Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №4 по курсу**

**«Операционные системы»**

**Работа между процессами посредством memory-mapped file**

Студент: Базаргармаев Нима Дондокович

Группа: М8О –301Б-18

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Тестирование
6. Демонстрация работы программы
7. Вывод

**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.

**Задание согласно 3 варианту:**

Рекурсивное вычисление суммы последовательности от 1 до n, где каждый отдельный уровень рекурсии вычисляется в отдельном процессе.

**Общие сведения о программе**

В 4.c используются заголовочные файлы stdio.h, stdlib.h, unistd.h, sys/types.h, sys/stat.h, fcntl.h, sys/wait.h, sys/mman.h, string.h. В программе используются следующие системные вызовы:

1. **open()** - Процедура открытия файла. При открытии файла операционная система проверяет, соответствуют ли права, которые запросил процесс для операций над файлом, правам доступа, установленным для этого файла. В случае соответствия она помещает необходимую информацию в системную таблицу файлов и, если этот файл не был ранее открыт другим процессом, в таблицу индексных дескрипторов открытых файлов. Далее операционная система находит пустой элемент в таблице открытых файлов процесса, устанавливает необходимую связь между всеми тремя таблицами и возвращает на пользовательский уровень дескриптор этого файла.
2. **off\_t lseek(int filedes, off\_t offset, int whence) –** установить смещение для файлового дескриптора filedes в значение аргумента offset в соответствии с директивой whence, которая может принимать одно из следующих значений: **SEEK\_SET (**смещение устанавливается в*offset* байт (от начала файла)),  **SEEK\_CUR (**смещение устанавливается как текущее смещение плюс*offset* байт), **SEEK\_END** (смещение устанавливается как размер файла плюс*offset* байт). При успешном выполнении lseek возвращает получившееся в результате смещение в байтах от начала файла. В противном случае, возвращается значение (off\_t)-1 и errno показывает ошибку**.**
3. **close(int fd)** - Функция закрывает файловый дескриптор, который после этого не ссылается ни на один файл и может быть использован повторно. При успешном завершении возвращает 0, в случае ошибки -1.
4. **void \* mmap(void \****start***, size\_t***length***, int***prot* **, int***flags***, int***fd***, off\_t***offset***)** - Системный вызов *mmap* служит для отображения предварительно открытого файла в адресное пространство вычислительной системы. Параметр start указывает начало области адресного пространства, в которую будет отображен файл, length отвечает за длину отображения файла в байтах, prot указывает возможные операции с файлом(операции не могу быть шире, чем они указаны при открытии). flags определяет способ отображения файла в адресное пространство.(MAX\_SHARED - полученное отображение используется другими процессами, вызвавшими mmap для этого файла, MAX\_PRIVATE - отображение файла в монопольном распоряжении процесса), fd файловый дескриптор для файла, offset - смещение отображения в память от начала файла в байтах. При успешном завершении возвращает начальный адрес в области памяти, в которую отображен файл, иначе MAP\_FAILED.
5. **int munmap (void \*start, size\_t length)** - Системный вызов *munmap* служит для прекращения отображения memory mapped файла в адресное пространство вычислительной системы. Параметр start является адресом начала области памяти, выделенной для отображения файла(то что вернул mmap), length - определяет ее длину, и его значение должно совпадать со значением соответствующего параметра в системном вызове *mmap()*. При нормально завершении возвращает 0, иначе -1;
6. **write** – предназначен для осуществления потоковых операций ввода информации над каналами связи, описываемыми файловыми дескрипторами, т.е. для pipe, файлов и для потокового ввода.
7. **int msync(void\* start, size\_t length, int flags) –** синхронизирует содержимое файла с его отражением в памяти. msync записывает на диск изменения, внесенные в файл, отраженные в память при помощи функции mmap.
8. **int fstat(int filedes, struct stat\* buf) –** возвращает информацию об указанном файле.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Записываю информацию в отображаемый файл.
2. Создаю дочерний процесс.
3. В дочернем процессе рекурсивно вызываю функцию подсчета.
4. Родительский процесс считывает информацию из отображенного файла и возвращает результат.

**Листинг программы.**

**Файл 4.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <string.h>

int radix\_count(int n) {

if (n / 10 == 0) {

return 1;

} else {

int count = 0;

while (n > 0) {

n = n / 10;

count += 1;

}

return count;

}

}

void reverse(char s[]) {

int i, j;

char c;

for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {

c = s[i];

s[i] = s[j];

s[j] = c;

}

}

void itoa(int n, char s[]) {

int i, sign;

if ((sign = n) < 0) {

n = -n;

}

i = 0;

do {

s[i++] = n % 10 + '0';

} while ((n /= 10) > 0);

if (sign < 0) {

s[i++] = '-';

}

s[i] = '\0';

reverse(s);

}

void Write(char\* path, int n) {

char str[radix\_count(n)];

itoa(n ,str);

int fd = open(path, O\_RDWR | O\_CREAT | O\_TRUNC, (mode\_t)0600);

if (fd == -1) {

perror("Error opening the file for writing");

exit(EXIT\_FAILURE);

}

size\_t str\_size = strlen(str);

if (lseek(fd, str\_size, SEEK\_SET) == -1) {

close(fd);

perror("Error while calling lseek()");

exit(EXIT\_FAILURE);

}

if (write(fd, "", 1) == -1) {

close(fd);

perror("Error writing last byte of the file");

exit(EXIT\_FAILURE);

}

char\* map = mmap(NULL, str\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (map == MAP\_FAILED) {

close(fd);

perror("Error while mapping the file (WRITE)");

exit(EXIT\_FAILURE);

}

for (size\_t i = 0; i < str\_size; ++i) {

map[i] = str[i];

}

if (msync(map, str\_size, MS\_SYNC) == -1) {

close(fd);

perror("Error while unmapping the file (WRITE)");

exit(EXIT\_FAILURE);

}

close(fd);

}

int Read(char\* path) {

int fd = open(path, O\_RDONLY, 0600);

if (fd == -1) {

perror("Error opening the file for reading");

exit(EXIT\_FAILURE);

}

struct stat info = {0};

if (fstat(fd, &info) == -1) {

perror("Error getting the file size");

exit(EXIT\_FAILURE);

}

if (info.st\_size == 0) {

perror("The file is empty");

exit(EXIT\_FAILURE);

}

char\* map = mmap(NULL, info.st\_size, PROT\_READ, MAP\_SHARED, fd, 0);

if (map == MAP\_FAILED) {

close(fd);

perror("Error while mapping the file (READ)");

exit(EXIT\_FAILURE);

}

int result = atoi(map);

if (munmap(map, info.st\_size) == -1) {

close(fd);

perror("Error while unmapping the file (READ)");

exit(EXIT\_FAILURE);

}

close(fd);

return result;

}

int sum(int n) {

if (n > 0) {

int result = 0, tmp = n - 1;

Write("in", tmp);

pid\_t p = fork();

if (p < 0) {

printf("Can't create fork\n");

exit(-1);

} else if (p > 0) {

wait(NULL);

result = Read("out");

return result + n;

} else {

int num = Read("in");

num = sum(num);

Write("out", num);

exit(0);

}

} else {

return n;

}

}

int main() {

int n = 0;

printf("Enter the number to calculate: \n");

scanf("%d", &n);

if (n > 0) {

n = sum(n);

} else {

printf("Number must be greater than zero\n");

return 0;

}

printf("Result: %d\n", n);

return 0;

}

**Тестирование.**

См. следующий пункт.

**Демонстрация работы программы.**

user@DESKTOP-4RFVSK2:~/os/4lab$ ./a.out

Enter the number to calculate:

0

Number must be greater than zero

user@DESKTOP-4RFVSK2:~/os/4lab$ ./a.out

Enter the number to calculate:

1

Result: 1

user@DESKTOP-4RFVSK2:~/os/4lab$ ./a.out

Enter the number to calculate:

2

Result: 3

user@DESKTOP-4RFVSK2:~/os/4lab$ ./a.out

Enter the number to calculate:

15

Result: 120

user@DESKTOP-4RFVSK2:~/os/4lab$ ./a.out

Enter the number to calculate:

8

Result: 36

**Вывод.**

Кажется, что отображение файла на дисковое пространство не имеет каких-либо преимуществ, но оно явно может ускорить перенос файла в память. Когда мы обращаемся к памяти, в которую отображен файл, данные загружаются с диска в кэш(если их там ещё нет), затем делается отображение кэша в адресное пространство нашей программы. Если эти данные удаляются — отображение отменяется. Таким образом, мы избавляемся от операции копирования из кэша в буфер. Кроме того, нам не нужно париться по поводу оптимизации работы с диском — всю грязную работу берёт на себя ядро ОС.

Помимо этого, если компьютер не обладает достаточным количеством оперативной памяти, то c помощью mmap() реализуется виртуальная память, которая виртуально расширяет объем оперативной памяти, и, тем самым обеспечивая более эффективный процесс решения разного рода задач, как для пользователей, так и для программистов.