26 de mayo de 2019

Laura Aguilera Checa

Ignacio Aguilera Gómez

Grado en ingeniería informática

Universidad de Almería

Práctica 3. Planificación de tareas en ADA

Sistemas de tiempo real

Contenido

[1. Autores 2](#_Toc9773526)

[2. Introducción 2](#_Toc9773527)

[3. Actividades a realizar 2](#_Toc9773528)

[3.1. Asignación de prioridades 2](#_Toc9773529)

[3.2. Planificación con ejecutivo cíclico 3](#_Toc9773530)

[3.3. Implementación con ejecutivo cíclico 3](#_Toc9773531)

[3.4.Planificación con herencia de prioridad 4](#_Toc9773532)

[3.5. Implementación con herencia de prioridad 4](#_Toc9773533)

[Anexo. Clases utilizadas para la implementación 7](#_Toc9773534)

[STR\_CPU.ads 7](#_Toc9773535)

[STR\_CPU.adb 7](#_Toc9773536)

[STR\_Chrono.ads 8](#_Toc9773537)

[STR\_Chrono.adb 8](#_Toc9773538)

[STR\_Tasking.ads 9](#_Toc9773539)

[control.ads 9](#_Toc9773540)

[control.adb 10](#_Toc9773541)

[io.ads 11](#_Toc9773542)

[io.adb 11](#_Toc9773543)

[modulos.ads 14](#_Toc9773544)

[modulos.adb 15](#_Toc9773545)

[semáforos.ads 16](#_Toc9773546)

[semáforos.adb 16](#_Toc9773547)

[tipos\_datos.ads 17](#_Toc9773548)

# 1. Autores

|  |  |  |
| --- | --- | --- |
| Apellidos | Aguilera | Checa |
| Nombre | Laura |  |
| Titulación | Grado de Ingeniería Informática | |

|  |  |  |
| --- | --- | --- |
| Apellidos | Aguilera | Gómez |
| Nombre | Ignacio |  |
| Titulación | Grado de Ingeniería Informática | |

# 2. Introducción

En este informe se describirá con detalle el proceso de la implementación del problema descrito en las prácticas 1 y 2 usando una planificación con ejecutivo cíclico y con herencia de prioridad.

# 3. Actividades a realizar

## 3.1. Asignación de prioridades

Contamos con tres tareas: el campo solar (CS), el módulo de destilación (MD) y el sistema de seguridad (S). Cada una de ellas cuenta con una serie de acciones (tabla 2), que acceden a dos recursos compartidos: el sistema de almacenamiento (x) y el sistema de visualización por pantalla (y).

La asignación de prioridades se ha realizado tomando como más prioritaria la tarea con un tiempo de cómputo menor. Como CS y MD tienen el mismo tiempo de cómputo, se asignará una mayor prioridad a la que tenga un periodo mayor, como es, en este caso, MD.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tarea | T | C | P | Acciones |
| CS | 100 | 10 | 2 | cs1, cs2, csx, csy |
| MD | 200 | 10 | 1 | md1, md2, mdx, mdy |
| S | 50 | 9 | 3 | s1, s2, s3, sy |

Tabla . Asignación de prioridades

|  |  |  |
| --- | --- | --- |
| Acción | C | Descripción |
| cs1 | 2 | Leer temperatura del campo solar (ST2) |
| cs2 | 2 | Controlar SC1 actuando sobre la bomba 1 en función a ST2 |
| csx | 3 | Escribir datos en el almacenamiento compartido |
| csy | 3 | Mostrar resultado por pantalla |
| md1 | 2 | Leer el caudal de agua de mar (SC2) |
| md2 | 2 | Controlar SD1 en función a ST2 actuando sobre la bomba 2 |
| mdx | 3 | Escribir datos en el almacenamiento compartido |
| mdy | 3 | Mostrar resultado por pantalla |
| s1 | 2 | Leer la temperatura la salida del campo solar (ST2) |
| s2 | 2 | Leer la temperatura a la entrada de MD (ST3) |
| s3 | 2 | Verificar los valores óptimos de temperatura |
| sy | 3 | Mostrar resultado por pantalla |

Tabla . Acciones

## 3.2. Planificación con ejecutivo cíclico

Dada la tabla de asignación de prioridades (Tabla 1), se realizará la planificación con ejecutivo cíclico. De este modo, tendremos un ciclo principal de 200 ms, dividido en ciclos secundarios de 50 ms, como se muestra en la gráfica siguiente, en la que se ha representado la planificación de las tres tareas mencionadas a lo largo de un ciclo.

## 3.3. Implementación con ejecutivo cíclico

La implementación en ADA sería la siguiente. El resto de clases, desarrolladas en la práctica anterior, se encuentran en el anexo.

|  |  |
| --- | --- |
| with modulos; | |
| with Ada.Real\_Time; | |
| use Ada.Real\_Time; | |
|  |  |
|  | procedure mainCiclico is |
|  |  |
|  | reps : integer := 10; -- Número de iteraciones en la ejecucion del ejercicio |
|  |  |
|  | type Ciclo is mod 4; |
|  | Turno : Ciclo := 0; |
|  |  |
|  | i : integer := 0; -- Variable auxiliar para el bucle del Campo Solar |
|  |  |
|  | Tiempo : Time; |
|  |  |
|  | begin |
|  |  |
|  | for i in 1 .. reps loop |
|  |  |
|  | Tiempo:=Tiempo + Milliseconds(50); |
|  |  |
|  | delay until (Tiempo); |
|  |  |
|  | -- En funcion del diseño realizado en la asignacio de prioridades |
|  | case Turno is |
|  | when 0 => Modulos.SistemaDeSeguridad; Modulos.CampoSolar; Modulos.ModuloDestilacion; |
|  | when 1 => Modulos.SistemaDeSeguridad; |
|  | when 2 => Modulos.SistemaDeSeguridad; Modulos.CampoSolar; |
|  | when 3 => Modulos.SistemaDeSeguridad; |
|  |  |
|  | end case; |
|  |  |
|  | Turno := Turno + 1; |
|  |  |
|  | end loop; |
|  |  |
|  | end mainCiclico; |

## 3.4.Planificación con herencia de prioridad

Los cálculos realizados para la planificación mediante herencia de prioridad en función a las tablas 1 y 2 son los que se muestran a continuación.

BCS = bx + by = 6

BMD = 0

BS = 6

|  |  |
| --- | --- |
| S |  |
| CS |  |
|  |
| MD |  |
|  |

Como todos los resultados finales entran dentro del periodo, se garantiza la ejecución dentro de los plazos previstos.

## 3.5. Implementación con herencia de prioridad

|  |  |
| --- | --- |
| with modulos, Ada.Real\_Time, System; | |
| use Ada.Real\_Time; | |
|  |
| procedure mainHerenciaPrioridad is |
|  |
| reps : integer := 10; -- Número de iteraciones en la ejecucion del ejercicio |
|  |
| i : integer := 0; -- Variable auxiliar para el bucle del Campo Solar |
| j : integer := 0; -- Variable auxiliar para el bucle del Modulo de Destilacion |
| k : integer := 0; -- Variable auxiliar para el bucle del Sistema de Seguridad |
|  |
| Tiempo1, Tiempo2, Tiempo3 : Time;  -- Variable para controlar el tiempo de cada modulo |
| Limite1, Limite2, Limite3 : Time;  -- Variable para almacenar el limite temporal de cada modulo |
| C1 : Time\_Span := Milliseconds (10);  -- Variable para almacenar el tiempo de computo de un modulo |
| C2 : Time\_Span := Milliseconds (10);  -- Variable para almacenar el tiempo de computo de un modulo |
| T1 : Time\_Span := Milliseconds (100);  -- Variable para almacenar el de periodo de un modulo |
| T2 : Time\_Span := Milliseconds (200);  -- Variable para almacenar el de periodo de un modulo |
| T3 : Time\_Span := Milliseconds (50);  -- Variable para almacenar el de periodo de un modulo |
|  |
| P\_CS : System.Priority := 2; |
| P\_MD : System.Priority := 1; |
| P\_SS : System.Priority := 3; |
|  |
| task type CS is |
| pragma Priority (P\_CS); |
| end CS; |
|  |
| type REF\_CS is access CS; |
|  |
| -- Tarea del campo solar |
| task body CS is |
| begin |
| Tiempo1 := clock; |
| for i in 1 .. reps loop |
| -- Añadimos el timepo de computo |
| limite1 := Tiempo1 + C1; |
| -- Ejecutamos los pasos que sigue el campo solar |
| Modulos.CampoSolar; |
| -- Añadimos el periodo |
| Tiempo1 := Tiempo1 + T1; |
| end loop; |
| end; |
|  |
| task type MD is |
| pragma Priority (P\_MD); |
| end MD; |
| type REF\_MD is access MD; |
|  |
| task body MD is |
| begin |
| Tiempo2 := clock; |
| for i in 1 .. reps loop |
| -- Añadimos el timepo de computo |
| limite2 := Tiempo2 + C2; |
| -- Ejecutamos los pasos que sigue el modulo de destilacion |
| Modulos.ModuloDestilacion; |
| -- Añadimos el periodo |
| Tiempo2 := Tiempo2 + T2; |
| end loop; |
| end; |
|  |
| task type SS is |
| pragma Priority (P\_SS); |
| end SS; |
| type REF\_SS is access SS; |
|  |
| task body SS is |
| begin |
| Tiempo3 := clock; |
| for i in 1 .. reps loop |
| limite3 := Tiempo3; |
| -- Ejecutamos los pasos que sigue el sistema de seguridad |
| Modulos.SistemaDeSeguridad; |
| -- Añadimos el periodo |
| Tiempo3 := Tiempo3 + T3; |
| end loop; |
| end; |
|  |
|  |
| begin |
|  |
| declare |
| m\_cs : REF\_CS := new CS; |
| m\_md : REF\_MD := new MD; |
| m\_ss : REF\_SS := new SS; |
| begin |
| null; |
| end; |
|  |
| end mainHerenciaPrioridad; |

# Anexo. Clases utilizadas para la implementación

## STR\_CPU.ads

|  |  |
| --- | --- |
| pragma Task\_Dispatching\_Policy (FIFO\_Within\_Priorities); | |
| pragma Queuing\_Policy (Priority\_Queuing); | |
| pragma Locking\_Policy (Ceiling\_Locking); | |
|  |  |
| with Ada.Numerics.Float\_Random; use Ada.Numerics.Float\_Random; | |
|  |  |
|  | package STR\_CPU is |
|  | sampling\_time : Float := 0.1; |
|  | task CPU is |
|  | pragma Priority(10); |
|  | entry run; |
|  | end CPU; |
|  | private |
|  | quantum : constant Float := 0.001; |
|  | end STR\_CPU; |

## STR\_CPU.adb

|  |  |  |
| --- | --- | --- |
| pragma Task\_Dispatching\_Policy (FIFO\_Within\_Priorities); | | |
| pragma Queuing\_Policy (Priority\_Queuing); | | |
| pragma Locking\_Policy (Ceiling\_Locking); | | |
|  |  | |
| with Ada.Text\_IO; | |
| with Ada.Real\_Time; use Ada.Real\_Time; | |
|  | |
| package body STR\_CPU is | |
| task body CPU is | |
| blocked : Boolean := False; | |
| i : Integer := 0; | |
| begin | |
| loop | |
| select | |
| when blocked = False => | |
| accept run do | |
| blocked := True; | |
| i := i + 1; | |
| delay Duration(sampling\_time-quantum/2.0); | |
| blocked := False; | |
| end run; | |
| or | |
| terminate; | |
| end select; | |
| delay Duration(quantum); | |
| end loop; | |
| end CPU; | |
| end STR\_CPU; | |

## STR\_Chrono.ads

|  |
| --- |
| pragma Task\_Dispatching\_Policy (FIFO\_Within\_Priorities); |
| pragma Queuing\_Policy (Priority\_Queuing); |
| pragma Locking\_Policy (Ceiling\_Locking); |
|  |
| with Ada.Numerics.Float\_Random; use Ada.Numerics.Float\_Random; |
|  |
| package STR\_Chrono is |
| procedure Count(id : Integer); |
| procedure PaintSimple; |
| procedure Paint; |
| private |
| N : constant Integer := 3; |
| M : constant Integer := 50; |
|  |
| data : array(1..N,1..M) of Boolean := (others => (others => False)); |
|  |
| procedure Shift; |
| end STR\_Chrono; |

## STR\_Chrono.adb

|  |
| --- |
| pragma Task\_Dispatching\_Policy (FIFO\_Within\_Priorities); |
| pragma Queuing\_Policy (Priority\_Queuing); |
| pragma Locking\_Policy (Ceiling\_Locking); |
|  |
| with Ada.Real\_Time; use Ada.Real\_Time; |
| with Ada.Numerics.Float\_Random; use Ada.Numerics.Float\_Random; |
| with Ada.Text\_IO; use Ada.Text\_IO; |
|  |
| with STR\_CPU; use STR\_CPU; |
| with STR\_Chrono; |
|  |
| package body STR\_Tasking is |
| procedure Tiempo\_Computo(tmin, tmax, id : Integer) is |
| waiting\_time : Float; |
| ttotal : Float := 0.0; |
| counter : Integer := 0; |
| begin |
| Reset(G); |
|  |
| waiting\_time := Float(tmin) + Float(tmax-tmin)\*Random(G); |
|  |
| loop |
| exit when ttotal > waiting\_time/1000.0; |
|  |
| ttotal := ttotal+sampling\_time; |
|  |
| counter := counter + 1; |
|  |
| if counter >= Integer(1.0/sampling\_time) then |
| --Put\_Line("Executing => Id:" & id'Img & ". Cycle execution time:" & Integer(ttotal)'Img); |
| STR\_Chrono.Count(id); |
| STR\_Chrono.Paint; |
| counter := 0; |
| end if; |
|  |
| CPU.run; |
| end loop; |
| end Tiempo\_Computo; |
| end STR\_Tasking; |

## STR\_Tasking.ads

|  |
| --- |
| pragma Task\_Dispatching\_Policy (FIFO\_Within\_Priorities); |
| pragma Queuing\_Policy (Priority\_Queuing); |
| *pragma Locking\_Policy (Ceiling\_Locking);* |
|  |
| *with Ada.Numerics.Float\_Random; use Ada.Numerics.Float\_Random;* |
|  |
| *package STR\_Tasking is* |
| *procedure Tiempo\_Computo(tmin, tmax, id : Integer);* |
| *private* |
| *G : Generator;* |
| *end STR\_Tasking;* |

## control.ads

|  |
| --- |
| with tipos\_datos |
| , Ada.Text\_IO; |
|  |
| use tipos\_datos |
| , Ada.Text\_IO; |
|  |
| package control is |
|  |
| function ControlarSC1 (t\_oSt1 : temperatura; |
| t\_oSt2 : temperatura; |
| t\_st2 : temperatura; |
| t\_st4 : temperatura; |
| t\_sr1 : radiacion; |
| t\_bomb : out estadoBomba) return string; |
|  |
| function ControlarSC2 (t\_oSt2 : temperatura; |
| t\_st3 : temperatura; |
| t\_sd1 : caudal; |
| t\_bomb : out estadoBomba) return string; |
|  |
| end control; |

## control.adb

|  |
| --- |
| with Ada.Text\_IO; |
| use Ada.Text\_IO; |
|  |
| package body control is |
|  |
| t\_bomb : estadoBomba := Off; |
|  |
| function ControlarSC1 (t\_oSt1 : temperatura; |
| t\_oSt2 : temperatura; |
| t\_st2 : temperatura; |
| t\_st4 : temperatura; |
| t\_sr1 : radiacion; |
| t\_bomb : out estadoBomba) return string is |
| begin |
| t\_bomb := On; |
|  |
| return ("Controlando SC1(" |
| & "), estado de la bomba: " & t\_bomb'image); |
| end; |
|  |
|  |
| function ControlarSC2 (t\_oSt2 : temperatura; |
| t\_st3 : temperatura; |
| t\_sd1 : caudal; |
| t\_bomb : out estadoBomba) return string is |
| begin |
| t\_bomb := On; |
|  |
| return ("Controlando SC2(" |
| & "), estado de la bomba: " & t\_bomb'image); |
| end; |
|  |
| end control; |

## io.ads

|  |
| --- |
| with tipos\_datos; |
| use tipos\_datos; |
|  |
| with semaforos; |
| use semaforos; |
|  |
| package IO is |
|  |
| Mutex : Semaphore; |
|  |
| m\_tiempoLectura : Integer := 2; |
| m\_tiempoCalculo : Integer := 2; |
| m\_tiempoEscritura : Integer := 3; |
| m\_tiempoEnvio : Integer := 3; |
|  |
| function Leer (t\_temp : out Temperatura) return string; |
| function Leer (t\_caud : out Caudal) return string; |
| function Leer (t\_rad : out Radiacion) return string; |
|  |
| function Escribir (t\_temp : in Temperatura) return string; |
| function Escribir (t\_caud : in Caudal) return string; |
| function Escribir (t\_rad : in Radiacion) return string; |
|  |
| function Enviar (t\_temp : in Temperatura) return string; |
| function Enviar (t\_caud : in Caudal) return string; |
| function Enviar (t\_rad : in Radiacion) return string; |
|  |
| function Verificar return string; |
|  |
| end IO; |

## io.adb

|  |
| --- |
| with Ada.Text\_IO; |
| use Ada.Text\_IO; |
|  |
| with STR\_Tasking; |
| use STR\_Tasking; |
|  |
| package body IO is |
|  |
| ----------------Leer-------------------- |
|  |
| function Leer (t\_temp : out Temperatura) return string is |
| begin |
| t\_temp := 2.0; |
| Wait(Mutex); |
| Tiempo\_Computo (0, m\_tiempoLectura, 0); |
| Signal(Mutex); |
| Return ("Leyendo temperatura: " & t\_temp'image); |
| end; |
|  |
| function Leer (t\_caud : out Caudal) return string is |
| begin |
| t\_caud := 2.0; |
| Wait(Mutex); |
| Tiempo\_Computo (0, m\_tiempoLectura, 1); |
| Signal(Mutex); |
| Return ("Leyendo caudal: " & t\_caud'image); |
| end; |
|  |
| function Leer (t\_rad : out Radiacion) return string is |
| begin |
| t\_rad := 2.0; |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoLectura, 2); |
| Signal(Mutex); |
| Return ("Leyendo radiacion: " & t\_rad'image); |
| end; |
|  |
| --------------Escribir------------------ |
|  |
| function Escribir (t\_temp : in Temperatura) return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoEscritura, 0); |
| Signal(Mutex); |
| Return ("Escribiendo temperatura: " & t\_temp'image); |
| end; |
|  |
| function Escribir (t\_caud : in Caudal) return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoEscritura, 1); |
| Signal(Mutex); |
| Return ("Escribiendo caudal: " & t\_caud'image); |
| end; |
|  |
| function Escribir (t\_rad : in Radiacion) return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoEscritura, 2); |
| Signal(Mutex); |
| Return ("Escribiendo radiacion: " & t\_rad'image); |
| end; |
|  |
| ---------------Enviar------------------- |
|  |
| function Enviar (t\_temp : in Temperatura) return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoEnvio, 0); |
| Signal(Mutex); |
| Return ("Enviando temperatura: " & t\_temp'image); |
| end; |
|  |
| function Enviar (t\_caud : in Caudal) return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoEnvio, 1); |
| Signal(Mutex); |
| Return ("Enviando caudal: " & t\_caud'image); |
| end; |
|  |
| function Enviar (t\_rad : in Radiacion) return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoEnvio, 2); |
| Signal(Mutex); |
| Return ("Enviando radiacion: " & t\_rad'image); |
| end; |
|  |
| -------------Verificar----------------- |
|  |
| function Verificar return string is |
| begin |
| Wait(Mutex); |
| Tiempo\_Computo(0, m\_tiempoCalculo, 3); |
| Signal(Mutex); |
| Return ("Comprobando temperaturas"); |
| end; |
|  |
| end IO; |

## modulos.ads

|  |
| --- |
| with Tipos\_Datos, |
| Ada.Text\_IO; |
| use Tipos\_Datos, |
| Ada.Text\_IO; |
|  |
| package body modulos is |
|  |
| procedure AlmacenamientoDeDatos is |
| begin |
| null; |
| end AlmacenamientoDeDatos; |
|  |
| procedure CampoSolar is |
| begin |
| Put\_Line("CS: " & io.Leer (m\_temp)); |
| Put\_Line("CS: " & control.ControlarSC1 (m\_temp, m\_temp, m\_temp, m\_temp, m\_rad, t\_bomb)); |
| Put\_Line("CS: " & io.Escribir (m\_temp)); |
| Put\_Line("CS: " & io.Enviar(m\_temp)); |
|  |
| Visualizador; |
|  |
| end CampoSolar; |
|  |
| procedure ModuloDestilacion is |
| begin |
| Put\_Line("MD: " & io.Leer (m\_caudal)); |
| Put\_Line("MD: " & control.ControlarSC2 (m\_temp, m\_temp, m\_caudal, t\_bomb)); |
| Put\_Line("MD: " & io.Escribir (m\_temp)); |
| Put\_Line("MD: " & io.Enviar (m\_temp)); |
|  |
| Visualizador; |
|  |
| end ModuloDestilacion; |
|  |
| procedure SistemaDeSeguridad is |
| begin |
| Put\_Line("SS: " & io.Leer (m\_temp)); |
| Put\_Line("SS: " & io.Leer (m\_temp)); |
| Put\_Line("SS: " & io.Verificar); |
| Put\_Line("SS: " & io.Enviar (m\_temp)); |
|  |
| Visualizador; |
|  |
| end SistemaDeSeguridad; |
|  |
| procedure Visualizador is |
| begin |
| Put\_Line("V: " & io.Escribir (m\_temp)); |
| Put\_Line("V: " & io.Escribir (m\_caudal)); |
| Put\_Line("V: " & io.Escribir (m\_rad)); |
| end Visualizador; |
|  |
| end modulos; |

## modulos.adb

|  |
| --- |
| with Tipos\_Datos, |
| Ada.Text\_IO; |
| use Tipos\_Datos, |
| Ada.Text\_IO; |
|  |
| package body modulos is |
|  |
| procedure AlmacenamientoDeDatos is |
| begin |
| null; |
| end AlmacenamientoDeDatos; |
|  |
| procedure CampoSolar is |
| begin |
| Put\_Line("CS: " & io.Leer (m\_temp)); |
| Put\_Line("CS: " & control.ControlarSC1 (m\_temp, m\_temp, m\_temp, m\_temp, m\_rad, t\_bomb)); |
| Put\_Line("CS: " & io.Escribir (m\_temp)); |
| Put\_Line("CS: " & io.Enviar(m\_temp)); |
|  |
| Visualizador; |
|  |
| end CampoSolar; |
|  |
| procedure ModuloDestilacion is |
| begin |
| Put\_Line("MD: " & io.Leer (m\_caudal)); |
| Put\_Line("MD: " & control.ControlarSC2 (m\_temp, m\_temp, m\_caudal, t\_bomb)); |
| Put\_Line("MD: " & io.Escribir (m\_temp)); |
| Put\_Line("MD: " & io.Enviar (m\_temp)); |
|  |
| Visualizador; |
|  |
| end ModuloDestilacion; |
|  |
| procedure SistemaDeSeguridad is |
| begin |
| Put\_Line("SS: " & io.Leer (m\_temp)); |
| Put\_Line("SS: " & io.Leer (m\_temp)); |
| Put\_Line("SS: " & io.Verificar); |
| Put\_Line("SS: " & io.Enviar (m\_temp)); |
|  |
| Visualizador; |
|  |
| end SistemaDeSeguridad; |
|  |
| procedure Visualizador is |
| begin |
| Put\_Line("V: " & io.Escribir (m\_temp)); |
| Put\_Line("V: " & io.Escribir (m\_caudal)); |
| Put\_Line("V: " & io.Escribir (m\_rad)); |
| end Visualizador; |
|  |
| end modulos; |

## semáforos.ads

|  |
| --- |
| package Semaforos is |
|  |
| type Semaphore (Initial : Natural := 1) is limited private; |
| procedure Wait (S : in out Semaphore); |
| procedure Signal (S : in out Semaphore); |
| private |
| protected type Semaphore (Initial : Natural := 1) is |
| entry Wait; |
| procedure Signal; |
| private |
| Count : Natural := Initial; |
| end Semaphore; |
|  |
| end Semaforos; |

## semáforos.adb

|  |
| --- |
| with Ada.Text\_IO; |
| use Ada.Text\_IO; |
|  |
| package body Semaforos is |
|  |
| procedure Wait (S : in out Semaphore) is |
| begin |
| S.Wait; |
| end Wait; |
| procedure Signal (S : in out Semaphore) is |
| begin |
| S.Signal; |
| end Signal; |
| protected body Semaphore is |
| entry Wait when Count > 0 is |
| begin |
| Count := Count |
| - 1; |
| end Wait; |
| procedure Signal is |
| begin |
| Count := Count + 1; |
| end Signal; |
| end Semaphore; |
|  |
|  |
| end Semaforos; |

## tipos\_datos.ads

|  |
| --- |
| package Tipos\_Datos is |
|  |
| type Temperatura is new float; |
| type Caudal is new float; |
| type Radiacion is new float; |
|  |
| type EstadoBomba is (On, Off); |
|  |
| end Tipos\_Datos; |