Лабораторная работа №5

Кислицин Данила 10702119

Работа с очередью сообщений (Message Queue)

ЦЕЛЬ. Изучить механизм очередей сообщений в LINUX.

ЗАДАНИЕ.

1. Подключить к очереди двух клиентов (один из них выполняется как поток).

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>
#include <iostream>

#define MAX_TEXT 512

using namespace std;

struct message
{
    long int message_type;
    char payload[BUFSIZ];
};

void sending(int msgid)
{
    struct message data;
    long int message_to_recieve = 0;
    char buffer[BUFSIZ];

    fgets(buffer, BUFSIZ, stdin);

    data.message_type = 1;

    strcpy(data.payload, buffer);

    if (msgsnd(msgid, (void*) & data, MAX_TEXT, 0) == -1)
    {
        cout << "msget failed!";
        exit(EXIT_FAILURE);
    }
}

```

```
int main()
{
    int running = 1;
    int msgid_1, msgid_2;
    struct message data;
    char buffer[BUFSIZ];

    msgid_1 = msgget((key_t) 1234, 0666 | IPC_CREAT);

    if (msgid_1 == -1 or msgid_2 == -1)
    {
        cout << "msget failed!";
        exit(EXIT_FAILURE);
    }

    cout << "enter" << endl;
    while(running)
    {
        sending(msgid_1);
    }
}

```

```
#include <thread>

#define MAX_TEXT 512

using namespace std;

struct message
{
    long int message_type;
    char payload[BUFSIZ];
};

void client(int msgid, int client_id)
{
    int running = 1;
    struct message data;
    char buffer[BUFSIZ];
    long int message_to_recieve = 0;

    while(running)
    {
        if (msgrcv(msgid, (void*) & data, BUFSIZ, message_to_recieve,0) == -1)
        {
            cout << "msgrcv failed!";
            exit(EXIT_FAILURE);
        }

        cout << client_id << " Message Recieved: " << data.payload << endl;

        if (strncmp(data.payload, "end", 3) == 0)
        {
            exit(EXIT_FAILURE);
        }
    }

}

```

```
int main()
{
    int msgid_1;
    msgid_1 = msgget((key_t) 1234, 0666 | IPC_CREAT);

    thread client_2(client, msgid_1, 2);
    client(msgid_1, 1);
    client_2.join();
}

```

```
gost@samurai:~/University/3_1/LinuxLabs/lab05/task5$ ./cli
1 Message Recieved:

2 Message Recieved: 3131

1 Message Recieved: 334134

2 Message Recieved:

1 Message Recieved:

2 Message Recieved: 341341

1 Message Recieved: 54

2 Message Recieved:
```

```
gost@samurai:~/University/3_1/LinuxLabs/lab05/task5$ ./ser
enter
3
enter

enter
3131
enter

enter
334134
enter

enter
1341
enter

enter
341341
enter

enter
54
enter

enter
```

2.Создать две очереди сообщений. Первый процесс пишет в первую очередь и читает из второй, второй процесс, наоборот, читает из первой очереди и пишет во вторую.

```
gost@samurai:~/University/3_1/LinuxLabs/lab05/task2$ cd ../task3/
gost@samurai:~/University/3_1/LinuxLabs/lab05/task3$ ./build.sh
gost@samurai:~/University/3_1/LinuxLabs/lab05/task3$ ./ser1
enter
g
Message Recieved: a

enter
g
Message Recieved: aq

enter
fdfd
Message Recieved: gfgf

enter
```

```
gost@samurai:~/University/3_1/LinuxLabs/lab05/task2$ cd ../task3
gost@samurai:~/University/3_1/LinuxLabs/lab05/task3$ chmod +x build.sh
gost@samurai:~/University/3_1/LinuxLabs/lab05/task3$ ./ser2
enter
a
Message Recieved: g

enter
aq
Message Recieved: g

enter
gfgf
Message Recieved: fdfd

enter
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>
#include <iostream>

#define MAX_TEXT 512

using namespace std;

struct message
{
    long int message_type;
    char payload[BUFSIZ];
};

void sending(int msgid)
{
    cout << "enter" << endl;

    struct message data;
    long int message_to_recieve = 0;
    char buffer[BUFSIZ];

    fgets(buffer, BUFSIZ, stdin);

    data.message_type = 1;

    strcpy(data.payload, buffer);

    if (msgsnd(msgid, (void*) & data, MAX_TEXT, 0) == -1)
    {
        cout << "msget failed!";
        exit(EXIT_FAILURE);
    }
}
```

```
void recieving(int msgid)
{
    int running = 1;
    struct message data;
    long int message_to_recieve = 0;
    if (msgrcv(msgid, (void*) & data, BUFSIZ, message_to_recieve,0) == -1)
    {
        cout << "msgrcv failed!";
        exit(EXIT_FAILURE);
    }

    cout << "Message Recieved: " << data.payload << endl;

    if (strncmp(data.payload, "end", 3) == 0)
    {
        exit(EXIT_FAILURE);
    }
}

int main()
{
    int running = 1;
    int msgid_1, msgid_2;
    struct message data;
    char buffer[BUFSIZ];
```

```
    msgid_1 = msgget((key_t) 1234, 0666 | IPC_CREAT);
    msgid_2 = msgget((key_t) 1235, 0666 | IPC_CREAT);

    if (msgid_1 == -1 or msgid_2 == -1)
    {
        cout << "msget failed!";
        exit(EXIT_FAILURE);
    }

    while(running)
    {
        sending(msgid_1);
        recieving(msgid_2);
    }
}
```

3. Сделать два сервера сообщений и одного клиента. Серверы используют одну и ту же очередь. Клиент читает и определяет, от какого сервера пришло сообщение.

```
3
enter
re
enter

enter
gfd
enter

enter
gf
enter

enter
gg
enter
g
enter
gg
enter

enter
g
enter

enter
ggg
enter
g
enter
```

```
gost@samurai:~/University/3_1/LinuxLabs/lab05/task4$ ./se
r2
enter
4
enter
ttr
enter

enter
trtrtr
enter

enter
tr
enter
tr
enter
tr
enter
tr
enter
gsdfg
enter

enter
12341234123412341234123
enter
12341234123412341
enter
112341234123123
```

```
gost@samurai:~/University/3_1/LinuxLabs/lab05/task4$ ./c
li
Server 2 Text captured: 4

Server 1 Text captured:

Server 2 Text captured:

Server 2 Text captured:

Server 2 Text captured: tr

Server 2 Text captured: tr

Server 1 Text captured:

Server 1 Text captured:

Server 1 Text captured: g

Server 1 Text captured:

Server 1 Text captured:

Server 1 Text captured: g

Server 1 Text captured: g

Server 2 Text captured:
```

```
using namespace std;

struct message
{
    int server_id;
    long int message_type;
    char payload[BUFSIZ];
};

int main()
{

    int running = 1;
    int msgid;
    struct message data;

    long int message_to_recieve = 0;

    msgid = msgget((key_t) 1234, 0666 | IPC_CREAT);
    if (msgid == -1)
    {
        cout << "msget failed";
        exit(EXIT_FAILURE);
    }

    while (running)
    {
        if (msgrcv(msgid, (void*) & data, BUFSIZ, message_to_recieve,0) == -1)
        {
            cout << "msgrcv failed!";
            exit(EXIT_FAILURE);
        }
```

```
13  struct message
14  {
15      int server_id;
16      long int message_type;
17      char payload[BUFSIZ];
18  };
19
20  void sending(int msgid, int server_id)
21  {
22      cout << "enter" << endl;
23
24      struct message data;
25      data.server_id = server_id;
26      long int message_to_recieve = 0;
27      char buffer[BUFSIZ];
28
29      fgets(buffer, BUFSIZ, stdin);
30
31      data.message_type = 1;
32
33      strcpy(data.payload, buffer);
34
35      if (msgsnd(msgid, (void*) & data, MAX_TEXT, 0) == -1)
36      {
37          cout << "msget failed!";
38          exit(EXIT_FAILURE);
39      }
40  }
```

```
41
42  int main()
43  {
44      int running = 1;
45      int msgid_1, msgid_2;
46      struct message data;
47      char buffer[BUFSIZ];
48
49      msgid_1 = msgget((key_t) 1234, 0666 | IPC_CREAT);
50
51      if (msgid_1 == -1 or msgid_2 == -1)
52      {
53          cout << "msget failed!";
54          exit(EXIT_FAILURE);
55      }
```