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**

Лабораторная работа №6

Отчёт

Чалый С.М 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ананьев В.В

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Великий Новгород

2021

**Цель работы:** научиться работать с семафорами в UNIX 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.

**Реализация:**

*void* sem(*int* *semId*, *int* *n*, *int* *state*)// Возвращает 0 в случае успешного завершения, –1 – в случае ошибки

{

*struct* sembuf op;

    op.sem\_op = state;//операция с семафором

    op.sem\_flg = 0;//флаги операции

    op.sem\_num = n;//номер семафора

    semop(semId, &op, 1);  используется для атомарного выполнения массива семафорных операций над набором семафоров

}

*void* rand(*int*\* *arr*, *int* *n*, *int* *mind*, *int* *maxd*)//рандомные числа

{

    srand(time(0));

*int* randnumber = maxd - mind + 1;

    for (*int* i = 0; i < n; ++i)

    {

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

    }

}

*void* print(*int* \**arr*, *int* *n*)//вывод на экран

{

    for (*int* i = 0; i < n; ++i)

    {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

*void* sort(*int*\* *arr*, *int* *semId*, *int* *memId*, const *size\_t* *n*)//сортировка

{

    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;

            sleep(1);

            sem(semId, i, 1);

            sem(semId, mInd, 1);

        }

    }

    exit(0);

}

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

{

*int* n = atoi(argc[1]);//преобразование

*int* mind = atoi(argc[2]);

*int* maxd = atoi(argc[3]);

*int* memId = shmget(IPC\_PRIVATE, sizeof(*int*) \* n, 0600 | IPC\_CREAT | IPC\_EXCL);//доступ к разд памяти

*int* semId = semget(IPC\_PRIVATE, n, 0600 | IPC\_CREAT);//получ идент сигмента

*int*\* arr = (*int*\*)shmat(memId, 0, 0);//адрес разделяемой памяти

    rand(arr, n, mind, maxd);//ранд числами заполняем

    printf("source: \r\n");

    print(arr, n);//вывод массива на экран

    printf("\n");

    for (*int* i = 0; i < n; ++i)

    {

        sem(semId, i, 1);

    }

*int* childId = fork();//порождение процесса

    if (childId == 0)//порожден

    {

        sort (arr, semId, memId, n); //сортировка по возрастанию

    }

    else

    {

*int* i = 0;

*int* status;

        do

        {

            printf("%d: ", i);

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

            {

                sem(semId, j, -1);

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

                fflush(stdout);  запись содержимого буфера в файл

                sem(semId, j, 1);

            }

            printf("\r\n");;

            status = waitpid(childId, NULL, WNOHANG); Проверка факта завершения дочернего процесса (без перевода текущего процесса в состояние ожидания)

            i++;

        } while(!status);

        printf("\nresult: \r\n");

        print(arr, n);//вывод массива на экран

        shmctl(memId, 0, IPC\_RMID);//удаление данных сигмента.IPC\_RMID

используется для пометки сегмента как удаленного.

        semctl(semId, 0, IPC\_RMID);//  производит операции управления семафорами 

    }

}