Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных систем и технологий»

**Очереди сообщений в UNIX и работа с ними**

Лабораторная работа №7 по учебной дисциплине «Операционные системы»

По направлению 09.03.01 «Информатика и вычислительная техника»

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_ Кудряшов И. С.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:** познакомиться с механизмами работы сообщений в UNIX.

**Задание:**

В данной работе требуется использовать параллельные потоки (threads), а не процессы.

Основной поток должен сгенерировать и вывести на экран 4 случайных числа.

Далее эти числа должны быть переданы во второй процесс через очередь сообщений.

Второй поток должен сгенерировать все возможные перестановки из этих чисел без повторений («1 2 3 4», «2 1 3 4», «4 3 1 2» и т.д.) и передать их обратно.

Первый поток должен получить все эти перестановки и вывести их на экран. В конце требуется вывести количество полученных перестановок.

Первый поток должен самостоятельно выполнить освобождение всех выделенных ресурсов в конце своей работы.

В отчете привести исходный код программы, а также результаты, выведенные на экран.

**Содержание файла message\_queue.c:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

int get\_random\_number(int min, int max)

{

    return rand() % (max - min + 1) + min;

}

int comp(const int \*a, const int \*b)

{

    return \*a > \*b;

}

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int next(int perm[], int n)

{

    int i = n - 1;

    while (--i >= 0 && perm[i] > perm[i + 1]);

    if (i == -1)

        return 0;

    for (int j = i + 1, k = n - 1; j < k; j++, k--)

        swap(perm + j, perm + k);

    int j = i + 1;

    while (perm[j] < perm[i])

        j++;

    swap(perm + i, perm + j);

    return 1;

}

struct Strmsg

{

    long mtype;

    int data[4];

    int islast;

};

void parentMainCode(int msgId)

{

    struct Strmsg localmsg;

    srand(time(NULL));

    for (int i = 0; i < 4; i++)

        localmsg.data[i] = get\_random\_number(0, 10000);

    printf("Parent: generate %i %i %i %i\n", localmsg.data[0], localmsg.data[1], localmsg.data[2], localmsg.data[3]);

    localmsg.islast = 1;

    localmsg.mtype = 1;

    msgsnd(msgId, &localmsg, sizeof(localmsg), 0);

    int count\_of\_r = 0;

    do

    {

        msgrcv(msgId, &localmsg, sizeof(localmsg), 2, 0);

        if (localmsg.islast)

            break;

        count\_of\_r++;

        printf("Parent: get %i: %i %i %i %i\n", count\_of\_r, localmsg.data[0], localmsg.data[1], localmsg.data[2], localmsg.data[3]);

    } while (!localmsg.islast);

    printf("Parent: wait until child is finished.\n");

    waitpid(0, 0, 0);

    printf("Parent: releasing the message queue.\n");

    msgctl(msgId, IPC\_RMID, NULL);

    printf("Parent: Process is finished.\n");

}

void childMainCode(int msgId)

{

    struct Strmsg childlocalmsg;

    msgrcv(msgId, &childlocalmsg, sizeof(childlocalmsg), 1, 0);

    printf("Child: data reading complete.\n");

    qsort(childlocalmsg.data, 4, sizeof(int), comp);

    childlocalmsg.islast = 0;

    childlocalmsg.mtype = 2;

    msgsnd(msgId, &childlocalmsg, sizeof(childlocalmsg), 0);

    do

    {

        childlocalmsg.islast = !next(childlocalmsg.data, 4);

        childlocalmsg.mtype = 2;

        msgsnd(msgId, &childlocalmsg, sizeof(childlocalmsg), 0);

    } while (!childlocalmsg.islast);

    printf("Child: the last message was send.\n");

    printf("Child: Process is finished.\n");

}

int main()

{

    const size\_t semCount = 10;

    int msgId = msgget(IPC\_PRIVATE, 0600 | IPC\_CREAT);

    if (msgId < 0)

    {

        perror("error with msgget()");

        return -1;

    }

    else

    {

        printf("message id = %i\n", msgId);

    }

    pid\_t childId = fork();

    if (childId < 0)

    {

        perror("error with fork()\n");

    }

    else if (childId > 0)

    {

        parentMainCode(msgId);

    }

    else

    {

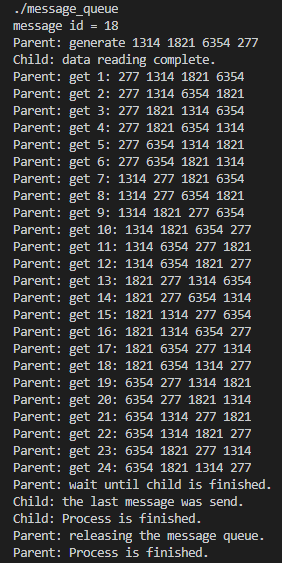
        childMainCode(msgId);

    }

    return 0;

}

**Результат, выведенный на экран:**

****

**Вывод:** В процессе выполнения лабораторной работы я на практике познакомился с работой сообщений в UNIX.