# Лабораторная работа №7

Исследование подсистемы управления процессами и потоками в ОС семейства UNIX

## 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.

## Постановка задачи

Получение практических навыков работы с операционной системой семейства UNIX, в частности с GNU/Linux, а также написания программ с использованием потоком в соответствии со стандартом POSIX.

## Ход выполнения работы

### Листинг 1 – программа Sort

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <unistd.h>

#define N 48

#define RANGE\_MIN 0

#define RANGE\_MAX 100

void sort(int array[], int size)

{

int tmp;

for (int startIndex = 0; startIndex < size - 1; ++startIndex)

{

int smallestIndex = startIndex;

for (int currentIndex = startIndex + 1; currentIndex < size; currentIndex++)

if (array[currentIndex] < array[smallestIndex])

smallestIndex = currentIndex;

tmp = array[startIndex];

array[startIndex] = array[smallestIndex];

array[smallestIndex] = tmp;

}

return;

}

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

{

if (size > 0)

{

\*array = rand() % (RANGE\_MAX + 1) + RANGE\_MIN;

fill\_array(array + 1, size - 1);

}

}

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

{

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

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

}

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

{

if (argc != 2) return 1;

pid\_t ppid = getppid(),

pid = getpid();

char prefix[1023];

memset(prefix, 0, sizeof(prefix));

sprintf(prefix, "PROC\_NUM: %s. PPID/PID: %d/%d", argv[1], ppid, pid);

int arr[N];

fill\_array (arr, N);

print\_array(arr, N, prefix);

sort(arr, N);

print\_array(arr, N, prefix);

return 0;

}

### Листинг 2 – программа Master

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <unistd.h>

#include <sys/types.h>

#define M 4

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/PID: %d/%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;

}

### Листинг 3 – программа Threads

/\*

Дополнительные параметры компиляции:

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

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.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 48

/\* 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 size)

{

int tmp;

for (int startIndex = 0; startIndex < size - 1; ++startIndex)

{

int smallestIndex = startIndex;

for (int currentIndex = startIndex + 1; currentIndex < size; currentIndex++)

if (array[currentIndex] < array[smallestIndex])

smallestIndex = currentIndex;

tmp = array[startIndex];

array[startIndex] = array[smallestIndex];

array[smallestIndex] = tmp;

}

return;

}

/\* Заполнение массива случайными числами \*/

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

{

if (size > 0)

{

\*array = rand() % (RANGE\_MAX + 1) + RANGE\_MIN;

fill\_array(array + 1, size - 1);

}

}

/\* Печать массива на экран \*/

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

{

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

printf("%s: array[%2d]: %3d.\n", prefix, 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,

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

);

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,

"PRINT\_THREAD"

);

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;

}

### Тестовые примеры

В начале были написаны программы Sort и Master, представленные в Листингах 1 и 2 соответственно. 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.

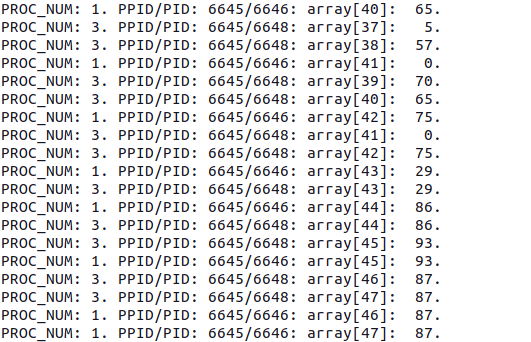


Рисунок 7.1 – Сортировка и печать массивов несколькими процессами

Затем была написана программа Threads, представленная в Листинге 3. Рисунок 7.2 демонстрирует результат запуска программы. По выводу программы были сделаны выводы:

1. Работа потоков сортировки и печати массива не синхронизирована, что подтверждают вывод несортированного массива и запуск потока сортировки лишь после вывода всех элементов массива.
2. В результате одновременного доступа к разделяемому ресурсу операционной системой было образован сигнал SIGSEGV.

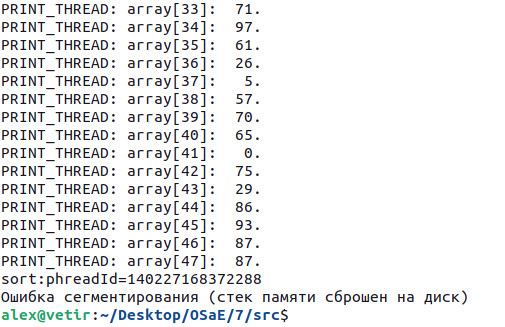


Рисунок 7.2 – Печать массива (без синхронизации потоков)

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

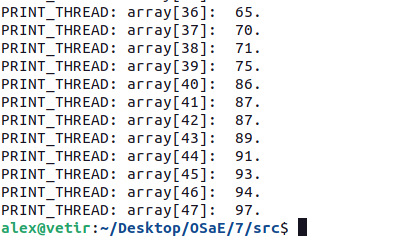


Рисунок 7.3 – Печать массива (с синхронизацией потоков)

## Вывод

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.