO, msg, length);
		}
```

```
		int child_status;
		wait(&child_status);

		if (child_status != EXIT_SUCCESS) {
			const char msg[] = "error: child exited with error\n";
			write(STDERR_FILENO, msg, sizeof(msg));
			exit(child_status);
		}
	} break;
	}

	// Освобождаем ресурсы
	munmap(shared_memory, 4096);
	shm_unlink("/my_shared_memory");
	close(shm_fd);
}
```

client.c

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h> // Для isupper()
#include <sys/mman.h>
#include <sys/stat.h>

// Проверить, что строка начинается с заглавной буквы

int main(int argc, char **argv) {
	char buf[4096];
	ssize_t bytes;

	pid_t pid = getpid();

	// Открываем разделяемую память
	int shm_fd = shm_open("/my_shared_memory", O_RDONLY, 0666);
	if (shm_fd == -1) {
		const char msg[] = "error: failed to open shared memory\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	}

	void *shared_memory = mmap(0, 4096, PROT_READ, MAP_SHARED, shm_fd, 0);
	if (shared_memory == MAP_FAILED) {
		const char msg[] = "error: failed to map shared memory\n";
```

```
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	}

	int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
	if (file == -1) {
		const char msg[] = "error: failed to open requested file\n";
		write(STDERR_FILENO, msg, sizeof(msg));
		exit(EXIT_FAILURE);
	}

	{
		char msg[128];
		int32_t len = snprintf(msg, sizeof(msg) - 1,
"%d: Start typing lines of text.Press 'Ctrl-D' or 'Enter' with no input to exit\n",
		 pid);
		write(STDOUT_FILENO, msg, len);
	}

	while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
		if (bytes < 0) {
			const char msg[] = "error: failed to read from stdin\n";
			write(STDERR_FILENO, msg, sizeof(msg));
			exit(EXIT_FAILURE);
		} else if (buf[0] == '\n') {
			break;
		}

		if (!isupper(buf[0])) {
			const char msg[] = "error: string does not start with uppercase
letter\n";
			write(STDERR_FILENO, msg, sizeof(msg));
			continue;
		}

		{
			char msg[32];
			int32_t len = snprintf(msg, sizeof(msg) - 1, "");

			int32_t written = write(file, msg, len);
			if (written != len) {
				const char msg[] = "error: failed to write to file\n";
				write(STDERR_FILENO, msg, sizeof(msg));
				exit(EXIT_FAILURE);
			}
		}

		{
			buf[bytes - 1] = '\0';
```

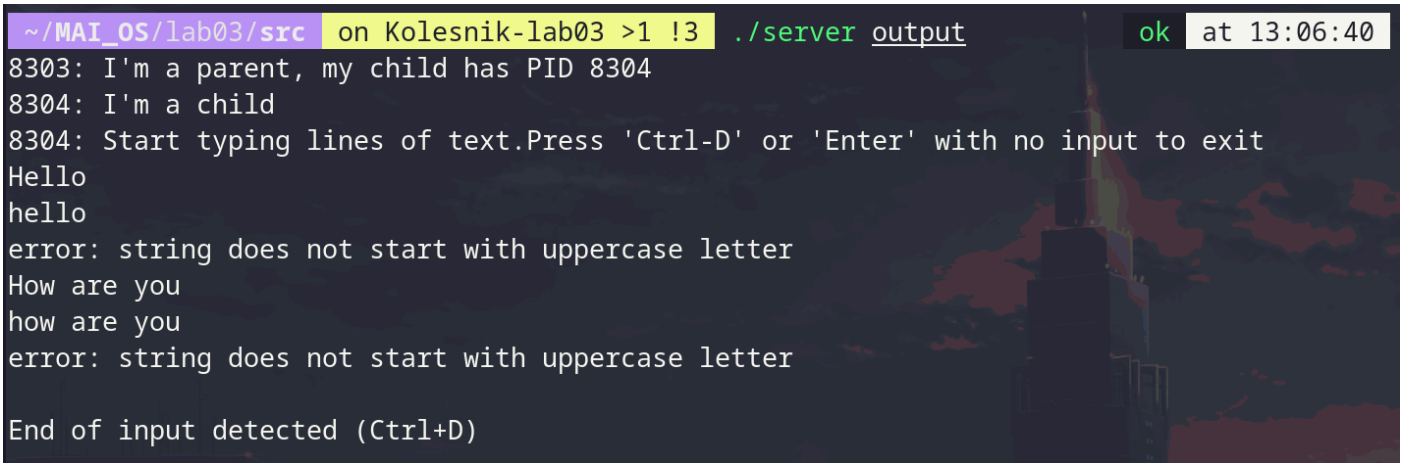
```
                        int32_t written = write(file, buf, bytes);
                        if (written != bytes) {
                                const char msg[] = "error: failed to write to file\n";
                                write(STDERR_FILENO, msg, sizeof(msg));
                                exit(EXIT_FAILURE);
                        }
                }
        }

    if (bytes == 0) {
                const char msg[] = "\nEnd of input detected (Ctrl+D)\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
        }

        const char term = '\0';
        write(file, &term, sizeof(term));

        close(file);
        munmap(shared_memory, 4096);
        close(shm_fd);
}
```

Протокол работы программы



Strace:

```
$ strace -f ./server
execve("./server", ["./server"], 0x7fff1ea40498 /* 64 vars */) = 0
brk(NULL)                               = 0x5852e4b40000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffec6d6d280) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a1617e56000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

```
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=117899, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 117899, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a1617e39000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"...,
68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a1617c00000
mprotect(0x7a1617c28000, 2023424, PROT_NONE) = 0
mmap(0x7a1617c28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x28000) = 0x7a1617c28000
mmap(0x7a1617dbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7a1617dbd000
mmap(0x7a1617e16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7a1617e16000
mmap(0x7a1617e1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
= 0x7a1617e1c000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a1617e36000
arch_prctl(ARCH_SET_FS, 0x7a1617e36740) = 0
set_tid_address(0x7a1617e36a10)         = 8739
set_robust_list(0x7a1617e36a20, 24)     = 0
rseq(0x7a1617e370e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7a1617e16000, 16384, PROT_READ) = 0
mprotect(0x5852e3a75000, 4096, PROT_READ) = 0
mprotect(0x7a1617e90000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7a1617e39000, 117899)          = 0
write(2, "error: no filename provided\n\0", 29error: no filename provided
) = 29
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Язык Си предоставляет большую гибкость в вопросе синхронизации разных приложений между собой. Shared_memory - механизм, позволяющий гибко настраивать приложения для использования одинакового ресурса и взаимодействия с файлом.