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

\_\_\_\_\_\_\_\_\_\_\_\_\_/Лопатин Даниил Михайлович/

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**

|  |
| --- |
| lab6.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"  #include <sys/ipc.h>  #define UNLOCK 1  #define LOCK -1  void arr\_rand\_fill(int\* arr, int size, int min, int max)  {  for (int i = 0; i < size; i++)  arr[i] = min + rand() % max;  }  void\* alloc\_shrd\_mem(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 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 set\_state\_nowait(int sem\_id, int num, int state)  {  struct sembuf op;  op.sem\_op = state;  op.sem\_flg = IPC\_NOWAIT;  op.sem\_num = num;  return semop(sem\_id, &op, 1);  }  void swap\_vals(int\* first, int\* second)  {  int temp = \*first;  \*first = \*second;  \*second = temp;  }  void parent\_code(int\* arr, int arr\_size, int sem\_id, pid\_t child\_id)  {  int count = 0;  while (!waitpid(child\_id, NULL, WNOHANG))  {  printf("Count %i\n", count);  for (int i = 0; i < arr\_size; i++)  {  if (set\_state\_nowait(sem\_id, i, LOCK) == -1)  printf("block ");  else  {  printf("%d ", arr[i]);  set\_state(sem\_id, i, UNLOCK);  }  }  printf("\n");  count++;  }  printf("Iteration is: %i\n", count);  }  void child\_code(int\* arr, int arr\_size, int sem\_id)  {  double factor = 1.2473309;  int step = arr\_size - 1;  while (step >= 1)  {  for (int i = 0; i + step < arr\_size; i++)  {  set\_state(sem\_id, i, LOCK);  set\_state(sem\_id, i + step, LOCK);  if (arr[i] > arr[i + step])  swap\_vals(&arr[i], &arr[i + step]);  set\_state(sem\_id, i + step, UNLOCK);  set\_state(sem\_id, i, UNLOCK);  }  step /= factor;  }  exit(0);  }  void free\_shrd\_memory(int\* mem\_id);  void free\_sems(int\* sem\_id);  int main(int argv, char\* argc[])  {  if (argv <= 3)  {  printf("Error! Not enough arguments! Required: 3 (arr\_size, min, max)\n");  return -1;  }    int arr\_size = atoi(argc[1]);  int arr\_min = atoi(argc[2]);  int arr\_max = atoi(argc[3]);  int mem\_id;  int\* arr\_ptr = alloc\_shrd\_mem(sizeof(int) \* arr\_size, &mem\_id);  arr\_rand\_fill(arr\_ptr, arr\_size, arr\_min, arr\_max);  int sem\_id = semget(IPC\_PRIVATE, arr\_size, 0600 | IPC\_CREAT);  if (sem\_id < 0)  {  perror("Error with semget()!\n");  return -1;  }  printf("Semaphore set id = %i\n", sem\_id);  for (int i = 0; i < arr\_size; i++)  set\_state(sem\_id, i, UNLOCK);  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\_code(arr\_ptr, arr\_size, sem\_id);  else  parent\_code(arr\_ptr, arr\_size, sem\_id, child\_process\_id);  free\_sems(&sem\_id);  free\_shrd\_mem(&mem\_id);  return 0;  }  void free\_shrd\_mem(int\* mem\_id)  {  char del\_cmd\_temp[124];  sprintf(del\_cmd\_temp, "ipcrm -m %i", \*mem\_id);  system(del\_cmd\_temp);  \*mem\_id = 0;  }  void free\_sems(int\* sem\_id)  {  char del\_cmd\_temp[124];  sprintf(del\_cmd\_temp, "ipcrm -s %i", \*sem\_id);  system(del\_cmd\_temp);  \*sem\_id = 0;  } |

**Результат выполнения программы**

|  |
| --- |
| Вывод в терминале |
| >> make  gcc lab5.c -o lab6  ./lab6 20 -50 100  Semaphore set id = 14  --- This is iteration 0 ---  33 36 27 -35 43 -15 36 42 -1 -29 12 -23 40 9 13 -24 -10 -24 22 33  --- This is iteration 1 ---  -24 -10 -24 -35 block -15 36 42 -1 -29 12 -23 40 9 13 -14 36 27 36 block  --- This is iteration 2 ---  -29 -10 block -35 33 -15 22 42 -1 -24 12 -23 40 33 13 22 42 27 36 43  --- This is iteration 3 ---  -29 -10 -24 -35 9 -15 -14 36 -1 -24 12 -23 40 33 13 22 42 27 36 43  --- This is iteration 4 ---  -29 -10 -24 -35 -23 -15 -14 13 -1 -24 block 9 40 33 36 22 42 27 36 43  --- This is iteration 5 ---  -29 -14 block -35 -23 -15 -10 13 -1 -23 12 block 40 33 36 22 42 27 36 43  --- This is iteration 6 ---  -29 -14 -24 -35 -24 -15 -10 13 -1 -23 12 9 27 33 36 22 42 40 36 43  --- This is iteration 7 ---  -29 -15 -24 -35 -24 -23 -10 9 -1 -14 12 13 27 block 36 22 42 40 36 43  --- This is iteration 8 ---  -29 -15 -24 -35 -24 -23 -10 9 -1 -14 12 13 27 33 36 22 42 40 36 43  --- This is iteration 9 ---  -35 -24 -24 -29 -15 -23 -14 9 -1 -10 12 13 22 33 36 block 42 40 36 43  --- This is iteration 10 ---  -35 -29 block -24 -15 block -14 9 -1 -10 12 13 22 33 36 27 42 40 36 43  --- This is iteration 11 ---  -35 -29 -24 -24 -15 -23 -14 -10 -1 9 12 13 22 27 36 33 36 40 42 43  --- This is iteration 12 ---  -35 -29 -24 -24 -23 -15 block -10 -1 9 12 13 22 27 36 33 36 40 42 43  --- This is iteration 13 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  --- This is iteration 14 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  --- This is iteration 15 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  --- This is iteration 16 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  --- This is iteration 17 ---  -35 -29 -24 -24 -23 -15 -14 -10 -1 9 12 13 22 27 33 36 36 40 42 43  ============== RESULT ==============  Iteration count: 18  ==================================== |

**Вывод**

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