**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика» Кафедра: 806 «Вычислительная математика и программирование» Дисциплина: «Операционные системы»

**Лабораторная работа № 4**

Студент: Зверев М.Е.

Группа: М8О-306Б-19

Преподаватель: Соколов А.А.

Дата:

Оценка:

Москва, 2020

**Цель работы**

Приобретение практических навыков в:

● Освоение принципов работы с файловыми системами

● Обеспечение обмена данных между процессами посредством технологии «File mapping»

**Задание**

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 (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Вариант:**

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float.

**Общие сведения о программе**

Программа компилируется из файла laba.c. Для реализации поставленной задачи в программе используются следующие системные вызовы: ***mmap*** - создает отображение файла в память.

***fork*** *-* создает копию текущего процесса, который является дочерним процессом для текущего процесса.

***open*** *-* открывает файл. Аргументом можно задать, открыть файл на запись или чтение*.*

***close*** *-* закрывает файл.

***mkstemp*** - создает временный файл с уникальным именем. ***unlink*** - удаляет файл, определенный по pathname.

**Листинг программы**

**parent.c**

**#include <stdio.h>**

**#include <unistd.h>**

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

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

**#include <fcntl.h>**

**#include <pthread.h>**

**int main(){**

**printf("enter file name:");**

**char filename[256];**

**if (scanf("%s", filename) <= 0){**

**printf("input format error\n");**

**return -1;**

**}**

**char pipeName[] = "pipe";**

**char mutexName[] = "mutex";**

**char mutex2Name[] = "mutex2";**

**char pipe1SizeName[] = "size";**

**int pipe1 = shm\_open(pipeName, O\_RDWR | O\_CREAT, S\_IRWXU);**

**int pipe1Size = shm\_open(pipe1SizeName, O\_RDWR | O\_CREAT, S\_IRWXU);**

**int mutex1 = shm\_open(mutexName, O\_RDWR | O\_CREAT, S\_IRWXU);**

**int mutex2 = shm\_open(mutex2Name, O\_RDWR | O\_CREAT, S\_IRWXU);**

**if (pipe1 == -1 || pipe1Size == -1 || mutex1 == -1 || mutex2 == -1){**

**perror("shm\_open\n");**

**return -1;**

**}**

**if (ftruncate(pipe1, getpagesize()) == -1){**

**perror("ftruncate error\n");**

**return -1;**

**}**

**if (ftruncate(pipe1Size, sizeof(int)) == -1){**

**perror("ftruncate error\n");**

**return -1;**

**}**

**if (ftruncate(mutex1, sizeof(pthread\_mutex\_t)) == -1){**

**perror("ftruncate error\n");**

**return -1;**

**}**

**if (ftruncate(mutex2, sizeof(pthread\_mutex\_t)) == -1){**

**perror("ftruncate error\n");**

**return -1;**

**}**

**float\* mmfData = (float \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1, 0);**

**int\* mmfDataSize = (int \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1Size, 0);**

**\*mmfDataSize = -1;**

**pthread\_mutex\_t\* mutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, mutex1, 0);**

**pthread\_mutex\_t\* secondMutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, mutex2, 0);**

**if (mmfData == MAP\_FAILED || mmfDataSize == MAP\_FAILED || mutex == MAP\_FAILED || secondMutex == MAP\_FAILED){**

**perror("mmap error\n");**

**return -1;**

**}**

**pthread\_mutexattr\_t mutexAttribute;**

**if (pthread\_mutexattr\_init(&mutexAttribute) != 0){**

**perror("pthread\_mutexattr\_init error\n");**

**return -1;**

**}**

**if (pthread\_mutexattr\_setpshared(&mutexAttribute, PTHREAD\_PROCESS\_SHARED) != 0){**

**perror("pthread\_mutexattr\_setpshared error\n");**

**return -1;**

**}**

**if (pthread\_mutex\_init(mutex, &mutexAttribute) != 0){**

**perror("pthread\_mutex\_init error\n");**

**return -1;**

**}**

**if (pthread\_mutex\_init(secondMutex, &mutexAttribute) != 0){**

**perror("pthread\_mutex\_init error\n");**

**return -1;**

**}**

**if (pthread\_mutex\_lock(secondMutex) != 0){**

**perror("pthread\_mutex\_lock error\n");**

**return -1;**

**}**

**if (pthread\_mutex\_lock(mutex) != 0){**

**perror("pthread\_mutex\_lock error\n");**

**return -1;**

**}**

**int id = fork();**

**if (id == -1){**

**perror("fork error\n");**

**return -1;**

**} else if (id == 0) {**

**char\* argv[] = {"child", filename, mutexName, mutex2Name, pipeName, pipe1SizeName, (char \*)NULL};**

**if (execv("child", argv) == -1){**

**perror("execl error\n");**

**return -1;**

**}**

**} else {**

**printf("insert sum terms:\n");**

**float currentTerm;**

**char c;**

**int cnt = 0;**

**while(scanf("%f%c", &currentTerm, &c) > 0){**

**mmfData[cnt] = currentTerm;**

**++cnt;**

**if (c == '\n'){**

**\*mmfDataSize = cnt;**

**if (pthread\_mutex\_unlock(mutex) != 0){**

**perror("pthread\_mutex\_unlock error\n");**

**return -1;**

**}**

**if (pthread\_mutex\_lock(secondMutex) != 0){**

**perror("pthread\_mutex\_lock error\n");**

**return -1;**

**}**

**cnt = 0;**

**}**

**}**

**printf("end\n");**

**\*mmfDataSize = -2;**

**if (pthread\_mutex\_unlock(mutex) != 0){**

**perror("pthread\_mutex\_unlock error\n");**

**return -1;**

**}**

**}**

**if (shm\_unlink(pipeName) != 0){**

**perror("shm\_unlink error\n");**

**return -1;**

**}**

**if (shm\_unlink(pipe1SizeName) != 0){**

**perror("shm\_unlink error\n");**

**return -1;**

**}**

**if (shm\_unlink(mutexName) != 0){**

**perror("shm\_unlink error\n");**

**return -1;**

**}**

**if (shm\_unlink(mutex2Name) != 0){**

**perror("shm\_unlink error\n");**

**return -1;**

**}**

**return 0;**

**}**

**child.c**

**#include <stdio.h>**

**#include <unistd.h>**

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

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

**#include <fcntl.h>**

**#include <pthread.h>**

**int main(int argc,char \*\*argv){**

**FILE \*file = fopen(argv[1],"a");**

**if (file == NULL){**

**perror("fopen error\n");**

**return -1;**

**}**

**//Open shared memory**

**//Create fd shared file**

**int mutex1 = shm\_open(argv[2], O\_RDWR | O\_CREAT, S\_IRWXU);**

**int mutex2 = shm\_open(argv[3], O\_RDWR | O\_CREAT, S\_IRWXU);**

**int pipe1 = shm\_open(argv[4],O\_RDWR | O\_CREAT, S\_IRWXU);**

**int pipe1Size = shm\_open(argv[5],O\_RDWR | O\_CREAT, S\_IRWXU);**

**if(pipe1 == -1 || pipe1Size == -1 || mutex1 == -1 || mutex2 ==-1){**

**printf("shm open\n");**

**return -1;**

**}**

**//Map shared files in memory**

**pthread\_mutex\_t\* mutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, mutex1, 0);**

**pthread\_mutex\_t\* secondMutex = (pthread\_mutex\_t \*) mmap(NULL, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, mutex2, 0);**

**float\* mmfData = (float \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1, 0);**

**int\* mmfDataSize = (int \*) mmap(NULL, getpagesize(), PROT\_READ | PROT\_WRITE, MAP\_SHARED, pipe1Size, 0);**

**if (mmfData == MAP\_FAILED || mmfDataSize == MAP\_FAILED || mutex == MAP\_FAILED || secondMutex == MAP\_FAILED){**

**printf("mmap error\n");**

**return -1;**

**}**

**while (\*mmfDataSize != -2){**

**if (pthread\_mutex\_lock(mutex) != 0){**

**perror("pthread\_mutex\_lock error\n");**

**return -1;**

**}**

**if (\*mmfDataSize == -2){**

**break;**

**}**

**float result = 0;**

**for (int i = 0; i < \*mmfDataSize; ++i){**

**result += mmfData[i];**

**}**

**fprintf(file, "%.3f\n", result);**

**\*mmfDataSize = -1;**

**if (pthread\_mutex\_unlock(secondMutex) != 0){**

**perror("pthread\_mutex\_unlock error\n");**

**return -1;**

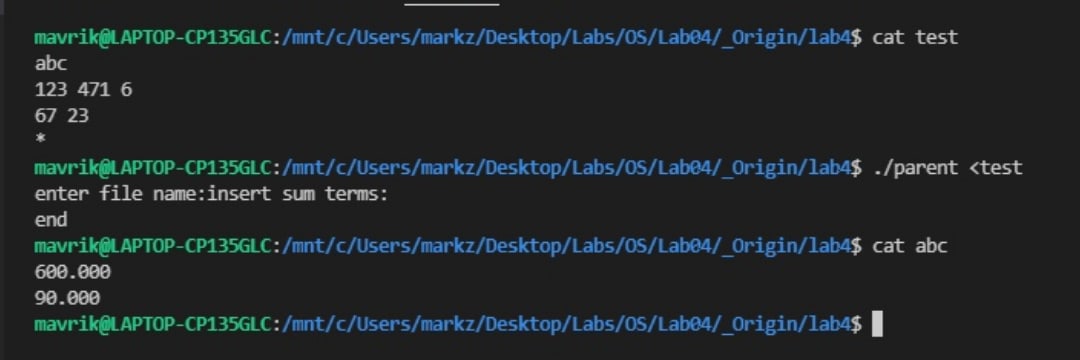
**}**

**}**

**fclose(file);**

**return 0;**

**}**

**Пример работы **

**Вывод**

В СИ помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например, отображение файла в память. Такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void\*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.