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»

**Семафоры в UNIX как средство синхронизации процессов**

Лабораторная работа №6 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:

\_\_\_\_\_\_\_ Сухарев И.Д.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Великий Новгород

2021

**Цель работы: О**знакомиться с механизмами работы семафоров.

**Задание:**

Основной процесс должен выделить область разделяемой памяти и заполнить ее N случайными числами. Количество чисел, а также минимальное и максимальное допустимые значения должны быть получены из параметров командной строки.

Каждому элементу массива должен быть сопоставлен свой собственный семафор, предотвращающий одновременное обращение двух процессов к одним и тем же элементам.

После заполнения массива должен быть порожден второй процесс, который начнет производить сортировку чисел по возрастанию.

Пока идет сортировка, первый процесс должен пытаться последовательно читать числа из массива и выводить на экран их текущее значение и результат работы с семафором: удалось ли получить доступ к числу сразу, или пришлось ждать освобождения семафора.

После окончания обхода массива первый процесс должен проверить, завершился ли уже процесс сортировки. Если да - то вывести массив и завершиться. Если еще нет - то начать новую итерацию обхода массива. В начале каждой итерации обхода требуется выводить ее порядковый номер.

Проверку факта завершения дочернего процесса (без перевода текущего процесса в состояние ожидания) можно сделать при помощи функции waitpid с параметром WHOHANG. (подробнее см. man waitpid)

Второй процесс в ходе сортировки также должен проверять и выставлять семафор перед обращением к каждому из элементов.

Далее первый процесс должен вывести на экран отсортированный массив. Второй процесс не должен осуществлять никакого вывода на экран.

Первый процесс должен самостоятельно выполнить освобождение всех выделенных ресурсов (в том числе семафоров) в конце своей работы.

**Содержание файла sem.c:**

#include "stdio.h"

#include "stdlib.h"

#include "unistd.h"

#include "time.h"

#include "stdlib.h"

#include "sys/shm.h"

#include "sys/types.h"

#include "sys/sem.h"

#include "sys/ipc.h"

#include "sys/wait.h"

#include <sys/ipc.h>

#define UNLOCK 1

#define LOCK -1

void arr\_rand\_fill(int\* arr, int size, int min, int max)

{

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

arr[i] = min + rand() % max;

}

void\* alloc\_shrd\_mem(size\_t mem\_size, int\* mem\_id)

{

\*mem\_id = shmget(IPC\_PRIVATE, mem\_size, 0600 | IPC\_CREAT | IPC\_EXCL);

if (\*mem\_id <= 0)

{

perror("error with memId");

return NULL;

}

void\* mem = shmat(\*mem\_id, 0, 0);

if (NULL == mem)

perror("error with shmat");

return mem;

}

void set\_state(int sem\_id, int num, int state)

{

struct sembuf op;

op.sem\_op = state;

op.sem\_flg = 0;

op.sem\_num = num;

semop(sem\_id, &op, 1);

}

char set\_state\_nowait(int sem\_id, int num, int state)

{

struct sembuf op;

op.sem\_op = state;

op.sem\_flg = IPC\_NOWAIT;

op.sem\_num = num;

return semop(sem\_id, &op, 1);

}

void swap\_vals(int\* first, int\* second)

{

int temp = \*first;

\*first = \*second;

\*second = temp;

}

void parent\_code(int\* arr, int arr\_size, int sem\_id, pid\_t child\_id)

{

int count = 0;

while (!waitpid(child\_id, NULL, WNOHANG))

{

printf("Count %i\n", count);

for (int i = 0; i < arr\_size; i++)

{

if (set\_state\_nowait(sem\_id, i, LOCK) == -1)

printf("block ");

else

{

printf("%d ", arr[i]);

set\_state(sem\_id, i, UNLOCK);

}

}

printf("\n");

count++;

}

printf("Iteration is: %i\n", count);

}

void child\_code(int\* arr, int arr\_size, int sem\_id)

{

double factor = 1.2473309;

int step = arr\_size - 1;

while (step >= 1)

{

for (int i = 0; i + step < arr\_size; i++)

{

set\_state(sem\_id, i, LOCK);

set\_state(sem\_id, i + step, LOCK);

if (arr[i] > arr[i + step])

swap\_vals(&arr[i], &arr[i + step]);

set\_state(sem\_id, i + step, UNLOCK);

set\_state(sem\_id, i, UNLOCK);

}

step /= factor;

}

exit(0);

}

void free\_shrd\_memory(int\* mem\_id);

void free\_sems(int\* sem\_id);

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

{

if (argv <= 3)

{

printf("Error! Not enough arguments! Required: 3 (arr\_size, min, max)\n");

return -1;

}

int arr\_size = atoi(argc[1]);

int arr\_min = atoi(argc[2]);

int arr\_max = atoi(argc[3]);

int mem\_id;

int\* arr\_ptr = alloc\_shrd\_mem(sizeof(int) \* arr\_size, &mem\_id);

arr\_rand\_fill(arr\_ptr, arr\_size, arr\_min, arr\_max);

int sem\_id = semget(IPC\_PRIVATE, arr\_size, 0600 | IPC\_CREAT);

if (sem\_id < 0)

{

perror("Error with semget()!\n");

return -1;

}

printf("Semaphore set id = %i\n", sem\_id);

for (int i = 0; i < arr\_size; i++)

set\_state(sem\_id, i, UNLOCK);

pid\_t child\_process\_id = fork();

if (child\_process\_id == -1)

perror("Error with fork() - process 1\n");

else if (child\_process\_id == 0)

child\_code(arr\_ptr, arr\_size, sem\_id);

else

parent\_code(arr\_ptr, arr\_size, sem\_id, child\_process\_id);

free\_sems(&sem\_id);

free\_shrd\_mem(&mem\_id);

return 0;

}

void free\_shrd\_mem(int\* mem\_id)

{

char del\_cmd\_temp[124];

sprintf(del\_cmd\_temp, "ipcrm -m %i", \*mem\_id);

system(del\_cmd\_temp);

\*mem\_id = 0;

}

void free\_sems(int\* sem\_id)

{

char del\_cmd\_temp[124];

sprintf(del\_cmd\_temp, "ipcrm -s %i", \*sem\_id);

system(del\_cmd\_temp);

\*sem\_id = 0;

}

**Результат, выведенный на экран:**

/ilya/Desktop/laba6/make

gcc sem.c -o s

./s 10 1 100

23 71 34 91 78 43 14 31 83 22

0: 23 71 34 91 78 43 14 31 83 22

1: 23 71 34 91 78 43 14 31 83 22

2: 23 71 34 91 78 43 14 31 83 22

3: 23 71 34 91 78 43 14 31 83 22

4: 23 71 34 91 78 43 14 31 83 22

5: 23 71 34 91 78 43 14 31 83 22

6: 23 71 34 91 78 43 14 31 83 22

7: 14 71 34 91 78 43 23 31 83 71

8: 14 22 23 91 78 43 34 91 83 71

9: 14 22 23 31 34 43 78 91 83 71

10: 14 22 23 31 34 43 71 78 83 91

11: 14 22 23 31 34 43 71 78 83 91

12: 14 22 23 31 34 43 71 78 83 91

13: 14 22 23 31 34 43 71 78 83 91

14: 14 22 23 31 34 43 71 78 83 91

15: 14 22 23 31 34 43 71 78 83 91

Sort finished

14 22 23 31 34 43 71 78 83 91

**Вывод:** В процессе выполнения лабораторной работы я в процессе изучения теории и на практике ознакомился с работой семафоров в UNIX.