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: Солодова С. М.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата:_____

Москва, 2024

Постановка задачи

Вариант 22.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс должен перенаправлять введенные пользователем строки в один из двух дочерних процессов на основе правила фильтрации. Обмен данными между процессами должен осуществляться с использованием разделяемой памяти и сигналов для уведомления о наличии новых данных. Процессы должны принимать строки из разделяемой памяти, инвертировать их (меняя порядок символов на обратный) и записывать результат в соответствующие файлы.

Правило фильтрации: с вероятностью 80% строки отправляются в child1, иначе в child2.

Общий метод и алгоритм решения

В данной работе использованы следующие системные вызовы и функции для работы с разделяемой памятью:

- **shm_open()**: открытие или создание объекта разделяемой памяти.
- **ftruncate()**: установка размера объекта разделяемой памяти.
- **mmap()**: отображение объекта разделяемой памяти в адресное пространство процесса.
- **munmap()**: удаление отображения объекта разделяемой памяти из адресного пространства процесса.
- **shm_unlink()**: удаление имени объекта разделяемой памяти из файловой системы.

Дополнительно использовались другие системные вызовы и функции для реализации межпроцессного взаимодействия и обработки сигналов:

- **fork()**: создание нового процесса.
- **execl()**: запуск нового процесса с заменой текущего изображения процесса.
- **kill()**: отправка сигнала другому процессу.
- **sigaction()**: установка обработчика сигнала.
- **pause()**: приостановка процесса до получения сигнала.
- **waitpid()**: ожидание завершения дочернего процесса.

1. Родительский процесс (`parent.c`) запрашивает у пользователя имена файлов для двух дочерних процессов.

2. Создание разделяемой памяти:

   - Создаются два объекта разделяемой памяти для обмена данными с каждым дочерним процессом.

   - С помощью shm_open() создаются или открываются объекты разделяемой памяти.

   - Размер объектов устанавливается с помощью ftruncate().

   - Объекты отображаются в адресное пространство родительского процесса с помощью mmap().
3. С помощью fork() создаются два дочерних процесса.

4. В каждом дочернем процессе вызывается execl(), чтобы запустить программу child с передачей аргументов: имя файла для записи, имя разделяемой памяти и название сигнала для обработки (SIGUSR1 или SIGUSR2).

5. Применение правила фильтрации 4:1:

   - С помощью генератора случайных чисел (rand() % 10) определяется, какому дочернему процессу отправить строку.

   - В 80% случаев (если число от 0 до 7) строка отправляется первому дочернему процессу.

   - В 20% случаев (если число 8 или 9) — второму.

6. Отправка строки в дочерний процесс:

   - Функция send_string() копирует строку в разделяемую память соответствующего дочернего процесса.

   - Устанавливается флаг flag в значение 1, указывающий на наличие новых данных.

   - Отправляется сигнал (SIGUSR1 или SIGUSR2) соответствующему дочернему процессу с помощью kill().

7. Открывает объект разделяемой памяти с помощью shm_open() и отображает его в свое адресное пространство с помощью mmap().

8. Открывает файл для записи результатов с помощью open(). Если файла не существует, он создается.

9. **Установка обработчика сигнала:**

10. С помощью sigaction() устанавливается обработчик для соответствующего сигнала (SIGUSR1 или SIGUSR2).

11. Обработчик сигнала signal_handler проверяет флаг flag в разделяемой памяти и обрабатывает данные при его установке.

12. **Обработка сигналов и данных:**

13. Процесс ждет поступления сигналов, используя pause().

14. При получении сигнала signal_handler проверяет наличие новых данных.

15. Записывает инвертированную строку в файл с помощью write().

16. После завершения ввода строк родительский процесс отправляет сообщение "TERMINATE" в оба дочерних процесса, чтобы они корректно завершили работу.

17. Ожидает завершения дочерних процессов с помощью waitpid().

18. Освобождает ресурсы, удаляя отображения разделяемой памяти и удаляя объекты разделяемой памяти с помощью shm_unlink().

Код программы

parent.c

```
#define _POSIX_C_SOURCE 200809L

#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <signal.h>
#include <time.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>

#define MAXLINE 1024
```

```
#define SHM_SIZE MAXLINE

typedef struct {
    volatile int flag;
    char data[MAXLINE];
} shared_memory_t;

void error_exit(const char *msg) {
    const char *err_str = strerror(errno);
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, ": ", 2);
    write(STDERR_FILENO, err_str, strlen(err_str));
    write(STDERR_FILENO, "\n", 1);
    exit(EXIT_FAILURE);
}

void send_string(shared_memory_t *shm, const char *str) {
    struct timespec req;
    req.tv_sec = 0;
    req.tv_nsec = 1000000L;

    while (shm->flag == 1) {
        nanosleep(&req, NULL);
    }
    size_t len = strlen(str);
    if (len >= MAXLINE) len = MAXLINE - 1;
    memcpy(shm->data, str, len);
    shm->data[len] = '\0';
    shm->flag = 1;
}

void read_input(char *buffer, size_t size) {
    ssize_t n = 0;
    size_t total_read = 0;
    while (total_read < size - 1) {
        n = read(STDIN_FILENO, buffer + total_read, 1);
        if (n <= 0) {
            if (n == 0 && total_read == 0) {
                buffer[0] = '\0';
```

```
                return;
            }
            break;
        }
        if (buffer[total_read] == '\n') {
            buffer[total_read] = '\0';
            return;
        }
        total_read += n;
    }
    buffer[total_read] = '\0';
}

int main() {
    char filename1[MAXLINE], filename2[MAXLINE];
    pid_t pid1, pid2;
    const char *child1_shm_name = "/child1_shm";
    const char *child2_shm_name = "/child2_shm";
    shared_memory_t *child1_shm, *child2_shm;
    int shm_fd1, shm_fd2;
    char line[MAXLINE];
    size_t len = 0;
    srand(time(NULL));

    const char *prompt1 = "Enter filename for child1: ";
    write(STDOUT_FILENO, prompt1, strlen(prompt1));
    read_input(filename1, MAXLINE);

    const char *prompt2 = "Enter filename for child2: ";
    write(STDOUT_FILENO, prompt2, strlen(prompt2));
    read_input(filename2, MAXLINE);

    shm_fd1 = shm_open(child1_shm_name, O_CREAT | O_RDWR, 0666);
    if (shm_fd1 == -1)
        error_exit("shm_open child1");
    if (ftruncate(shm_fd1, sizeof(shared_memory_t)) == -1)
        error_exit("ftruncate child1");
    child1_shm = mmap(NULL, sizeof(shared_memory_t), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd1, 0);
```

```
    if (child1_shm == MAP_FAILED)
        error_exit("mmap child1");
    child1_shm->flag = 0;

    shm_fd2 = shm_open(child2_shm_name, O_CREAT | O_RDWR, 0666);
    if (shm_fd2 == -1)
        error_exit("shm_open child2");
    if (ftruncate(shm_fd2, sizeof(shared_memory_t)) == -1)
        error_exit("ftruncate child2");
    child2_shm = mmap(NULL, sizeof(shared_memory_t), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd2, 0);
    if (child2_shm == MAP_FAILED)
        error_exit("mmap child2");
    child2_shm->flag = 0;

    pid1 = fork();
    if (pid1 < 0) {
        error_exit("fork child1");
    }
    if (pid1 == 0) {
        execl("./child", "./child", filename1, child1_shm_name, "SIGUSR1", (char
*)NULL);
        error_exit("execl child1");
    }

    pid2 = fork();
    if (pid2 < 0) {
        error_exit("fork child2");
    }
    if (pid2 == 0) {
        execl("./child", "./child", filename2, child2_shm_name, "SIGUSR2", (char
*)NULL);
        error_exit("execl child2");
    }

    const char *input_prompt = "Enter strings (Ctrl+D to end):\n";
    write(STDOUT_FILENO, input_prompt, strlen(input_prompt));

    while (1) {
        read_input(line, MAXLINE);
```

```
        if (strlen(line) == 0) {
            break;
        }

        int r = rand() % 10;
        if (r < 8) {
            send_string(child1_shm, line);
            if (kill(pid1, SIGUSR1) == -1) {
                error_exit("kill SIGUSR1 to child1");
            }
        } else {
            send_string(child2_shm, line);
            if (kill(pid2, SIGUSR2) == -1) {
                error_exit("kill SIGUSR2 to child2");
            }
        }
    }

    send_string(child1_shm, "TERMINATE");
    send_string(child2_shm, "TERMINATE");
    kill(pid1, SIGUSR1);
    kill(pid2, SIGUSR2);

    if (waitpid(pid1, NULL, 0) == -1)
        error_exit("waitpid pid1");
    if (waitpid(pid2, NULL, 0) == -1)
        error_exit("waitpid pid2");

    munmap(child1_shm, sizeof(shared_memory_t));
    munmap(child2_shm, sizeof(shared_memory_t));
    shm_unlink(child1_shm_name);
    shm_unlink(child2_shm_name);

    return 0;
}
```

child.c

```
#define _POSIX_C_SOURCE 200809L
```

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>

#define MAXLINE 1024

typedef struct {
    volatile int flag;
    char data[MAXLINE];
} shared_memory_t;

shared_memory_t *shm;
int shm_fd;
int terminate = 0;
char *filename;
int file_fd;
int sig_num;

void error_exit(const char *msg) {
    const char *err_str = strerror(errno);
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, ": ", 2);
    write(STDERR_FILENO, err_str, strlen(err_str));
    write(STDERR_FILENO, "\n", 1);
    exit(EXIT_FAILURE);
}

void signal_handler(int signum) {
    if (shm->flag == 1) {
        if (strcmp(shm->data, "TERMINATE") == 0) {
            terminate = 1;
        } else {
            size_t len = strlen(shm->data);
            char inverted[MAXLINE];
            for (size_t i = 0; i < len; i++) {
                inverted[i] = shm->data[len - i - 1];
            }
            inverted[len] = '\n';
            inverted[len + 1] = '\0';
            ssize_t written = write(file_fd, inverted, len + 1);
            if (written == -1) {
                error_exit("write to file");
            }
```

```
        }
        shm->flag = 0;
    }
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        const char *msg = "Usage: ./child <filename> <shm_name> <signal_name>\n";
        write(STDERR_FILENO, msg, strlen(msg));
        exit(EXIT_FAILURE);
    }

    filename = argv[1];
    const char *shm_name = argv[2];
    const char *signal_name = argv[3];

    shm_fd = shm_open(shm_name, O_RDWR, 0666);
    if (shm_fd == -1)
        error_exit("shm_open child");
    shm = mmap(NULL, sizeof(shared_memory_t), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shm == MAP_FAILED)
        error_exit("mmap child");

    file_fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (file_fd == -1)
        error_exit("open file");

    if (strcmp(signal_name, "SIGUSR1") == 0) {
        sig_num = SIGUSR1;
    } else if (strcmp(signal_name, "SIGUSR2") == 0) {
        sig_num = SIGUSR2;
    } else {
        error_exit("Invalid signal name");
    }
    struct sigaction sa;
    memset(&sa, 0, sizeof(sa));
    sa.sa_handler = signal_handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(sig_num, &sa, NULL) == -1) {
        error_exit("sigaction");
    }

    while (!terminate) {
        pause();
    }

    close(file_fd);
```

```
    munmap(shm, sizeof(shared_memory_t));
    close(shm_fd);

    return 0;
}
```

Протокол работы программы

Тестирование:

```
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa/os_lab3$ gcc -o parent parent.c
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa/os_lab3$ gcc -o child child.c
wabisabi@wabisabi:/mnt/c/Users/zlata/OneDrive/Рабочий стол/labs/sofa/os_lab3$ ./parent
Enter filename for child1: ch1
Enter filename for child2: ch2
Enter strings (Ctrl+D to end):
str1
str2
str3
str4
str5
str6
str7
str8
str9
```

Ch1:

sofa > os_lab3 > ch1

```
1rts
2rts
3rts
5rts
6rts
7rts
9rts
aaahhhhaaaaha
```

Ch2:

sofa > os_lab3 > ch2

```
4rts
8rts
```

Strace: strace ./parent

execve("./parent", ["./parent"], 0x7ffc45befa10 /* 35 vars */) = 0

brk(NULL) = 0x559ddbb10000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7efe68225000

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

fstat(3, {st_mode=S_IFREG|0644, st_size=19243, ...}) = 0

mmap(NULL, 19243, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7efe68220000

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\0\220\243\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

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7efe6800e000

mmap(0x7efe68036000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7efe68036000

mmap(0x7efe681be000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7efe681be000

mmap(0x7efe6820d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7efe6820d000

mmap(0x7efe68213000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7efe68213000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7efe6800b000

arch_prctl(ARCH_SET_FS, 0x7efe6800b740) = 0

set_tid_address(0x7efe6800ba10) = 8468

set_robust_list(0x7efe6800ba20, 24) = 0

rseq(0x7efe6800c060, 0x20, 0, 0x53053053) = 0

mprotect(0x7efe6820d000, 16384, PROT_READ) = 0

mprotect(0x559dc3c69000, 4096, PROT_READ) = 0

mprotect(0x7efe6825d000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7efe68220000, 19243) = 0

write(1, "Enter filename for child1: ", 27Enter filename for child1:) = 27

read(0, ch1

"c", 1) = 1

read(0, "h", 1) = 1

read(0, "1", 1) = 1

read(0, "\n", 1) = 1

write(1, "Enter filename for child2: ", 27Enter filename for child2:) = 27

```
        read(0, ch2
        "c", 1)                 = 1
        read(0, "h", 1)                 = 1
        read(0, "2", 1)                 = 1
        read(0, "\n", 1)                = 1
        openat(AT_FDCWD, "/dev/shm/child1_shm",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
        ftruncate(3, 1028)              = 0
        mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) =
0x7efe68224000
        openat(AT_FDCWD, "/dev/shm/child2_shm",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
        ftruncate(4, 1028)              = 0
        mmap(NULL, 1028, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) =
0x7efe68223000
        clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7efe6800ba10) = 8550
        clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7efe6800ba10) = 8551
        write(1, "Enter strings (Ctrl+D to end):\n", 31Enter strings (Ctrl+D to end):
        ) = 31
        read(0, str1
        "s", 1)                 = 1
        read(0, "t", 1)                 = 1
        read(0, "r", 1)                 = 1
        read(0, "1", 1)                 = 1
        read(0, "\n", 1)                = 1
        kill(8550, SIGUSR1)             = 0
        read(0, 1
        "1", 1)                 = 1
        read(0, "\n", 1)                = 1
        kill(8550, SIGUSR1)             = 0
        read(0, 1
        "1", 1)                 = 1
        read(0, "\n", 1)                = 1
        kill(8551, SIGUSR2)             = 0
        read(0, 1
        "1", 1)                 = 1
        read(0, "\n", 1)                = 1
        kill(8551, SIGUSR2)             = 0
        read(0, 1
        "1", 1)                 = 1
        read(0, "\n", 1)                = 1
        kill(8550, SIGUSR1)             = 0
        read(0, 1
        "1", 1)                 = 1
        read(0, "\n", 1)                = 1
        kill(8550, SIGUSR1)             = 0
```

```
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8550, SIGUSR1)             = 0
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8550, SIGUSR1)             = 0
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8550, SIGUSR1)             = 0
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8550, SIGUSR1)             = 0
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8551, SIGUSR2)             = 0
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8551, SIGUSR2)             = 0
read(0, 1
"1", 1)                 = 1
read(0, "\n", 1)                = 1
kill(8550, SIGUSR1)             = 0
read(0, "", 1)                  = 0
kill(8550, SIGUSR1)             = 0
kill(8551, SIGUSR2)             = 0
wait4(8550, NULL, 0, NULL)          = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=8551, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
wait4(8550, NULL, 0, NULL)          = 8550
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=8550, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
wait4(8551, NULL, 0, NULL)          = 8551
munmap(0x7efe68224000, 1028)        = 0
munmap(0x7efe68223000, 1028)        = 0
unlink("/dev/shm/child1_shm")       = 0
unlink("/dev/shm/child2_shm")       = 0
exit_group(0)                   = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы была успешно разработана программа на языке C, которая демонстрирует межпроцессное взаимодействие с использованием разделяемой памяти и сигналов. Основными системными вызовами для этого являются: shm_open(), shm_unlink(), ftruncate(), mmap(). Лабораторная работа послужила отличным

практическим примером использования теоретических знаний о системах, работе с процессами и управлении памятью, что особенно актуально для дальнейшего изучения программирования на языке C и разработки приложений в Linux.