Параллельное программирование

Т. П. ГРЫЗЛОВА

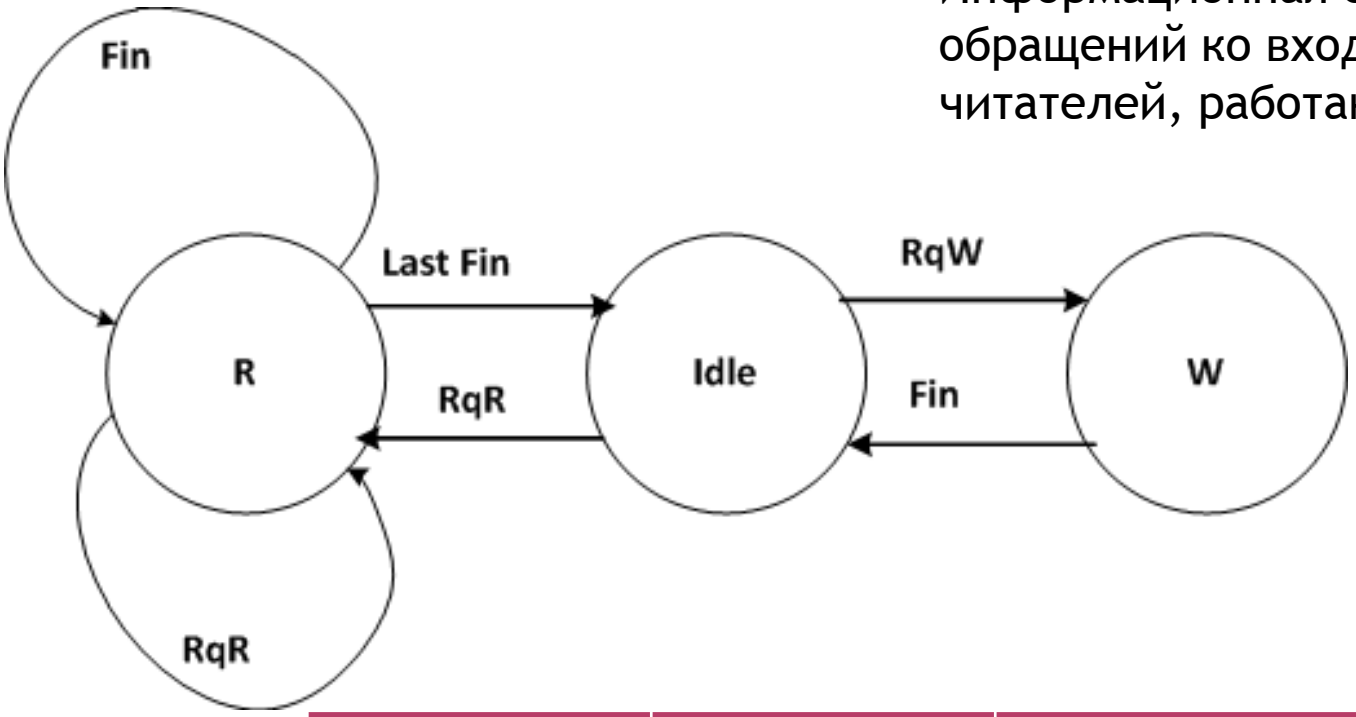
ПРОЕКТИРОВАНИЕ ПРОГРАММ НА ADA И C#. ЧИТАТЕЛИ - ПИСАТЕЛИ

РГАТА им. П. А. Соловьева

Читатели-писатели. Синхронизация процессов чтения – записи с помощью управляющей задачи sheduler

Управляющий автомат в задаче Sheduler

Информационная среда – счетчик обращений ко входу RqR – число читателей, работающих с ресурсом



States=> Signals	Idle, Count = 0	R	W
RqR	R, Count++	R, Count++	W
RqW	W		W
Fin	-	R, Count--	Idle
Last Fin	-	Idle, Count = 0	-

```
with Gnat.io; use Gnat.io;
procedure RW1 is
type tStates is (IDLE,R,W);
state: tStates := IDLE;
nReaders: integer := 0;

task Sheduler is
task Writer is
task Reader1 is
task Reader2 is
task Reader3 is
task body Sheduler is

task body Writer is
task body Reader1 is

task body Reader2 is

task body Reader3 is

begin
   put("");
end RW1;
```

АДА-ПРОГРАММЫ. ОТ АВТОМАТНОЙ МОДЕЛИ УПРАВЛЯЮЩЕГО МОДУЛЯ К ПРОГРАММЕ

Грызлова Т. П.

Читатели-писатели, вар. 1

```
with Text_IO; use text_IO;
procedure RsW_1 is
  task Res_Sheduler is
    entry Req_Read;
    entry Req_Write;
    entry FIN;
  end Res_Sheduler;
  task Reader;
  task Writer;
 Task body Reader is
  begin
    loop
      Res_Sheduler.Req_Read;
      Res_Sheduler.FIN;
    end loop;
  end Reader;
```

```
Task body Writer is
  begin
    loop
      Res_Sheduler.Req_Write;
      Res_Sheduler.FIN;
    end loop;
  end Writer;
```

```
task body Res_Sheduler is
   type tState is (Idle, Reading, Writing);
   State : tState := Idle;
   Readers : integer := 0;
   begin
   loop
     select
      when State = Idle => accept Req_Write do
        State := Writing; end;
      or when State /= Writing => accept Req_Read do
        State := Reading; Readers := Readers + 1;
      end Req_Read;
      or when State /= Idle => accept FIN do
        case State is
          when Reading => Readers := Readers - 1;
          if Readers = 0 then State := Idle; end if;
          when Writing => State := Idle;
                    when Idle => State := Idle; -- без этой строки не работает
       end case;
      end FIN;
     end select;
   end loop;
 end Res_Sheduler;
```

Окончание «Читатели-писатели», вар. 1

```
begin
  put("    m");
end RsW_1;
```

Семафоры в C#.
Синхронизация процессов чтения - записи с дополнительными ограничениями

Семафоры

Семафоры. Операции с семафорами
Объекты типа **Semaphore** ограничивают число потоков, которые могут одновременно получать доступ к ресурсу или пулу ресурсов. Метод **WaitOne** – вход в семафор, **Release** – освобождение семафора. Объект **Semaphore** создается с помощью конструктора, в который передаются параметры – начальное значение семафора и количество процессов, которые могут одновременно иметь доступ к ресурсу.
SemaphoreSlim – упрощенная альтернатива семафору **Semaphore**, ограничивающая количество потоков, которые могут параллельно обращаться к ресурсу или пулу ресурсов.

Определяется семафор

```
static Semaphore rw1;
```

Количество итераций фиксировано

```
NI = 15;
```

Фиксировано число читателей и писателей:

```
NR = 3; NW = 2;
```

Переменные – iteration с начальным значением

```
iteration = 0;
```

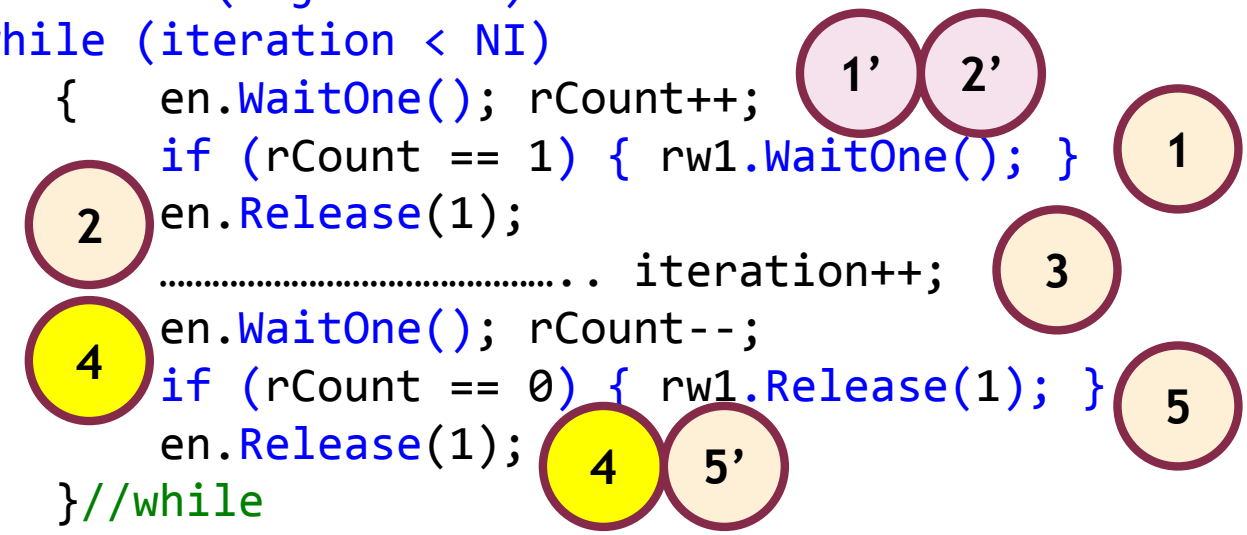
```
rw1 = new Semaphore(1, 1);
TimeSpan t0 = new TimeSpan(0, 0, 0);
Console.WriteLine(t0.ToString());
DateTime date1 = DateTime.Now;
long d0 = date1.Ticks;
```

Создание нитей, выполняющей заданные функции

```
                Thread Reader = new Thread(new
ParameterizedThreadStart(ToRead));
                Reader.Start(i);
                Thread Writer = new Thread(new
ParameterizedThreadStart(ToWrite));
                Writer.Start(i);
        static void ToRead(object num)

            while (iteration < NI)

                rw1.WaitOne();
                Thread.Sleep(200);
                iteration++;
                rw1.Release(1);
            }//while

        }//Reader
        static void ToWrite (object num
```

```
static Semaphore rw1;
        static Semaphore en;
        static int rCount;


rw1 = new Semaphore(1, 1);
            en = new Semaphore(1, 1);
            rCount = 0;


 static void ToWrite(object num)
         {while (iteration < NI)
             { rw1.WaitOne();
               Thread.Sleep(500);
               iteration++;
               rw1.Release(1);
             }//while
 }//Writer
```

```
static void ToRead(object num)
        {while (iteration < NI)
            {   en.WaitOne(); rCount++;
                if (rCount == 1) { rw1.WaitOne(); }
                en.Release(1);
                …….. iteration++;
                en.WaitOne(); rCount--;
                if (rCount == 0) { rw1.Release(1); }
                en.Release(1);
            }//while
}//Reader
```



1) Первый читатель ждет на семафоре **rw1**, остальные ждут на семафоре **en**;
2) Если первый читатель захватил семафор **rw1**, то он освобождает семафор **en**, и тем самым дает возможность всем остальным процессам чтения войти по очереди в секцию, защищенную семафором **en**, и увеличить значение глобального счетчика
3) Все читатели могут читать ресурс, они все находятся в критической секции, защищенной семафором **rw1**
4) Перед выходом из режима чтения читатель снова захватывает семафор **en**, уменьшает значение счетчика и освобождает семафор **en**, давая возможность остальным процессам чтения корректно завершить работу с ресурсом
5) Последний читатель должен освободить семафор **rw1**