# ЛАБОРАТОРНАЯ РАБОТА №7

# ИССЛЕДОВАНИЕ ПОДСИСТЕМЫ УПРАВЛЕНИЯ ПРОЦЕССАМИ И ПОТОКАМИ В ОС СЕМЕЙСТВА UNIX

Цель работы

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

Задачи

1. Написать на языке C программу Sort, реализующую следующие действия:
   1. Получить целое число (номер процесса) из первого аргумента программной строки;
   2. Получить с использованием системных вызовов getppid() и getpid() идентификаторы родительского и текущего процесса;
   3. Вывести на экран номер процесса, идентификаторы родительского и текущего процесса;
   4. Заполнить массив целых чисел случайными значениями из диапазона 0-100;
   5. Отсортировать массив;
   6. Вывести на экран отсортированный массив (перед каждым элементом массива выводить номер процесса, полученный в п. 1.1);

Метод и направление сортировки, а также количество элементов массива N выбирается в соответствии с вариантом задания (Таблица 1).

1. Написать на языке C программу Master, выполняющую следующие действия:
   1. Получить с использованием системных вызовов getppid() и getpid() идентификаторы родительского и текущего процесса и вывести их на экран;
   2. Используя системные вызовы fork() и exec(), создать M процессов (функция для выполнения системного вызова exec() и количество процессов M определяются по Таблице 1);
2. 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:
   1. Заполнить массив случайными числами;
   2. Используя системные вызовы pthread\_create(), создать потоки sort и mass\_print и получить идентификаторы созданных потоков;
   3. Вывести на экран идентификаторы созданных потоков (в формате threads: sort: pthreadId = threadId; mass\_print: pthreadId = threadId);
   4. Ожидать завершения потоков (с использованием вызова pthread\_join);
3. 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);

Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № варианта | Метод сортировки | Направление сортировки | N | Функция для выполнения *exec()* | Количество процессов М |
| 11 | Шелла | Убыв. | 32 | *execle()* | 3 |

Таблица 1 – Вариант задания

Ход работы

Была запущена программа Master. В результате её выполнения в одну консоль был произведён вывод трёх процессов программы Sort (Рисунок 1). Поскольку вызываемые процессы не были синхронизированы, при выводе данные перемешались.

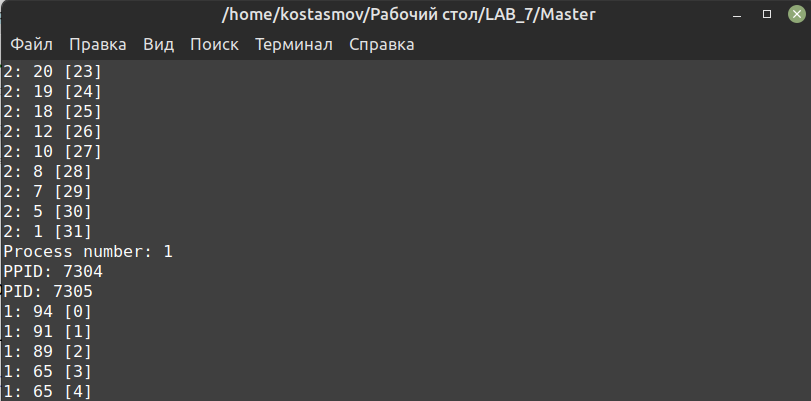


Рисунок 1 – Вызов программы Master

Была запущена первая версия программы Threads. Несмотря на отсутствие в ней механизма синхронизации потоков, программа верно вывела отсортированный массив (Рисунок 2). Скорее всего, это связано с высокой скоростью сортировки.

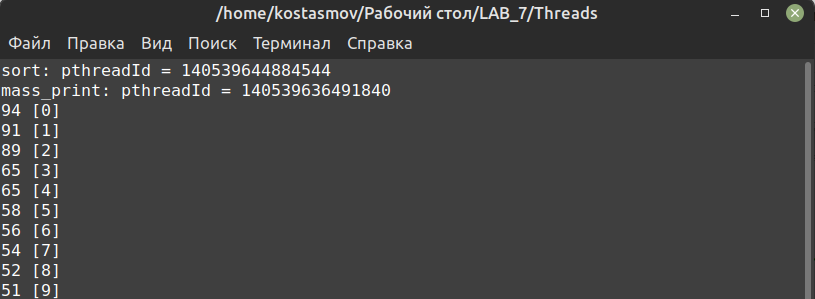


Рисунок 2 – Вызов программы Threads (без синхронизации)

Тем не менее, при увеличении длины массива (N = 132) потоки всё же перекрыли друг друга, что привело к ошибке вывода (Рисунок 3).

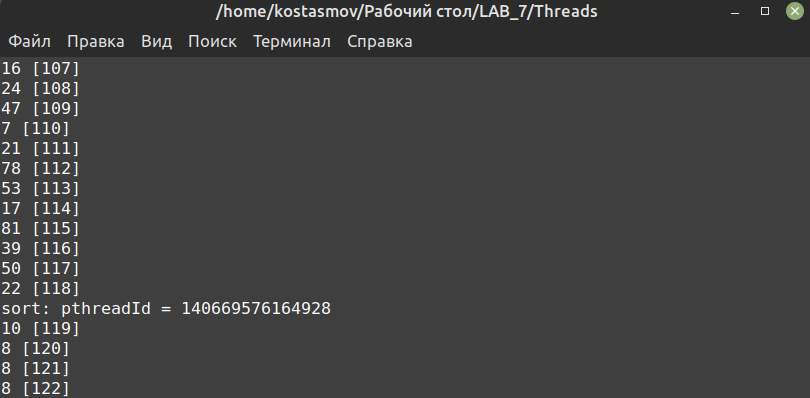


Рисунок 3 – Вызов программы Threads (без синхронизации) для N = 132

Программа Threads была модифицирована таким образом, чтобы потоки сортировки и вывода были синхронизированы. Для этого в поток вывода был передан идентификатор потока сортировки. В результате при вызове программы Threads вывод отсортированного массива производился правильно при любом N (Рисунок 4).

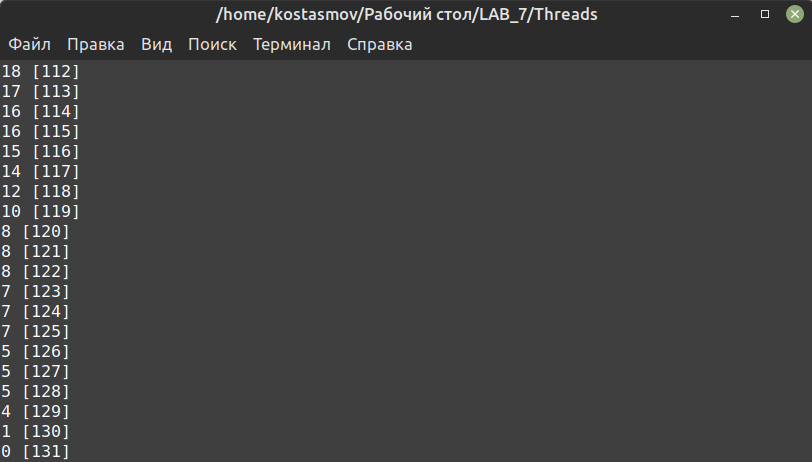


Рисунок 4 – Вызов модифицированной программы Threads

Текст программы

Программа Sort:

#include <iostream>

#include <unistd.h>

using namespace std;

const int N = 32; // длина массива, задана вариантом

// Заполнение массива

void fillArr(int arr[], int len)

{

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

{

array[i] = rand() % 101 + 0;

}

return;

}

// Сортировка методом Шелла

void sortArr(int arr[], int len)

{

int i, j, step, temp;

for (step = len / 2; step > 0; step /= 2)

for (i = step; i < len; i++)

{

temp = arr[i];

for (j = i; j >= step; j -= step)

{

if (temp > arr[j - step])

arr[j] = arr[j - step];

else break;

}

arr[j] = temp;

}

return;

}

// Вывод массива

void printArr(int arr[], int len, char\* pnum)

{

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

{

cout << pnum << ": " << arr[i];

cout << " [" << i << "]" << endl;

}

return;

}

// Главная функция - процесс обработки массива

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

{

// проверка наличия параметра (№ процесса)

if (argc < 2)

{

cout << "Sort: ERROR - No parameter" << endl;

return 1;

}

// получение данных процесса

char\* pnum = argv[1]; // номер вызванного процесса

pid\_t ppid = getppid(); // идент. родителя

pid\_t pid = getpid(); // идент. процесса

// вывод данных о процессе

cout << "Process number: " << pnum << endl;

cout << "PPID: " << ppid << endl;

cout << "PID: " << pid << endl;

// обработка массива

int arr[N];

fillArr(arr, N);

sortArr(arr, N);

printArr(arr, N, pnum);

return 0;

}

Программа Master:

#include <iostream>

#include <cstring>

#include <unistd.h>

using namespace std;

const int M = 3; // число запускаемых процессов

// Главная функция - запуск М процессов Sort

int main()

{

char pnum[1]; // номер процесса

// получение и вывод данных о процессе

pid\_t ppid = getppid();

cout << "Master: PPID = " << ppid << endl;

pid\_t pid = getpid();

cout << "Master: PID = " << pid << endl;

// цикл запусков процессов Sort

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

{

sprintf(pnum, "%d", i + 1);

if (!fork())

execle("./Sort", "Sort", pnum, 0, NULL);

}

sleep(1);

return 0;

}

Программа Threads:

#include <iostream>

#include <unistd.h>

#include <pthread.h>

using namespace std;

const int N = 132; // длина массива, задана вариантом

int arr[N]; // обрабатываемый массив

// Заполнение массива

void mass\_fill(int arr[], int len)

{

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

{

arr[i] = rand() % 101 + 0;

}

return;

}

// Поток сортировки массива

void\* sort(void\* arg)

{

// получение ID потока sort

pthread\_t ID = pthread\_self();

printf("sort: pthreadId = %lu\n", ID);

// сортировка массива (метод Шелла)

int i, j, step, temp;

for (step = N / 2; step > 0; step /= 2)

for (i = step; i < N; i++)

{

temp = arr[i];

for (j = i; j >= step; j -= step)

{

if (temp > arr[j - step])

arr[j] = arr[j - step];

else break;

}

arr[j] = temp;

}

return 0;

}

// Поток вывода массива

void\* mass\_print(void\* arg)

{

// получение ID потока sort и ожидание его завершения

pthread\_t sortID = (pthread\_t)arg;

pthread\_join(sortID, NULL);

// получение ID потока mass\_print

pthread\_t ID = pthread\_self();

printf("mass\_print: pthreadId = %lu\n", ID);

// вывод массива

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

{

cout << arr[i] << " [" << i << "]" << endl;

}

return 0;

}

// Главная функция - управление потоками обработки массива

int main()

{

mass\_fill(arr, N); // заполнение массива

pthread\_t sortID, mass\_printID; // идентификаторы потоков

// Запуск потоков

pthread\_create(&sortID, NULL, sort, NULL);

pthread\_create(&mass\_printID, NULL, mass\_print, (void\*)sortID);

// Ожидание окончания потоков

//pthread\_join(sortID, NULL);

pthread\_join(mass\_printID, NULL);

return 0;

}

Вывод

В ходе работы были получены навыки управления процессами и потоками в ОС семейства UNIX. Были изучены особенности обработки процессов операционной системой, принципы поведения потоков по стандарту POSIX, метод синхронизации потоков. В результате был написан ряд программ, в которых были реализованы изменение пользовательского контекста процесса-потомка и многопоточная обработка данных.