|  |  |
| --- | --- |
|  | **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»

# Отчет

**Лабораторная работа № 5**

|  |  |
| --- | --- |
| Тема: Взаимодействие параллельных процессов. **Студент** ВоякинА. Я.  **Группа ИУ7-54Б**  **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 программы:**

#**include** <sys/shm.h>  
#**include** <sys/sem.h>  
#**include** <fcntl.h>  
#**include** <unistd.h>  
#**include** <sys/wait.h>  
#**include** <stdio.h>  
#**include** <time.h>  
#**include** <stdlib.h>  
#**include** <signal.h>  
  
#**define** SE 0  
#**define** SF 1  
#**define** SB 2  
  
**const** **int** buffer\_size = 10;  
**const** **int** permissions = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH;  
**const** **int** consumer\_num = 3;  
**const** **int** producer\_num = 3;  
  
**struct** **sembuf** **semops\_produce\_begin**[2] = {  
 {SE, -1, 0},  
 {SB, -1, 0}  
};  
  
**struct** **sembuf** **semops\_produce\_end**[2] = {  
 {SB, 1, 0},  
 {SF, 1, 0}  
};  
  
**struct** **sembuf** **semops\_consume\_begin**[2] = {  
 {SF, -1, 0},  
 {SB, -1, 0}  
};  
  
**struct** **sembuf** **semops\_consume\_end**[2] = {  
 {SB, 1, 0},  
 {SE, 1, 0}  
};  
  
**int**\*\* shm\_cons\_pos;  
**int**\*\* shm\_prod\_pos;  
**int**\* shm\_buffer;  
  
**pid\_t**\* pid\_arr;  
  
**void** **kill\_producers**()   
{  
 **for** (**int** i = 0; i < producer\_num; i++) {  
 **if** (pid\_arr[i] == getpid()) {  
 **continue**;  
 }  
 kill(pid\_arr[i], SIGTERM);  
 }  
 exit(0);  
}  
  
**void** **kill\_consumers**()   
{  
 **for** (**int** i = producer\_num; i < producer\_num + consumer\_num; i++) {  
 **if** (pid\_arr[i] == getpid()) {  
 **continue**;  
 }  
 kill(pid\_arr[i], SIGTERM);  
 }  
 kill(getpid(), SIGKILL);  
}  
  
**int**\* **get\_shm**()  
{  
 **int** shm\_id;  
 **int**\* shm;  
  
 shm\_id = shmget(100, **sizeof**(**char**) \* (buffer\_size + 1) + (producer\_num + consumer\_num) \* **sizeof**(**pid\_t**) + **sizeof**(**int**\*) \* 2, IPC\_CREAT|permissions);  
   
 **if** (shm\_id == -1)  
 {  
 perror("Shmget failed.\n");  
 exit(1);  
 }  
  
 shm = (**int**\*) shmat(shm\_id, NULL, 0);  
 **if** (\*shm == -1)  
 {  
 perror("Shmat failed.\n");  
 exit(2);  
 }  
  
 **return** shm;  
}  
  
**int** **get\_sem\_fd**()  
{  
 **int** sem\_fd = semget(101, 3, IPC\_CREAT | permissions);  
 **if** (sem\_fd == -1)  
 {  
 perror("Semget failed.\n");  
 exit(3);  
 }  
 **return** sem\_fd;  
}  
  
**int** **produce**(**int** sem\_fd, **int** number)  
{  
 srand(time(NULL));  
 sleep(rand() % 5);  
   
 **if** (semop(sem\_fd, semops\_produce\_begin, 2) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(4);  
 }  
 **int** \*\*shm\_prod\_pos = shm\_buffer - 3;  
 **int** \*prod\_pos = \*shm\_prod\_pos;  
  
 **if** (\*(prod\_pos - 1) >= buffer\_size)  
 {  
 **if** (semop(sem\_fd, semops\_produce\_end, 2) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(5);  
 }  
 kill\_producers();  
 **return** 0;  
 }  
   
 \*prod\_pos = \*(prod\_pos - 1) + 1;  
 printf("Производитель %d, pid=%d создал значение %c\n", number, getpid(), \*prod\_pos);  
 prod\_pos++;  
 \*shm\_prod\_pos = prod\_pos;  
  
 **if** (semop(sem\_fd, semops\_produce\_end, 2) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(6);  
 }  
  
 **return** 0;  
}  
  
**int** **consume**(**int** sem\_fd, **int** number)  
{  
 srand(time(NULL));  
 sleep(rand() % 5 + 5);  
   
 **if** (semop(sem\_fd, semops\_consume\_begin, 2) == -1)  
 {   
 perror("Semop failed.\n");  
 exit(7);  
 }  
 **int** \*\*shm\_cons\_pos = shm\_buffer - 5;  
 **int** \*cons\_pos = \*shm\_cons\_pos;  
 **if** (cons\_pos - shm\_buffer >= buffer\_size)  
 {  
 **if** (semop(sem\_fd, semops\_consume\_end, 2) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(1);  
 }  
 kill\_consumers();  
 **return** 0;  
 }  
 printf("Потребитель %d, pid=%d прочитал значение: %c\n", number, getpid(), \*cons\_pos);  
 cons\_pos++;  
 \*shm\_cons\_pos = cons\_pos;  
  
 **if** (semop(sem\_fd, semops\_consume\_end, 2) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(8);  
 }  
  
 **return** 0;  
}  
  
**void** **get\_producer**(**int** number, **int** sem\_fd) {  
 **pid\_t** pid;  
 **if** ((pid = fork()) == -1)   
 {  
 printf("Fork failed.\n");  
 exit(4);  
 }  
  
 **if**(pid == 0)   
 {  
 printf("Создан производитель %d, pid: %d\n", number, getpid());  
 **while**(1)  
 produce(sem\_fd, number);  
 }  
 **else**  
 {  
 pid\_arr[number] = pid;  
 }  
}  
  
**void** **get\_consumer**(**int** number, **const** **int** sem\_fd) {  
 **pid\_t** pid;  
  
 **if** ((pid = fork()) == -1)   
 {  
 printf("Fork failed.\n");  
 exit(1);  
 }  
  
 **if**(pid == 0)   
 {  
 printf("Создан потребитель %d, pid: %d\n", number, getpid());  
 **while**(1)  
 consume(sem\_fd, number);  
 }   
 **else**  
 {  
 pid\_arr[number] = pid;  
 }  
}  
  
**int** **main**()  
{  
 **int** \*shm = get\_shm();  
  
 shm\_cons\_pos = shm;  
 shm\_prod\_pos = shm + 2;  
 \*(shm + 4) = 0;  
 shm\_buffer = shm + 5;  
 pid\_arr = shm + 5 + buffer\_size;  
  
 \*shm\_cons\_pos = shm\_buffer;  
 \*shm\_prod\_pos = shm\_buffer;  
   
 **int** sem\_fd = get\_sem\_fd();  
  
 **if** (semctl(sem\_fd, SE, SETVAL, consumer\_num + producer\_num) == -1 ||  
 semctl(sem\_fd, SF, SETVAL, 0) == -1 ||  
 semctl(sem\_fd, SB, SETVAL, 1) == -1)  
 {  
 perror("Semctl failed");  
 exit(1);  
 }  
  
 **int** i = 0;  
 **while**(i < producer\_num)  
 {  
 get\_producer(i, sem\_fd);  
 i++;  
 }  
  
 **int** j = producer\_num;  
 **while**(j < producer\_num + consumer\_num)  
 {  
 get\_consumer(j, sem\_fd);  
 j++;  
 }  
   
 **for** (**int** i = 0; i < consumer\_num + producer\_num; i++)  
 {  
 **int** \*status;  
 **if** (wait(status) == -1)  
 {  
 printf("Waitpid");  
 exit(9);  
 }  
 **if** (WIFEXITED(status))  
 {  
 printf("\tChild exited with code = %d\n", WEXITSTATUS(status));  
 }  
 **else** **if** (WIFSIGNALED(status))  
 {  
 printf("\tChild with (signal %d) killed\n", WTERMSIG(status));  
 }  
 **else** **if** (WIFSTOPPED(status))  
 {  
 printf("\tChild with (signal %d) stopped\n", WSTOPSIG(status));  
 }  
 **else**  
 printf("\tChild terminated abnormally \n");  
 }  
  
 **if** (shmdt(shm) == -1)  
 {  
 perror("Shmdt failed.\n");  
 exit(10);  
 }  
  
 **return** 0;  
}

**Результат работы программы:**

Создан производитель 0, pid: 57842  
Создан производитель 1, pid: 57843  
Создан производитель 2, pid: 57844  
Создан потребитель 0, pid: 57845  
Создан потребитель 1, pid: 57846  
Создан потребитель 2, pid: 57847  
Производитель 0, pid=57842 создал значение A  
Производитель 1, pid=57843 создал значение B  
Производитель 2, pid=57844 создал значение C  
Производитель 1, pid=57843 создал значение D  
Производитель 0, pid=57842 создал значение E  
Производитель 2, pid=57844 создал значение F  
Потребитель 0, pid=57845 прочитал значение: A  
Производитель 2, pid=57844 создал значение G  
Потребитель 1, pid=57846 прочитал значение: B  
Производитель 1, pid=57843 создал значение H  
Потребитель 2, pid=57847 прочитал значение: C  
Производитель 0, pid=57842 создал значение I  
Потребитель 1, pid=57846 прочитал значение: D  
Производитель 2, pid=57844 создал значение J  
Потребитель 0, pid=57845 прочитал значение: E  
Потребитель 2, pid=57847 прочитал значение: F  
Потребитель 0, pid=57845 прочитал значение: G  
Потребитель 1, pid=57846 прочитал значение: H  
Потребитель 2, pid=57847 прочитал значение: I  
Потребитель 0, pid=57845 прочитал значение: J

**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. Для реализации взаимоисключения используются семафоры.

**Листинг программы:**

#**include** <sys/sem.h>  
#**include** <sys/shm.h>  
#**include** <sys/stat.h>  
#**include** <sys/wait.h>  
#**include** <stdio.h>  
#**include** <stdlib.h>  
#**include** <time.h>  
#**include** <unistd.h>  
  
#**define** ACTIVE\_WRITER 0  
#**define** WAITING\_WRITERS 1  
#**define** ACTIVE\_READERS 2  
#**define** WAITING\_READERS 3  
  
#**define** WRITERS\_NUM 3  
#**define** READERS\_NUM 5  
  
**struct** **sembuf** **start\_read**[5] = {{WAITING\_READERS, 1, SEM\_UNDO},   
 {WAITING\_WRITERS, 0, SEM\_UNDO},   
 {ACTIVE\_WRITER, 0, SEM\_UNDO},   
 {WAITING\_READERS, -1, SEM\_UNDO},   
 {ACTIVE\_READERS, 1, SEM\_UNDO}};  
  
**struct** **sembuf** **stop\_read**[1] = {{ACTIVE\_READERS, -1, SEM\_UNDO}};  
  
**struct** **sembuf** **start\_write**[5] = {{WAITING\_WRITERS, 1, SEM\_UNDO},   
 {ACTIVE\_WRITER, 0, SEM\_UNDO},   
 {ACTIVE\_READERS, 0, SEM\_UNDO},   
 {WAITING\_WRITERS, -1, SEM\_UNDO},   
 {ACTIVE\_WRITER, 1, SEM\_UNDO}};  
  
**struct** **sembuf** **stop\_write**[1] = {{ACTIVE\_WRITER, -1, SEM\_UNDO}};  
  
**int** \*shm;  
  
**void** **writer**(**const** **int** semid, **const** **int** index)  
{  
 srand(time(NULL));  
 sleep(rand() % 10);  
 **if** (semop(semid, start\_write, 5) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(2);  
 }  
 (\*shm)++;  
 printf("Писатель %d написал %d\n", index + 1, \*shm);  
 **if** (semop(semid, stop\_write, 1) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(2);  
 }  
}  
  
**void** **reader**(**const** **int** semid, **const** **int** index)  
{  
 srand(time(NULL));  
 sleep(rand() % 10);  
 **if** (semop(semid, start\_read, 5) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(1);  
 }  
 printf("Читатель %d прочитал %d\n", index - WRITERS\_NUM + 1, \*shm);  
 **if** (semop(semid, stop\_read, 1) == -1)  
 {  
 perror("Semop failed.\n");  
 exit(1);  
 }  
}  
  
**int** **main**()  
{  
 **int** semid, shmid;  
 **int** ctl\_ar, ctl\_wr, ctl\_aw, ctl\_ww;  
 **int** status;  
 **pid\_t** pid[WRITERS\_NUM + READERS\_NUM];  
 **const** **int** PERMS = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH;  
  
 **if** ((semid = semget(IPC\_PRIVATE, 4, IPC\_CREAT | PERMS)) == -1)  
 {  
 perror("Semget failed.\n");  
 exit(3);  
 }  
 ctl\_ar = semctl(semid, ACTIVE\_READERS, SETVAL, 0);  
 ctl\_wr = semctl(semid, WAITING\_READERS, SETVAL, 0);  
 ctl\_aw = semctl(semid, ACTIVE\_WRITER, SETVAL, 0);  
 ctl\_ww = semctl(semid, WAITING\_WRITERS, SETVAL, 0);  
 **if** (ctl\_ar == -1 || ctl\_wr == -1 || ctl\_aw == -1 || ctl\_ww == -1)  
 {  
 perror("Semctl failed.\n" );  
 exit(4);  
 }  
   
 **if** ((shmid = shmget(IPC\_PRIVATE, **sizeof**(**int**), IPC\_CREAT | PERMS)) == -1)  
 {  
 perror("Shmget failed.\n");  
 exit(5);  
 }  
   
 shm = shmat(shmid, 0, 0);  
 **if** (\*shm == -1)  
 {  
 perror("Shmat failed.\n");  
 exit(6);  
 }  
 (\*shm) = 0;  
   
 **for** (**int** i = 0; i < WRITERS\_NUM; i++)  
 {  
 pid[i] = fork();  
 **if** (pid[i] == -1)  
 {  
 perror("Fork failed.\n");  
 exit(7);  
 }  
 **if** (pid[i] == 0)  
 {  
 **while**(1)  
 writer(semid, i);  
 **return** 0;  
 }  
 }  
   
 **for** (**int** i = WRITERS\_NUM; i < WRITERS\_NUM + READERS\_NUM; i++)  
 {  
 pid[i] = fork();  
 **if** (pid[i] == -1)  
 {  
 perror("Fork failed.\n");  
 exit(8);  
 }  
 **if** (pid[i] == 0)  
 {  
 **while**(1)  
 reader(semid, i);  
 **return** 0;  
 }  
 }  
   
 **while** (wait(&status) != -1){}  
 **if** (shmdt(shm) == -1)  
 {  
 perror("Shmdt failed.\n");  
 exit(10);  
 }  
   
 **return** 0;  
}

**Результат работы программы:**

Писатель 3 написал 1  
Писатель 1 написал 2  
Читатель 1 прочитал 2  
Читатель 2 прочитал 2  
Читатель 3 прочитал 2  
Писатель 2 написал 3  
Читатель 4 прочитал 3  
Читатель 5 прочитал 3  
Писатель 3 написал 4  
Писатель 1 написал 5  
Читатель 1 прочитал 5  
Читатель 3 прочитал 5  
Читатель 2 прочитал 5  
Писатель 2 написал 6  
Читатель 4 прочитал 6  
Читатель 5 прочитал 6  
Читатель 3 прочитал 6  
Читатель 4 прочитал 6  
Читатель 1 прочитал 6  
Читатель 2 прочитал 6  
Читатель 5 прочитал 6  
Писатель 1 написал 7  
Писатель 3 написал 8  
Писатель 2 написал 9  
Читатель 3 прочитал 9  
Читатель 1 прочитал 9  
Читатель 5 прочитал 9  
Читатель 2 прочитал 9  
Читатель 4 прочитал 9  
Писатель 1 написал 10  
Писатель 3 написал 11  
Писатель 2 написал 12