Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6
за I семестр
по дисциплине: "Операционные системы и системное программирование"
Тема: "Средства межпроцессного взаимодействия"

ЛР.190333.ПО4.06 81 00

Листов 9

Выполнил:
студент 2-ого курса
IV-ого семестра
факультета ЭИС
группы ПО-4(1)
Галанин П. И.

Проверил:
ст. преподаватель
Давидюк Ю. И.

Брест 2021

Лабораторная работа №6

Тема: "Средства межпроцессного взаимодействия".

Цель:

Ход работы:

Условие:

Таблица 1 – Варианты

Вариант	Средство взаимодействия	Действия
...	...	...
5	Очереди сообщений	Родитель передает потомку три стороны треугольника, потомок возвращает его площадь
...	...	...

Решение:

Листинг: CMakeLists.txt

```
cmake_minimum_required(
    VERSION 3.13
    FATAL_ERROR
)

add_executable(
    main
    src/main.c
    src/message_queue/message_queue.h
    src/message_queue/message_queue.c
    src/my_ftoa/my_ftoa.h
    src/my_ftoa/my_ftoa.c
)

target_link_libraries(
    main
    -lrt
    -lm
)
```

Листинг: Terminal

```
tree --charset ascii
```

Листинг: Out

```
.
|-- build
|-- CMakeLists.txt
|-- README.md
|-- README-ru.md
`-- src
    |-- main.c
    |-- message_queue
    |   |-- message_queue.c
    |   `-- message_queue.h
    `-- my_ftoa
        |-- my_ftoa.c
        `-- my_ftoa.h
```

Листинг: main.c

```c
#include <unistd.h>     //fork()
#include <sys/types.h>
#include <stdio.h>      //printf()
#include <signal.h>     //SIGUSR1
#include <math.h>       //sqrt()

#include "message_queue/message_queue.h"
#include "my_ftoa/my_ftoa.h"

void my_handler(int nsig);
void input_a_b_c(double* a, double* b, double* c);
void send_a_b_c(double a, double b, double c);
void get_S();
void get_a_b_c();

int main()
{
    pid_t pid;

    // print 0
    printf("::::: start [process 0] PID = %d PPID = %d :::::\n", getpid(), getppid());

    signal(SIGUSR1, my_handler);

    double a,b,c;
    input_a_b_c(&a, &b, &c);

    if ((pid = fork()) == -1)                                   //процесс не создался
    {
        printf("Err\n");
    }
    else if (pid > 0)                                           //в родительском процессе
    {
        printf("\t===== start part 1 =====\n");
        printf("\t::::: start [parent process] PID = %d PPID = %d :::::\n", getpid(), getppid());
        send_a_b_c(a, b, c);
        printf("\t::::: kill [parent process] PID = %d PPID = %d :::::\n", getpid(), getppid());
        printf("\t===== end part 1 =====\n");

        kill(pid, SIGUSR1);
        sleep(2);

        printf("\t===== start part 3 =====\n");
        printf("\t::::: respawn [parent process] PID = %d PPID = %d :::::\n", getpid(), getppid());
        get_S();
        printf("\t::::: end [parent process] PID = %d PPID = %d :::::\n", getpid(), getppid());
        printf("\t===== end part 3 =====\n");
        exit(0);
    }
    else if (pid == 0)                                          //в дочернем процессе
    {
        printf("\t===== start part 2 =====\n");
        printf("\t::::: start [child process] PID = %d PPID = %d :::::\n", getpid(), getppid());
        get_a_b_c();
        printf("\t::::: kill [child process] PID = %d PPID = %d :::::\n", getpid(), getppid());
        printf("\t===== end part 2 =====\n");

        kill(getppid(), SIGUSR1);
```

```
59
60          printf("\t= = = = = start part 4 = = = = =\n");
61          printf("\t::::: respawn [child process] PID = %d PPID = %d :::::\n", getpid(), getppid());
62          printf("\t::::: end [child process] PID = %d PPID = %d :::::\n", getpid(), getppid());
63          printf("\t= = = = = end part 4 = = = = =\n");
64          //exit(0);
65      }
66      printf("::::: end [process 0] PID = %d PPID = %d :::::\n", getpid(), getppid());
67
68      return 0;
69  }
70
71  void my_handler(int nsig)
72  {
73      printf("\t= = = = = my_handler = = = = =\n");
74  }
75
76  void input_a_b_c(double* a, double* b, double* c)
77  {
78      printf("a = ");
79      scanf("%lf", a);                                    // ввод стороны треугольника A
80
81      printf("b = ");
82      scanf("%lf", b);                                    // ввод стороны треугольника B
83
84      printf("c = ");
85      scanf("%lf", c);                                    // ввод стороны треугольника C
86
87      if (!(*a + *b > *c))                                // условие существование треугольника
88      {
89          printf("Not triangle\n");
90          printf("%lf * %lf not > %lf\n", *a, *b, *c);
91          printf("%lf not > %lf\n", *a + *b, *c);
92          printf("\n");
93
94          input_a_b_c(a, b, c);
95          return;
96      }
97      else
98      {
99          printf("Is triagle\n");
100         printf("%lf * %lf > %lf\n", *a, *b, *c);
101         printf("%lf > %lf\n", *a + *b, *c);
102         printf("\n");
103     }
104 }
105
106 void send_a_b_c(double a, double b, double c)
107 {
108     char* string_a = my_ftoa(a);                        // double to char*
109     char* string_b = my_ftoa(b);                        // double to char*
110     char* string_c = my_ftoa(c);                        // double to char*
111
112     mqd_t queue_send_a_b_c = get_opened_message_queue("/abc");// открытие очереди /abc
113
114     send_message_queue(string_a, queue_send_a_b_c);     // отправка сообщения в очередь /abc
115     free(string_a);                                     // очистка памяти
116
117     send_message_queue(string_b, queue_send_a_b_c);     // отправка сообщения в очередь /abc
118     free(string_b);                                     // очистка памяти
```

```
119
120     send_message_queue(string_c, queue_send_a_b_c);          // отправка сообщения в очередь /abc
121     free(string_c);                                          // очистка памяти
122 }
123
124 void get_S()
125 {
126     mqd_t queue_get_s = get_opened_message_queue("/s");      // открытие очереди /s
127
128     reveiving_message_queue(queue_get_s);                    // принятие сообщения из очереди /s
129
130     delete_message_queue("/s");                              // удаление очереди /s
131 }
132
133 void get_a_b_c()
134 {
135     mqd_t bbb = get_opened_message_queue("/abc");            // открываю очередь /abc
136
137     char* msg_a = reveiving_message_queue(bbb);              // получаю сообщение с очереди /abc
138
139     char* msg_b = reveiving_message_queue(bbb);              // получаю сообщение с очереди /abc
140
141     char* msg_c = reveiving_message_queue(bbb);              // получаю сообщение с очереди /abc
142
143     delete_message_queue("/abc");                            // удаляем очередь /abc
144
145     float a = atof(msg_a);                                   // char* to float
146     free(msg_a);                                             // очистка памяти
147
148     float b = atof(msg_b);                                   // char* to float
149     free(msg_b);                                             // очистка памяти
150
151     float c = atof(msg_c);                                   // char* to float
152     free(msg_c);                                             // очистка памяти
153
154     float p = (a + b + c) / 2;                               // полупериметр
155     float S = sqrt( p * (p - a) * (p - b) * (p - c) );       // площадь через теорему Герона
156     printf(
157         "\nS = sqrt(\n\t%f *\n\t* (%f - %f) *\n\t* (%f - %f) *\n\t* (%f - %f)\n) = %f\n\n",
158         p, p, a, p, b, p, c, S
159     );
160
161     char* message_S = my_ftoa(S);                            // double to char*
162
163     mqd_t queue_s = get_opened_message_queue("/s");          // открытие очереди /s
164
165     send_message_queue(message_S, queue_s);                  // отправка сообщения в очередь /s
166
167     free(message_S);                                         // очистка памяти
168 }
```

Листинг: message_queue.c

```c
#include "message_queue.h"

mqd_t get_opened_message_queue(const char message_queue_name[])
{
    printf("\"queue\": \"%s\" - открывается очередь\n", message_queue_name);

    struct mq_attr mqAttr;                                  // создаем структуру
    mqAttr.mq_maxmsg = 10;                                  // максимальное количество сообщений
    mqAttr.mq_msgsize = 1024;                               // максимальная длина сообщения в байт

    mqd_t handler = mq_open(
        message_queue_name,                                 // название очереди
        O_RDWR | O_CREAT,                                   // int флаги
        S_IWUSR | S_IRUSR,                                  // флаги
        &mqAttr                                             // ссылка на структуру
    );

    if (handler < 0)                                        // если очередь не открылась
    {
        printf("Error %d (%s) mq_open for send.\n", errno, strerror(errno));
        exit(-1);
    }

    printf("mqd_t %d - очередь открыта успешно\n", handler);

    return handler;                                         // возвращаем номер очереди
}

void send_message_queue(const char message[], mqd_t handler)
{
    printf("\"message\": \"%s\" - сообщение отправляется\n", message);

    int rc = mq_send(
        handler,                                            // номер очереди
        message,                                            // сообщение
        strlen(message),                                    // длина сообщения
        1                                                   // приоритет
    );

    if (rc < 0)                                             // если сообщение не отправилось
    {
        printf("Error %d (%s) mq_send.\n", errno, strerror(errno));
        exit(-1);
    }

    printf("mqd_t %d - сообщение отправлено успешно\n", handler);
}

char* reveiving_message_queue(mqd_t handler)
{
    char buffer[2048];                                      // сюда запишется сообщение
    printf("mqd_t %d - получаем сообщение\n", handler);

    int rc = mq_receive(
        handler,                                            // номер очереди
        buffer,                                             // строка в которую запишется сообщение
        sizeof(buffer),                                     // длина строки
        NULL                                                // приоритет
```

```
    );

    if (rc < 0)
    {
        printf("Error %d (%s) mq_receive.\n", errno, strerror(errno));
        exit(-1);
    }

    char* msg = (char*) calloc(strlen(buffer), sizeof(char));// выделяем память (не 2048)
    strcpy(msg, buffer);                                     // msg = buffer

    printf("\"message\": \"%s\" - сообщение получено успешно\n", msg);

    return msg;                                              // возвращаем строку (её нужно очищать)
}

void delete_message_queue(const char message_queue_name[])
{
    printf("\"queue\": \"%s\" - очередь закрывается\n", message_queue_name);
    if (mq_unlink(message_queue_name) < 0)
    {
        printf("Warning %d (%s) mq_unlink.\n", errno, strerror(errno));
    }
    printf("\"queue\": \"%s\" - очередь закрыта успешно\n", message_queue_name);
}
```

Листинг: my_ftoa.c

```
#include "my_ftoa.h"

char* my_ftoa(float number)
{
    char* str = (char*) calloc(100, sizeof(char));           // выделяем память под строку
    if (str == NULL)                                         // выделилась память?
    {
        printf("Память не выделилась\n");
        return NULL;
    }

    sprintf(str, "%f", number);                              // копируем float в строку

    char* result = (char*) calloc(strlen(str), sizeof(char));// строка уже не длинны 100
    if (str == NULL)                                         // память выделилась
    {
        printf("Память не выделилась\n");
        return NULL;
    }

    strcpy(result, str);                                     // result = str
    free(str);                                               // очищаем память

    return result;                                           // возвращем строку (её надо очищать!)
}
```

Листинг: Terminal

```
mkdir -p build
cd build
cmake ..
cmake --build .
./main
```

Листинг: Out

```
::::: start [process 0] PID = 14003 PPID = 12430 :::::
a = 2
b = 2
c = 5
Not triangle
2.000000 * 2.000000 not > 5.000000
4.000000 not > 5.000000

a = 3
b = 4
c = 5
Is triagle
3.000000 * 4.000000 > 5.000000
7.000000 > 5.000000

        = = = = = start part 1 = = = = =
        ::::: start [parent process] PID = 14003 PPID = 12430 :::::
"queue": "/abc" - открывается очередь
mqd_t 3 - очередь открыта успешно
"message": "3.000000" - сообщение отправляется
mqd_t 3 - сообщение отправлено успешно
"message": "4.000000" - сообщение отправляется
mqd_t 3 - сообщение отправлено успешно
"message": "5.000000" - сообщение отправляется
mqd_t 3 - сообщение отправлено успешно
        ::::: kill [parent process] PID = 14003 PPID = 12430 :::::
        = = = = = end part 1 = = = = =
        = = = = = my_handler = = = = =
        = = = = = start part 2 = = = = =
        ::::: start [child process] PID = 14007 PPID = 14003 :::::
"queue": "/abc" - открывается очередь
mqd_t 3 - очередь открыта успешно
mqd_t 3 - получаем сообщение
"message": "3.000000" - сообщение получено успешно
mqd_t 3 - получаем сообщение
"message": "4.000000" - сообщение получено успешно
mqd_t 3 - получаем сообщение
"message": "5.000000" - сообщение получено успешно
"queue": "/abc" - очередь закрывается
"queue": "/abc" - очередь закрыта успешно

S = sqrt(
        6.000000 *
        * (6.000000 - 3.000000) *
        * (6.000000 - 4.000000) *
        * (6.000000 - 5.000000)
) = 6.000000

"queue": "/s" - открывается очередь
mqd_t 4 - очередь открыта успешно
```

```
"message": "6.000000" - сообщение отправляется
mqd_t 4 - сообщение отправлено успешно
        ::::: kill [child process] PID = 14007 PPID = 14003 :::::
        = = = = = end part 2 = = = = =
        = = = = = my_handler = = = = =
        = = = = = start part 3 = = = = =
        ::::: respawn [parent process] PID = 14003 PPID = 12430 :::::
"queue": "/s" - открывается очередь
mqd_t 4 - очередь открыта успешно
mqd_t 4 - получаем сообщение
"message": "6.0000000?H?" - сообщение получено успешно
"queue": "/s" - очередь закрывается
"queue": "/s" - очередь закрыта успешно
        ::::: end [parent process] PID = 14003 PPID = 12430 :::::
        = = = = = end part 3 = = = = =
        = = = = = start part 4 = = = = =
        ::::: respawn [child process] PID = 14007 PPID = 14003 :::::
        ::::: end [child process] PID = 14007 PPID = 1 :::::
        = = = = = end part 4 = = = = =
::::: end [process 0] PID = 14007 PPID = 1 :::::
```