Programación Concurrente 2019

Explicación práctica Rendezvous (ADA)

Facultad de Informática

UNLP

El lenguaje ADA

- Desarrollado por el Departamento de Defensa de USA para que sea el estándar en programación de aplicaciones de defensa (desde sistemas de Tiempo Real a grandes sistemas de información).
- Desde el punto de vista de la concurrencia, un programa Ada tiene ***tasks*** (tareas) que pueden ejecutar independientemente y contienen primitivas de sincronización.
- Los puntos de invocación (entrada) a una tarea se denominan ***entrys*** y están especificados en la parte visible (encabezado de la tarea).
- Una tarea puede decidir si acepta la comunicación con otro proceso, mediante la primitiva ***accept***.
- Se puede declarar un ***type task***, y luego crear instancias de procesos (tareas) identificado con dicho tipo (arreglo, puntero, instancia simple).

Tasks

- La forma más común de especificación de task es:

  TASK *nombre* **IS**
  declaraciones de ENTRYs
  end;

- La forma más común de cuerpo de task es:

  TASK BODY *nombre* **IS**
  declaraciones locales
  BEGIN
  sentencias
  END *nombre***;**

- Una especificación de TASK define una única tarea.
- Una instancia del correspondiente ***task body*** se crea en el bloque en el cual se declara el TASK.

Sincronización
Entry

- El ***rendezvous*** es el principal mecanismo de sincronización en Ada y también es el mecanismo de comunicación primario.
- ***Entry:***
  - Declaración de ***entry simples y familia de entry*** (parámetros IN, OUT y IN OUT)**.**

```
TASK nombre IS
        ENTRY e1;
        ENTRY e2 (p1: IN integer; p2: OUT char; p3: IN OUT float);
end;
```

Los entry's funcionan de manera semejante a los procedimientos: solo pueden recibir o enviar información por medio de los parámetros del entry. *No retornan valores como las funciones.*

Sincronización
Call: *Entry Call*

- ***Entry:***
  - ***Entry call.*** La ejecución demora al llamador hasta que la operación termine (o aborte o alcance una excepción).
  - ***Entry call condicional (SELECT-ELSE)*** :

```
select entry call;
        sentencias adicionales;
else
        sentencias;
end select;
```

  - ***Entry call temporal (SELECT-OR DELAY):***

```
select entry call;
        sentencias adicionales;
or delay tiempo
        sentencias;
end select;
```

Sincronización

Sentencia de Entrada: *Accept*

- La tarea que declara un entry sirve llamados al entry con *accept*:

  accept *nombre* **(parámetros formales) do**

  sentencias

  end *nombre*;

- Demora la tarea hasta que haya una invocación, copia los parámetros reales en los parámetros formales, y ejecuta las sentencias. Cuando termina, los parámetros formales de salida son copiados a los parámetros reales. Luego ambos procesos continúan.

- Ejemplo:

  accept e2 (p1: IN integer; p2: OUT char; p3: IN OUT float**) do**

  sentencias

  end *e2*;

Sincronización

Sentencia de Entrada: *Accept*

- Es posible que una tarea tenga más de un entry call pendiente.
- La ***sentencia wait selectiva*** soporta comunicación guardada.

select *when* $B_1 \Rightarrow$ ***accept*** E_1; sentencias$_1$
or **...**
or ***when*** $B_n \Rightarrow$ ***accept*** E_n; sentencias$_n$
end select;

  - Cada línea se llama ***alternativa***. Las cláusulas ***when*** son opcionales.
  - Puede contener una alternativa ***else, or delay.***
  - Uso de atributos del entry: ***count.***

En los SELECT no es posible mezclar entry call con accept's

Ejemplo

Atención en un banco

Modele la atención de un banco con un único empleado. Los clientes llegan y son atendidos de acuerdo al orden de llegada.

Ejemplo 1

Atención en un banco

```
Procedure Banco is
TASK Type Cliente;

TASK Empleado IS
    entry atencion;
end Empleado;

clientes: array (1..C) of Cliente;

TASK Body Cliente IS
Begin
       Empleado.atencion;
End Cliente;
```

```
TASK Body Empleado IS
Begin
    loop
          Accept atención do
                -- atender
           End atención
    End loop;
End Empleado;

Begin
      Null;
End Banco
```

Si los clientes esperan a lo sumo 10 minutos para ser atendidos, ¿qué modificaciones debería hacer?

Ejemplo 2

Atención en un banco

```
Procedure Banco2 is
TASK Type Cliente;

TASK Type Empleado IS
   entry atencion;
end Empleado;

clientes: array (1..C) of Cliente;

TASK Body Cliente IS
Begin
   SELECT
      Empleado.atencion;
   OR DELAY 600.0
       Null;
   End SELECT;
End Cliente;
```

```
TASK Body Empleado IS
Begin
   loop
      Accept atención do
            -- atender
      End atención
   End loop;
End Empleado;

Begin
    Null;
End Banco2;
```

Si los clientes no son atendidos inmediatamente, entonces se retiran. ¿Qué modificaciones debería hacer?

Ejemplo 3

Atención en un banco

```
Procedure Banco3 is
TASK Type Cliente;

TASK Type Empleado IS
   entry atencion;
end Empleado;

clientes: array (1..C) of Cliente;

TASK Body Cliente IS
Begin
   SELECT
      Empleado.atencion;
   ELSE
      Null;
   End SELECT;
End Cliente;
```

```
TASK Body Empleado IS
Begin
   loop
      Accept atención do
            -- atender
      End atención
   End loop;
End Empleado;

Begin
    Null;
End Banco3;
```

Si ahora hay 2 tipos de clientes (regular y prioritario). ¿Qué modificaciones debería hacer?

Ejemplo 4

Atención en un banco

```
Procedure Banco4 is
TASK Type Cliente;
TASK Type ClienteProritario;

TASK Empleado IS
   entry atencion;
   entry atencionPrioritaria;
end Empleado;

clientes: array (1..C) of Cliente;
clientesP: array (1..D) of ClientePrioritario;

TASK Body Cliente IS
Begin
     Empleado.atencion;
End Cliente;
```

```
TASK Body ClientePrioritario IS
Begin
      Empleado.atencionPrioritaria;
End ClientePrioritario;


TASK Body Empleado IS
Begin
   loop
      SELECT
         When (atencionPrioritaria'count == 0) =>
            Accept atención do
               -- atender
            End atención
      OR
         Accept atenciónPrioritaria do
               -- atender
         End atención
       End SELECT;
   End loop;
End Empleado;

Begin
     Null;
End Banco4;
```

Ejemplo 5
Alocador SJN

```
Procedure SchedulerSJN is

Task Alocador_SJN is
        entry pedir (tiempo, id: IN integer);
        entry liberar;
End Alocador_SJN ;

Task Type Cliente Is
    entry Ident (A: IN integer);
    entry usar;
End Cliente;

ArrC: array (1..C) of Cliente;

Task Body Cliente Is
    id: integer;   tiempo: integer;
BEGIN
    ACCEPT Ident (A : IN integer) do
        id := A;
    End Identificar;
    loop
        //trabaja y determina el valor de tiempo
        Alocador_SJN.pedir(tiempo, id);
        Accept usar;
        //Usa el recurso
        Alocador_SJN.liberar;
    end loop;
End Cliente;
```

```
Task Body Alocador_SJN is
    libre: boolean := true;
    espera: colaOrdenada;
    tiempo, aux: integer;
Begin
  loop
    aux := -1;
    select
      accept Pedir (tiempo, id: IN integer) do
        if (libre) then  libre:= false;  aux := id;
        else agregar(espera, (id, tiempo));  end if;
      end Pedir;
    or   accept  liberar;
      if (empty (espera)) then libre := true;
      else sacar(espera, (aux, tiempo);  end if;
    end select;
    if (aux <> -1) then ArrC(aux).usar;  end if;
  end loop;
End Alocador_SJN ;

Begin
   for i in 1..C loop     ArrC(i).ident (i);    end loop;
End  SchedulerSJN;
```

Ejemplo 6

Contar ocurrencias

Se debe modelar un buscador para contar la cantidad de veces que aparece un número dentro de un vector distribuido entre las ***N*** tareas ***contador***. Además existe un administrador que decide el número que se desea buscar y se lo envía a los ***N contadores*** para que lo busquen en la parte del vector que poseen.

```
Procedure ContadorOcurrencias is
Task Admin;
Task type Contador is
      entry Contar (num: in integer; res: out integer);
End contador;

ArrC: array (1..N) of Contador;

Task body Admin is
     num: integer := elegirNumero;
     parcial, total: integer := 0;
Begin
     for i in 1..N loop
          ArrC(i).Contar (num, parcial);
          total:= total + parcial;
     end loop;
End Admin;
```

```
Task body Contador is
     vec: array (1..V) of integer := InicializarVector;
     cant: integer :=0;
Begin
     Accept Contar(num: in integer; res: out integer) do
          for i in 1..V loop
               if (vec(i) = num) then
                    cant:=cant+1;
               end if;
          end loop;
          res := cant;
     end contar;
End contador;

Begin
   null;
End ContadorOcurrencias;
```

SECUENCIAL

Ejemplo 6

Contar ocurrencias

```
Procedure ContadorOcurrencias is

Task Admin is
      entry Resultado (res: in integer);
End admin;
Task type Contador is
     entry Contar (num: in integer);
End contador;

ArrC: array (1..N) of Contador;

Task body Admin is
    num: integer := elegirNumero;
    total: integer := 0;
Begin
    for i in 1..N loop
        ArrC(i).Contar (num);
    end loop;
    for i in 1..N loop
        accept Resultado (res: in integer) do
              total:= total + res;
        end Resultado;
    end loop;
End Admin;
```

```
Task body Contador is
    vec: array (1..V) of integer := InicializarVector;
    valor, cant: integer :=0;
Begin
    Accept Contar(num: in integer) do
        valor := num;
    end contar;
    for i in 1..V loop
        if (vec(i) = valor) then
            cant:=cant+1;
        end if;
    end loop;
    Admin.Resultado(cant);
End contador;

Begin
   null;
End ContadorOcurrencias;
```

Solución aceptable pero con posible demora innecesaria

Ejemplo 6

Contar ocurrencias

```
Procedure ContadorOcurrencias is
Task Admin is
       entry Valor (num: out integer);
       entry Resultado (res: in integer);
End admin;

Task body Admin is
     numero: integer := elegirNumero;
     total: integer := 0;
Begin
     for i in 1..2*N loop
          select
               accept Valor (num: out integer) do
                     num := numero;
               end Valor;
          or
               accept Resultado (res: in integer) do
                  total:= total + res;
               end Resultado;
          end select;
     end loop;
End Admin;

Task type Contador;
ArrC: array (1..N) of Contador;


Task body Contador is
     vec: array (1..V) of integer := InicializarVector;
     valor, cant: integer :=0;
Begin
     Admin.valor(valor);

     for i in 1..V loop
          if (vec(i) = valor) then
               cant:=cant+1;
          end if;
     end loop;

     Admin.Resultado(cant);
End contador;


Begin
   null;
End ContadorOcurrencias;
```