**7 ЛАБОРАТОРНАЯ РАБОТА №7**

**«Исследование подсистемы управления процессами и потоками в ОС семейства UNIX»**

**7.1 Цель работы**

Изучение понятий процесса и потока ОС семейства UNIX. Приобретение практических навыков разработки программ на языке Cи в ОС UNIX, а также написания программ с использованием системных вызовов создания и управления процессами и потоками в ОС UNIX.

**7.2 Вариант задания – 17**

Требуется написать на языке C программу **Sort**, реализующую следующие действия:

Получить целое число (номер процесса) из первого аргумента программной строки;

Получить с использованием системных вызовов getppid() и getpid() идентификаторы родительского и текущего процесса;

Вывести на экран номер процесса, идентификаторы родительского и текущего процесса;

Заполнить массив целых чисел случайными значениями из диапазона 0-100;

Отсортировать массив;

Вывести на экран отсортированный массив (перед каждым элементом массива выводить номер процесса).

Метод сортировки: быстрая; Сортировка по возрастанию; кол-во элементов массива 38.

Написать на языке C программу **Master**, выполняющую следующие действия:

Получить с использованием системных вызовов getppid() и getpid() идентификаторы родительского и текущего процесса и вывести их на экран;

Используя системные вызовы fork() и execlp() создать 3 процесса

Написать на языке C программу **Threads**, содержащую процедуру сортировки массива - **sort**, и процедуру вывода массива на экран – **mass\_print**. Процедура **sort** должна получать идентификатор собственного потока и выводить его на экран (в формате **sort:pthreadId**=threadId), после чего сортировать массив. Процедура **mass\_print** должна получать идентификатор собственного потока, выводить его на экран (в формате **mass\_print:pthreadId**=threadId) и выводить на экран массив.

Программа **Threads** должна выполнять следующие действия:

Заполнить массив случайными числами;

Используя системные вызовы pthread\_create(), создать потоки **sort** и **mass\_print** и получить идентификаторы созданных потоков.

Вывести на экран идентификаторы созданных потоков (в формате **threads:sort:pthreadId**=threadId; **mass\_print:pthreadId**=threadId).

Ожидать завершения потоков (с использованием вызова pthread\_join).

Модифицировать программу **Threads** так, чтобы обеспечить синхронизацию потоков. Поток **mass\_print**, после запуска должен ожидать завершения потока **sort** (с использованием вызова pthread\_join), и лишь затем выводить массив на экран (для этого необходимо при создании потока **mass\_print** в параметре arg передать идентификатор потока **sort**).

**7.3 Ход работы**

7.3.1 Были написаны программы на языке Си согласну варианту

Текст программы Sort:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <unistd.h>

#define N 38

#define RANGE\_MIN 0

#define RANGE\_MAX 100

void fill\_array(int \*array, int size)

{

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

{

array[i] = rand() % (RANGE\_MAX + 1) + RANGE\_MIN;

}

}

void print\_array(int \*array, int size, char \*PROC\_NUM)

{

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

{

printf("PROC\_NUM: [%s] - array[%2d]: %3d;\n", PROC\_NUM, i, array[i]);

}

}

void quick\_sort\_array(int \*array, int left, int right)

{

int pivot;

int l\_hold = left;

int r\_hold = right;

pivot = array[left];

while (left < right)

{

while((array[right] >= pivot) && (left < right))

right--;

if (left != right)

{

array[left] = array[right];

left++;

}

while ((array[left] <= pivot) && (left < right))

left++;

if (left != right)

{

array[right] = array[left];

right--;

}

}

array[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot)

quick\_sort\_array(array, left, pivot-1);

if (right > pivot)

quick\_sort\_array(array, pivot+1, right);

}

int main (int argc, char \*\*argv, char \*\*envp)

{

if (argc != 2) return 1;

pid\_t ppid = getppid(),

pid = getpid();

char \*PROC\_NUM = argv[1];

printf("PROC\_NUM: %s; PPID: %d; PID: %d;\n", PROC\_NUM, ppid, pid);

int arr[N];

fill\_array(arr, N);

//print\_array(arr, N, PROC\_NUM);

quick\_sort\_array(arr, 0, N-1);

//printf("\n...SORTING...\n\n");

print\_array(arr, N, PROC\_NUM);

return 0;

}

Текст программы Master:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#define M 3

int main (int argc, char \*\*argv, char \*\*envp)

{

char arg\_char[1023]; memset (arg\_char, 0, sizeof(arg\_char));

pid\_t ppid = getppid(),

pid = getpid();

/\* Вывод PID и PPID Master-процесса \*/

printf("MASTER. PPID: %d, PID: %d;\n", ppid, pid);

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

{

sprintf(arg\_char, "%d", i + 1);

// Запустить другую программу, если форк, иначе продолжить выполнение цикла

if (!fork())

execlp ("./sort", "sort", arg\_char, NULL);

}

sleep(1);

return 0;

}

Текст программы :

/\*

Дополнительные параметры компиляции:

\* -DUSE\_SYNC -> Использовать синхронизацию потоков.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#ifndef USE\_SYNC

#define USE\_SYNC 0

#endif

#define RANGE\_MIN 0

#define RANGE\_MAX 100

#define N 38

/\* Структура экспорта данных в поток \*/

typedef struct export\_data\_struct

{

pthread\_t neighbor\_thread\_id; // идентификатор соседского потока

int \* array\_ptr; // Указатель на массив

int array\_size; // Размер массива

}

export\_data\_t;

/\* Сортировка данных в массиве \*/

void sort(int \*array, int left, int right)

{

int pivot;

int l\_hold = left;

int r\_hold = right;

pivot = array[left];

while (left < right)

{

while((array[right] >= pivot) && (left < right))

right--;

if (left != right)

{

array[left] = array[right];

left++;

}

while ((array[left] <= pivot) && (left < right))

left++;

if (left != right)

{

array[right] = array[left];

right--;

}

}

array[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot)

sort(array, left, pivot-1);

if (right > pivot)

sort(array, pivot+1, right);

}

/\* Заполнение массива случайными числами \*/

void fill\_array(int \*array, int size)

{

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

{

array[i] = rand() % (RANGE\_MAX + 1) + RANGE\_MIN;

}

}

/\* Печать массива на экран \*/

void print\_array(int \*array, int size)

{

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

{

printf("array[%2d]: %3d;\n", i, array[i]);

}

}

/\* Обёртка для sort для работы с потоками \*/

void \* sort\_wrap (void \* import\_data)

{

/\* Печать имени потока \*/

printf("sort:phreadId=%lu\n", pthread\_self());

/\* Выполнить основное действие потока \*/

sort (

((export\_data\_t \*) import\_data)->array\_ptr,

0,

((export\_data\_t \*) import\_data)->array\_size - 1

);

return NULL;

}

/\* Обёртка для print\_array для работы с потоками \*/

void \* print\_array\_wrap (void \* import\_data)

{

/\* Печать имени потока \*/

printf("print:phreadId=%lu\n", pthread\_self());

#if USE\_SYNC

/\* Ожидать завершения сортировки массива \*/

pthread\_join(((export\_data\_t \*) import\_data)->neighbor\_thread\_id, NULL);

#endif

/\* Выполнить основное действие потока \*/

print\_array(

((export\_data\_t \*) import\_data)->array\_ptr,

((export\_data\_t \*) import\_data)->array\_size

);

return NULL;

}

int main (int argc, char \*\* argv, char \*\* envp)

{

int array[N]; // Массив

export\_data\_t print\_st, sort\_st; // export-структуры

/\* Заполнение массива числами \*/

fill\_array (array, N);

/\* Запись в структуры размеров массива и указателя на этот массив \*/

print\_st.array\_size = sort\_st.array\_size = N;

print\_st.array\_ptr = sort\_st.array\_ptr = ((int \*) array);

/\* Создание потоков \*/

pthread\_create(&print\_st.neighbor\_thread\_id, NULL, sort\_wrap, (void \*) &sort\_st);

pthread\_create(&sort\_st.neighbor\_thread\_id, NULL, print\_array\_wrap, (void \*) &print\_st);

/\* Ожидание завершения работы потока печати массива \*/

pthread\_join(sort\_st.neighbor\_thread\_id, NULL);

return 0;

}

7.3.2 В начале были написаны программы Sort и Master, представленные выше. Рисунок 7.1 демонстрирует вывод процессов. Был сделан вывод, что создание и запуск процессов были осуществлены правильно, а также вследствие несинхронизированной работы процессов выводы процессов перемешаны между собой.

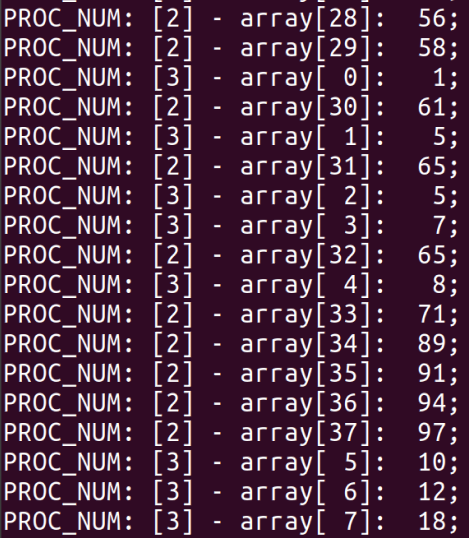


Рисунок 7.1 – Сортировка и печать массивов несколькими процессами

Затем была написана программа Threads. Рисунок 7.2 демонстрирует результат запуска программы. В результате работы программы были сделаны выводы:

Работа потоков сортировки и печати массива не синхронизирована, что подтверждают вывод несортированного массива и запуск потока сортировки лишь после вывода всех элементов массива.

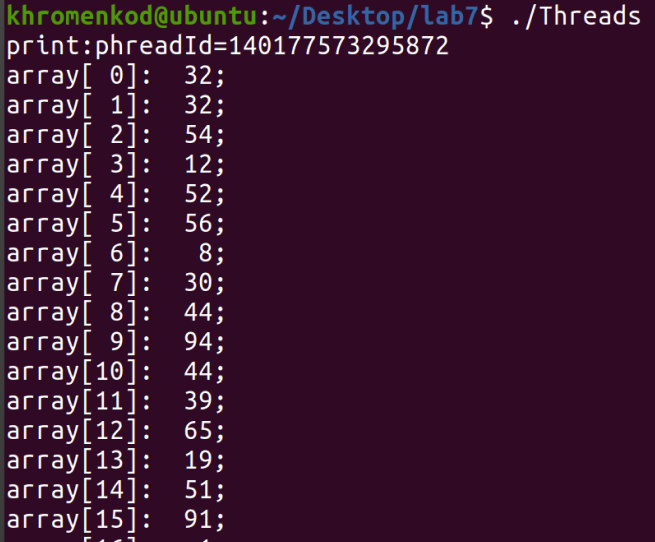


Рисунок 7.2 – Печать массива без синхронизации потоков

После чего программа была переписана с применением pthread\_join для синхронизации потоков. Рисунок 7.3 демонстрирует корректную работу программы.

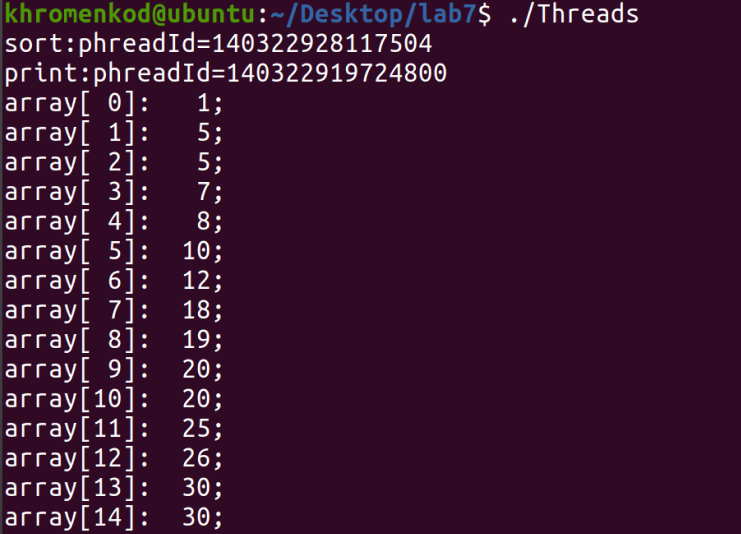


Рисунок 7.3 – Печать массива с синхронизацией потоков

Результаты тестирования полностью соответствуют ожиданиям.

**Выводы**

При выполнении данной лабораторной работы были получены навыки создания процессов и потоков в операционной системе семейства UNIX. Также были повторно закреплены навыки работы в операционной системе семейства UNIX. Полученные знания, опыт и навыки позволят создавать многопоточные программы в ОС семейства UNIX.