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»

**ОРГАНИЗАЦИЯ РАБОТЫ С РАЗДЕЛЯЕМОЙ ПАМЯТЬЮ. ПОНЯТИЕ ПОТОКОВ В UNIX**

Лабораторная работа №3 по учебной дисциплине «Операционные системы»

По направлению 09.03.01 «Информатика и вычислительная техника»

Отчёт

Принял преподаватель:

\_\_\_\_\_\_\_\_\_ Ананьев В. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Выполнил студент группы 9091:

\_\_\_\_\_\_\_ Баринов Д. А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Великий Новгород

2021

**Цель работы:** научиться работать с разделяемой памятью и потоками в UNIX системах.

**Задание:**

## Задание по разделяемой памяти

Написать две программы на C или C++.

Первая программа должна сформировать массив из 20 случайных чисел. Для массива должна быть выделена область в разделяемой памяти. Затем эта программа должна запустить вторую программу и передать ей ключ для доступа к общей области памяти в качестве параметра командной строки.

Вторая программа должна прочитать из общего массива все числа и вывести их на экран.

Затем она должна отсортировать их по возрастанию и вывести результат сортировки на экран.

## Задание по потокам

Написать многопоточную программу. Основной поток (который начинается в функции main) должен породить два новых потока, дождаться их завершения и закончить работу. Первый из новых потоков должен вывести на экран "Hello Threads (1)", "Hello Threads (2)" и т.д., всего 10 раз. Второй поток должен выводить в цикле строки "This is iteration 1", "This is iteration 2", ... "This is iteration 12".

Каждый из порожденных потоков после вывода каждой строки должен делать системный вызов sleep() с параметром 1 для первого потока и параметром 2 для второго потока (чтобы можно было успевать наблюдать переключение между потоками).

В отчете привести исходные коды всех программ, а также результаты, выведенные ими на экран.

**Реализация:**

**Задание по разделяемой памяти:**

sender

#include <stdio.h>

#include <sys/shm.h>

#include <stdlib.h>

int main()

{

const size\_t memSize = 20\*sizeof(int);

int arr[20];

for (int i = 0; i < 20; i++)

{

arr[i] = rand()%100;

}

int memId = shmget(IPC\_PRIVATE, memSize, 0600|IPC\_CREAT|IPC\_EXCL);

printf("shmid = %i\n" , memId);

if (memId <= 0)

{

printf("error with shmget()\n");

return -1;

}

int \*mem = (int\*)shmat(memId, NULL,0);

if (NULL == mem)

{

printf("error with shmat()\n");

return -2;

}

for (int i = 0; i < 20; i++)

{

mem[i]=arr[i];

}

char callbuf[1024];

sprintf(callbuf, "./receiver %i", memId);

system(callbuf);

return 0;

}

receiver

#include <stdio.h>

#include <sys/shm.h>

#include <stdlib.h>

void Sort(int data[], int lenD)

{

int j = 0;

int tmp = 0;

for(int i=0; i<lenD; i++){

j = i;

for(int k = i; k<lenD; k++){

if(data[j]>data[k]){

j = k;

}

}

tmp = data[i];

data[i] = data[j];

data[j] = tmp;

}

}

int main(int argv, char \*argc[])

{

if (argv <= 1)

{

printf("not enough params\n");

return -1;

}

char \*paramStr = argc[1];

int memId = atoi(paramStr);

if (memId == 0)

{

printf("incorrect parameter string: %s\n", paramStr);

return -2;

}

printf("receivning the memory data: shmid = %i\n", memId);

int \*mem = (int \*)shmat(memId, NULL, 0);

if (NULL == mem)

{

printf("error with shmat()\n");

return -3;

}

for(int i = 0; i < 20; i++)

{

printf("%d\n",mem[i] );

}

printf("Sorted array:\n" );

Sort(mem, 20);

for(int i = 0; i < 20; i++)

{

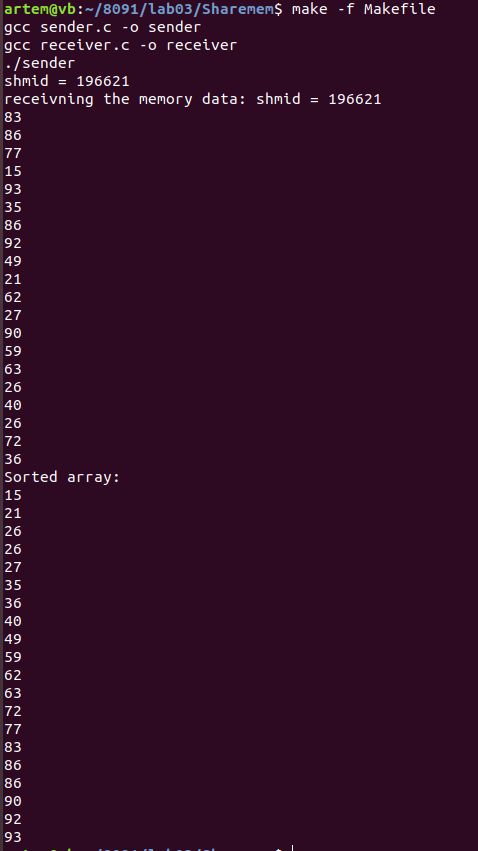
printf("%d\n",mem[i] );

}

return 0;

}

Скриншот результата работы программы:



**Задание по потокам:**

threader

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

void\* print\_message\_function1(void \*ptr)

{

char \*message = (char \*)ptr;

for (int i = 1; i <= 10; i++)

{

printf("%s (%d)\n", message,i);

sleep(1);

}

}

void\* print\_message\_function2(void \*ptr)

{

char \*message = (char \*)ptr;

for (int i = 1; i <= 12; i++)

{

printf("%s %d\n", message,i);

sleep(2);

}

}

int main()

{

pthread\_t thread1, thread2;

const char \*message1 = "Hello Threads ";

const char \*message2 = "This is iteration ";

int res1 = pthread\_create(&thread1, NULL, print\_message\_function1,

(void\*)message1);

int res2 = pthread\_create(&thread2, NULL, print\_message\_function2,

(void\*)message2);

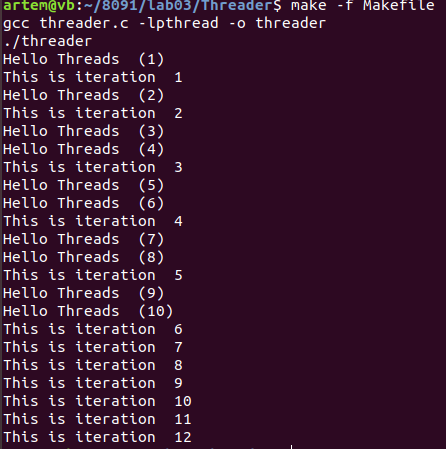
pthread\_join(thread1, (void \*\*)NULL);

pthread\_join(thread2, (void \*\*)NULL);

return 0;

}

Скриншот результата работы программы:



**Вывод:** В процессе выполнения лабораторной работы я научился работать с разделяемой памятью и потоками.