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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 3

Название: Взаимодействие параллельных процессов.

Дисциплина: Операционные системы

Студент ИУ7-52Б _____ Е.В. Брянская
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Ю. Рязанова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Задание 1.

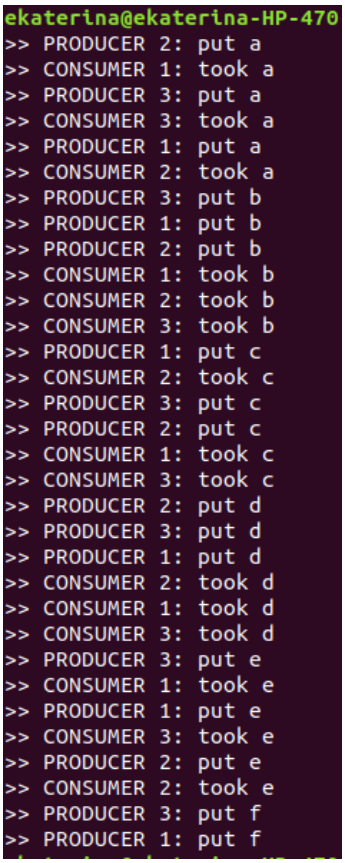
Написать программу, реализующую задачу «Производство-потребление» по алгоритму Э. Дейкстры с тремя семафорами: двумя считающими и одним бинарным. В программе должно создаваться не менее 3
изводителей и 3х процессов – потребителей. В программе надо обеспечить случайные задержки выполнения созданных процессов. 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.

```
#define N 10
#define NUM_PR 3

#define SE 0
#define SF 1
#define SB 2

struct sembuf before_putting_in_buf[2] =
{
{SE, -1, SEM_UNDO},
{SB, -1, SEM_UNDO}
};

struct sembuf after_putting_in_buf[2] =
{
{SB, 1, SEM_UNDO},
{SF, 1, SEM_UNDO}
};

struct sembuf before_taking_from_buf[2] =
{
{SF, -1, SEM_UNDO},
{SB, -1, SEM_UNDO}
};

struct sembuf after_taking_from_buf[2] =
{
{SB, 1, SEM_UNDO},
{SE, 1, SEM_UNDO}
};


int action_producer(int num_pr, int id_sem, int n, char* addr)
{
	int temp;
	char cur_letter = 'a';

	while (1)
	{
		sleep(rand() % 4 + 1);

		temp = semop(id_sem, before_putting_in_buf, 2);
		if (temp == -1)
		{
		perror("semop error");
		return 6;
		}

		addr[addr[1]] = cur_letter;
		printf(">> PRODUCER %d: put %c\n", num_pr + 1, cur_letter);
		cur_letter++;

		if (cur_letter > 'z')
			cur_letter = 'a';
```

```
		(addr[1])++;

		if (addr[1] > n - 1)
			addr[1] = 2;

		temp = semop(id_sem, after_putting_in_buf, 2);
		if (temp == -1)
		{
			perror("semop error");
			return 6;
		}
	}

	return 0;
}

int action_consumer(int num_cn, int cur_id, int n, char* addr)
{
	int temp;

	while (1)
	{
		sleep(rand() % 4 + 1);

		temp = semop(cur_id, before_taking_from_buf, 2);
		if (temp == -1)
		{
			perror("semop error");
			return 6;
		}

		printf(">> CONSUMER %d: took %c\n", num_cn + 1, addr[addr[0]]);
		addr[0]++;

		if (addr[0] > n - 1)
			addr[0] = 2;

		temp = semop(cur_id, after_taking_from_buf, 2);
		if (temp == -1)
		{
			perror("semop error");
			return 6;
		}
	}

	return 0;
}


int main()
{
	srand(time(NULL));

	int perms = S_IRWXU | S_IRWXG | S_IRWXO;
	int se, sf, sb, res, status, temp_id1, temp_id2;
	pid_t producers[NUM_PR], consumers[NUM_PR];

	int id_shm = shmget(IPC_PRIVATE, (N + 2) * sizeof(char), IPC_CREAT | perms);
	if (id_shm == -1)
	{
		perror("shmget error");
		exit(1);
	}

	char* addr = (char*)shmat(id_shm, 0, 0);
	if (addr == (char*)-1)
	{
		perror("shmat error");
		exit(2);
	}

	addr[0] = (char)2;
	addr[1] = (char)2;

	int id_sem = semget(IPC_PRIVATE, 3, IPC_CREAT | perms);
	if (id_sem == -1)
```

```
	{
		perror("semget error");
		exit(3);
	}

	se = semctl(id_sem, SE, SETVAL, N);
	sf = semctl(id_sem, SF, SETVAL, 0);
	sb = semctl(id_sem, SB, SETVAL, 1);

	if (se == -1 || sf == -1 || sb == -1)
	{
		perror("semctl error");
		exit(4);
	}
	for (int i = 0; i < NUM_PR; i++)
	{
		producers[i] = fork();
		if (producers[i] == -1)
		{
			perror("fork error");
			exit(5);
		}

		if (producers[i] == 0)
		{
			res = action_producer(i, id_sem, N + 2, addr);
			if (res)
				exit(res);
		}

		rand();

		consumers[i] = fork();
		if (consumers[i] == -1)
		{
			perror("fork error");
			exit(5);
		}

		if (consumers[i] == 0)
		{
			res = action_consumer(i, id_sem, N + 2, addr);
			if (res)
				exit(res);
		}

		rand();
	}

	for (int i = 0; i < NUM_PR; i++)
	{
		temp_id1 = wait(&status);
		temp_id2 = wait(&status);
		if (temp_id1 == -1 || temp_id2 == -1)
		{
			perror("wait error");
			exit(7);
		}
	}

	if (shmctl(id_shm, IPC_RMID, NULL) == -1)
	{
		perror("shmctl error");
		exit(4);
	}

	if (shmctl(id_sem, 0, IPC_RMID) == -1)
	{
		perror("shmctl error");
		exit(4);
	}

	return 0;
}
```

Результат выполнения программы:



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.