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

«Семафоры в UNIX как средство синхронизации процессов»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Федотов Дмитрий Борисович/

Подпись ФИО

Принял преподаватель

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

Подпись ФИО

Великий Новгород

2021

**Цель лабораторной работы**

Цель работы: познакомиться с механизмом синхронизации процессов в UNIX.

**Исходный текст программы**

|  |
| --- |
| main.c |
| #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 array\_fill\_random\_value(int\* array, int size, int min, int max)  {  for (int i = 0; i < size; i++)  array[i] = min + rand() % max;  }  void\* allocate\_shared\_memory(size\_t mem\_size, int\* mem\_id)  {  \*mem\_id = shmget(IPC\_PRIVATE, mem\_size, 0600 | IPC\_CREAT | IPC\_EXCL);  if (\*mem\_id <= 0)  {  perror("error with memId");  return NULL;  }  void\* mem = shmat(\*mem\_id, 0, 0);  if (NULL == mem)  perror("error with shmat");    return mem;  }  void semaphore\_set\_state(int sem\_id, int num, int state)  {  struct sembuf op;  op.sem\_op = state;  op.sem\_flg = 0;  op.sem\_num = num;  semop(sem\_id, &op, 1);  }  char semaphore\_lock(int sem\_id, int num, char\* array\_check\_is\_lock\_ptr)  {  char is\_lock = array\_check\_is\_lock\_ptr[num];  semaphore\_set\_state(sem\_id, num, SEMAPHORE\_LOCK);  array\_check\_is\_lock\_ptr[num] = 1;  return is\_lock;  }  void semaphore\_unlock(int sem\_id, int num, char\* array\_check\_is\_lock\_ptr)  {  semaphore\_set\_state(sem\_id, num, SEMAPHORE\_UNLOCK);  array\_check\_is\_lock\_ptr[num] = 0;  }  void swap\_values(int\* first, int\* second)  {  int temp = \*first;  \*first = \*second;  \*second = temp;  }  void child\_main\_code(int\* array, char\* array\_check\_ptr, int array\_size, int sem\_id)  {  double factor = 1.2473309;  int step = array\_size - 1;  while (step >= 1)  {  for (int i = 0; i + step < array\_size; i++)  {  semaphore\_lock(sem\_id, i, array\_check\_ptr);  semaphore\_lock(sem\_id, i + step, array\_check\_ptr);  if (array[i] > array[i + step])  swap\_values(&array[i], &array[i + step]);  semaphore\_unlock(sem\_id, i + step, array\_check\_ptr);  semaphore\_unlock(sem\_id, i, array\_check\_ptr);  }  step /= factor;  }  exit(0);  }  void parent\_main\_code(int\* array, char\* array\_check\_ptr, int array\_size, int sem\_id, pid\_t child\_id)  {  int iteration = 0;  while (!waitpid(child\_id, NULL, WNOHANG))  {  printf("--- This is iteration %i ---\n", iteration);  for (int i = 0; i < array\_size; i++)  {  if (semaphore\_lock(sem\_id, i, array\_check\_ptr))  printf("[%d] ", array[i]);  else  printf("%d ", array[i]);  semaphore\_unlock(sem\_id, i, array\_check\_ptr);  }  printf("\n");  iteration++;  }  printf("============== RESULT ==============\n");  printf("Iteration count: %i\n", iteration);  printf("====================================\n");  }  void free\_shared\_memory(int\* mem\_id)  {  char resource\_delete\_command\_buff[124];  sprintf(resource\_delete\_command\_buff, "ipcrm -m %i", \*mem\_id);  system(resource\_delete\_command\_buff);  \*mem\_id = 0;  }  void free\_semaphores(int\* sem\_id)  {  char resource\_delete\_command\_buff[124];  sprintf(resource\_delete\_command\_buff, "ipcrm -s %i", \*sem\_id);  system(resource\_delete\_command\_buff);  \*sem\_id = 0;  }  int main(int argv, char\* argc[])  {  if (argv <= 3)  {  printf("Error! Not enough arguments! Required: 3 (array\_size, min, max)\n");  return -1;  }    /\* --- array mem init --- \*/    int array\_size = atoi(argc[1]);  int array\_min\_value = atoi(argc[2]);  int array\_max\_value = atoi(argc[3]);  int mem\_id;  int\* array\_ptr = allocate\_shared\_memory(sizeof(int) \* array\_size, &mem\_id);  array\_fill\_random\_value(array\_ptr, array\_size, array\_min\_value, array\_max\_value);    /\* --- semaphore init --- \*/  int sem\_id = semget(IPC\_PRIVATE, array\_size, 0600 | IPC\_CREAT);  int check\_mem\_id;  char\* array\_check\_is\_lock = allocate\_shared\_memory(sizeof(char) \* array\_size, &check\_mem\_id);  if (sem\_id < 0)  {  perror("Error with semget()!\n");  return -1;  }  printf("Semaphore set id = %i\n", sem\_id);  for (int i = 0; i < array\_size; i++)  semaphore\_set\_state(sem\_id, i, SEMAPHORE\_UNLOCK);  /\* --- lab --- \*/  pid\_t child\_process\_id = fork();  if (child\_process\_id == -1)  perror("Error with fork() - process 1\n");  else if (child\_process\_id == 0)  child\_main\_code(array\_ptr, array\_check\_is\_lock, array\_size, sem\_id);  else  parent\_main\_code(array\_ptr, array\_check\_is\_lock, array\_size, sem\_id, child\_process\_id);  free\_semaphores(&sem\_id);  free\_shared\_memory(&mem\_id);  free\_shared\_memory(&check\_mem\_id);  return 0;  } |

**Результат выполнения программы**

|  |
| --- |
| Вывод в терминале |
| >> make  gcc main.c -o main  ./main 20 -50 100  Semaphore set id = 40  --- This is iteration 0 ---  [-14] 36 27 -35 43 -15 36 42 -1 -29 12 -23 40 9 13 -14 36 27 22 43  --- This is iteration 1 ---  -29 -10 -24 -35 9 [-15] -14 36 -1 -24 [12] -23 40 33 13 22 42 27 36 43  --- This is iteration 2 ---  -29 -10 -24 -35 -23 [-15] -10 13 -1 -23 12 9 40 33 36 22 42 40 36 43  --- This is iteration 3 ---  -29 -15 -24 -35 -24 -23 -10 9 -1 [-14] [12] 13 27 33 [36] [22] 42 40 36 43  --- This is iteration 4 ---  -35 -24 -24 -29 -15 -23 -14 9 -1 -10 12 13 22 33 36 27 42 40 36 43  --- This is iteration 5 ---  -35 -29 -24 -24 -15 -23 -14 -10 -1 9 12 13 22 [27] 36 33 36 40 42 43  --- This is iteration 6 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 [22] 27 33 36 36 [40] 42 43  --- This is iteration 7 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  --- This is iteration 8 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  --- This is iteration 9 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  ============== RESULT ==============  Iteration count: 10  ==================================== |

**Вывод**

Вывод: выполняя лабораторную работу, я познакомился с механизмом синхронизации процессов в UNIX.