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 C++ 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

10

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.