|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
|  | | |
| Лабораторная работа №5 | | |
| по дисциплине «Управление ресурсами в вычислительных системах». | | |
| «МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ ПРОГРАММ» | | |
|  | | |
|  | | |
|  | Факультет: | ФПМИ |
| Группа: | ПМИ-12 |
| Студент: | Курдюков Иван, Омельницкая Екатерина |
| Преподаватели: | Стасышин В.М.,  Сивак М.А. |
|  |  |
|  |  |
|  |  |
|  |  |
| Новосибирск, 2024 | | |

1. **Цель работы**

Освоение средств IPC. Написание программ, использующих механизм семафоров, очередей сообщений, сегментов разделяемой памяти.

1. **Задание**

Программа моделирует работу монитора обработки сообщений. Порожденные процессы, обладающие различными приоритетами и выполняющие некоторые циклы работ, посредством очереди сообщений передают родительскому процессу имена программ из предыдущих лабораторных работ, которые им должны быть запущены. Родительский процесс, обрабатывая сообщения в соответствии с их приоритетами, следит, чтобы одновременно было запущено не более трех программ.

1. **Описание используемых структур**

Структура message

Эта структура используется для обмена сообщениями между процессами через очередь сообщений.

*struct message {*

*long mtype;*

*char mtext[MSGSZ];*

*};*

**Функции работы с очередями сообщений**

*mqd\_t mq\_open(const char*[*\*name*](https://vk.com/name)*, int oflag, ...)* - Открывает или создает очередь сообщений.  
  
Параметры:  
  
name: Имя очереди сообщений. Должно начинаться с символа /.  
oflag: Флаги управления (например, O\_CREAT для создания очереди).  
mode (необязательный): Права доступа к создаваемой очереди (например, 0644).  
attr (необязательный): Указатель на структуру mq\_attr, определяющую атрибуты очереди (максимальное количество сообщений, максимальный размер сообщения и т.д.).  
Возвращаемое значение:  
  
Дескриптор очереди сообщений (mqd\_t) при успехе. -1 при ошибке и установка errno.

*int mq\_send(mqd\_t mqdes, const char \*msg\_ptr, size\_t msg\_len, unsigned int msg\_prio)*  
Отправляет сообщение в очередь.  
  
Параметры:  
  
mqdes: Дескриптор очереди сообщений.  
msg\_ptr: Указатель на сообщение.  
msg\_len: Длина сообщения.  
msg\_prio: Приоритет сообщения.  
Возвращаемое значение:  
  
0 при успехе.  
-1 при ошибке и установка errno.

*ssize\_t mq\_receive(mqd\_t mqdes, char \*msg\_ptr, size\_t msg\_len, unsigned int \*msg\_prio)*  
  
Получает сообщение из очереди.  
  
Параметры:  
  
mqdes: Дескриптор очереди сообщений.  
msg\_ptr: Указатель на буфер для получения сообщения.  
msg\_len: Размер буфера.  
msg\_prio: Указатель для получения приоритета сообщения (может быть NULL).  
Возвращаемое значение:  
  
Длина полученного сообщения при успехе.  
-1 при ошибке и установка errno.

*int mq\_close(mqd\_t mqdes) -* Закрывает очередь сообщений.  
  
Параметры:  
  
mqdes: Дескриптор очереди сообщений.  
Возвращаемое значение:  
  
0 при успехе.  
-1 при ошибке и установка errno.

*int mq\_unlink(const char*[*\*name*](https://vk.com/name)*)*  
  
Удаляет очередь сообщений.  
  
Параметры:  
  
name: Имя очереди сообщений.  
Возвращаемое значение:  
  
0 при успехе.  
-1 при ошибке и установка errno.

1. **Спецификация**

Программа разработана и протестирована на компьютере с Cygwin. Cygwin — UNIX-подобная среда и интерфейс командной строки для Microsoft Windows. В качестве компилятора используется GCC версии 11.2.0.

Директория, содержащая файл: C:\cygwin64\home\Ivan

Название файла с программой: lab5\_10.c

Компиляция программы: gcc -o <результат компиляции> lab5\_10.c

Запуск программы: ./<результат компиляции>

1. **Описание алгоритма на языке Си**

1. Создание очереди сообщений:  
  
В начале программы создается очередь сообщений с помощью функции mq\_open.

2. Создание дочерних процессов:  
  
Процесс main() создает несколько дочерних процессов с помощью функции fork().  
Каждый дочерний процесс отправляет сообщение в очередь сообщений с помощью функции mq\_send.

3. Создание процесса-родителя:  
  
Процесс main() создает несколько процессов-родителей с помощью функции fork().  
Каждый процесс-родитель получает сообщения из очереди с помощью функции mq\_receive, обрабатывает их и использует семафор для синхронизации.

4. Взаимодействие через очередь сообщений:  
  
Процесс-родитель использует семафор для ограничения одновременного выполнения заданий.  
Каждый дочерний процесс отправляет сообщение в очередь сообщений.  
Процесс-родитель получает сообщение, обрабатывает его и использует семафор для синхронизации выполнения.

5. Завершение работы и очистка ресурсов:  
  
После завершения работы процессы закрывают очередь сообщений с помощью функции mq\_close.  
Удаляют очередь сообщений с помощью функции mq\_unlink.  
Освобождают разделяемую память и удаляют семафор.

**Код программы на языке Си**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <errno.h>

#include <mqueue.h>

#include <pthread.h>

#include <semaphore.h>

#include <sys/wait.h>

#define MAX\_PROGRAMS 3

#define NUM\_PROCESSES 5

#define MSGSZ 128

#define QUEUE\_NAME "/msg\_queue"

struct message {

long mtype;

char mtext[MSGSZ];

};

void child\_process(mqd\_t mqd, int id) {

    struct message msg;

    msg.mtype = id;

    // Генерация имени программы

    snprintf(msg.mtext, MSGSZ, "Lab%d", id);

    // Отправка сообщения в очередь

    if (mq\_send(mqd, (char\*)&msg, sizeof(msg),NUM\_PROCESSES + 1 - id) == -1) { // здесь NUM\_PROCESSES+1-id это приоритет сообщения

    perror("mq\_send");

    exit(1);

    }

    printf("Child %d sent message: %s with priority %d\n", getpid(), msg.mtext, NUM\_PROCESSES + 1 - id);

    exit(0);

}

void parent\_process(mqd\_t mqd, int shmid) {

    sem\_t\* semaphore = shmat(shmid, NULL, 0);

    if (semaphore == (sem\_t\*)-1){

        perror("shmat");

        exit(1);

    }

    struct message msg;

    unsigned int prio;

    int curlen = NUM\_PROCESSES;

    while (curlen > 0) {

        if (mq\_receive(mqd, (char\*)&msg, sizeof(msg), &prio) == -1) {

            perror("mq\_receive");

            exit(1);

        }

        // Уменьшение счетчика семафора

        sem\_wait(semaphore);

        // Симуляция выполнения программы

        printf("executing a program: %s with priority %d\n", msg.mtext, prio);

        //sleep(NUM\_PROCESSES + 1 - prio); // Симуляция работы программы в зависимости от приоритета

        sleep(1);

        printf("Finished program: %s with priority %d\n", msg.mtext, prio);

        sem\_post(semaphore);// Увеличение счетчика семафора

        curlen --;

        // отсоединение разделяемой обл. памяти

        if (shmdt(semaphore)== -1){

            perror("shmdt");

            exit(1);

        }

        exit(0);

    }

}

int main() {

    key\_t key = 1234;

    // создание разделяемой памяти

    int shmid = shmget(key, sizeof(sem\_t), IPC\_CREAT | 0666);

    if (shmid<0){

        perror("shmget");

        exit(1);

    }

    // присоединение разделяемой памяти

    sem\_t\* semaphore = shmat(shmid, NULL, 0);

    if (semaphore == (sem\_t\*)-1){

        perror("shmat");

        exit(1);

    }

    // Инициализация семафора

    if (sem\_init(semaphore, 1, MAX\_PROGRAMS) == -1) {

        perror("sem\_init");

        exit(1);

    }

    // Создание очереди сообщений

    struct mq\_attr attr;

    attr.mq\_flags = 0;

    attr.mq\_maxmsg = 10;

    attr.mq\_msgsize = sizeof(struct message);

    attr.mq\_curmsgs = 0;

    mqd\_t mqd = mq\_open(QUEUE\_NAME, O\_CREAT | O\_RDWR, 0666, &attr);

    if (mqd == (mqd\_t)-1) {

        perror("mq\_open");

        exit(1);

    }

    for (int i = 0; i < NUM\_PROCESSES; i++) {

        if (fork() == 0) {

        child\_process(mqd, i + 1);

        }

    }

    for (int i = 0; i < NUM\_PROCESSES; i++){ wait(NULL); }

    for (int i = 0; i < NUM\_PROCESSES; i++) {

        if (fork() == 0) {

        parent\_process(mqd, shmid);

        }

    }

    for (int i = 0; i < NUM\_PROCESSES; i++){ wait(NULL); }

    // Удаление семафора и очереди сообщений

    sem\_destroy(semaphore);

    mq\_close(mqd);

    mq\_unlink(QUEUE\_NAME);

    // отсоединение разделяемой обл. памяти

    if (shmdt(semaphore)== -1){

        perror("shmdt");

        exit(1);

    }

    //удаление разделяемой памяти

    if(shmctl(shmid,IPC\_RMID, NULL)==-1){

        perror("shmctl");

        exit(1);

    }

    return 0;

}

1. **Тестирование**

