Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Новгородский Государственный Университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

Процессы в ОС UNIX

Лабораторная работа №6 по дисциплине

Операционные системы

Выполнили студенты гр. 8091

Кулаков Игорь Юрьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

Принял

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

### Великий Новгород

### 2020

# Лабораторная работа №6–«Семафоры в UNIX как средство синхронизации процессов»

## Задание на лабораторную работу

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.

**Код:**

#include <stdio.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <fcntl.h>

#include <algorithm>

#include <sys/stat.h>

#include <sys/sem.h>

void initSemValue(int semId, int n)

{

struct sembuf op;

op.sem\_op = 1;

op.sem\_flg = 0;

op.sem\_num = n;

semop(semId, &op, 1);

}

void tryToGetSemOrWait(int semId, int n)

{

struct sembuf op;

op.sem\_op = -1;

op.sem\_flg = 0;

op.sem\_num = n;

semop(semId, &op, 1);

}

void releaseSem(int semId, int n)

{

initSemValue(semId, n);

}

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

{

int N;

N= atoi(argv[1]);

int min = atoi(argv[2]);

int max = atoi(argv[3]);

int memId = shmget(IPC\_PRIVATE, 4\*N, 0600|IPC\_CREAT|IPC\_EXCL);

if (memId <= 0)

{

printf("error with shmget()\n");

return -1;

}

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

if (NULL == mem)

{

printf("error with shmat()\n");

return -2;

}

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

{

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

printf("%d\n",mem[i]);

}

const size\_t semCount = N;

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

if (semId < 0)

{

perror("error with semget()");

return -1;

}

else

{

printf("semaphore set id = %i\n", semId);

}

pid\_t childId = fork();

if (childId < 0)

{

perror("error with fork()\n");

}

else if (childId > 0)

{

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

{

printf("Parent: Freeing semaphore %i\n", i);

releaseSem(semId, i);

}

int status=0;

while (waitpid(-1, &status, 0|WNOHANG)==0)

{

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

{

tryToGetSemOrWait(semId, i);

printf("%d\n", mem[i]);

releaseSem(semId, i);

}

}

printf("Sort is end\n");

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

{

tryToGetSemOrWait(semId, i);

printf("%d\n", mem[i]);

releaseSem(semId, i);

}

shmctl(memId, IPC\_RMID, NULL);

semctl (semId, IPC\_RMID, 0);

}

else

{

int key =0;

int i=0;

for(int j=1;j<N;j++){

tryToGetSemOrWait(semId, j);

key = mem[j];

releaseSem(semId, j);

i=j-1;

while(i>=0 && mem[i]>key){

tryToGetSemOrWait(semId, i);

tryToGetSemOrWait(semId, i+1);

mem[i+1]=mem[i];

releaseSem(semId, i);

releaseSem(semId, i+1);

i=i-1;

tryToGetSemOrWait(semId, i+1);

mem[i+1]=key;

releaseSem(semId, i+1);

}

}

exit(0);

}

}