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.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_ Шляханов Д. А.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:** познакомиться 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 semaphore.c:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <string.h>

#include <time.h>

void sem(int semId, int n, int d)

{

struct sembuf op;

op.sem\_op = d;

op.sem\_flg = 0;

op.sem\_num = n;

semop(semId, &op, 1);

}

void unlockSem(int semId, int n)

{

sem(semId, n, 1);

}

void lockSem(int semId, int n)

{

sem(semId, n, -1);

}

void sort(int semId, int memId, const size\_t n)

{

int \*nums = (int \*)shmat(memId, 0, 0);

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

{

int minInd = i;

for (int j = i + 1; j < n; j++)

{

lockSem(semId, i);

lockSem(semId, j);

if (nums[j] < nums[minInd])

{

minInd = j;

}

unlockSem(semId, i);

unlockSem(semId, j);

}

if (i != minInd)

{

lockSem(semId, i);

lockSem(semId, minInd);

int t = nums[i];

nums[i] = nums[minInd];

nums[minInd] = t;

unlockSem(semId, i);

unlockSem(semId, minInd);

}

}

}

void fill\_random\_nums(int \*nums, int n, int min, int max)

{

srand((unsigned)(time(0)));

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

{

nums[i] = min + rand() % (max - min + 1);

}

}

void print\_nums(int \*nums, int n)

{

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

{

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

}

printf("\n");

}

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

{

const int N = atoi(argv[1]);

const int min = atoi(argv[2]);

const int max = atoi(argv[3]);

int memId = shmget(IPC\_PRIVATE, sizeof(int) \* N, 0600 | IPC\_CREAT | IPC\_EXCL);

int semId = semget(IPC\_PRIVATE, N, 0600 | IPC\_CREAT);

int \*numbers = (int \*)shmat(memId, 0, 0);

fill\_random\_nums(numbers, N, min, max);

print\_nums(numbers, N);

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

{

unlockSem(semId, i);

}

int childId = fork();

if (childId == 0)

{

sort(semId, memId, N);

}

else

{

int i = 0;

int status;

do

{

printf("%d: ", i);

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

{

time\_t before = time(0);

lockSem(semId, j);

if (before == time(0))

{

printf("%d ", numbers[j]);

}

else

{

printf("[%d] ", numbers[j]);

}

fflush(stdout);

unlockSem(semId, j);

}

printf("\r\n");

status = waitpid(childId, NULL, WNOHANG);

i++;

} while (!status);

printf("Sort finished \r\n");

print\_nums(numbers, N);

shmctl(memId, 0, IPC\_RMID);

semctl(semId, 0, IPC\_RMID);

}

}

**Результат, выведенный на экран:**

den@gremlin:~/labs/6/semaphore $ make

gcc semaphore.c -o semaphore

./semaphore 10 50 100

72 80 77 79 76 56 69 52 51 78

0: 72 80 77 79 76 56 69 52 51 78

1: 51 52 56 69 72 76 77 78 79 80

Sort finished

51 52 56 69 72 76 77 78 79 80

**Вывод:** В процессе выполнения лабораторной работы я на практике познакомился с работой семафоров в UNIX.