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:

\_\_\_\_\_\_\_ Григорьев Д. И.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г.

Великий Новгород

2020

**Цель работы:** научиться работать с семафорами в UNIX-like системах.

**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.

В отчете привести исходный код программы, а также результаты, выведенные на экран.

**Реализация:**

#include <sys/shm.h>

#include <sys/wait.h>

#include <sys/sem.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

*void* sem(*int* *semId*, *int* *n*, *int* *o*)

{

*struct* sembuf op;

    op.sem\_op = o;

    op.sem\_flg = 0;

    op.sem\_num = n;

    semop(semId, &op, 1);

}

*void* rand\_fill(*int*\* *arr*, *int* *n*, *int* *mind*, *int* *maxd*)

{

    srand(time(NULL));

*int* randVar = maxd - mind + 1;

    for (*int* i = 0; i < n; ++i)

    {

        arr[i] = mind + rand() % randVar;

    }

}

*void* arr\_output(*int* \**arr*, *int* *n*)

{

    for (*int* i = 0; i < n; ++i)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

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

{

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

    for (*int* i = 0; i < n; ++i)

    {

*int* mInd = i;

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

        {

            sem(semId, i, -1);

            sem(semId, j, -1);

            if (arr[j] < arr[mInd])

            {

                mInd = j;

            }

            sem(semId, i, 1);

            sem(semId, j, 1);

        }

        if (i != mInd)

        {

            sem(semId, i, -1);

            sem(semId, mInd, -1);

*int* temp = arr[i];

            arr[i] = arr[mInd];

            arr[mInd] = temp;

            sem(semId, i, 1);

            sem(semId, mInd, 1);

        }

    }

}

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

{

*int* n = atoi(argc[1]);

*int* mind = atoi(argc[2]);

*int* maxd = atoi(argc[3]);

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

*int* semId = shmget(IPC\_PRIVATE, n, 0600 | IPC\_CREAT);

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

    rand\_fill(arr, n, mind, maxd);

    printf("source: \r\n");

    arr\_output(arr, n);

    printf("\n");

    for (*int* i = 0; i < n; ++i)

    {

        sem(semId, i, 1);

    }

*int* childId = fork();

    if (childId == 0)

    {

        sort\_sem(semId, memId, n);

    }

    else

    {

*int* i = 0;

*int* status;

        do

        {

            printf("%d: ", i);

            for (*int* j = 0; j < n; ++j)

            {

*time\_t* before = time(NULL);

                sem(semId, j, -1);

                if (before == time(NULL))

                {

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

                }

                else

                {

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

                }

                fflush(stdout);

                sem(semId, j, -1);

            }

            printf("\r\n");

            status = waitpid(childId, NULL, WNOHANG);

            ++i;

        } while(!status);

        printf("\nresult: \r\n");

        arr\_output(arr, n);

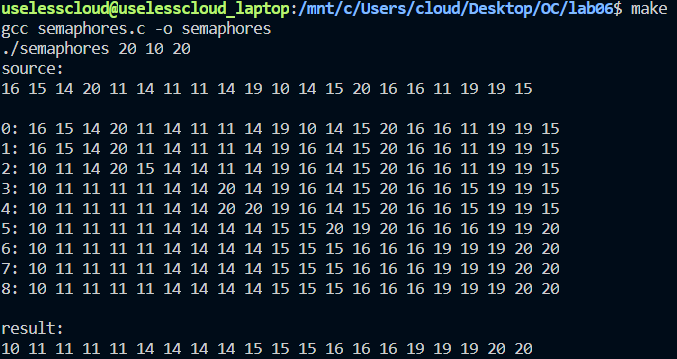
        shmctl(memId, 0, IPC\_RMID);

        semctl(semId, 0, IPC\_RMID);

    }

}

**Результат выполнения программы:**

****

**Вывод:** В процессе выполнения лабораторной работы я научился работать с семафорами.