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

Студент: Савинова Екатерина Ильинична
Группа: М8О-207Б-21
Вариант: 15
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/savinova-kati/operating-systems/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.

Группа вариантов 4:

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. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 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 представлена файлом lab.cpp.

Общий метод и алгоритм решения

Опишу новые для себя системные вызовы:

-sem_open

Функция sem_open() создаёт новый семафор POSIX или открывает существующий семафор.

-sem_wait

Функция sem_wait() уменьшает (блокирует) семафор, на который указывает sem. Если значение семафор больше нуля, то выполняется уменьшение и функция сразу завершается. Если значение семафора равно нулю, то вызов блокируется до тех пор, пока не станет возможным выполнить уменьшение

-sem_post

Функция sem_post() увеличивает (разблокирует) семафор, на который указывает sem. Если значение семафора после этого становится больше нуля, то другой процесс или нить заблокированная в вызове sem_wait, проснётся и заблокирует семафор.

Алгоритм решения:

В родительском процессе принимаем из ввода строчку пользователя, затем открываем объект общей памяти, устанавливаем ему размер текста и отображаем на него текст.

Далее создаем семафор, вызываем sem_post и переходим в дочерний процесс для проверки верности правилу строчки.

Исходный код

lab.cpp

```
#include <iostream>
#include <string>
#include <sys/types.h>
```

```
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <unistd.h>
#include <fstream>
#include <errno.h>
#include <sys/mman.h>
#include <cstdio>

using namespace std;
int flag_ = 0;

int child(string filename, char *mapped, string sem_file) {
	int count = 1;
	fstream file_1;
	file_1.open(filename, fstream::in | fstream::out | fstream::app);
	sem_t *semaphore = sem_open(sem_file.c_str(), 1);

	while (true) {

		if (sem_wait(semaphore) == -1) {
			perror("Semaphore error");
			exit(EXIT_FAILURE);
		}

		if (mapped[count] == '!') {
			break;
		}
```

```
            int str_size = (int)mapped[count];

            int start = count;
            char mas[str_size];
            int i = 0;

            for(; count < start + str_size; count++) {
                mas[i] = mapped[count + 1];
                i += 1;
            }

            string result;

            if (mas[0] >= 65 && mas[0] <= 90) {
                for(int i = 0; i < str_size; i++) {
                    result.push_back(mas[i]);
                    file_1 << mas[i];
                }
                file_1 << endl;
                cout << "Added string " << result << " to file!" << endl;
            } else {
                mapped[0] = 1;
            }
            sem_post(semaphore);

            count++;

    }
        return 0;
```

```
}

int main ()
{
	string filename;
	int flag;
	int strings_size;
	string sem_file = "a.semaphore";

	cout << "Enter name of file ";
	cin >> filename;
	cout << endl;

	cout<<"Enter amount of strings: ";
	int amount;
	cin >> amount;
	cout << endl;

	const int mapsize = amount*256;

	int flaccess = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH; //права семафора

	sem_t *semaphore = sem_open(sem_file.c_str(), O_CREAT, flaccess, 0);

	if (semaphore == SEM_FAILED) {
		perror("Semaphore error");
		exit(EXIT_FAILURE);
	}
```

```
	char *mapped = (char *)mmap(0, mapsize, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);


	pid_t id = fork();


	if (id == -1){


		perror("fork");

		cout << "1";

		exit(EXIT_FAILURE);

	}


	else if (id == 0) {


		child(filename, mapped, sem_file);

		return 0;

		//execl("./child_", to_string(truba[0]).c_str(), to_string(truba[1]).c_str(), to_string(truba_2[0]).c_str(), to_string(truba_2[1]).c_str(), name.c_str(), NULL);


	}


	if (id != 0) {

		string string_r;

		int start = 1;

		mapped[0] = 0;

		for (int i = 0; i < amount + 1; ++i) {

			if (i == amount) {

				mapped[start] = '!';

				if (mapped[0] == 1) {

					cout << "The string does not fit the rule" << endl;
```

```
                        mapped[0] = 0;
                    }
                    sem_post(semaphore);
                    break;
                }
                cin >> string_r;
                for (int j = 0; j < string_r.size() + 1; j++){
                    if (j == 0) {
                        mapped[start] = (char)string_r.size();
                        continue;
                    }
                    mapped[start + j] = string_r[j - 1];
                }
                sem_post(semaphore); //разблакировка семафора
                sem_wait(semaphore);
                if (mapped[0] == 1) {
                    cout << "The string does not fit the rule" << endl;
                    mapped[0] = 0;
                }
                start += string_r.size() + 1;
            }
        }
        munmap(mapped, mapsize);
        sem_close(semaphore);
        sem_unlink(sem_file.c_str());

        return 0;
}
```

Демонстрация работы программы

Ввод в консоль:

```
MacBook-Air-Ekaterina:src ekaterina$ ./main
Enter name of file 22.txt

Enter amount of strings: 5

Fgjsc
Added string Fgjsc to file!
ojhjklo
The string does not fit the rule
IKJHjki
The string does not fit the rule
OIJHG
The string does not fit the rule
aswdefg
The string does not fit the rule
```

Выводы

Благодаря данной лабораторной работе, я получила больше информации о работе с отображаемой памятью и семафорами.