6.2. Задача производители и потребители

```
Lock mutex;                          // блокировка доступа к складу
Semaphore empty(n), full(0);         // n – вместимость склада

void producer()
{
        while(true)
        {
                produceProduct();        // произвести продукт
                empty.wait();            // уменьшить кол-во свободных ячеек на складе
                mutex.acquire();         // вход в критическую секцию
                putProduct();            // поместить продукт на склад
                mutex.release();         // выход из критической секции
                full.signal();           // увеличить кол-во продуктов на складе
        }
}

void consumer()
{
        while(true)
        {
                full.wait();             // уменьшить кол-во продуктов на складе
                mutex.acquire();         // вход в критическую секцию
                getProduct();            // взять продукт со склада
                mutex.release();         // выход из критической секции
                empty.signal();          // увеличить кол-во своб. ячеек на складе
                consumeProduct();        // потребить продукт
        }
}
```

6.3. Задача «Читатели и писатели»

```
Integer readerCount =0;              // количество читателей
Lock mutex;
Semaphore write(1);                  // количество писателей

void writer()
{
        write.wait();                // захватить доступ к ресурсу
        modifyData();                // изменить данные
        write.signal();              // освободить доступ к ресурсу
}
void reader()
{
        mutex.acquire();             // войти в критическую секцию
```

```
        ++readerCount;              // увеличить количество читателей
        if (readerCount == 1)       // если первый читатель,
                write.wait();       // то запретить писать
        mutex.release();            // выйти из критической секции
        readData();                 // читать данные
        mutex.acquire();            // войти в критическую секцию
        --readerCount;              // уменьшить количество читателей
        if (readerCount == 0)       // если читателей нет,
                write.signal();     // разрешить писать
        mutex.release();            // выйти из критической секции
}
```

6.4. Задача Дейкстры о спящем парикмахере

```
Semaphore customer(0);
Semaphore barber(0);
Lock accessSeat;
int nFreeSeat = n;              // количество свободных мест

void barber()
{
        while(true)                     // runs in an infinite loop
        {
                customer.wait();        // ждет посетителя
                accessSeat.acquire();   // закр. вход в зал ожидания
                ++nFreeSeat;            // приглашает клиента
                accessSeat.release();   // откр. вход в зал ожидания
                barber.signal();        // приступает к работе
                cuttingHair();          // стрижет
        }
}

void customer()
{
        accessSeat.acquire();           // заходит в зал ожидания и закрывает вход
        if (nFreeSeat > 0)              // если есть свободные места
        {
                --nFreeSeat;            // занимает место
                customer.release();     // становится в очередь на обслуживание
                accessSeat.release();   // открывает вход в зал ожидания
                barber.wait();          // ждет в очереди обслуживания парикмахера
                haveCutingHair();       // стрижется
                exit();                 // уходит
        }
        else                            // свободных мест нет
        {
```

```
                accessSeat.release();       // открывает вход в зал ожидания
                exit();                     // уходит
        }
}
```

6.5. Задача Дейкстры об обедающих философах

```
#define THINKING        0
#define HUNGRY          1
#define EATING          2

#define LEFT    (i+4)%5
#define RIGHT   (i+1)%5

int state[5];                               // состояния философов
Semaphore s[5] = {0, 0, 0, 0, 0};           // один семафор на философа
Semaphore mutex(1);                         // взаимное исключение

void philosopher(int i)
{
        while(true)
        {
                think();
                take_forks(i);
                eat();
                put_forks(i);
        }
}

void take_forks(int i)
{
        mutex.wait();
        state[i]=HUNGRY;
        test(i);                    // попробовать взять две вилки
        mutex.signal();
        s[i].wait();                // заблокироваться, если вилки недоступны
}

void put_forks(int i)
{
        mutex.wait();
        state[i]=THINKING;
        test(LEFT);                 // разрешить есть соседу слева
        test(RIGHT);                // разрешить есть соседу справа
        mutex.signal();
}
```