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»

Дисциплина «Операционные системы»

Отчет по лабораторной работе

«Синхронизация процессов с помощью семафоров в UNIX»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Алентьев Александр Григорьевич/

Подпись ФИО

Принял преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_/Ананьев Владислав Валерьевич/

Подпись ФИО

Великий Новгород

2021

**Цель лабораторной работ:**

Цель работы: познакомиться с механизмом синхронизации процессов в UNIX.

**Задание:**

1. Основной процесс должен сгенерировать N случайных числе и вывести их на экран.

Количество чисел, а также минимальное и максимальное допустимые значения должны быть получены из параметров командной строки.

1. Каждому элементу массива должен быть сопоставлен свой собственный семафор, предотвращающий одновременное обращение двух процессов к одним и тем же элементам.
2. После заполнения массива должен быть порожден второй процесс, который начнет производить сортировку чисел по возрастанию.
3. 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. Передача отсортированного массива обратно в главный процесс осуществить через PIPE.
4. 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.
5. Второй процесс в ходе сортировки также должен проверять и выставлять семафор перед обращением к каждому из элементов.
6. 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.

**Ход работы:**

**Исходный текст программы:**

|  |
| --- |
| main.cpp |
| #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"  #define SEMAPHORE\_UNLOCK 1  #define SEMAPHORE\_LOCK -1  void\* allocate\_shared\_memory(size\_t memSize, int\* memId)  {  \*memId = shmget(IPC\_PRIVATE, memSize, 0600 | IPC\_CREAT | IPC\_EXCL);  if (\*memId <= 0)  {  perror("error with memId");  return NULL;  }  void\* mem = shmat(\*memId, 0, 0);  if (NULL == mem)  perror("error with shmat");    return mem;  }  void fullArr(int \*array, int arrSize, int minValue, int maxValue)  {  srand(time(NULL));  for (int i = 0; i < arrSize; i++)  array[i] = minValue+ rand()%maxValue;  }  void printArr(int\* array, int size)  {  for (size\_t i = 0; i < size; i++)  printf("%i ",array[i]);  }  //------------------------SemaphoreStart-----------------------------------  void semaphoreSetState(int semId, int num, int state)  {  struct sembuf op;  op.sem\_op = state;  op.sem\_flg = 0;  op.sem\_num = num;  semop(semId, &op, 1);  }  char semaphore\_lock(int semId, int num, char\* arrayCheck)  {  if(arrayCheck[num])  return 1;  semaphoreSetState(semId, num, SEMAPHORE\_LOCK);  arrayCheck[num] = 1;  return 0;  }  void semaphore\_unlock(int semId, int num, char\* arrayCheck)  {  semaphoreSetState(semId, num, SEMAPHORE\_UNLOCK);  arrayCheck[num] = 0;  }  //------------------------SemaphoreEnd----------------------------------------  void childMainCode(int\* array, char\* arrayCheck, int size, int semId)  {  int temp;  for (int i = 0; i < size - 1; i++) {  for (int j = 0; j < size - i - 1; j++) {  semaphore\_lock(semId, i, arrayCheck);  semaphore\_lock(semId, i+1, arrayCheck);  if (array[j] > array[j + 1]) {  temp = array[j];  array[j] = array[j + 1];  array[j + 1] = temp;  }  semaphore\_unlock(semId, i, arrayCheck);  semaphore\_unlock(semId, i+1, arrayCheck);  }  }    exit(0);  }  void parentMainCode(int\* array, char\* arrayCheck, int size, int semId, pid\_t child)  {  int iteration = 0;  while (!waitpid(child, NULL, WNOHANG))  {  printf("\nIteration %i\n\n", iteration);  for (int i = 0; i < size; i++)  {  if (semaphore\_lock(semId, i, arrayCheck))  printf("Bloke\t");  else  printf("%d\t", array[i]);  semaphore\_unlock(semId, i, arrayCheck);  }  printf("\n");  iteration++;  }  printf("Result: %i\n", iteration);  printArr(array, size);  }  int main(int argv, char\* argc[])  {  if (argv <= 3)  {  printf("Error! Not enough arguments! Required: 3 (array\_size, min, max)\n");  return -1;  }  int arrSize = atoi(argc[1]);  int minValue = atoi(argc[2]);  int maxValue = atoi(argc[3]);  int memId;  int\* array = allocate\_shared\_memory(sizeof(int) \* arrSize, &memId);  fullArr(array, arrSize, minValue, maxValue);  int semId;  if(semId = semget(IPC\_PRIVATE, arrSize, 0600 | IPC\_CREAT)<0){  perror("Error with semget()!\n");  return -1;  }  printf("Semaphore set id = %i\n", semId);  int checkId;  char\* arrayCheck = allocate\_shared\_memory(sizeof(char) \* arrSize, &checkId);  for (size\_t i = 0; i < arrSize; i++)  {  semaphoreSetState(semId, i, SEMAPHORE\_UNLOCK);  }  pid\_t childProcess = fork();  if(childProcess < 0){  perror("Error with fork() - process 1\n");  }  else if(childProcess > 0){  parentMainCode(array, arrayCheck, arrSize, semId, childProcess);  }  else{  childMainCode(array, arrayCheck, arrSize, semId);  }    char deleteCommand[124];  sprintf(deleteCommand, "ipcrm -m %i", memId);  system(deleteCommand);  sprintf(deleteCommand, "ipcrm -m %i", checkId);  system(deleteCommand);  sprintf(deleteCommand, "ipcrm -s %i", semId);  system(deleteCommand);  } |

|  |
| --- |
| Output Terminal |
| >> make  gcc main.c -o main  ./main 10 1 99  Semaphore set id = 0  Iteration 0  48 29 60 58 5 69 86 92 70 54  Iteration 1  29 Bloke 5 58 60 69 70 54 86 92  Iteration 2  5 29 48 54 58 60 Bloke 70 Bloke 92  Iteration 3  5 29 48 54 58 60 69 70 86 92  Iteration 4  5 29 48 54 58 60 69 70 86 92  Iteration 5  5 29 48 54 58 60 69 70 86 92  Iteration 6  5 29 48 54 58 60 69 70 86 92  Result: 7  5 29 48 54 58 60 69 70 86 92 |

**Вывод:**

Входе выполнения лабораторной работы, я познакомился с механизмом синхронизации процессов при помощи семафоров в операционной системе UNIX.