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: Коваль М.Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

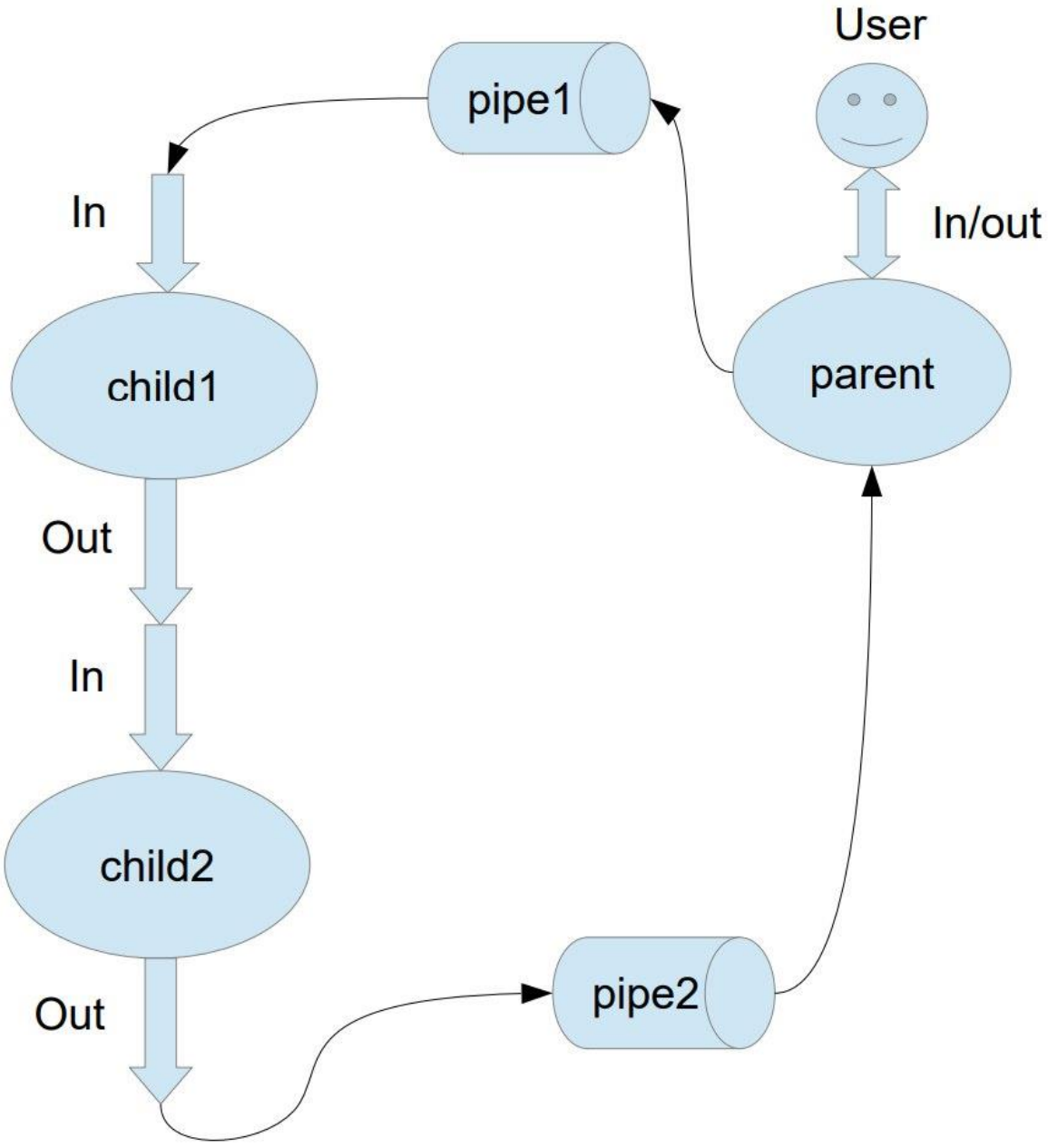
Дата: 13.02.24

Москва, 2024

Постановка задачи

Вариант 14.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа должна создать дочерние процессы для решения поставленной задачи. Взаимодействие между процессами осуществляется через общую память.



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.

14 вариант) Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int execv(const char *filename, char *const argv[]) - замена образа памяти процесса
- int shm_open(const char *name, int oflag, mode_t mode) - открытие объекта общей памяти
- int sem_init(sem_t *sem, int pshared, unsigned int value) - инициализация семафора
- int sem_post(sem_t *sem) - инкремент счётчика семафора
- int sem_wait(sem_t *sem) - декремент счетчика семафора
- int shm_unlink(const char *name) - закрытие объекта общей памяти
- int open(const char *pathname, int flags, mode_t mode) - открытие\создание файла
- int close(int fd) - закрытие файла
- void *mmap(void addr[.length], size_t length, int prot, int flags, int fd, off_t offset) - отображение памяти в виртуальное адресное пространство
- int munmap(void addr[.length], size_t length) - отсоединение отображения

Код программы

main.c:

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include "pshm_ucase.h"

#if 1
char *get_string(int *len) {
    *len = 0;
    int capacity = 1;
    char *s = (char *)malloc(sizeof(char));

    char c = getchar();

    while (c != '\n' && c != EOF) {
        s[(*len)++] = c;

        if (*len >= capacity) {
            capacity *= 2;  // увеличиваем ёмкость строки в два раза
            s = (char *)realloc(s, capacity * sizeof(char));  // перевыделяем память на строку с увеличенной ёмкостью
        }

        c = getchar();
    }

    s[*len] = '\0';  // завершаем строку символом конца строки

    return s;
}
#endif
char *get_string(int *len);


int main() {
    const char *args[] = {"./trash", NULL};
    const char* shmpath = args[1];
    struct shmbuf *shmp;
    int len;
```

```
char *s = get_string(&len); // считываем динамическую строку
const char name[] = "/sosal"; // Имя общей памяти
int fd = shm_open(name, O_CREAT | O_RDWR, 0600);
if (ftruncate(fd, sizeof(struct shmbuf)) == -1)
    errExit("ftruncate");
shmp = mmap(NULL, sizeof(*shmp), PROT_READ | PROT_WRITE,
                    MAP_SHARED, fd, 0);

if (shmp == MAP_FAILED)
    errExit("mmap");

if (sem_init(&shmp->sem1, 1, 0) == -1)
    errExit("sem_init-sem1");
if (sem_init(&shmp->sem2, 1, 0) == -1)
    errExit("sem_init-sem2");
if (sem_init(&shmp->sem3, 1, 0) == -1)
    errExit("sem_init-sem3");

shmp->cnt = len;
memcpy(shmp->buf, s, len); // записываем строку в общую память
sem_post(&shmp->sem1); // Разрешаем чтение первому ребенку

pid_t pId1, pId2;
pId1 = fork();
if (pId1 == 0) {
    int status = execv("./child1", (char **)args);
    exit(status);
}

if (pId1){
    pId2 = fork();
}

if (pId2 == 0){
    int status = execv("./child2", (char **)args);
    exit(status);
}
if (sem_post(&shmp->sem2) == -1)
    errExit("sem_post");

sem_wait(&shmp->sem3);

printf("Received array of size %ld: %s\n", strlen(shmp->buf), shmp->buf);


munmap(shmp, sizeof(struct shmbuf));
```

```
    shm_unlink(name);
    free(s);

    return 0;
}
```

ChildOne.c

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include "pshm_ucase.h"

int main() {
    const char name[] = "/sosal";
    struct shmbuf *shmp;
    int fd = shm_open(name, O_RDWR, 0);
    shmp = mmap(NULL, sizeof(shmp), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    sem_wait(&shmp->sem1);

    int len = strlen(shmp->buf);
    for (int i = 0; i < len; i++) {
            char c = shmp->buf[i];
            if (c >= 'A' && c <= 'Z') {
                shmp->buf[i] += 'a' - 'A';
            }
        }
    sem_post(&shmp->sem2);
    return 0;
}
```

ChildTwo.c

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

```c
#include "pshm_ucase.h"

int main() {
    const char name[] = "/sosal";
    struct shmbuf *shmp;
    int fd = shm_open(name, O_RDWR, 0);

    shmp = mmap(NULL, sizeof(shmp), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);

    sem_wait(&shmp->sem2);

    int len_cpy = 0, capacity = 2;
    char *cpy = (char*) malloc(capacity * sizeof(char));
    cpy[0] = shmp->buf[0];

    for(int i = 0; i < shmp->cnt; i++) {
        if (len_cpy >= capacity) {
            capacity *= 2;
            cpy = (char*) realloc(cpy, capacity * sizeof(char));
        }

        if (!(shmp->buf[i] == ' ' && shmp->buf[i + 1] == ' ')) {
            cpy[len_cpy++] = shmp->buf[i];
        }
    }

    cpy = (char*) realloc(cpy, (capacity + 1) * sizeof(char));
    cpy[++len_cpy] = '\0';

    for(int i = 0; i < len_cpy; i++) {
        shmp->buf[i] = cpy[i];
    }

    sem_post(&shmp->sem3); // Разрешаем чтение родителю

    free(cpy);
    return 0;
}
```

Протокол работы программы

Тестирование

```
(base) lab@DESKTOP-KPRMFIO:~/mk/OSlabs/lr3/src$ ./main.o
 TeSttED    FoR FFEDO_Skaaa   !
    !ived array of size 30: testted for ffedo_skaaa !
```

Strace uname

/usr/lib/locale/C.UTF-8/LC_ADDRESS", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_ADDRESS", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=127, ...}) = 0

mmap(NULL, 127, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf4413b000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_NAME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_NAME", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=62, ...}) = 0

mmap(NULL, 62, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf4413a000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_PAPER", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_PAPER", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=34, ...}) = 0

mmap(NULL, 34, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43f1d000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_MESSAGES", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MESSAGES", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MESSAGES/SYS_LC_MESSAGES", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=48, ...}) = 0

mmap(NULL, 48, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43f1c000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_MONETARY", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_MONETARY", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=270, ...}) = 0

mmap(NULL, 270, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43f1b000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_COLLATE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_COLLATE", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=1406, ...}) = 0

mmap(NULL, 1406, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43f1a000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_TIME", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_TIME", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=3360, ...}) = 0

mmap(NULL, 3360, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43f19000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_NUMERIC", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_NUMERIC", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=50, ...}) = 0

mmap(NULL, 50, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43f18000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/locale/C.UTF-8/LC_CTYPE", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/usr/lib/locale/C.utf8/LC_CTYPE", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=360460, ...}) = 0

mmap(NULL, 360460, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7faf43ebf000

close(3) = 0

uname({sysname="Linux", nodename="DESKTOP-KPRMFIO", ...}) = 0

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}) = 0

write(1, "Linux\n", 6Linux

) = 6

close(1) = 0

close(2) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы я освоил практическое понимание работы с shared memory. В рамках выполнения работы была создана и отлажена программа на си. Самым сложным по сей день является отлов ошибки, ведь где-то теряется память, но вывод результата остается корректным, проделанной работой доволен.