Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Тема работы**

**“Межпроцессорное взаимодействие через memory-mapped files”**

Студент: Молчанов Владислав Дмитриевич

Группа: М8О-208Б-20

Вариант: 8

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

<https://github.com/molch4nov/OS>

**Постановка задачи**

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 решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Общие сведения о программе**

Программа написана на языке С++ в UNIX-подобной операционной системе. Для компиляции требуется указать ключ –pthread и -lrt.

**Общий метод и алгоритм решения**

Программа на вход требует названия файла. Если такого файла не существует программа сразу завершается. Создаём два семафора, которые будут регулировать взаимодействие между дочерним и родительским процессором. Также создаем два файловых дескриптора, с помощью которых будет делать отображение на память вызовом mmap. Считываем построчно информацию из файла и передаем от родительского процессора через memptr1 дочернему. Он обрабатывает строку, полученную из memptr1 и результат кладёт в memptr2, который передаёт информацию из дочернего процесса в родительский. После завершения снимаем отображение файлов на память с помощью munmap и удаляем семафор функцией sem\_destroy.

**Исходный код**

**main.cpp**

**#include <fcntl.h>**

**#include <semaphore.h>**

**#include <stdbool.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

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

**#include <unistd.h>**

**#include "shrmem.h"**

**#define max\_filename\_size 10**

**int main() {**

**int fd = shm\_open(BackingFile, O\_CREAT | O\_RDWR, AccessPerms);**

**printf("Enter filename\n");**

**char \*filename = (char \*)malloc(sizeof(char) \* max\_filename\_size);**

**scanf("%s", filename);**

**int file = open(filename, O\_RDONLY);**

**if (fd == -1 || file == -1) {**

**perror("open");**

**exit(EXIT\_FAILURE);**

**}**

**sem\_t \*semptr = sem\_open(SemaphoreName, O\_CREAT, AccessPerms, 1);**

**if (semptr == SEM\_FAILED){**

**perror("sem\_open");**

**exit(EXIT\_FAILURE);**

**}**

**int val;**

**ftruncate(fd, map\_size);**

**char\* memptr = mmap(**

**NULL,**

**map\_size,**

**PROT\_READ | PROT\_WRITE,**

**MAP\_SHARED,**

**fd,**

**0);**

**if(memptr == MAP\_FAILED){**

**perror("mmap");**

**exit(EXIT\_FAILURE);**

**}**

**if(sem\_getvalue(semptr, &val) != 0){**

**perror("sem\_getvalue");**

**exit(EXIT\_FAILURE);**

**}**

**memset(memptr, '\0', map\_size);**

**while (val-- > 1) {**

**sem\_wait(semptr);**

**}**

**while (val++ < 0) {**

**sem\_post(semptr);**

**}**

**pid\_t pid = fork();**

**if(pid == 0){**

**if (dup2(file, fileno(stdin)) == -1) {**

**perror("DUP2");**

**exit(EXIT\_FAILURE);**

**}**

**execl("child", "child", NULL);**

**}**

**else if(pid == -1){**

**perror("fork");**

**exit(EXIT\_FAILURE);**

**}**

**while (true) {**

**if (sem\_getvalue(semptr, &val) != 0) {**

**perror("SEM\_GETVALUE");**

**exit(EXIT\_FAILURE);**

**}**

**if (val == 0) {**

**if (sem\_wait(semptr) != 0) {**

**perror("SEM\_WAIT");**

**exit(EXIT\_FAILURE);**

**}**

**printf("%s", memptr);**

**memset(memptr, '\0', map\_size);**

**if (sem\_post(semptr) != 0) {**

**perror("SEM\_POST");**

**exit(EXIT\_FAILURE);**

**}**

**} else {**

**if (sem\_wait(semptr) != 0) {**

**perror("SEM\_WAIT");**

**exit(EXIT\_FAILURE);**

**}**

**if (memptr[0] == EOF) {**

**break;**

**}**

**if (sem\_post(semptr) != 0) {**

**perror("SEM\_POST");**

**exit(EXIT\_FAILURE);**

**}**

**}**

**}**

**close(file);**

**free(filename);**

**if(munmap(memptr, map\_size) != 0){**

**perror("munmap");**

**exit(EXIT\_FAILURE);**

**}**

**close(fd);**

**if(sem\_close(semptr) != 0){**

**perror("sem\_close");**

**exit(EXIT\_FAILURE);**

**}**

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

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

**exit(EXIT\_FAILURE);**

**}**

**return EXIT\_SUCCESS;**

**}**

**Child.cpp**

#include <semaphore.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <unistd.h>

#include "shrmem.h"

#define NEG -1

#define PRIME 0

#define NORM 1

void print(char\* memptr, sem\_t \*semptr, const char \*empty\_string, int stat, int n) {

  while (true) {

    if ((sem\_wait(semptr)) == 0) {

      if (strcmp(memptr, empty\_string) != 0) {

        if (sem\_post(semptr) != 0) {

          perror("SEM\_POST");

          exit(EXIT\_FAILURE);

        }

        continue;

      }

      switch (stat) {

        case NEG:

          sprintf(memptr, "Negative number %d\n", n);

          break;

        case PRIME:

          sprintf(memptr, "Prime number %d\n", n);

          break;

        case NORM:

          sprintf(memptr, "%d\n", n);

          break;

        default:

          sprintf(memptr, "Programmist ne ochen\n");

      }

      if ((sem\_post(semptr)) != 0) {

        perror("SEM\_POST");

        exit(EXIT\_FAILURE);

      }

      break;

    } else {

      perror("SEM\_WAIT");

      exit(EXIT\_FAILURE);

    }

  }

}

int main() {

  int n;

  char c;

  int map\_fd = shm\_open(BackingFile, O\_RDWR, AccessPerms);

  if (map\_fd < 0) {

    perror("SHM\_OPEN");

    exit(EXIT\_FAILURE);

  }

  char\* memptr = mmap(

      NULL,

      map\_size,

      PROT\_READ | PROT\_WRITE,

      MAP\_SHARED,

      map\_fd,

      0);

  if (memptr == MAP\_FAILED) {

    perror("MMAP");

    exit(EXIT\_FAILURE);

  }

  sem\_t \*semptr = sem\_open(SemaphoreName, O\_CREAT, AccessPerms, 2);

  if (semptr == SEM\_FAILED) {

    perror("SEM\_OPEN");

    exit(EXIT\_FAILURE);

  }

  char \*empty\_string = (char \*)malloc(sizeof(char) \* map\_size);

  memset(empty\_string, '\0', map\_size);

  while (scanf("%d%c", &n, &c) != EOF) {

    if (n < 0) {

      print(memptr, semptr, empty\_string, NEG, n);

      break;

    }

    bool prost = true;

    for (int i = 2; i \* i <= n; ++i) {

      if (n % i == 0) {

        prost = false;

        print(memptr, semptr, empty\_string, NORM, n);

        break;

      }

    }

    if (prost) {

      print(memptr, semptr, empty\_string, PRIME, n);

      break;

    }

  }

  usleep(00500000);

  memptr[0] = EOF;

  free(empty\_string);

  close(map\_fd);

  sem\_close(semptr);

  return EXIT\_SUCCESS;

}

**Shrmem.h**

#ifndef SRC\_\_SHRMEM\_H\_

#define SRC\_\_SHRMEM\_H\_

#include <fcntl.h>

const int map\_size = 4096;

const char \*BackingFile = "lab\_4.back";

const char \*SemaphoreName = "lab4.semaphore";

unsigned AccessPerms = S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH;

#endif//SRC\_\_SHRMEM\_H\_

**Демонстрация работы программы**

vladislav@DESKTOP-OL36FK8:/mnt/c/Users/vlad-/Desktop/my\_lab$ ./a.out

Enter filename

file

4

Prime number 2

testtestfdadwa

**Выводы**

Эта лабораторная работа ознакомила и научила меня работать с расширяемой памятью. Научился синхронизировать работу процессов и потоков с помощью семафоров. В отличие от лабораторной работы №2, где мы вызывали read и write, взаимодействие между процессами через mmaped – files происходит эффективнее и требует меньше памяти.