



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

---

---

**Algorithm 2** Дочерний процесс

---

```
1: SUBPROC_CNT ← 10                                ▷ Количество подпроцессов
2: FILES_CNT ← 4                                    ▷ Количество файлов
3:
4: процедура Дочерний процесс
5:     создать IPC ключ key
6:
7:     получить сегмент разделяемой памяти data с помощью key
8:
9:     подключиться к очереди сообщений с помощью key
10:
11:    получить случайное имя файла filename из data
12:    получить соответствующий filename номер семафора fid
13:
14:    получить семафоры с помощью key в колчестве FILES_CNT
15:    захватить семафор fid
16:    выполнить программу с именем filename
17:    освободить семафор
18:
19:    отправить информационное сообщение по очереди сообщений
```

---

## Пример вывода программы

```
1 | Hello, process 7093!
2 | Hello, process 7095!
3 | Got message: olds/p1 executed by 7093
4 | Got message: olds/p4 executed by 7095
5 | Hello, process 7105!
6 | Got message: olds/p3 executed by 7105
7 | Hello, process 7094!
8 | Got message: olds/p2 executed by 7094
9 | Hello, process 7102!
10 | Got message: olds/p1 executed by 7102
11 | Hello, process 7099!
12 | Got message: olds/p4 executed by 7099
13 | Hello, process 7098!
14 | Got message: olds/p2 executed by 7098
15 | Hello, process 7101!
16 | Got message: olds/p4 executed by 7101
17 | Hello, process 7103!
18 | Got message: olds/p4 executed by 7103
19 | Hello, process 7104!
20 | Got message: olds/p4 executed by 7104
```

## Вывод

Мы научились пользоваться средствами 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. процсами
│   ├── p.c..... Исходный код программ для запуска доч. процсами
│   ├── p1..... Первая программа для запуска доч. процсами
│   ├── p2..... Вторая программа для запуска доч. процсами
│   ├── p3..... Третья программа для запуска доч. процсами
│   └── p4..... Четвёртая программа для запуска доч. процсами
```

### main.c

```
1  #include "main.h"
2  #include "subprocess.h"
3  #include "msg.h"
4
5  // * Вариант 9
6  // Родительский процесс помещает в сегмент разделяемой памяти имена
7  // программ из предыдущих лабораторных работ, которые могут быть запущены.
8  // Выполняя некоторые циклы работ, порожденные процессы случайным образом
9  // выбирают имена программ из таблицы сегмента разделяемой памяти, запускают эти
10 // программы, и продолжают свою работу. Посредством аппарата семафоров должно
11 // быть обеспечено, чтобы не были одновременно запущены две одинаковые
12 // программы. В процессе работы через очередь сообщений родительский процесс
13 // информируется, какие программы и от имени кого запущены.
14 // P.S. Вместо программ из предыдущих лабораторных работ, мы сделали программу,
15   ↳ которая в качестве аргумента принимает строку, которую затем в консоль как чьё-то
16   ↳ имя, а затем ждёт случайное количество времени.
17
18 // Функция, нужная только для того чтобы
19 // засунуть в разделяемую память нужную информацию.
20 // Код вынесен сюда, чтобы не мешаться под ногами.
21 void put_data(char *data) {
22     char *str, *token, *context;
23
24     data[0] = '\0';
25     strcat(data, "olds/p1 olds/p2 olds/p3 olds/p4");
26
27     token = strtok_r(data, " ", &context);
28
29     for (int i = 0; i < FILES_CNT - 1; i++) {
30         token = strtok_r(NULL, " ", &context);
31     }
32 }
33
34 // Создаём семафоры
35 int init_sems(key_t key, int nsems) {
36     union semun arg;
37     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); // Отчистка
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     // записываем в него нужную информацию
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) {

```

```

97         perror("msgrcv");
98         exit(EXIT_FAILURE);
99     }
100
101     printf("Got message: %s\n", msg.mtext);
102 }
103
104 // Закрываем очередь сообщений
105 if (msgctl(msgid, IPC_RMID, NULL) == -1) {
106     perror("msgctl");
107     exit(EXIT_FAILURE);
108 }
109
110 // Закрываем разделяемую память
111 if (shmctl(shmid, IPC_RMID, NULL) == -1) {
112     perror("shmctl");
113     exit(EXIT_FAILURE);
114 }
115
116 // Уничтожаем набор семафоров
117 if (semctl(semid, FILES_CNT, IPC_RMID, NULL) == -1) {
118     perror("semctl");
119     exit(EXIT_FAILURE);
120 }
121
122 return EXIT_SUCCESS;
123 }

```

## main.h

```

1  #pragma once
2
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  #include<sys/ipc.h>
8  #include<sys/sem.h>
9  #include<sys/msg.h>
10 #include<sys/shm.h>
11
12 void put_data(char *data);
13 int init_sems(key_t key, int nsems);

```

## subprocess.c

```

1  #include "subprocess.h"
2  #include "msg.h"
3
4  // Получаем случайное слово из data
5  // * res -- сюда записывается имя файла
6  // * data -- буффер из разделяемой памяти со словами, разделёнными '\0'
7  // * words -- количество слов в буфере разделяемой памяти
8  // Программа возвращает номер слова в списке слов
9  static int get_random_word(char *res, char *data, int words) {
10     char buf[MTEXT_SIZE];
11
12     // Получаем случайный id
13     int random_id = rand() % words;

```

```

14
15 // Указатели на начало строки
16 char *word = data;
17 char *symbol = data;
18
19 // Перебираем слово за словом (в качестве разделителя '\0')
20 for (int word_id = 0; word_id <= random_id; word_id++) {
21
22     // Буква за буквой, пока не '\0'
23     for (; *symbol != 0; symbol++) {
24
25         // Нам нужно только то самое случайное слово
26         if (word_id == random_id) {
27             buf[symbol - word] = *symbol;
28         }
29     }
30
31     // Пропускаем нулевой символ
32     word = ++symbol;
33 }
34
35 // Записываем найденное слово в буфер
36 memcpy(res, buf, MTEXT_SIZE);
37
38 return random_id;
39 }
40
41 // Функция для подпроцесса
42 static void run() {
43     key_t key;
44     int msgid, shmid;
45     char *data;
46     char filename[MTEXT_SIZE];
47
48     // Запускаем генератор случайных чисел
49     srand(getpid());
50
51     // Получаем уникальный ключ
52     if ((key = ftok("main.c", 'F')) == -1) {
53         perror("ftok");
54         exit(EXIT_FAILURE);
55     }
56
57     // Создаём сегмент разделяемой памяти
58     if ((shmid = shmget(key, 1024, 0)) == -1) {
59         perror("shmget");
60         exit(EXIT_FAILURE);
61     }
62
63     // Получаем ссылку на сегмент памяти
64     data = shmat(shmid, (void *)0, 0);
65
66     // Подключаемся к очереди сообщений
67     if ((msgid = msgget(key, 0666 | IPC_CREAT)) == -1) {
68         perror("msgget");
69         exit(EXIT_FAILURE);
70     }
71
72     // Получаем случайное имя файла и его номер в списке
73     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) {

```



```

134         run();
135         exit(EXIT_SUCCESS);
136     }
137
138     return pid;
139 }

```

## subprocess.h

```

1  #pragma once
2
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7
8  #include<sys/ipc.h>
9  #include<sys/sem.h>
10 #include<sys/msg.h>
11 #include<sys/shm.h>
12
13
14 static int get_random_word(char *res, char *data, int words);
15 static void run();
16 int subprocess();

```

## msg.h

```

1  #define MSGBUF_SIZE 68
2  #define MTEXT_SIZE 64
3
4  typedef struct {
5      long mtype;
6      char mtext[MTEXT_SIZE];
7  } msgbuf;
8
9  #define SUBPROC_CNT 10
10 #define FILES_CNT 4
11
12 union semun {
13     int val;
14     struct semid_ds *buf;
15     unsigned short *array;
16 };

```

## olds/p.c

```

1  #include <unistd.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <time.h>
6
7  int main(int argc, char *argv[]) {
8      int num = atoi(argv[1]);
9
10     if (argc >= 2 && num != 0) {
11         srand(num);

```

```

12         sleep(rand() % 3 + 1);
13         printf("Hello, process %s!\n", argv[1]);
14
15     } else {
16         srand(time(0));
17         sleep(rand() % 3 + 1);
18         printf("Hello, unknown!\n");
19     }
20
21     return 0;
22 }

```

## Makefile

```

1 SHELL=/bin/bash
2
3 CC = gcc
4 CFLAGS = -c -g
5 LDFLAGS = -W
6
7 PROGRAM_NAME = lab5
8
9 SRCS := $(wildcard *.c)
10 OBJS := $(SRCS:%.c=%.o)
11 BINS := $(SRCS:%.c=%~)
12
13 .INTERMEDIATE: $(OBJS)
14
15 all: remove_bin link
16
17 remove_bin:
18     rm -rvf $(PROGRAM_NAME)
19
20 link: $(OBJS)
21     $(CC) $(LDFLAGS) $(OBJS) -o $(PROGRAM_NAME)
22
23 %.o: %.c
24     @echo "Compiling '$<' to object file '$@'..."
25     $(CC) $(CFLAGS) $< -o $@
26
27 clean:
28     @echo "Cleaning up..."
29     rm -rvf $(PROGRAM_NAME)

```