МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | **Факультет прикладной математики и информатики**

Кафедра теоретической и прикладной информатики

Лабораторная работа №5
по дисциплине «Управление ресурсами в вычислительных системах»

Межпроцессное взаимодействие программ

Бригада 09 БЕГИЧЕВ АЛЕКСАНДР

Группа ПМ-92 ШИШКИН НИКИТА

Вариант 09

Преподаватели СИВАК МАРИЯ АЛЕКСЕЕВНА

СТАСЫШИН ВЛАДИМИР МИХАЙЛОВИЧ

Новосибирск, 2022

Цель работы

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

Вариант 9

Родительский процесс помещает в сегмент разделяемой памяти имена программ из предыдущих лабораторных работ, которые могут быть запущены. Выполняя некоторые циклы работ, порожденные процессы случайным образом выбирают имена программ из таблицы сегмента разделяемой памяти, запускают эти программы, и продолжают свою работу. Посредством аппарата семафоров должно быть обеспечено, чтобы не были одновременно запущены две одинаковые программы. В процессе работы через очередь сообщений родительский процесс информируется, какие программы и от имени кого запущены.

Готовая программа

Algorithm 1 Родительский процесс

```
1:  SUBPROC_CNT ← 10                                  ▷ Количество подпроцессов
2:  FILES_CNT ← 4                                     ▷ Количество файлов
3:
4:  процедура Родительский процесс
5:      создать IPC ключ key
6:
7:      создать сегмент разделяемой памяти data с помощью key
8:
9:      data ← "olds/p1 olds/p2 olds/p3 olds/p4"      ▷ 4 файла
10:     заменить  в data на NULL-разделитель
11:
12:     создать семафоры с помощю key в колчестве FILES_CNT
13:     создать подпроцессы в колчестве SUBPROC_CNT
14:
15:     подключиться к очереди сообщений с помощью key
16:
17:     пока не получено SUBPROC_CNT сообщний делать
18:         получить сообщение msg
19:         вывести msg
20:
21:     уничтожить семафоры, очередь сообщений и разделяемую память
```

Algorithm 2 Дочерний процесс

```
SUBPROC_CNT ← 10                                  ▷ Количество подпроцессов
FILES_CNT ← 4                                     ▷ Количество файлов

процедура Дочерний процесс
    создать IPC ключ key

    получить сегмент разделяемой памяти data с помощью key

    подключиться к очереди сообщений с помощью key

    получить случайное имя файла filename из data
    получить соответствующий filename номер семафора fid

    получить семафоры с помощью key в колчестве FILES_CNT
    захватить семафор fid
    выполнить программу с именем filename
    освободить семафор

    отправить информационное сообщение по очереди сообщений
```

Пример вывода программы

```
Hello, process 7093!
Hello, process 7095!
Got message: olds/p1 executed by 7093
Got message: olds/p4 executed by 7095
Hello, process 7105!
Got message: olds/p3 executed by 7105
Hello, process 7094!
Got message: olds/p2 executed by 7094
Hello, process 7102!
Got message: olds/p1 executed by 7102
Hello, process 7099!
Got message: olds/p4 executed by 7099
Hello, process 7098!
Got message: olds/p2 executed by 7098
Hello, process 7101!
Got message: olds/p4 executed by 7101
Hello, process 7103!
Got message: olds/p4 executed by 7103
Hello, process 7104!
Got message: olds/p4 executed by 7104
```

Вывод

Мы научились пользоваться средстваи IPC для коммуникации между процессами, а именно: общаться путём очереди сообщений, создавать сегменты разделяемой памяти (самый быстрый способ общения между процессами) и управлять ходом выполнения программы и контролировать использование ресурсов с помощью семафоров.

Исходный код

Структура проекта

```
code ..... Корневая папка
├── main.c ..... Код родительского процесса
├── main.h ..... Его заголовочный файл
├── subprocess.c ..... Код дочерних процессов
├── subprocess.h ..... Его заголовочный файл
├── msg.h ..... Файл со структурами и константами для main и subprocess
└── olds ..... Папка с программами для запуска доч. процссами
    ├── p.c ..... Исходный код программ для запуска доч. процссами
    ├── p1 ..... Первая программа для запуска доч. процссами
    ├── p2 ..... Вторая программа для запуска доч. процссами
    ├── p3 ..... Третья программа для запуска доч. процссами
    └── p4 ..... Четвёраяй программа для запуска доч. процссами
```

main.c

```c
#include "main.h"
#include "subprocess.h"
#include "msg.h"

// * Вариант 9
// Родительский процесс помещает в сегмент разделяемой памяти имена
// программ из предыдущих лабораторных работ, которые могут быть запущены.
// Выполняя некоторые циклы работ, порожденные процессы случайным образом
// выбирают имена программ из таблицы сегмента разделяемой памяти, запускают эти
// программы, и продолжают свою работу. Посредством аппарата семафоров должно
// быть обеспечено, чтобы не были одновременно запущены две одинаковые
// программы. В процессе работы через очередь сообщений родительский процесс
// информируется, какие программы и от имени кого запущены.
// P.S. Вместо программ из предыдущих лабораторных работ, мы сделали программу,
↪  которая в качестве аргумента принимает строку, которую затем в консоль как чьё-то
↪  имя, а затем ждёт случайное количество времени.


// Функция, нужная только для того чтобы
// засунуть в разделяемую память нужную информацию.
// Код вынесен сюда, чтобы не мешаться под ногами.
void put_data(char *data) {
    char *str, *token, *context;

    data[0] = '\0';
    strcat(data, "olds/p1 olds/p2 olds/p3 olds/p4");

    token = strtok_r(data, " ", &context);

    for (int i = 0; i < FILES_CNT - 1; i++) {
        token = strtok_r(NULL, " ", &context);
    }
}

// Создаём семафоры
int init_sems(key_t key, int nsems) {
    union semun arg;
    int semid;
```

```
37
38      semid = semget(key, nsems, IPC_CREAT | 0666);
39      arg.val = 1;
40
41      for(int isem = 0; isem < nsems; isem++) {
42          if (semctl(semid, isem, SETVAL, arg) == -1) {
43              semctl(semid, 0, IPC_RMID); // Отщистка
44              return -1; // Ошибка
45          }
46      }
47
48      return semid;
49  }
50
51  int main(int argc, char *argv[]) {
52      pid_t pids[SUBPROC_CNT];
53      key_t key;
54
55      int msgid, shmid, semid;
56      msgbuf msg;
57
58      char *data; // Разделяемая память
59
60      // Получаем уникальный ключ
61      if ((key = ftok("main.c", 'F')) == -1) {
62          perror("ftok");
63          exit(EXIT_FAILURE);
64      }
65
66      // Создаём сегмент разделяемой памяти
67      if ((shmid = shmget(key, 1024, 0666 | IPC_CREAT)) == -1) {
68          perror("shmget");
69          exit(EXIT_FAILURE);
70      }
71
72      // Получаем ссылку на сегмент памяти и
73      // запихиваем в него нужную информацю
74      data = shmat(shmid, NULL, 0);
75      put_data(data);
76
77      // Создаём семафоры
78      if ((semid = init_sems(key, FILES_CNT)) == -1) {
79          perror("init_sems");
80          exit(EXIT_FAILURE);
81      }
82
83      // Создаём подпроцессы
84      for (int num = 0; num < SUBPROC_CNT; num++) {
85          pids[num] = subprocess();
86      }
87
88      // Подключаемся к очереди сообщений
89      if ((msgid = msgget(key, 0666 | IPC_CREAT)) == -1) {
90          perror("msgget");
91          exit(EXIT_FAILURE);
92      }
93
94      // Просушиваем сообщения из очереди
95      for (int i = 0; i < SUBPROC_CNT; i++) {
96          if (msgrcv(msgid, &msg, sizeof(msg.mtext), 1, 0) == -1) {
```

```
            perror("msgrcv");
            exit(EXIT_FAILURE);
        }

        printf("Got message: %s\n", msg.mtext);
    }

    // Закрываем очередь сообщений
    if (msgctl(msgid, IPC_RMID, NULL) == -1) {
        perror("msgctl");
        exit(EXIT_FAILURE);
    }

    // Закрываем разделяемую память
    if (shmctl(shmid, IPC_RMID, NULL) == -1) {
        perror("shmctl");
        exit(EXIT_FAILURE);
    }

    // Уничтожаем набор семафоров
    if (semctl(semid, FILES_CNT, IPC_RMID, NULL) == -1) {
        perror("semctl");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}
```

main.h

```
#pragma once

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/msg.h>
#include<sys/shm.h>

void put_data(char *data);
int init_sems(key_t key, int nsems);
```

subprocess.c

```
#include "subprocess.h"
#include "msg.h"

// Получаем случайное слово из data
// * res -- сюда записывается имя файла
// * data -- буффер из разделяемой памяти со словами, разделёнными '\0'
// * words -- количество слов в буфере разделяемой памяти
// Программа возвращает номер слова в спсике слов
static int get_random_word(char *res, char *data, int words) {
    char buf[MTEXT_SIZE];

    // Получаем случайный id
    int random_id = rand() % words;
```

```c

    // Указатели на начало строки
    char *word = data;
    char *symbol = data;

    // Перебираем слово за словом (в качестве разделителя '\0')
    for (int word_id = 0; word_id <= random_id; word_id++) {

        // Буква за буквой, пока не '\0'
        for (; *symbol != 0; symbol++) {

            // Нам нужно только то самое случайное слово
            if (word_id == random_id) {
                buf[symbol - word] = *symbol;
            }
        }

        // Пропускаем нулевой символ
        word = ++symbol;
    }

    // Запихиваем найденное слово в буфер
    memcpy(res, buf, MTEXT_SIZE);

    return random_id;
}

// Функция для подпроцесса
static void run() {
    key_t key;
    int msgid, shmid;
    char *data;
    char filename[MTEXT_SIZE];

    // Запускаем генератор случайных чисел
    srand(getpid());

    // Получаем уникальный ключ
    if ((key = ftok("main.c", 'F')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }

    // Создаём сегмент разделяемой памяти
    if ((shmid = shmget(key, 1024, 0)) == -1) {
        perror("shmget");
        exit(EXIT_FAILURE);
    }

    // Получаем ссылку на сегмент памяти
    data = shmat(shmid, (void *)0, 0);

    // Подключаемся к очереди сообщений
    if ((msgid = msgget(key, 0666 | IPC_CREAT)) == -1) {
        perror("msgget");
        exit(EXIT_FAILURE);
    }

    // Получаем случайное имя файла и его номер в списке
    int fid = get_random_word(filename, data, FILES_CNT);
```

```
74
75      // Начинается работа с семафором
76      struct sembuf sb;
77      int semid;
78
79      semid = semget(key, FILES_CNT, 0);
80
81      sb.sem_op = -1;
82      sb.sem_num = fid;
83      sb.sem_flg = 0;
84
85      // Закрываем семафор
86      if (semop(semid, &sb, 1) == -1) {
87          perror("semop");
88          exit(EXIT_FAILURE);
89      }
90
91      char kek[MTEXT_SIZE + 32] = "./";
92      char pidstr[16];
93      sprintf(pidstr, "%d", getpid());
94
95      strcat(kek, filename);
96      strcat(kek, " ");
97      strcat(kek, pidstr);
98
99      int status = system(kek);
100     sb.sem_op = 1;
101
102     // Открываем семафор
103     if (semop(semid, &sb, 1) == -1) {
104         perror("semop");
105         exit(EXIT_FAILURE);
106     }
107
108     // Готовим сообщение
109     msgbuf msg;
110     msg.mtype = 1;
111     strcpy(msg.mtext, filename);
112     strcat(msg.mtext, " executed by ");
113     strcat(msg.mtext, pidstr);
114
115     // Отправляем сообщение
116     if (msgsnd(msgid, &msg, sizeof(msg.mtext), 0) == -1) {
117         perror("msgsnd");
118         exit(EXIT_FAILURE);
119     }
120 }
121
122 // Функция для создания подпроцесса
123 int subprocess() {
124     pid_t pid;
125
126     // Создаём подпроцесс
127     if ((pid = fork()) == -1) {
128         perror("Fork failed\n");
129         exit(EXIT_FAILURE);
130     }
131
132     // Выполняем дочерний процесс
133     if (pid == 0) {
```

```
        run();
        exit(EXIT_SUCCESS);
    }

    return pid;
}
```

subprocess.h

```h
#pragma once

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/msg.h>
#include<sys/shm.h>


static int get_random_word(char *res, char *data, int words);
static void run();
int subprocess();
```

msg.h

```h
#define MSGBUF_SIZE 68
#define MTEXT_SIZE 64

typedef struct {
    long mtype;
    char mtext[MTEXT_SIZE];
} msgbuf;

#define SUBPROC_CNT 10
#define FILES_CNT 4

union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};
```

olds/p.c

```c
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>

int main(int argc, char *argv[]) {
    if (argc >= 2) {
        int num = atoi(argv[1]);

        if (num != 0) {
```

```
            srand(num);
            sleep(rand() % 3 + 1);
            printf("Hello, process %s!\n", argv[1]);

        } else {
            srand(time(0));
            sleep(rand() % 3 + 1);
            printf("Hello, unknown!\n");
        }

    } else {
        srand(time(0));
        sleep(rand() % 3 + 1);
        printf("Hello, unknown!\n");
    }

    return 0;
}
```

Makefile

```
SHELL=/bin/bash

CC = gcc
CFLAGS = -c -g
LDFLAGS = -W

PROGRAM_NAME = lab5

SRCS := $(wildcard *.c)
OBJS := $(SRCS:%.c=%.o)
BINS := $(SRCS:%.c=%~)

.INTERMEDIATE: $(OBJS)

all: remove_bin link

remove_bin:
  rm -rvf $(PROGRAM_NAME)

link: $(OBJS)
  $(CC) $(LDFLAGS) $(OBJS) -o $(PROGRAM_NAME)

%.o: %.c
  @echo "Compiling "$<" to object file "$@"..."
  $(CC) $(CFLAGS) $< -o $@

clean:
  @echo "Cleaning up..."
  rm -rvf $(PROGRAM_NAME)
```