Sistemas Operativos

Uso de ADA y Mailbox avanzado

Curso 2025

Facultad de Ingeniería, UDELAR

Agenda

1. ADA: Equivalencia con semáforos
2. ADA: Equivalencia con monitores
3. ADA: Equivalencia con mailbox
4. ADA: Problemas clásicos de concurrencia
5. ADA: Errores comunes
6. Mailbox: Errores comunes
7. Mailbox: Problemas avanzados

ADA: Equivalencia con semáforos

Semáforos con ADA

```
task type SEMAFORO is
    entry INIT(v: Integer);
    entry P;
    entry V;
end SEMAFORO;
```

```
task body SEMAFORO is
    valor: Integer;
begin
    accept INIT(v: Integer) DO
        valor := v;
    end INIT;
    loop
        select
            when valor > 0 =>
                accept P;
                valor := valor - 1;
        or
            accept V;
            valor := valor + 1;
        end select;
    end loop;
end SEMAFORO;
```

ADA: Equivalencia con monitores

Monitores con ADA

```
procedure function_N()
    MONITOR.ENTRAR();
    ...
    MONITOR.SALIR();
end

procedure wait()
    MONITOR.SALIR();
    MONITOR.WAIT();
    MONITOR.ENTRAR();
end
procedure signal()
    MONITOR.SIGNAL();
end

task type MONITOR is
    entry ENTRAR;
    entry SALIR;
    entry SIGNAL;
    entry WAIT;
end MONITOR;
```

```
task body MONITOR is
    libre:bool;
begin
    libre := True;
    loop
        select
            when libre =>
                accept ENTRAR;
                libre := False;
        or
            accept SALIR;
            libre := True;
        or
            accept SIGNAL;
            select
                accept WAIT;
            else
            end select;
        end select;
    end loop;
end MONITOR;
```

ADA: Equivalencia con mailbox

Mailbox infinito con ADA

```
task type MAILBOX is
  entry ENVIAR(m: Integer);
  entry RECIBIR(out m: Integer);
end MAILBOX;
```

```
task body MAILBOX is
  mensajes: cola;
begin
  loop
    select
      when size(mensajes) > 0 =>
        accept RECIBIR(out m: Integer) do
          m := remove(mensajes);
        end RECIBIR;
    or
      accept ENVIAR(m: Integer) do
        add(mensajes, m);
      end ENVIAR;
    end select;
  end loop;
end MAILBOX;
```

ADA: Problemas clásicos de concurrencia

Problema de Alicia y Bernardo

```
task PATIO is
    entry ENTRAR;
    entry SALIR;
end PATIO;
task body PATIO is
begin
    loop
        accept ENTRAR;
        accept SALIR;
    end loop;
end PATIO;

task ALICIA;
task body ALICIA is
    loop
        PATIO.ENTRAR();
        pasear_perro();
        PATIO.SALIR();
    end loop;
end ALICIA;
```

```
task BERNARDO;
task body BERNARDO is
    loop
        PATIO.ENTRAR();
        pasear_perro();
        PATIO.SALIR();
    end loop;
end BERNARDO;

begin
    -- procedimiento principal
end;
```

Problema de productor-consumidor (tamaño N)

```
task BUFFER is
  entry AGREGAR(m: Integer);
  entry SACAR(out m: Integer);
end BUFFER;
task body BUFFER is
  contenido: cola;
begin
  loop
    select
       when size(contenido) < N =>
         accept AGREGAR(m: Integer) do
           add(contenido, m);
         end AGREGAR;
    or
       when size(contenido) > 0 =>
         accept SACAR(out m: Integer) do
           m := remove(contenido);
         end SACAR;
    end select;
  end loop;
end BUFFER;
```

```
task type PRODUCTOR;
task body PRODUCTOR is
begin
  BUFFER.AGREGAR(producir());
end PRODUCTOR;

task type CONSUMIDOR;
task body CONSUMIDOR is
  p: Integer;
begin
  BUFFER.SACAR(p)
  consumir(p);
end CONSUMIDOR;

ps: array(NP) of PRODUCTOR;
cs: array(NC) of CONSUMIDOR;

begin
  -- procedimiento principal
end;
```

Problema de lectores-escritores (prioridad escritores) I

```
task DOC is
  entry INI_LECT;
  entry FIN_LECT;
  entry INI_ESCR;
  entry FIN_ESCR;
end DOC;
task body DOC is
  cantLect: Integer;
  escritor: Boolean;
begin
  cantLect := 0;
  escritor := False;
  loop
    select
      when cantLect = 0
           and not escritor =>
        accept INI_ESCR;
        escritor := True;
      or
        accept FIN_ESCR;
        escritor := False;
      or
        when INI_ESCR'Count = 0
             and not escritor =>
          accept INI_LECT;
          cantLect := cantLect + 1;
      or
        accept FIN_LECT;
        cantLect := cantLect - 1;
    end select;
  end loop;
end DOC;
```

Problema de lectores-escritores (prioridad escritores) II

```
task type LECTOR;
task body LECTOR is
begin
    loop
        DOC.INI_LECT();
        leer();
        DOC.FIN_LECT();
    end loop
end LECTOR;

task type ESCRITOR;
task body ESCRITOR is
begin
    loop
        DOC.INI_ESCR();
        escribir();
        DOC.FIN_ESCR();
    end loop
end ESCRITOR;
```

```
ls: array(NL) of LECTOR;
es: array(NE) of ESCRITOR;

begin
    -- procedimiento principal
end;
```

Problema de N filósofos comensales I

```
task MESA is
  entry ENTRAR;
  entry SALIR;
end MESA;

task body MESA is
  cant: Integer;
begin
  cant := 0;
  loop
    select
      when cant < N-1 =>
        accept ENTRAR;
        cant := cant + 1;
      or
        accept SALIR;
        cant := cant - 1;
    end select;
  end loop;
end MESA;
```

```
task type TENEDOR is
  entry TOMAR;
  entry DEJAR;
end TENEDOR;

task body TENEDOR is
begin
  loop
    accept TOMAR;
    accept DEJAR;
  end loop;
end TENEDOR;
```

Problema de N filósofos comensales II

```
task type FILOSOFO is
  entry INIT(i: Integer);
end FILOSOFO;
task body FILOSOFO is
  id: Integer;
begin
  accept INIT(i: Integer) do
    id := i;
  end INIT;
  loop
    pensar();
    MESA.ENTRAR();
    T[id].TOMAR();
    T[id + 1 mod N].TOMAR();
    comer();
    T[id].DEJAR();
    T[id + 1 mod N].DEJAR();
    MESA.SALIR();
  end loop;
end FILOSOFO;
```

```
T:array(0..N-1) of TENEDOR;
F:array(0..N-1) of FILOSOFO;

begin
  for i in (0..N-1) loop
    F(i).INIT(i);
  end loop;
end;
```

ADA: Errores comunes

Errores comunes

- Usar Cobegin-Coend
  - No es necesario en ada porque los task inician solos al comenzar el programa
- Código extra en el accept
  - Durante la ejecución del cuerpo del accept el proceso que invoca está bloqueado.
  - No tiene sentido bloquearlo al actualizar variables locales de la tarea.
- Los selects no son ifs, no se usan igual y no se pueden intercambiar.
  - Luego de un select solo puede venir un accept o una guarda.

Mailbox: Errores comunes

Comunicación entre procesos

- Tengo procesos cliente (varios) y servidor (uno)
- Quiero que el cliente le envíe un mensaje al servidor y recibir una respuesta

Posible solución:

```
mb: mailbox of integer;

procedure cliente()
   var resp: integer
   enviar(mb, pedido());
   recibir(mb, resp);
   procesar_respuesta(resp);
end procedure
```

```
procedure servidor()
   var pedido, respuesta: integer
   recibir(mb, pedido);
   respuesta := procesar(pedido);
   enviar(mb, respuesta);
end procedure
```

Puedo recibir mi mismo mensaje

Otra implementación

```
   mb_pedido, mb_respuesta: mailbox of integer;

procedure cliente()
   var resp: integer
   enviar(mb_pedido, pedido());
   recibir(mb_respuesta, resp);
   procesar_respuesta(resp);
end procedure
```

```
procedure servidor()
   var pedido, respuesta: integer
   recibir(mb_pedido, pedido);
   respuesta := procesar(pedido);
   enviar(mb_respuesta, respuesta);
end procedure
```

¿Cuál es el problema?

Mezcla de mensajes: solución 1

```
   mb_pedido, mb_respuesta: mailbox of integer;
   mb_mutex: mailbox of NIL;

procedure cliente()
   var resp: integer
   recibir(mb_mutex);
   enviar(mb_pedido, pedido());
   resp := recibir(mb_respuesta);
   enviar(mb_mutex, NIL);
   procesar_respuesta(resp);
end procedure

procedure servidor()
   var pedido, respuesta: integer
   pedido := recibir(mb_pedido);
   respuesta := procesar(pedido);
   enviar(mb_respuesta, respuesta);
end procedure

   Begin
      enviar(mb_mutex, NIL);
   End
```

Mezcla de mensajes: solución 2

- Lo anterior no funciona si tengo varios servidores.
- Pero si tengo un número acotado de clientes:

```
mb_pedido: mailbox of {integer, integer};
mb_respuesta: array [1..n] of mailbox of integer;
```

```
procedure cliente(id: integer)
   var resp: integer
   enviar(mb_pedido, {pedido(), id});
   resp := recibir(mb_respuesta[id]);
   procesar_respuesta(resp);
end procedure
```

```
procedure servidor()
   var pedido, resp: integer
   var id: integer
   {pedido, id}:=recibir(mb_pedido);
   resp := procesar(pedido);
   enviar(mb_respuesta[id], resp);
end procedure
```

Asignación de números a procesos

- Forma fácil de asignar números únicos a procesos además de controlar el número

```
procedure cliente()
   var mi_numero: integer
   mi_numero := recibir(numeros);
   hago_cosas;
   enviar(numeros, mi_numero);
end procedure
```

```
numeros: mailbox of integer;
Begin
   for i := 1..N do
      enviar(numeros, i);
   end for
End
```

Mailbox: Problemas avanzados

Problema de lectores-escritores (con prioridad escritores)

```
mtxdoc, mtxprio: mailbox of NIL;
cantlect, cantesc: mailbox of integer;
```

```
procedure lector()
   recibir(mtxprio, m);
   recibir(cantlect, cant);
   if cant = 0 then
      recibir(mtxdoc, m);
   end if
   enviar(cantlect, cant + 1);
   enviar(mtxprio, NIL);
   leer();
   recibir(cantlect, cant);
   if cant = 1 then
      enviar(mtxdoc, NIL);
   end if
   enviar(cantlect, cant - 1);
end procedure
```

```
procedure escritor()
   recibir(cantesc, esc);
   if esc = 0 then
      recibir(mtxprio, m);
   end if
   enviar(cantesc, esc + 1);
   recibir(mtxdoc, m);
   escribir();
   enviar(mtxdoc, NIL);
   recibir(cantesc, esc);
   if esc = 1 then
      enviar(mtxprio, NIL);
   end if
   enviar(cantesc, esc - 1);
end procedure
```

Problema de lectores-escritores (con prioridad escritores)

```
Begin
   enviar(mtxdoc, NIL);
   enviar(mtxprio, NIL);
   enviar(cantlect, 0);
   enviar(cantesc, 0);
   Cobegin
      lector();
      ...
      lector();
      escritor();
      ...
      escritor();
   Coend
End
```