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

\_\_\_\_\_\_\_\_\_\_\_\_\_/Флейшер Освальд Кевин/

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\* allocateSharedMemory(size\_t memorySize, int\* memID)  {  \*memID = shmget(IPC\_PRIVATE, memorySize, 0600 | IPC\_CREAT | IPC\_EXCL);  if(\*memID <= 0)  {  perror("Error with shmget");  return NULL;  }  void\* memory = shmat(\*memID, 0, 0);  if(NULL == memory)  {  perror("Error with shmat");  }  return memory;  }  void fillArray(int \*array, int arraySize, int minValue, int maxValue)  {  srand(time(NULL));  for(int i = 0; i < arraySize; i++)  {  array[i] = minValue+ rand()%maxValue;  }  }  void printArray( int\* array, int size)  {  for(size\_t i = 0; i < size; i++)  {  printf("%i ", array[i]);  }  }  void semaphoreSetStatus(int semID, int num, int status)  {  struct sembuf op;  op.sem\_op = status;  op.sem\_flg = 0;  op.sem\_num = num;  semop(semID, &op, 1);  }  char semaphoreLock(int semID, int num, char\* semaphoreStatusArray)  {  semaphoreSetStatus(semID, num, SEMAPHORE\_LOCK);  semaphoreStatusArray[num] = 1;  }  char semaphoreLockParent(int semID, int num, char\* semaphoreStatusArray)  {  if(semaphoreStatusArray[num])  {  return -1;  }  semaphoreSetStatus(semID, num, SEMAPHORE\_LOCK);  semaphoreStatusArray[num] = 1;  return 0;  }  void semaphoreUnlock(int semID, int num, char\* semaphoreStatusArray)  {  semaphoreSetStatus(semID, num, SEMAPHORE\_UNLOCK);  semaphoreStatusArray[num] = 0;  }  void childCode(int\* array, char\* semaphoreStatusArray, int size, int semID)  {  int temp;  for(int i = 0; i < size - 1; i++)  {  for(int j = 0; j < size - i - 1; j++)  {  semaphoreLock(semID, j, semaphoreStatusArray);  semaphoreLock(semID, j + 1, semaphoreStatusArray);  if(array[j] > array[j + 1])  {  temp = array[j];  array[j] = array[j + 1];  array[j + 1] = temp;  }  semaphoreUnlock(semID, j, semaphoreStatusArray);  semaphoreUnlock(semID, j + 1, semaphoreStatusArray);  }  }  exit(0);  }  void parentCode(int\* array, char\* semaphoreStatusArray, int size, int semID, pid\_t child)  {  int iteration = 0;  while(!waitpid(child, NULL, WNOHANG))  {  printf("\nIteration %i\n", iteration);  for(int i = 0; i < size; i++)  {  if(semaphoreLockParent(semID, i, semaphoreStatusArray))  {  printf("Blocked\t");  }  else  {  printf("%d\t", array[i]);  }  semaphoreUnlock(semID, i, semaphoreStatusArray);  }  printf("\n");  iteration++;  }  printf("Result: %i\n", iteration);  printArray(array, size);  }  int main(int argv, char\* argc[])  {  if( argv <= 3)  {  printf("Error! Not enough params! Params required: array size, min, max");  return -1;  }  int arraySize = atoi(argc[1]);  int minValue = atoi(argc[2]);  int maxValue = atoi(argc[3]);  int memID;  int\* array = allocateSharedMemory(sizeof(int) \* arraySize, &memID);  fillArray(array, arraySize, minValue, maxValue);  int semID;  if(semID = semget(IPC\_PRIVATE, arraySize, 0600 | IPC\_CREAT)< 0)  {  perror("Error with semget()!\n");  return -1;  }  printf("Semaphore set ID = %i\n", semID);  int semaphoreStatusID;  char\* semaphoreStatusArray = allocateSharedMemory(sizeof(int) \* arraySize, &semaphoreStatusID);  for(size\_t i = 0; i < arraySize; i++)  {  semaphoreSetStatus(semID, i, SEMAPHORE\_UNLOCK);  }  pid\_t childProcess = fork();  if(childProcess < 0)  {  perror("Error with fork - process 1\n");  }  else if(childProcess > 0)  {  parentCode(array, semaphoreStatusArray, arraySize, semID, childProcess);  }  else  {  childCode(array, semaphoreStatusArray, arraySize, semID);  }  char freeMemoryCommand[124];  sprintf(freeMemoryCommand, "ipcrm -m %i", memID);  system(freeMemoryCommand);  sprintf(freeMemoryCommand, "ipcrm -m %i", semaphoreStatusID);  system(freeMemoryCommand);  sprintf(freeMemoryCommand, "ipcrm -s %i", semID);  system(freeMemoryCommand);  return 0;  } |

**Результат выполнения программы**

|  |
| --- |
| Вывод в терминале |
| >> make  gcc main.c -o main  ./main 10 1 99  Semaphore set ID = 0  Iteration 0  22 43 20 30 31 45 51 37 29 64  Iteration 1  22 43 20 30 31 45 51 37 29 64  Iteration 2  22 43 20 30 31 45 51 37 29 64  Iteration 3  22 20 30 31 43 45 37 29 Blocked Blocked  Iteration 4  20 22 30 31 43 37 29 45 51 64  Iteration 5  20 22 30 31 Blocked Blocked 43 45 51 64  Iteration 6  Blocked Blocked 30 29 31 37 43 45 51 64  Iteration 7  20 22 29 30 31 37 43 45 51 64  Iteration 8  20 22 29 30 31 37 43 45 51 64  Iteration 9  20 22 29 30 31 37 43 45 51 64  Iteration 10  20 22 29 30 31 37 43 45 51 64  Result: 11  20 22 29 30 31 37 43 45 51 64 |

**Вывод**

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