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: Ханнанов Руслан Маратович
Группа: М8О-208Б-20
Вариант: 14
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/Naksen/OS

Постановка задачи

Составить и отладить программу на языке C\C++, 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.
Сборка проекта происходит при помощи Cmake-файла.

Общий метод и алгоритм решения

Программа создает и отражает на общую для процессов память буферный файл. В этот буферный файл построчно считывается информация, его размер при это фиксированный и заранее установленный равным максимальной длине строки. В программе используется три семафора: первый – для общения между родительский и первым дочерним процессами, второй – для общения между первым и вторым дочерними процессами, третий – для общения между вторым дочерним и родительским процессом. Как только считывается новая строка, дочерни процесс понимает, что появились данные, которые нужно обработать, он переводит строку в нижний регистр и передает сигнал о том, что нужно строку нужно обработать второму

дочернему процессу, после чего второй дочерний процесс убирает задвоенные пробелы и сигнализирует с помощью третьего семафора о том, что строка полностью обработана. Родительский процесс понимает это и печатает обработанную строчку. Благодаря такому подходу в любой момент времени мы используем кусок памяти равный только лишь максимальной длине строки. После завершения снимает отображение файлов на память с помощью minmap и удаляем семафор функцией sem_destroy.

Исходный код

main.cpp:

```cpp
#include <iostream>
#include <unistd.h>
#include <inttypes.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fstream>
#include <sys/mman.h>
#include <string>
#include <semaphore.h>
#include <fcntl.h>
#include <wait.h>

int main() {

  int LINE_MAX_SIZE = 256;
  int val1;
  int val2;
  int val3;


  sem_t *semptr = sem_open( name: "CurSem", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 2);
  if (sem_getvalue(semptr, &val1) != 0) {
    perror( s: "SEM_GETVALUE");
    exit(EXIT_FAILURE);
  }
  while (val1++ < 2) {
    sem_post(semptr);
  }
  while (val1-- > 3){
    sem_wait(semptr);
  }

  sem_t *sem2 = sem_open( name: "Sem2", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 0);
  sem_getvalue(sem2,&val2);

  sem_t *sem3 = sem_open( name: "Sem3", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 0);

  sem_getvalue(semptr,&val1);
  sem_wait(semptr);

  sem_getvalue(semptr,&val1);

  int fd = shm_open( name: "BackingFile",  oflag: O_RDWR | O_CREAT,  mode: S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
  if (fd == -1) {
```

```
    perror( s: "OPEN");
    exit(EXIT_FAILURE);
}

ftruncate(fd, (off_t)LINE_MAX_SIZE);

char* memptr = (char*)mmap(
    addr: NULL,
    LINE_MAX_SIZE,
    prot: PROT_READ | PROT_WRITE,
    MAP_SHARED,
    fd,
    offset: 0);
if (memptr == MAP_FAILED) {
    perror( s: "MMAP");
    exit(EXIT_FAILURE);
}

int str_size = 0;
char *in = (char *)calloc( nmemb: 1, sizeof(char));
char c;


pid_t pid_0; //Идентификатор текущего процесса
if ((pid_0 = fork()) > 0) {

    while((c = getchar()) != EOF) {

        in[str_size] = c;
        in = (char *)realloc(in, size: (++str_size + 1) * sizeof(char));
        if (c == '\n') {
            in[str_size] = '\0';
            memset(memptr, c: '\0', LINE_MAX_SIZE);
            sprintf(memptr, format: "%s",in);
            sem_getvalue(semptr,&val1);
            sem_post(semptr);
            while(true) {
                sem_getvalue(sem3,&val3);
                if (val3 == 1) {
                    printf( format: "%s",memptr);
                    break;
                }
            }
            str_size = 0;
        }
```

```
    }
    sem_post(sem2);
    sem_post(sem2);
    sem_post(sem2);
    sem_post(semptr);
    sem_post(semptr);
    sem_post(semptr);

    close(fd);
    sem_destroy(semptr);
    sem_destroy(sem2);
    sem_destroy(sem3);
    sem_getvalue(semptr,&val1);
    munmap(memptr,LINE_MAX_SIZE);

  } else if (pid_0 == 0) {
    while(true) {
      if (sem_getvalue(semptr, &val1) != 0) {
        perror( s: "SEM_GETVALUE");
        exit(EXIT_FAILURE);
      }
      if (val1 == 2) {
        execl( path: "4_lab_child_1",  arg: "4_lab_child_1", NULL);
        perror( s: "EXECL");
        exit(EXIT_FAILURE);
      }
    }

  } else if (pid_0 < 0) {

    perror( s: "FORK");
    exit(EXIT_FAILURE);

  }
  return 0;
}
```

child1.cpp:

```cpp
#include <iostream>
#include <unistd.h>
#include <inttypes.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fstream>
#include <sys/mman.h>
#include <string>
#include <semaphore.h>
#include <fcntl.h>

int main(int argc, char **argv) {
  int val1;
  int val2;
  int LINE_MAX_SIZE = 256;

  int map_fd = shm_open( name: "BackingFile", O_RDWR,  mode: S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
  if (map_fd < 0) {
    perror( s: "SHM_OPEN");
    exit(EXIT_FAILURE);
  }

  char* memptr = (char*)mmap(
      addr: NULL,
      LINE_MAX_SIZE,
      prot: PROT_READ | PROT_WRITE,
      MAP_SHARED,
      map_fd,
      offset: 0);
  if (memptr == MAP_FAILED) {
    perror( s: "MMAP");
    exit(EXIT_FAILURE);
  }

  sem_t *semptr = sem_open( name: "CurSem", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 2);
  if (semptr == SEM_FAILED) {
    perror( s: "SEM_OPEN");
    exit(EXIT_FAILURE);
  }
  if (sem_wait(semptr) != 0) {
    perror( s: "SEM_WAIT");
    exit(EXIT_FAILURE);
  }
```

```
if (sem_getvalue(semptr, &val1) != 0) {
  perror( s: "SEM_GETVALUE");
  exit(EXIT_FAILURE);
}

while (val1++ < 3) {
  sem_post(semptr);
}
while (val1-- > 4){
  sem_wait(semptr);
}

sem_t *sem2 = sem_open( name: "Sem2", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 0);
sem_getvalue(sem2,&val2);
while (val2++ < 0) {
  sem_post(sem2);
}
while (val2-- > 1){
  sem_wait(sem2);
}

sem_getvalue(semptr,&val1);

pid_t pid_0; //Идентификатор текущего процесса
if ((pid_0 = fork()) > 0) {

  while(true) {
    if (sem_getvalue(semptr, &val1) != 0) {
      perror( s: "SEM_GETVALUE");
      exit(EXIT_FAILURE);
    }
    if (val1 == 4) {
      munmap(memptr, LINE_MAX_SIZE);
      close(map_fd);
      break;
    } else if (val1 == 3) {

      //cout << "New String" << endl;
      int i = 0;
      memptr[0] = tolower( c: memptr[0]);
      char c = memptr[0];
      while (c != '\0') {
```

```
            ++i;
            c = memptr[i];
            memptr[i] = tolower(memptr[i]);
        }
        ++i;
        //printf("%s",memptr);

        sem_post(sem2);
        sem_wait(semptr);
        //sem_post(sem2);
        sem_getvalue(sem2,&val2);
        //cout << "VAL2 = " << val2 << endl;
        sem_getvalue(semptr,&val1);
        //cout << "VALuE = " << val1 << endl;

    }
  }

} else if (pid_0 == 0) {
  while(true) {
    if (sem_getvalue(sem2, &val2) != 0) {
      perror( s: "SEM_GETVALUE");
      exit(EXIT_FAILURE);
    }
    if (val2 == 1) {
      //cout << "CHILD 2" << endl;
      //break;
      //cout << "CHILD in PARENT: Semaphore value = " << val1 << endl;
      //printf("CHILD BUF = %s",buf);
      //cout << "CHILD BUF = " << memptr << endl;
      //fflush(stdout);
      execl( path: "4_lab_child_2",  arg: "4_lab_child_2", NULL);
      //perror("EXECL");
      //exit(EXIT_FAILURE);
    }
    //sem_wait(semptr);
  }

} else if (pid_0 < 0) {

  perror( s: "FORK");
  exit(EXIT_FAILURE);

}
return EXIT_SUCCESS;
}
```

child2.cpp:

```cpp
#include <iostream>
#include <unistd.h>
#include <inttypes.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fstream>
#include <sys/mman.h>
#include <string>
#include <semaphore.h>
#include <fcntl.h>

int main(int argc, char **argv) {
  int val2;
  int val3;
  int LINE_MAX_SIZE = 256;

  sem_t *sem2 = sem_open( name: "Sem2", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 0);
  sem_getvalue(sem2,&val2);

  sem_t *sem3 = sem_open( name: "Sem3", O_CREAT, S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH, 0);
  sem_getvalue(sem3,&val3);
  while (val3++ < 0) {
    sem_post(sem3);
  }
  while (val3-- > 1){
    sem_wait(sem3);
  }
  sem_getvalue(sem3,&val3);

  int map_fd = shm_open( name: "BackingFile", O_RDWR,  mode: S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
  if (map_fd < 0) {
    perror( s: "SHM_OPEN");
    exit(EXIT_FAILURE);
  }

  char* memptr = (char*)mmap(
      addr: NULL,
      LINE_MAX_SIZE,
      prot: PROT_READ | PROT_WRITE,
      MAP_SHARED,
      map_fd,
      offset: 0);
  if (memptr == MAP_FAILED) {
    perror( s: "MMAP");
```

```
    exit(EXIT_FAILURE);
  }

  while(true) {
    if (sem_getvalue(sem2, &val2) != 0) {
      perror( s: "SEM_GETVALUE");
      exit(EXIT_FAILURE);
    }
    if (val2 == 2) {
      munmap(memptr, LINE_MAX_SIZE);
      close(map_fd);
      break;
    }
    if (val2 == 1) {

      sem_getvalue(sem3,&val3);
      char *out = (char *)calloc( nmemb: 1, sizeof(char));
      size_t m_size = 0;
      for (int i = 0; i + 1 < LINE_MAX_SIZE; ++i) {// преобразование
        if (memptr[i] == ' ' && memptr[i + 1] == ' ') {
          ++i;
          continue;//
        }
        out[m_size] = memptr[i];
        out = (char *)realloc(out,  size: (++m_size + 1) * sizeof(char));
      }
      out[m_size++] = '\0';

      memset(memptr,  c: '\0', LINE_MAX_SIZE);
      sprintf(memptr,  format: "%s", out);
      free(out);

      sem_post(sem3);

      sem_wait(sem3);
      sem_wait(sem2);
    }
  }

  return 0;
}
```

Демонстрация работы программы

```
HHHHH   hHHH asdsad DDDDD
AAA  AAnnBBd DD
^D
hhhhh hhhh asdsad ddddd
aaaaannbbd dd
```

Выводы

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