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-211Б-23

Студент:Косов В.В.

Преподаватель: Бахарев В.Д.

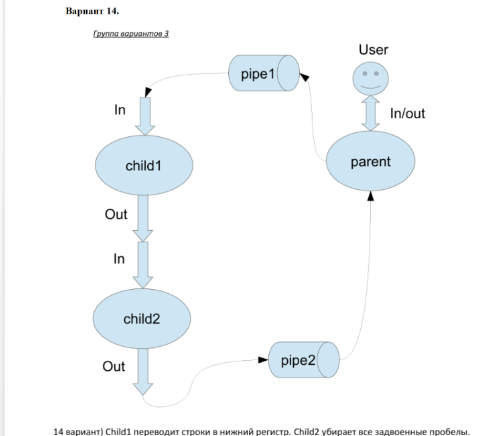
Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 26.12.24

Москва, 2024

**Постановка задачи**

**Вариант 14.**

****

Реализовать программу с обменом данными через shared memory а не pipe.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

1. **int shmctl(int shmid, int cmd, struct shmid\_ds \*buf**); – выполняет операции управления над сегментом разделяемой памяти.
2. **void \*shmat(int shmid, const void \*shmaddr, int shmflg);** – присоединяет сегмент разделяемой памяти к адресному пространству процесса.
3. **int shmget(key\_t key, size\_t size, int shmflg);** – создает или получает доступ к сегменту разделяемой памяти.
4. **int execl(const char \*path, const char \*arg, ...);** – заменяет образ текущего процесса на образ нового процесса, определенного в пути path.
5. **int close(int fd);** – закрывает файловый дескриптор.
6. **ssize\_t read(int fd, void \*buf, size\_t count);** – читает данные из файлового дескриптора.
7. **ssize\_t write(int fd, const void \*buf, size\_t count);** – записывает данные в файловый дескриптор.

**Описание лабораторной работы**

В рамках лабораторной работы была разработана программа, которая демонстрирует межпроцессное взаимодействие с использованием каналов (pipes) и системных вызовов. Программа состоит из родительского процесса и двух дочерних процессов, которые обрабатывают данные последовательно.

**Цель лабораторной работы**

Целью лабораторной работы было изучение и применение системных вызовов для создания и управления процессами, а также использование shared memory для межпроцессного взаимодействия.

**Описание программы**

Программа состоит из трех частей:

1. **Родительский процесс (parent.c)**: Считывает данные из стандартного ввода (stdin), передает их первому дочернему процессу через канал, затем передает обработанные данные второму дочернему процессу и выводит окончательный результат в стандартный вывод (stdout).
2. **Первый дочерний процесс (child1.c)**: Считывает данные из shared memory, преобразует их в верхний регистр и записывает обратно в shared memory.
3. **Второй дочерний процесс (child2.c)**: Считывает данные из shared memory, заменяет пробельные символы на подчеркивания и записывает обратно в shared memory.

**Код программы**

parent.c

#include "main.h"

int main() {

    int shmid;

    struct SharedData \*shmaddr;

    int child1, child2;

    shmid = shmget(SHM\_KEY, sizeof(struct SharedData), IPC\_CREAT | 0666);

    if (shmid < 0)

        exit(EXIT\_FAILURE);

    shmaddr = shmat(shmid, NULL, 0);

    if (shmaddr == (void \*)-1)

        exit(EXIT\_FAILURE);

    read(STDIN\_FILENO, shmaddr->message, sizeof(shmaddr->message));

    shmaddr->flag = 1;

    child1 = fork();

    if (child1 == -1)

        exit(EXIT\_FAILURE);

    if (child1 == 0) {

        execl("./out/child1", "child1", NULL);

        exit(EXIT\_FAILURE);

    }

    waitpid(child1, NULL, 0);

    while (shmaddr->flag != 2) {

        printf("Parent process waiting...\n");

    }

    child2 = fork();

    if (child2 == -1) {

        exit(EXIT\_FAILURE);

    }

    if (child2 == 0) {

        execl("./out/child2", "child2", NULL);

        exit(EXIT\_FAILURE);

    }

    waitpid(child2, NULL, 0);

    printf("%s\n", shmaddr->message);

    if (shmdt(shmaddr) < 0)

        exit(EXIT\_FAILURE);

    if (shmctl(shmid, IPC\_RMID, NULL) < 0)

        exit(EXIT\_FAILURE);

    return 0;

}

child1.c

#include "main.h"

int main() {

    int shmid;

    struct SharedData \*shmaddr;

    shmid = shmget(SHM\_KEY, sizeof(struct SharedData), 0666);

    if (shmid < 0)

        exit(EXIT\_FAILURE);

    shmaddr = shmat(shmid, NULL, 0);

    if (shmaddr == (void \*)-1)

        exit(EXIT\_FAILURE);

    int cur = 0;

    while (shmaddr->message[cur] != '\0') {

        shmaddr->message[cur] = tolower(shmaddr->message[cur]);

        cur++;

    }

    shmaddr->flag = 2;

    if (shmdt(shmaddr) < 0)

        exit(EXIT\_FAILURE);

    return 0;

}

child2.c

#include "main.h"

#include "main.h"

int main() {

int shmid;

struct SharedData \*shmaddr;

shmid = shmget(SHM\_KEY, sizeof(struct SharedData), 0666);

if (shmid < 0)

exit(EXIT\_FAILURE);

shmaddr = shmat(shmid, NULL, 0);

if (shmaddr == (void \*)-1)

exit(EXIT\_FAILURE);

int cur = 0, write\_index = 0;

int prev\_was\_space = 0;

while (shmaddr->message[cur] != '\0') {

if (shmaddr->message[cur] == ' ') {

if (prev\_was\_space) {

cur++;

continue;

}

prev\_was\_space = 1;

} else {

prev\_was\_space = 0;

}

shmaddr->message[write\_index++] = shmaddr->message[cur++];

}

shmaddr->message[write\_index] = '\0';

shmaddr->flag = 3;

if (shmdt(shmaddr) < 0)

exit(EXIT\_FAILURE);

return 0;

}

main.h

#pragma once

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/wait.h>

#include <ctype.h>

#define BUFFER\_SIZE 256

#define SHM\_KEY 1234

#define SHM\_SIZE 1024

struct SharedData {

    char message[SHM\_SIZE];

    int flag;

};

**Протокол работы программы**

**Тестирование:**

**vsevolod@DESKTOP-K08EACJ:~/os\_labs/laba\_3$ ./test.sh**

**--------------------------------------------------**

**Start program...**

**--------------------------------------------------**

**gcc -Wall -Werror -c parent.c -o parent.o**

**gcc -o out/parent parent.o**

**gcc -Wall -Werror -c child1.c -o child1.o**

**gcc -o out/child1 child1.o**

**gcc -Wall -Werror -c child2.c -o child2.o**

**gcc -o out/child2 child2.o**

execve("./out/parent", ["./out/parent"], 0x7ffe4804d3b0 /\* 27 vars \*/) = 0

brk(NULL) = 0x5654d7706000

**mmap (NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f4b0c84b000**

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=20335, ...}) = 0

mmap(NULL, 20335, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f4b0c846000

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) = 0x7f4b0c634000

mmap(0x7f4b0c65c000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f4b0c65c000

mmap(0x7f4b0c7e4000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f4b0c7e4000

mmap(0x7f4b0c833000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f4b0c833000

mmap(0x7f4b0c839000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f4b0c839000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f4b0c631000

arch\_prctl(ARCH\_SET\_FS, 0x7f4b0c631740) = 0

set\_tid\_address(0x7f4b0c631a10) = 34278

set\_robust\_list(0x7f4b0c631a20, 24) = 0

rseq(0x7f4b0c632060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f4b0c833000, 16384, PROT\_READ) = 0

mprotect(0x5654bdf69000, 4096, PROT\_READ) = 0

mprotect(0x7f4b0c883000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f4b0c846000, 20335) = 0

**shmget(0x4d2, 1028, IPC\_CREAT|0666) = 12**

**shmat(12, NULL, 0) = 0x7f4b0c84a000**

**read(0, "hello world\nthis is a test\n", 1024) = 27**

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f4b0c631a10) = 34279**

**wait4(34279, NULL, 0, NULL) = 34279**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=34279, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f4b0c631a10) = 34280**

**wait4(34280, NULL, 0, NULL) = 34280**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=34280, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=1 /\* 0.01 s \*/} ---

fstat(1, {st\_mode=S\_IFREG|0600, st\_size=0, ...}) = 0

getrandom("\x83\x68\x3b\x9f\x41\x03\xf8\xb9", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5654d7706000

brk(0x5654d7727000) = 0x5654d7727000

**shmdt(0x7f4b0c84a000) = 0**

**shmctl(12, IPC\_RMID, NULL) = 0**

**write(1, "HELLO\_WORLD\nTHIS\_IS\_A\_TEST\n\n", 28) = 28**

exit\_group(0) = ?

+++ exited with 0 +++

**--------------------------------------------------**

**hello world**

**--------------------------------------------------**

**rm -f out/parent out/child1 out/child2 parent.o child1.o child2.o**

**Вывод**

В рамках лабораторной работы была разработана программа, демонстрирующая межпроцессное взаимодействие с использованием shared memory и системных вызовов. Программа состоит из родительского процесса и двух дочерних процессов, которые обрабатывают данные последовательно. Родительский процесс считывает данные из стандартного ввода, передает их первому дочернему процессу, который преобразует их в нижний регистр, затем передает обработанные данные второму дочернему процессу, который убирает задвоенные пробелы, и выводит окончательный результат в стандартный вывод.