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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

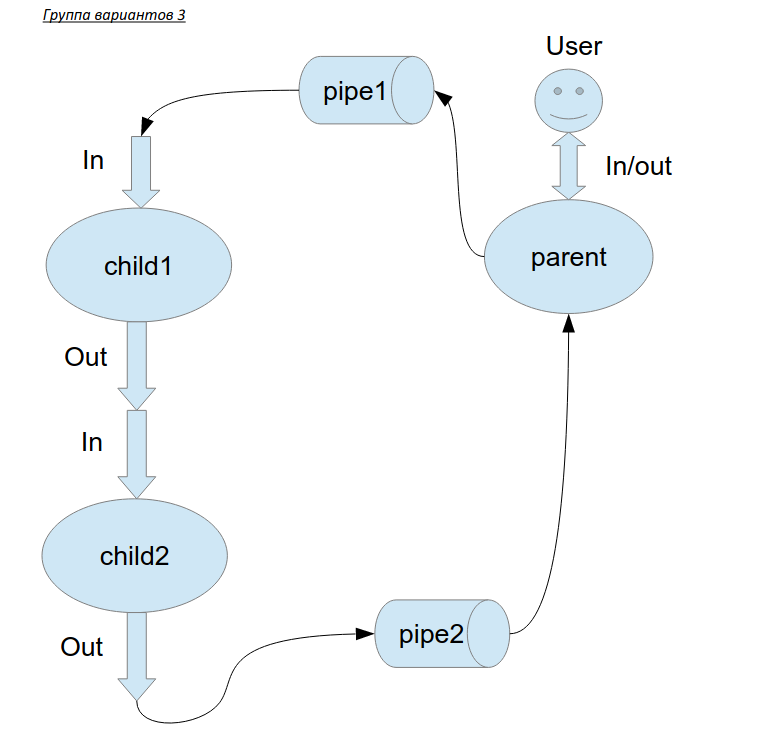
Дата: 12.12.24

Москва, 2024

**Постановка задачи**

**Постановка задачи**

**Вариант 12.**

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 argsc, 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 semaphorepc1 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 argsc, 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);

// }

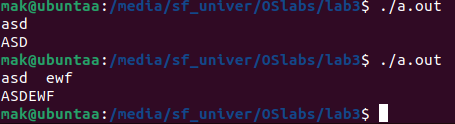
sem\_unlink(SEM\_NAME\_C2P);

return 0;

}

}**Протокол работы программы**

**Тестирование:**



**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\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) = 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

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7333e66af000

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7333e6664a10) = 3889

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7333e6664a10) = 3890

openat(AT\_FDCWD, "/dev/shm/sem.sync\_semaphoasdasdsdfertere1", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

getrandom("\x45\xb0\xc1\x0e\x6f\x83\x88\x53", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.PYYZhX", 0x7ffd2fc87bf0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT\_FDCWD, "/dev/shm/sem.PYYZhX", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 4

write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7333e6675000

link("/dev/shm/sem.PYYZhX", "/dev/shm/sem.sync\_semaphoasdasdsdfertere1") = 0

newfstatat(4, "", {st\_mode=S\_IFREG|0664, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

getrandom("\xbd\xf4\x8f\xfd\x79\x6d\xaf\xdf", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5a2fa1993000

brk(0x5a2fa19b4000) = 0x5a2fa19b4000

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

close(4) = 0

read(0, asdf

"asdf\n", 4096) = 5

openat(AT\_FDCWD, "/dev/shm/sem.sync\_semaphoasdasdsardsftetre3", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

getrandom("\x71\xd5\x37\x4c\xff\xc2\x4f\x17", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.nqVXdn", 0x7ffd2fc87bf0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT\_FDCWD, "/dev/shm/sem.nqVXdn", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 4

write(4, "\0\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7333e6674000

link("/dev/shm/sem.nqVXdn", "/dev/shm/sem.sync\_semaphoasdasdsardsftetre3") = 0

newfstatat(4, "", {st\_mode=S\_IFREG|0664, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

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\_utime=0, si\_stime=0} ---

unlink("/dev/shm/sem.sync\_semaphoasdasdsdfertere1") = 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\_utime=0, si\_stime=0} ---

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

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе лабораторной работе 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.