|  |  |
| --- | --- |
| **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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ДИСЦИПЛИНА «Операционные системы»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 5**

|  |  |
| --- | --- |
| **Тема** «Взаимодействие параллельных процессов»  **Студент** Шиленков А. А.  **Группа** ИУ7-55Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Рязанова Н.Ю. |  |

Москва.

2020 г.

**Цели и задачи**

В лабораторной работе исследуются вопросы и формируются навыки использования в приложения таких средств меж процессного взаимодействия, как семафоры и разделяемая память. Работа выполняется на примере двух задач, характерных для взаимодействия асинхронных параллельных процессов: «производство-потребление» и «читатели-писатели».

В программах осуществляется консольный вывод, т.е. никакого интерфейса не нужно. Работающая программа должна выводить на экран какой процесс что записал, какой процесс что считал.

**Задание 1**

Реализация задачи «Производство-потребление»: 3 процесса производителя, 3 процесса потребителя. 3 семафора: 2 считающих и 1 бинарный.

**Код программы:**

// Производство-потребление

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define CNT\_BUF 5

#define VALUES 10

#define PROD 3

#define CONS 3

#define BIN 0

#define EMPTY 1

#define FULL 2

struct sembuf producer\_P[2] = {{EMPTY, -1, 0}, {FULL, -1, 0}};

struct sembuf producer\_V[2] = {{BIN, 1, 0}, {FULL, 1, 0}};

struct sembuf consumer\_P[2] = {{BIN, -1, 0}, {FULL, -1, 0}};

struct sembuf consumer\_V[2] = {{EMPTY, 1, 0}, {FULL, 1, 0}};

int \*shared\_buffer;

int \*shared\_pos\_consumer;

int \*shared\_pos\_producer;

int \*shared\_letter;

int producer(int semaphores, pid\_t pid)

{

while (1)

{

sleep(rand() % 5);

if (semop(semaphores, producer\_P, 2) == -1)

{

perror("Can't SEMOP!\n");

exit(1);

}

shared\_buffer[\*shared\_pos\_producer] = \*shared\_letter + 'a';

printf("\x1b[7mPROD #%d: %c to buf[%d]\n", pid, shared\_buffer[\*shared\_pos\_producer], \*shared\_pos\_producer);

(\*shared\_pos\_producer) = (\*shared\_pos\_producer + 1) % CNT\_BUF;

(\*shared\_letter) = (\*shared\_letter + 1) % 26;

if (semop(semaphores, producer\_V, 2) == -1)

{

perror("Can't SEMOP!\n");

exit(1);

}

}

return 0;

}

int consumer(int semaphores, pid\_t pid)

{

while (1)

{

sleep(rand() % 2);

if (semop(semaphores, consumer\_P, 2) == -1)

{

perror("Can't SEMOP!\n");

exit(1);

}

printf("\x1b[0mCONS #%d: %c from buf[%d]\n", pid, shared\_buffer[\*shared\_pos\_consumer], \*shared\_pos\_consumer);

(\*shared\_pos\_consumer) = (\*shared\_pos\_consumer + 1) % CNT\_BUF;

if (semop(semaphores, consumer\_V, 2) == -1)

{

perror("Can't SEMOP!\n");

exit(1);

}

}

return 0;

}

int main()

{

srand(NULL);

int perms = S\_IRWXU | S\_IRWXG | S\_IRWXO;

int fd = shmget(IPC\_PRIVATE, (CNT\_BUF + 1) \* sizeof(char), IPC\_CREAT | perms);

if (fd == -1)

{

perror("Can't SHMGET!\n");

exit(1);

}

shared\_pos\_producer = shmat(fd, 0, 0);

if (\*shared\_pos\_producer == -1)

{

perror("Can't SHMAT!\n");

exit(1);

}

shared\_buffer = shared\_pos\_producer + 3 \* sizeof(char);

shared\_pos\_consumer = shared\_pos\_producer + 2 \* sizeof(char);

shared\_letter = shared\_pos\_producer + sizeof(char);

(\*shared\_pos\_producer) = 0;

(\*shared\_pos\_consumer) = 0;

(\*shared\_letter) = 0;

int semaphores = semget(IPC\_PRIVATE, 3, IPC\_CREAT | perms);

if (semaphores == -1)

{

perror("Can't SEMGET!\n");

exit(1);

}

int ctlb = semctl(semaphores, BIN, SETVAL, 0);

int ctle = semctl(semaphores, EMPTY, SETVAL, CNT\_BUF);

int ctlf = semctl(semaphores, FULL, SETVAL, 1);

if (ctlf == -1 || ctle == -1 || ctlb == -1)

{

perror("Can't SEMCTL!\n");

exit(1);

}

pid\_t pid;

for (int i = 0; i < PROD && pid != 0; i++)

{

if ((pid = fork()) == -1)

{

perror("Can't FORK prod!\n");

exit(1);

}

if (pid == 0)

{

producer(semaphores, getpid());

return 0;

}

}

for (int i = 0; i < CONS && pid != 0; i++)

{

if ((pid = fork()) == -1)

{

perror("Can't FORK cons!\n");

exit(1);

}

if (pid == 0)

{

consumer(semaphores, getpid());

return 0;

}

}

if (pid != 0)

{

int status;

for (int i = 0; i < CNT\_BUF; i++)

wait(&status);

if (shmdt(shared\_pos\_producer) == -1)

{

perror("Can't SHMDT!\n");

exit(1);

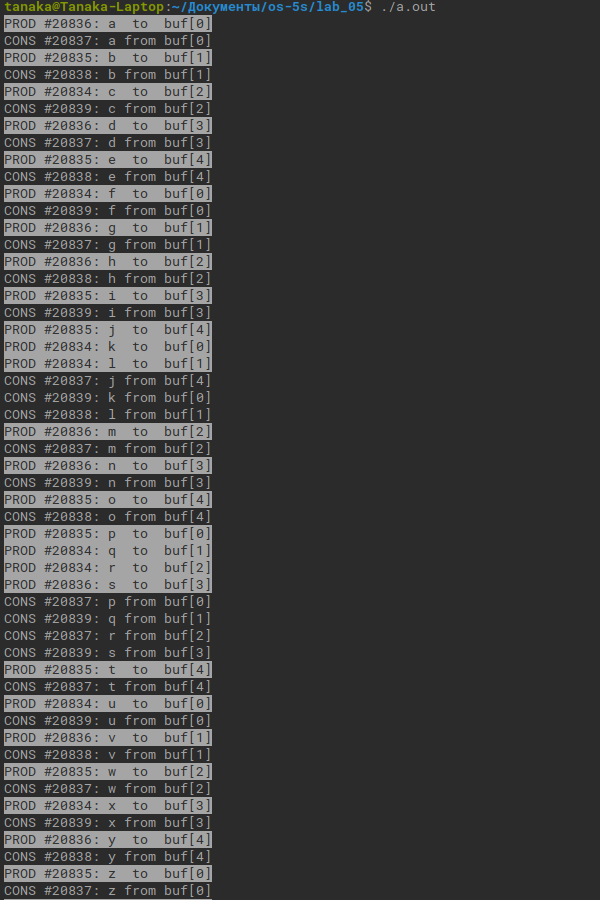
}

}

return 0;

}

**Пример работы программы:**

**

*Изображение 1. – Работа программы.*

**Задание 2**

Реализация задачи «Читатели – писатели»: 3 процесса писателя, 5 процессов потребителей. 4 семафора: 3 считающих и 1 бинарный.

**Код программы:**

// Читатели писатели

#include <stdio.h>

#include <unistd.h>

#include <sys/sem.h>

#include <sys/stat.h>

#include <sys/shm.h>

#include <stdlib.h>

#include <sys/wait.h>

// Количество читателей и писателей

#define COUNT\_READERS 4

#define COUNT\_WRITERS 4

// Операции на семафорах

#define SEM\_OP\_INC 1

#define SEM\_OP\_DEC -1

#define SEM\_OP\_WAIT 0

// Ключи

#define ACTIVE\_READERS 0

#define ACTIVE\_WRITER 1

#define WAITING\_WRITERS 2

#define WAITING\_READERS 3

struct sembuf start\_writer[] = {{WAITING\_WRITERS, SEM\_OP\_INC, 0},

{ACTIVE\_READERS, SEM\_OP\_WAIT, 0},

{ACTIVE\_WRITER, SEM\_OP\_WAIT, 0},

{ACTIVE\_WRITER, SEM\_OP\_INC, 0},

{WAITING\_WRITERS, SEM\_OP\_DEC, 0},},

stop\_writer[] = {{ACTIVE\_WRITER, -1, 0}},

start\_reader[] = {{WAITING\_READERS, SEM\_OP\_INC, 0},

{WAITING\_WRITERS, SEM\_OP\_WAIT, 0},

{ACTIVE\_WRITER, SEM\_OP\_WAIT, 0},

{ACTIVE\_READERS, SEM\_OP\_INC, 0},

{WAITING\_READERS, SEM\_OP\_DEC, 0},},

stop\_reader[] = {{ACTIVE\_READERS, -1, 0}};

void Reader(int nomer, int sem\_id, int\* buf)

{

while (1)

{

semop(sem\_id, start\_reader, 5);

printf("Reader #%d -> ", nomer);

printf("read %d\n", \*buf);

semop(sem\_id, stop\_reader, 1);

sleep(rand() % 2);

}

}

void Writer(int nomer, int sem\_id, int\* buf)

{

while (1)

{

semop(sem\_id, start\_writer, 5);

printf("Writer #%d -> ", nomer);

printf("write %d\n", ++(\*buf));

semop(sem\_id, stop\_writer, 1);

sleep(rand() % 3);

}

}

int main()

{

int perms = S\_IRWXU | S\_IRWXG | S\_IRWXO;

int shm\_id;

int sem\_id;

int \*mem\_ptr;

// возвращает идентификатор разделяемому сегменту памяти

if ((shm\_id = shmget(IPC\_PRIVATE, sizeof(int), IPC\_CREAT | perms)) == -1)

{

perror("Unable to create a shared area.\n");

return 1;

}

// возвращает указатель на сегмент разделяемой памяти

if ((mem\_ptr = shmat(shm\_id, NULL, 0)) == -1)

{

perror("Can't attach memory.\n");

return 1;

}

// создание набора семафоров

if ((sem\_id = semget(IPC\_PRIVATE, 4, perms)) == -1)

{

perror("Can’t semget.\n");

return 1;

}

// изменение управляющих параметров набора семафоров

semctl(sem\_id, ACTIVE\_READERS, SETVAL, 0);

semctl(sem\_id, ACTIVE\_WRITER, SETVAL, 0);

semctl(sem\_id, WAITING\_READERS, SETVAL, 0);

semctl(sem\_id, WAITING\_WRITERS, SETVAL, 0);

// создание процессов

int count\_processes = 0;

pid\_t pid;

\*mem\_ptr = 0;

for (int i = 0; i < COUNT\_WRITERS; i++)

{

if ((pid = fork())== -1)

{

perror("Can’t fork.\n");

return 1;

}

if (!pid)

Writer(i + 1, sem\_id, mem\_ptr);

else

count\_processes++;

}

for (int i = 0; i < COUNT\_READERS; i++)

{

if ((pid = fork())== -1)

{

perror("Can’t fork.\n");

return 1;

}

if (!pid)

Reader(i + 1, sem\_id, mem\_ptr);

else

count\_processes++;

}

// ожидание завершение процессов

int status;

for (int i = 0; i < count\_processes; i++)

{

wait(&status);

if (!WIFEXITED(status))

printf("exit-error, code = %d\n", status);

}

// освобождение ресурсов

shmdt(mem\_ptr);

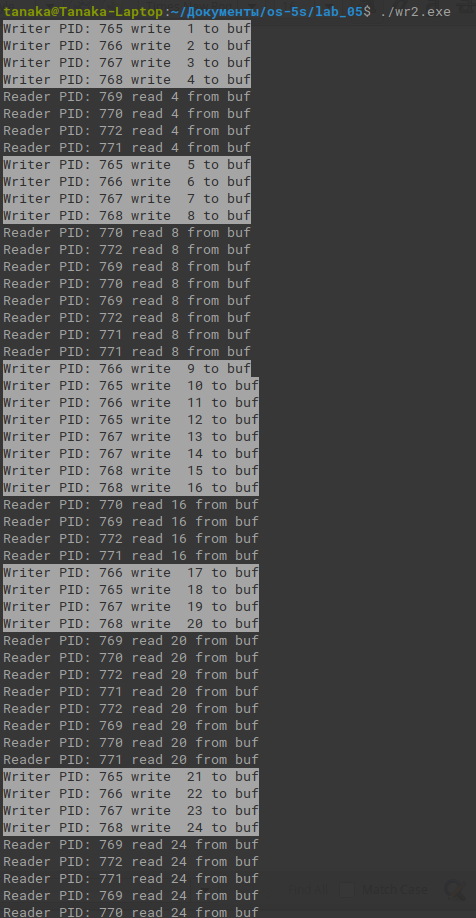
semctl(sem\_id, 0, IPC\_RMID);

shmctl(shm\_id, IPC\_RMID, 0);

return 0;

}

**Пример работы программы:**



*Изображение 3. – Работа второй программы.*