|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Отчёт по лабораторной работе №5 по курсу**

**«Операционные системы»**

**Тема**  Взаимодействие параллельных процессов

**Студент**  Якуба Д.В.

**Группа**  ИУ7-53Б

**Оценка (баллы)**

**Преподаватель**  Рязанова Н.Ю.

*Москва, 2020 г*

# Задание 1.

Написать программу, реализующую задачу «Производство-потребление» по алгоритму Э. Дейкстры с тремя семафорами: двумя считающими и одним бинарным. В программе должно создаваться не менее 3х процессов -производителей и 3х процессов – потребителей. В программе надо обеспечить случайные задержки выполнения созданных процессов. В программе для взаимодействия производителей и потребителей буфер создается в разделяемом сегменте. Обратите внимание на то, чтобы не работать с одиночной переменной, а работать именно с буфером, состоящим их N ячеек по алгоритму. Производители в ячейки буфера записывают буквы алфавита по порядку. Потребители считывают символы из доступной ячейки. После считывания буквы из ячейки следующий потребитель может взять букву из следующей ячейки.

Листинг кода 1

|  |
| --- |
| #include <stdio.h>  #include "stdlib.h"  #include "time.h"  #include "sys/stat.h"  #include <sys/ipc.h>  #include <sys/sem.h>  #include <sys/shm.h>  #include <sys/wait.h>  #include <unistd.h>  #define SEM\_AMOUNT 3  #define SEM\_BIN 0  #define SEM\_E 1  #define SEM\_F 2  #define EMPTY\_NUM 20  #define PRODUCE\_NUM 6  #define CONSUME\_NUM 6  #define CYCLES 3  #define SEM\_ERROR 1  #define SEM\_SET\_ERR 2  #define SHM\_ERROR 3  #define MEM\_ERR 4  #define FORK\_ERR 5  #define SEMOP\_ERR 6  int \*sharedMemoryPtr = NULL;  char \*sharedCharMemoryPtr = NULL;  char \*alphabet = "abcdefghijklmnopqrstuvwxyz";  struct sembuf prodStart[2] =  {  {SEM\_E, -1, 1},  {SEM\_BIN, -1, 1}  };  struct sembuf prodEnd[2] =  {  {SEM\_BIN, 1, 1},  {SEM\_F, 1, 1}  };  struct sembuf readStart[2] =  {  {SEM\_F, -1, 1},  {SEM\_BIN, -1, 1}  };  struct sembuf readEnd[2] =  {  {SEM\_BIN, 1, 1},  {SEM\_E, 1, 1}  };  void **producer**(int semID, int prodID)  {  srand(time(NULL));  sleep(rand() % 4);  if (semop(semID, prodStart, 2) == -1)  {  perror("Producer semop error");  exit(SEMOP\_ERR);  }  sharedCharMemoryPtr[sharedMemoryPtr[0]] = alphabet[sharedMemoryPtr[0]];  printf("<<---Producer[ID = %d]: wrote %c\n", prodID, sharedCharMemoryPtr[sharedMemoryPtr[0]]);  sharedMemoryPtr[0]++;  if (semop(semID, prodEnd, 2) == -1)  {  perror("Producer semop error");  exit(SEMOP\_ERR);  }  }  void **consumer**(int semID, int consID)  {  srand(time(NULL));  sleep(rand() % 4);  if (semop(semID, readStart, 2) == -1)  {  perror("Consumer semop error");  exit(SEMOP\_ERR);  }  printf("->>Consumer[ID = %d]: read %c\n", consID, sharedCharMemoryPtr[sharedMemoryPtr[1]]);  sharedMemoryPtr[1]++;  if (semop(semID, readEnd, 2) == -1)  {  perror("Consumer semop error");  exit(SEMOP\_ERR);  }  }  int **main**()  {  int semID = semget(IPC\_PRIVATE, SEM\_AMOUNT, IPC\_CREAT | S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  if (semID == -1)  {  perror("Semaphore creation error.");  exit(SEM\_ERROR);  }  if (semctl(semID, SEM\_BIN, SETVAL, 1) == -1 ||  semctl(semID, SEM\_E, SETVAL, EMPTY\_NUM) == -1 ||  semctl(semID, SEM\_F, SETVAL, 0) == -1)  {  perror("Semaphore set error.");  exit(SEM\_SET\_ERR);  }  int shmID = shmget(IPC\_PRIVATE, 2 \* sizeof(int) + EMPTY\_NUM \* sizeof(char), S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  if (shmID == -1)  {  perror("Shared memory creation error.");  exit(SHM\_ERROR);  }  sharedMemoryPtr = shmat(shmID, 0, 0);  if (\*sharedMemoryPtr == -1)  {  perror("Memory all error.");  exit(MEM\_ERR);  }  sharedCharMemoryPtr = (char \*)(sharedMemoryPtr + 2 \* sizeof(int));  pid\_t childID = -1;  for (int i = 0; i < CYCLES; ++i)  {  if ((childID = fork()) == -1)  {  perror("Producer fork error");  exit(FORK\_ERR);  }  else if (childID == 0)  {  for (int j = 0; j < PRODUCE\_NUM; j++)  producer(semID, i);  exit(0);  }  if ((childID = fork()) == -1)  {  perror("Consumer fork error");  exit(FORK\_ERR);  }  else if (childID == 0)  {  for (int j = 0; j < CONSUME\_NUM; j++)  consumer(semID, i);  exit(0);  }  }  int status;  for (int i = 0; i < CONSUME\_NUM + PRODUCE\_NUM; i++)  wait(&status);  if (shmdt(sharedMemoryPtr) == -1)  {  perror("SHMDT error");  exit(MEM\_ERR);  }  if (shmctl(shmID, IPC\_RMID, NULL) == -1)  {  perror("SHMCTL error");  exit(MEM\_ERR);  }  sharedCharMemoryPtr = NULL;  printf("\n\nThis is the end of the task\n");  exit(0);  } |

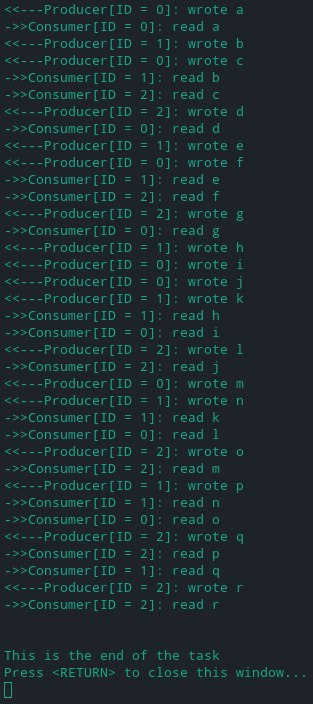


Рисунок 1, Демонстрация работы программы (задержки производителей от 0 до 4, задержки потребителей от 0 до 4)

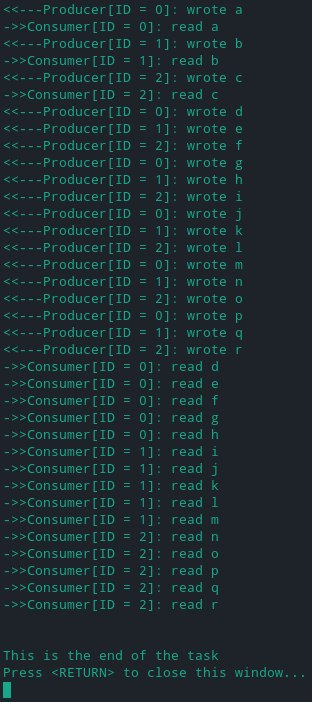


Рисунок 2, Демонстрация работы программы (задержки производителей от 0 до 3, задержки потребителей от 0 до 4)

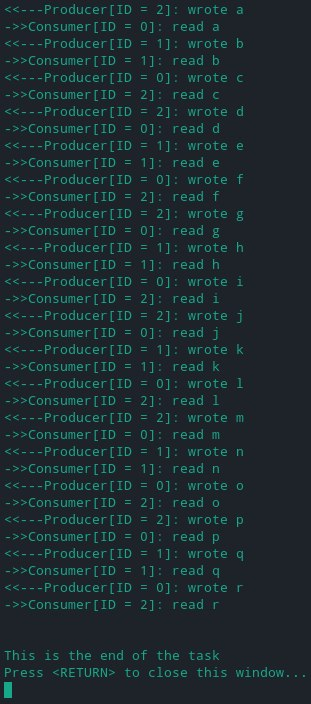


Рисунок 3, Демонстрация работы программы (задержки производителей от 0 до 4, задержки потребителей от 0 до 2)

# Задание 2.

Написать программу, реализующую задачу «Читатели – писатели» по монитору Хоара с четырьмя функциями: Начать\_чтение, Закончить\_чтение, Начать\_запись, Закончить\_запись. В программе всеми процессами разделяется одно единственное значение в разделяемой памяти. Писатели ее только инкрементируют, читатели могут только читать значение.

Листинг кода 2

|  |
| --- |
| #include <stdio.h>  #include "stdlib.h"  #include "time.h"  #include "sys/stat.h"  #include <sys/ipc.h>  #include <sys/sem.h>  #include<sys/shm.h>  #include <sys/wait.h>  #include <unistd.h>  #define SEM\_AMOUNT 4  #define WRITERS\_AMOUNT 3  #define READERS\_AMOUNT 5  #define ACTIVE\_READERS 0  #define ACTIVE\_WRITERS 1  #define WAIT\_READERS 2  #define WAIT\_WRITERS 3  #define SEM\_ERROR 1  #define SEM\_SET\_ERR 2  #define SHM\_ERR 3  #define MEM\_ERR 4  #define FORK\_ERR 5  #define SEMOP\_ERR 6  struct sembuf startWrite[] =  {  { WAIT\_WRITERS, 1, 0 },  { ACTIVE\_READERS, 0, 0 },  { ACTIVE\_WRITERS, 0, 0 },  { ACTIVE\_WRITERS, 1, 0 },  { WAIT\_WRITERS, -1, 0 }  };  struct sembuf stopWrite[] =  {  { ACTIVE\_WRITERS, -1, 0 }  };  struct sembuf startRead[] =  {  { WAIT\_READERS, 1, 0 },  { ACTIVE\_WRITERS, 0, 0 },  { WAIT\_WRITERS, 0, 0 },  { ACTIVE\_READERS, 1, 0 },  { WAIT\_READERS, -1, 0 }  };  struct sembuf stopRead[] =  {  { ACTIVE\_READERS, -1, 0 }  };  int \*sharedMemoryPtr = NULL;  void **writer**(int semID, int writerID)  {  srand(time(NULL));  if (semop(semID, startWrite, 5) == -1)  {  perror("Semop error");  exit(SEMOP\_ERR);  }  (\*sharedMemoryPtr)++;  printf("<<---Writer[ID = %d]: write value %d\n", writerID, \*sharedMemoryPtr);  if (semop(semID, stopWrite, 1) == -1)  {  perror("Writer semop error");  exit(SEMOP\_ERR);  }  sleep(rand() % 3 + 1);  }  void **reader**(int semID, int readerID)  {  srand(time(NULL));  if (semop(semID, startRead, 5) == -1)  {  perror("Semop error");  exit(SEMOP\_ERR);  }  printf("->>Reader[ID = %d]: reads value %d\n", readerID, \*sharedMemoryPtr);  if (semop(semID, stopRead, 1) == -1)  {  perror("Writer semop error");  exit(SEMOP\_ERR);  }  sleep(rand() % 3 + 1);  }  int **main**()  {  int semID = semget(IPC\_PRIVATE, SEM\_AMOUNT, IPC\_CREAT | S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  if (semID == -1)  {  perror("Semaphore creation error.");  exit(SEM\_ERROR);  }  if (semctl(semID, 2, SETVAL, 1) == -1)  {  perror("Semaphore set error.");  exit(SEM\_SET\_ERR);  }  int shmID = shmget(IPC\_PRIVATE, sizeof(int), S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);  if (shmID == -1)  {  perror("Shared memory creation error.");  exit(SHM\_ERR);  }  sharedMemoryPtr = shmat(shmID, 0, 0);  if (\*sharedMemoryPtr == -1)  {  perror("Memory all error.");  exit(MEM\_ERR);  }  pid\_t childID = -1;  for (int i = 0; i < WRITERS\_AMOUNT; i++)  {  if ((childID = fork()) == -1)  {  perror("Write fork error");  exit(FORK\_ERR);  }  else if (childID == 0)  {  for (;;)  writer(semID, i);  exit(0);  }  }  for (int i = 0; i < READERS\_AMOUNT; i++)  {  if ((childID = fork()) == -1)  {  perror("Reader fork error");  exit(FORK\_ERR);  }  else if (childID == 0)  {  for (;;)  reader(semID, i);  exit(0);  }  }  int status;  for (int i = 0; i < WRITERS\_AMOUNT + READERS\_AMOUNT; i++)  wait(&status);  if (shmdt(sharedMemoryPtr) == -1)  {  perror("SHMDT error");  exit(MEM\_ERR);  }  if (shmctl(shmID, IPC\_RMID, NULL) == -1)  {  perror("SHMCTL error");  exit(MEM\_ERR);  }  } |

Пример работы программы предоставлен на следующей странице (Рисунок 4).

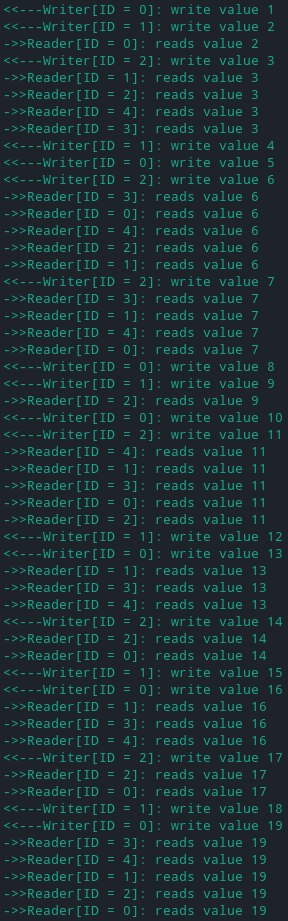


Рисунок 4, Демонстрация работы программы (задержки писателей и читателей от 1 до 3)