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**

**File mapping**

Студент: Велесов Даниил Игоревич

Группа: М8О-208Б-20

Вариант:

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/Kalambur4k/OS/tree/main/lab4

**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 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.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в semaphore A. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в semaphore B выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

*Вариант 15*

*Правило проверки: строка должна начинаться с заглавной буквы*

**Общие сведения о программе**

Программа состоит из двух файлов: master\_mm.c и check\_mm.c.

В master\_mm.c реализована основная работа программы – направление потоков ввода и вывода между процессами, работа родительского процесса.

check\_mm.c – код для программы дочернего процесса. В нем реализована проверка входящих строк на соответствие правилу варианта и работа семафоров.

**Общий метод и алгоритм решения**

master\_mm.c

Так как мы работаем на системе UNIX, то для создания дочернего процесса мы будем использовать утилиту fork(), предварительно создав семафоры для обмена данными между процессорами и открыв файл для записи. В дочернем процессе с помощью dup2 организуем ввод/вывод и используем утилиту exec для исполнения кода программы проверки на валидность правилу check\_mm.c.

При правильных строках запись происходит в файл, иначе, информация об ошибке пишется в SEMB. Также в дочернем процессе реализованы семафоры для контроля доступа к разделяемой памяти. Первым семафором (А) родитель дает сигнал на дочерний процесс, вторым семафором (Б) наоборот.

Так же здесь используется функция очистки ресурсов при нарушении работы программы void clean\_all().

**Исходный код**

***master\_mm.c***

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <string.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/mman.h>

#define MAXLEN 200

#define SEMA\_NAME "/semaphore\_for\_pipe1\_1d5LFo4"

#define SEMB\_NAME "/semaphore\_for\_pipe2\_1d5LFo4"

#define SHARED\_MEMORY\_OBJECT\_NAME "shared\_memory\_1d5LFo4"

#define SHARED\_MEMORY\_SIZE 2

sem\_t \*sema, \*semb;

int shm;

char \*addr;

FILE\* fp;

int sema\_ok = 0, semb\_ok = 0, shm\_ok = 0, mmap\_ok = 0, file\_ok = 0;

void clean\_all() {

if ( file\_ok ) {

fclose( fp );

}

if ( mmap\_ok ) {

munmap( addr, SHARED\_MEMORY\_SIZE );

}

if ( shm\_ok ) {

close( shm );

shm\_unlink(SHARED\_MEMORY\_OBJECT\_NAME);

}

if ( semb\_ok ) {

sem\_close(semb);

sem\_unlink(SEMB\_NAME);

}

if ( sema\_ok ) {

sem\_close(sema);

sem\_unlink(SEMA\_NAME);

}

}

int main(void)

{

sema = sem\_open(SEMA\_NAME, O\_CREAT, 0777, 0);

if ( sema == SEM\_FAILED ) {

perror("Cannot create semaphore A");

return EXIT\_FAILURE;

}

sema\_ok = 1;

semb = sem\_open(SEMB\_NAME, O\_CREAT, 0777, 0);

if ( semb == SEM\_FAILED ) {

perror("Cannot create semaphore B");

clean\_all();

return EXIT\_FAILURE;

}

semb\_ok = 1;

shm = shm\_open(SHARED\_MEMORY\_OBJECT\_NAME, O\_CREAT|O\_RDWR, 0777);

if ( shm < 0 ) {

perror("Cannot create shared memory object");

clean\_all();

return EXIT\_FAILURE;

}

shm\_ok = 1;

if ( ftruncate(shm, SHARED\_MEMORY\_SIZE) == -1 ) {

perror("Cannot set shared memory object size");

clean\_all();

return EXIT\_FAILURE;

}

addr = mmap(0, SHARED\_MEMORY\_SIZE, PROT\_WRITE|PROT\_READ, MAP\_SHARED, shm, 0);

if ( addr == (char\*)-1 ) {

perror("Cannot do mmap");

clean\_all();

return EXIT\_FAILURE;

}

mmap\_ok = 1;

char fname[MAXLEN];

//input filename

printf("Input file name: ");

scanf("%[^\n]s", fname);

getc(stdin); // убрать перевод строки

fp = fopen(fname, "w");

if (!fp) {

perror("Cannot create file");

clean\_all();

return EXIT\_FAILURE;

}

file\_ok = 1;

// FORK

int id = fork();

//error FORK

if (id == -1)

{

perror("Fork error");

clean\_all();

return EXIT\_FAILURE;

}

//CHILD WORK

else if (id == 0)

{

execlp("./check\_mm", NULL);

}

//PARENT WORK

else

{

char \*test = addr + 1;

char buffer[MAXLEN];

char \*buf = buffer;

int buf\_size = MAXLEN;

int len = getline(&buf,&buf\_size,stdin);

while ( len != EOF ) {

//printf( "Master: %zu len %s", len, buf );

\*test = buffer[0];

if ( sem\_post( sema ) < 0 ) {

perror("Cannot post semaphore a");

clean\_all();

return EXIT\_FAILURE;

}

if (sem\_wait(semb) < 0 ) {

perror("Cannot wait semaphore b");

clean\_all();

return EXIT\_FAILURE;

}

if ( \*addr ) {

printf( "Error: %s", buf );

} else {

fprintf( fp, buf );

}

len = getline(&buf,&buf\_size,stdin);

}

\*test = 0;

if ( sem\_post( sema ) < 0 ) {

perror("Cannot post semaphore a");

clean\_all();

return EXIT\_FAILURE;

}

wait(NULL);

clean\_all();

}

return 0;

}

***check\_mm.c***

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/mman.h>

#define SEMA\_NAME "/semaphore\_for\_pipe1\_1d5LFo4"

#define SEMB\_NAME "/semaphore\_for\_pipe2\_1d5LFo4"

#define SHARED\_MEMORY\_OBJECT\_NAME "shared\_memory\_1d5LFo4"

#define SHARED\_MEMORY\_SIZE 2

sem\_t \*sema, \*semb;

int shm;

char \*addr;

int sema\_ok = 0, semb\_ok = 0, shm\_ok = 0, mmap\_ok = 0, file\_ok = 0;

void clean\_all() {

if ( mmap\_ok ) {

munmap( addr, SHARED\_MEMORY\_SIZE );

}

if ( shm\_ok ) {

close( shm );

}

if ( semb\_ok ) {

sem\_close(semb);

}

if ( sema\_ok ) {

sem\_close(sema);

}

}

int main()

{

sem\_t \*sema, \*semb;

sema = sem\_open(SEMA\_NAME, 0);

if ( sema == SEM\_FAILED ) {

perror("Cannot open semaphore A");

return EXIT\_FAILURE;

}

sema\_ok = 1;

semb = sem\_open(SEMB\_NAME, 0);

if ( semb == SEM\_FAILED ) {

perror("Cannot open semaphore B");

clean\_all();

return EXIT\_FAILURE;

}

semb\_ok = 1;

int shm = shm\_open(SHARED\_MEMORY\_OBJECT\_NAME, O\_RDWR, 0777);

if ( shm < 0 ) {

perror("Cannot open shared memory object");

clean\_all();

return EXIT\_FAILURE;

}

shm\_ok = 1;

char \*addr = mmap(0, SHARED\_MEMORY\_SIZE, PROT\_WRITE|PROT\_READ, MAP\_SHARED, shm, 0);

if ( addr == (char\*)-1 ) {

perror("Cannot do mmap");

clean\_all();

return EXIT\_FAILURE;

}

mmap\_ok = 1;

char \*test = addr + 1;

if ( sem\_wait(sema) < 0 ) {

perror("Cannot wait semaphore a");

clean\_all();

return EXIT\_FAILURE;

}

while ( \*test ) {

if ( isupper( \*test ) ) {

\*addr = 0;

} else {

\*addr = 1;

}

if ( sem\_post( semb ) < 0 ) {

perror("Cannot post semaphore b");

clean\_all();

return EXIT\_FAILURE;

}

if (sem\_wait(sema) < 0 ) {

perror("Cannot wait semaphore a");

clean\_all();

return EXIT\_FAILURE;

}

}

clean\_all();

}

**Демонстрация работы программы**

user@user-Inspiron-3584:~/Рабочий стол/lab4os$ ./master

Input file name: Output

Hello, how are you?

im fine, thanks. What about you?

Error: im fine, thanks. What about you?

iM fInE toO ThX!

Error: iM fInE toO ThX!

OK, Bye!

-Bye!

Error: -Bye!

user@user-Inspiron-3584:~/Рабочий стол/lab2os$ cat Output

Hello, how are you?

OK, Bye!

**Выводы**

Благодаря этой работе я научился работать с процессами с помощью File Mapping. А также применил знания на практике.