

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-210Б-23

Студент: Иванченко Макар Дмитриевич

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 12.12.24

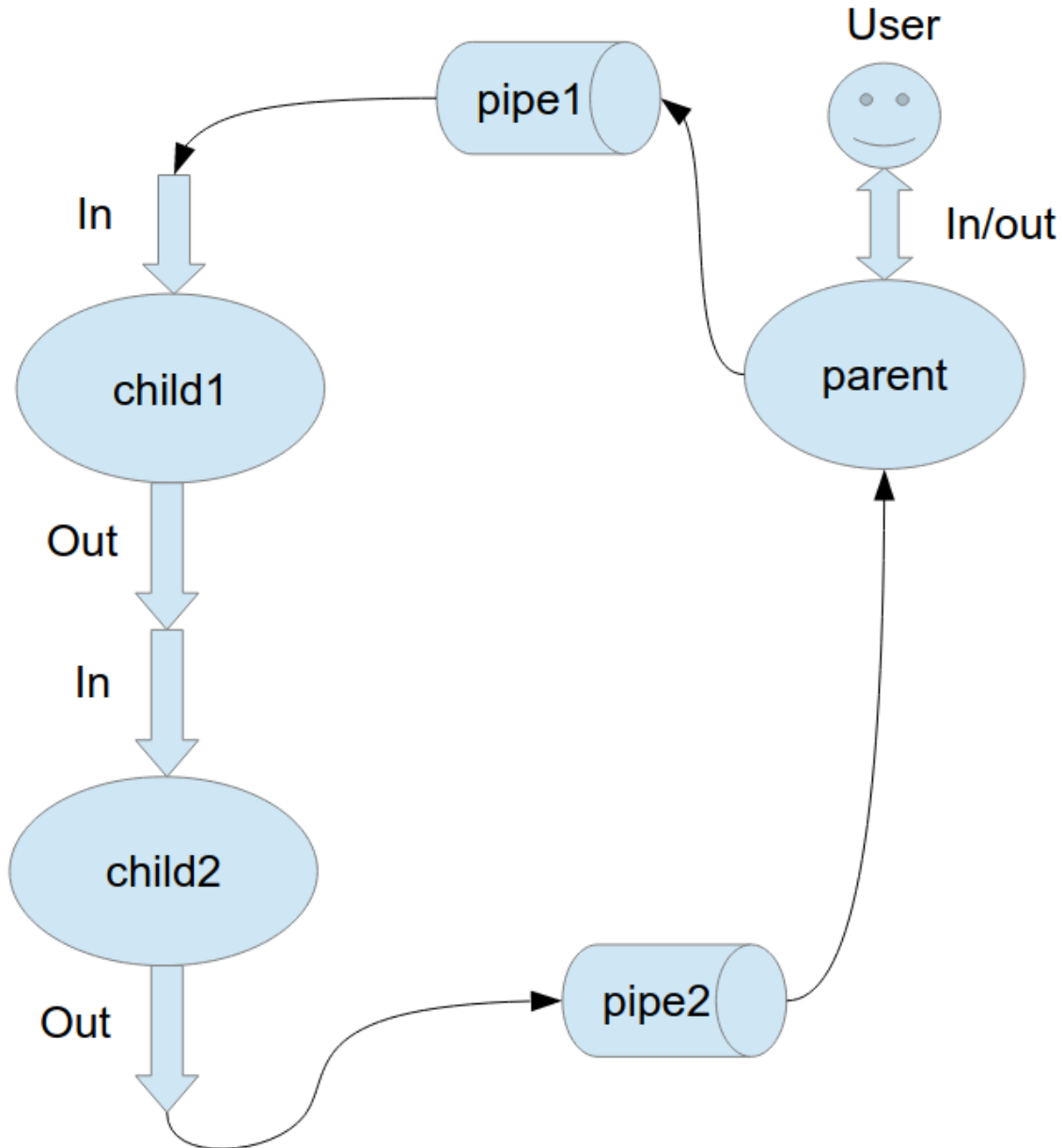
Москва, 2024

Постановка задачи

Постановка задачи

Вариант 12.

Группа вариантов 3



Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- mmap – отображение файла в память
- fork – создание дочернего процесса
- execv – замена исполняемого кода
- sem_open – создание/подключение к семафору
- sem_init – инициализация семафора
- sem_post – поднятие семафора
- sem_wait – опускание семафора
- waitpid – ожидание завершения процесса
- kill – завершение процесса

Код программы

parent.c

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/time.h>

#include <bits/mman-linux.h>

#include <stdint.h>

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include "string.h"

include <signal.h>

#include <fcntl.h>

#include <semaphore.h>

#include "parent.h"

int main()
```

```

{

char path1[4096] = "/media/sf_univer/OSlabs/lab3/child1";

char path2[4096] = "/media/sf_univer/OSlabs/lab3/child2";


int shared_fd = open("temp.txt", O_RDWR | O_CREAT, 0);

if (shared_fd == -1){

    exit(EXIT_FAILURE);

}


char* data = mmap(NULL, getpagesize(), PROT_READ | PROT_WRITE, MAP_SHARED, shared_fd, 0);

data[10] = '4';


pid_t child1 = fork();

int status;

int res_status;

if (child1 == -1)

{

    exit(EXIT_FAILURE);

}


if (child1 == 0)

{

    char *args[] = {NULL};

    status = execv(path1, args);

    if (status == -1)

    {

        const char msg[] = "error: failed to exec into new exectuable image\n";

        write(STDERR_FILENO, msg, sizeof(msg));

        exit(EXIT_FAILURE);

    }

}

```

```

}

// write(shared_fd, "aaaaa", 6);

// sem_post(semaphore1);

// wait(NULL);

// printf("%s", data);

// return 0;

else

{

    //

    // parent


    pid_t child2 = fork();

    if (child2 == -1)

    {

        exit(EXIT_FAILURE);

    }

    if (child2 == 0)

    {

        char *args[] = {NULL};

        status = execv(path2, args);


        if (status == -1)

        {

            const char msg[] = "error: failed to exec into new exectuable image\n";

            write(STDERR_FILENO, msg, sizeof(msg));

            kill(child1, SIGKILL);

            exit(EXIT_FAILURE);

        }

    }

}

else

```

```

{

char input_string[4096];

char output_string[4096];

sem_t *semaphore_p_c1;

semaphore_p_c1 = sem_open(SEM_NAME_PC1, O_CREAT, 0666, 0);

if (semaphore_p_c1 == SEM_FAILED){

    write(STDERR_FILENO, "Error connecting to semaphorepc1 in parent.\n", 60);

    kill(child1, SIGKILL);

    kill(child2, SIGKILL);

    return 1;

}

int a;

sem_init(semaphore_p_c1, 1, 0);

int n_inp = read(STDIN_FILENO, input_string, sizeof(input_string));

int n_out;

sem_t *semaphore_c2p;

semaphore_c2p = sem_open(SEM_NAME_C2P, O_CREAT, 0666, 0);

if (semaphore_c2p == SEM_FAILED){

    write(STDERR_FILENO, "Error connecting to semaphorec2p in parent.\n", 60);

    kill(child1, SIGKILL);

    kill(child2, SIGKILL);

    return 1;

}

sem_init(semaphore_c2p, 1, 0);

// while (input_string[0] != '\n' && input_string[0] != EOF)

// {

    // fprintf(stderr, "p %s", input_string);

    input_string[n_inp - 1] = '\0';

    write(shared_fd, input_string, n_inp);

```

```

// sem_post(semaphore_p_c1);

// fprintf(stderr, "wait");

// n_out = read(p2[0], output_string, sizeof(output_string));

// fprintf(stderr, "parent read %s end", output_string);

// int really_written = write(STDOUT_FILENO, output_string, n_out);

char temp = '\n';

sem_post(semaphore_p_c1);

// fprintf(stderr, "posted");

sem_wait(semaphore_c2p);

write(STDOUT_FILENO, data, strlen(data));

write(STDOUT_FILENO, "\n", 1);

// if (really_written != n_out || write(STDOUT_FILENO, &temp, 1) != 1)

// {

//     const char msg[] = "error: failed to write to stdout\n";

//     write(STDERR_FILENO, msg, sizeof(msg));

//     kill(child1, SIGKILL);

//     kill(child2, SIGKILL);

//     exit(EXIT_FAILURE);

// }


// n_inp = read(STDIN_FILENO, input_string, sizeof(input_string));

// }

// fprintf(stderr, "exit");

// if(write(p1[1], &temp, 1) != 1){

//     const char msg[] = "error: failed to write to pipe\n";

//     write(STDERR_FILENO, msg, sizeof(msg));

//     kill(child1, SIGKILL);

//     kill(child2, SIGKILL);

//     exit(EXIT_FAILURE);

// }

wait(&res_status);

```

```

        sem_unlink(SEM_NAME_PC1);
    }

    int status1, status2;

    waitpid(child1, &status1, WNOHANG);

    waitpid(child2, &status2, WNOHANG);

    if (status1 != 0 || status2 != 0)
    {
        if (status1 == 0){
            kill(child1, SIGKILL);
        } else {
            kill(child2, SIGKILL);
        }
    }
}

return res_status;

```

child1_source.c

```
#include <stdint.h>
```

```
#include <stdbool.h>
```

```
#include <unistd.h>
```

```
#include <sys/wait.h>
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <ctype.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <sys/mman.h>
```

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```



```

#include <fcntl.h>

#include <unistd.h>

#include <sys/time.h>

#include <bits/mman-linux.h>

#include <semaphore.h>

#include "parent.h"


int main(int argc, char** args){

    int shared_fd = open("temp.txt", O_RDWR | O_CREAT, 0);

    if (shared_fd == -1){

        exit(EXIT_FAILURE);

    }

    sem_t *semaphore_pc1;

    semaphore_pc1 = sem_open(SEM_NAME_PC1, O_CREAT);

    if (semaphore_pc1 == SEM_FAILED) {

        write(STDERR_FILENO, "Error connecting to semaphore pc1 in child.\n", 54);

        return 1;

    }

    sem_t *semaphore_c1c2;

    // fprintf(stderr, "waiting");

    semaphore_c1c2 = sem_open(SEM_NAME_C1C2, O_CREAT, 0666, 0);

    if (semaphore_c1c2 == SEM_FAILED) {

        sem_unlink(SEM_NAME_C1C2);

        write(STDERR_FILENO, "Error connecting to semaphore c1c2 in c1.\n", 54);

        return 1;

    }

    sem_init(semaphore_c1c2, 1, 0);

    char* data = mmap(NULL, getpagesize(), PROT_READ | PROT_WRITE, MAP_SHARED, shared_fd, 0);


    sem_wait(semaphore_pc1);

    // fprintf(stderr, "go");

```

```

size_t bytes = strlen(data);

// while(data[0] != '\n'){

    // fprintf(stderr, "%s\n", buf);

    int n = strlen(data);

    for (int i = 0; i < n; i++){

        data[i] = toupper(data[i]);

        // fprintf(stderr, "%c", data[i]);

    }

    sem_post(semaphore_c1c2);

// }

sem_unlink(SEM_NAME_C1C2);

return 0;

}

```

Child2_source.c

```

#include <stdint.h>

#include <stdbool.h>


#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

```

```

#include <sys/time.h>

#include <bits/mman-linux.h>

#include <semaphore.h>

#include "parent.h"


int main(int argc, char **args)
{
    sem_t *semaphore_c1c2;

    semaphore_c1c2 = sem_open(SEM_NAME_C1C2, O_CREAT, 0666, 0);

    if (semaphore_c1c2 == SEM_FAILED){

        write(STDERR_FILENO, "Error connecting to semaphorec1c2 in childc2.\n", 62);

        return 1;

    }

    int shared_fd = open("temp.txt", O_RDWR | O_CREAT, 0);

    char* data = mmap(NULL, getpagesize(), PROT_READ | PROT_WRITE, MAP_SHARED, shared_fd, 0);

    sem_wait(semaphore_c1c2);

    // char buf[4096];

    // size_t bytes;

    // while (bytes = read(STDIN_FILENO, buf, sizeof(buf)))

    // {

        // if (buf[0] == '\n'){

            // int written = write(STDOUT_FILENO, &buf[0], 1);

            // if (written != 1)

                // {

                    // const char msg[] = "error: failed to write to pipe\n";

                    // write(STDERR_FILENO, msg, sizeof(msg));

                    // exit(EXIT_FAILURE);

                // }

            // _exit(0);

        // }

        // if (bytes < 0)

```

```

// {

//  const char msg[] = "error: failed to read from stdin\n";

//  write(STDERR_FILENO, msg, sizeof(msg));

//  exit(EXIT_FAILURE);

// }

// fprintf(stderr, "c2read %s", buf);

// buf[bytes] = '\0';

int n = strlen(data);

char res[4096];

strcpy(res, "");


// fprintf(stderr, "%s", buf);

for (int i = 0; i < n - 1; i++)

{

    if (data[i] == ' ' && data[i + 1] == ' ')

    {

        i++;

    }

    else

    {

        strncat(res, &data[i], 1);

    }

}

strncat(res, &data[n - 1], 1);

// fprintf(stderr, "c2");

// fprintf(stderr, "%ld", bytes);

// fprintf(stderr, "2 %s", res);


sem_t *semaphore_c2p;

semaphore_c2p = sem_open(SEM_NAME_C2P, 0, 0666, 0);

if (semaphore_c2p == SEM_FAILED) {

```

```

    write(STDERR_FILENO, "Error connecting to semaphore c2p in child2.\n", 62);

}

strcpy(data, res);

// fprintf(stderr, "%s\n", res);

// fprintf(stderr, "%s\n", data);

sem_post(semaphore_c2p);

// }

sem_unlink(SEM_NAME_C2P);

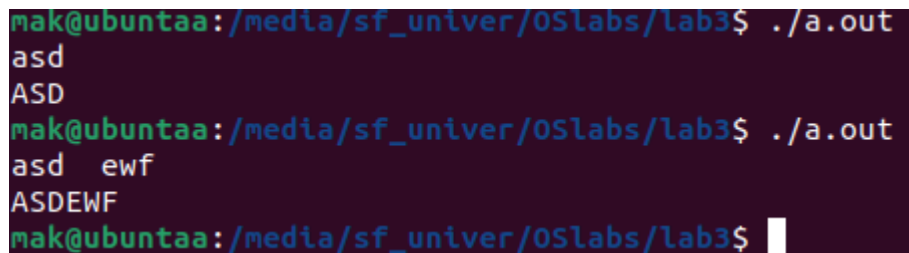
return 0;

}

```

}Протокол работы программы

Тестирование:



```

mak@ubuntaa: /media/sf_univer/05labs/lab3$ ./a.out
asd
ASD
mak@ubuntaa: /media/sf_univer/05labs/lab3$ ./a.out
asd ewf
ASDEWF
mak@ubuntaa: /media/sf_univer/05labs/lab3$

```

Strace:

```
execve("./a.out", [ "./a.out" ], 0x7ffd1f5863b0 /* 57 vars */) = 0
```

```
brk(NULL) = 0x5a2fa1993000
```

```
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd2fc8af00) = -1 EINVAL (Недопустимый аргумент)
```

```
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7333e6676000
```

```
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
```

```
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
```

```
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=59683, ...}, AT_EMPTY_PATH) = 0
```

mmap(NULL, 59683, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7333e6667000

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\3\0>\0\1\0\0\0P\237\2\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"..., 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) = 0x7333e6400000

mprotect(0x7333e6428000, 2023424, PROT_NONE) = 0

mmap(0x7333e6428000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7333e6428000

mmap(0x7333e65bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7333e65bd000

mmap(0x7333e6616000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7333e6616000

mmap(0x7333e661c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7333e661c000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7333e6664000

arch_prctl(ARCH_SET_FS, 0x7333e6664740) = 0

set_tid_address(0x7333e6664a10) = 3888

set_robust_list(0x7333e6664a20, 24) = 0

rseq(0x7333e66650e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7333e6616000, 16384, PROT_READ) = 0

mprotect(0x5a2fa0ad3000, 4096, PROT_READ) = 0

mprotect(0x7333e66b0000, 8192, PROT_READ) = 0

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

munmap(0x7333e6667000, 59683) = 0

openat(AT_FDCWD, "temp.txt", O_RDWR|O_CREAT, 000) = 3


```
unlink("/dev/shm/sem.nqVXdn")    = 0
```

close(4) = 0

write(3, "asdf\0", 5) = 5

futex(0x7333e6675000, FUTEX_WAKE, 1) = 1

futex(0x7333e6674000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0

write(1, "ASDF", 4ASDF) = 4

write(1, "\n", 1

) = 1

wait4(-1, [{ WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 3890

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3890, si_uid=1000, si_status=0, si_etime=0, si_stime=0} ---

unlink("/dev/shm/sem.sync_semaphoreasdasdsdfertere1") = 0

wait4(3889, 0x7ffd2fc87f64, WNOHANG, NULL) = 0

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3889, si_uid=1000, si_status=0, si_etime=0, si_stime=0} ---

wait4(3890, 0x7ffd2fc87f68, WNOHANG, NULL) = -1 ECHILD (Нет дочерних процессов)

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе лабораторной работы я приобрел базовые навыки по работе с разделяемой памятью в си. Я научился создавать объект разделяемой памяти, записывать в него данные и читать их из него. Также я узнал о работе с семафорами, научился использовать их для синхронизации при работе с разделяемой памятью. Помимо этого, я узнал о файловых системах и памяти в целом.