Explicación Práctica de Pasaje de Mensajes Asincrónicos

Ejercicios

Links a los archivos con audio

El archivo en formato MP4 de la explicación con audio se encuentra comprimido en el siguiente link:

- https://drive.google.com/u/0/uc?id=1xF55YT57IsS3aDCKrDTQdHSpZ4E228B6&export=download

Pasaje de Mensajes Asincrónicos (PMA)

- Los programas se componen **SÓLO** de procesos y canales (NO EXISTEN LAS VARIABLES COMPARTIDAS).

- Los canales actúan como "colas" de mensajes enviados y no recibidos. Son de tipo mailbox (todos los procesos los pueden usar para enviar o recibir mensajes).

- Los procesos interactúan entre ellos ÚNICAMENTE por medio del envío de mensajes (tanto para comunicación como para sincronización por condición).

- No se requiere sincronización por exclusión mutua ya que no existen las variables compartidas.

Sintaxis

- Declaración de canales: se deben declarar los canales indicando la estructura de datos de los mensajes, puede ser:

  - Un canal:

    chan nombrecanal (tipoDato)

  - Un arreglo (de una o mas dimensiones) de canales:

    chan nombrearreglo[1..m](tipoDato)

Sintaxis

- Sentencias de comunicación (*send* / *receive*): el uso de cada canal es atómico, por lo que no se harán al mismo tiempo 2 operaciones (send y/o receive) sobre el mismo canal.
  - Envío (send): la operación es no bloqueante, deposita el mensaje al final del canal y continua su ejecución.

    send nombrecanal (mensaje)

    send nombrearreglo[i] (mensaje)

  - Recepción (receive): la operación es bloqueante, si el canal está vacío se demora hasta que haya al menos un mensaje en él, luego saca el primer mensaje del canal (el mas viejo).

    receive nombrecanal (Variables para el mensaje)

    receive nombrearreglo[i](Variables para el mensaje)

Sintaxis

- Consultas por mensajes pendientes (*empty*): esta función retorna un booleano que indica si el canal está o no vacío. Usar con cuidado cuando el canal tiene más de UN posible receptor.

  empty (nombrecanal)

  empty (nombrearreglo[i])

- Uso de Sentencias de Alternativa Múltiple (IF no determinístico) y Alternativas Iterativa Múltiple (DO no determinístico). Puede generar Busy Waiting que en PMA está permitido (igual hay que tratar de evitarlo).

EJERCICIO 1

En una empresa de software hay N personas que prueban un nuevo producto para encontrar errores, cuando encuentran uno generan un reporte para que un empleado corrija el error (las personas no deben reciben ninguna respuesta). El empleado toma los reportes de acuerdo al orden de llegada, los evalúan y hace las correcciones necesarias.

Lo primero es definir la estructura del programa: que procesos y como se van a comunicar

En este problema las personas sólo deben dejar su reporte y el empleado los resuelve de acuerdo al orden de llegada. Por lo tanto tendremos N procesos persona y un proceso empleado.

¿Cómo se resuelve la atención en orden de los pedidos?

Los canales actúan como COLAS de mensajes, por lo que mantienen el orden de los mismos → USAREMOS UN CANAL COMO BUFFER "ORDENADO" DE REPORTES

EJERCICIO 1

```
chan Reportes(texto);

Process Persona[id: 0..N-1]
{ texto R;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( R );
    };
}
```

```
Process Empleado
{ texto Rep;
  while (true)
    { receive Reportes ( Rep );
      resolver (Rep);
    };
}
```

Usamos el canal ***Reportes*** como buffer/cola donde se guardan ordenados los reportes dejados por las personas.
Las personas dejan su reporte en el canal y continua trabajando SIN esperar a que sea recibido el mensaje.
El empleado espera hasta que haya al menos un reporte en el canal y lo toma para resolver.

EJERCICIO 2

Hacemos una pequeña modificación al ejercicio 1, la cual es marcada en verde.

En una empresa de software hay N personas que prueban un nuevo producto para encontrar errores, cuando encuentran uno generan un reporte para que un empleado corrija el error **y esperan la respuesta del mismo.** El empleado toma los reportes de acuerdo al orden de llegada, los evalúan, hace las correcciones necesarias **y le responde a la persona que hizo el reporte**.

A diferencia del ejercicio anterior cuando la persona envía un reporte no debe seguir trabajando hasta que le responda el empleado, es decir que deben "interactuar" ambos procesos "sincronizando" de alguna manera el final de la misma.

¿Alcanza con usar sólo el canal *Reportes* tanto para enviar el reporte como para la respuesta?

NO. Se mezclarías ambas cosas en el mismo canal, por lo que el empleado podría recibir una respuesta y la persona un reporte → DEBEMOS ENVIAR LAS RESPUESTAS POR OTRO CANAL

EJERCICIO 2

Partimos de la solución del ejercicio 1 y agregamos un canal *Respuestas* para que el empleado envíe las respuestas a las personas.
Las personas, después de dejar su reporte se quedan esperando en este nuevo canal hasta que el empleado le envía la respuesta.

```
chan Reportes(texto);
chan Respuestas(texto);

Process Persona[id: 0..N-1]
{ texto R, Res;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( R );
      receive Respuestas ( Res );
    };
}
```

```
Process Empleado
{ texto Rep, Res;
  while (true)
    { receive Reportes ( Rep );
      Res = resolver (Rep);
      send Respuestas ( Res );
    };
```

¿Cómo sabe la persona si es su respuesta o la de otra persona?

EJERCICIO 2

Al usar el mismo canal *Respuestas* para todas las personas, una puede tomar la respuesta de otra, y esto no es correcto. Necesito un canal "privado" para cada persona, de esta manera el empleado manda el mensaje con la respuesta SÓLO al canal de la persona que hizo ese reporte.

```
chan Reportes(texto);
chan Respuestas[N](texto);

Process Persona[id: 0..N-1]
{ texto R, Res;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( R );
      receive Respuestas[id] ( Res );
    };
}
```

```
Process Empleado
{ texto Rep, Res;
  int idP;
  while (true)
    { receive Reportes ( Rep );
      Res = resolver (Rep);
      send Respuestas[idP] ( Res );
    };
}
```

¿Cómo sabe a quien enviarle la respuesta?

EJERCICIO 2

Para saber quien envió el reporte para luego poder mandarle la respuesta, la persona debe enviar junto con el Reporte su identificador (en el mismo mensaje).

```
chan Reportes(int, texto);
chan Respuestas[N](texto);

Process Persona[id: 0..N-1]
{ texto R, Res;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( id , R );
      receive Respuestas[id] ( Res );
    };
}
```

```
Process Empleado
{ texto Rep, Res;
  int idP;
  while (true)
    { receive Reportes ( idP , Rep );
      Res = resolver (Rep);
      send Respuestas[idP] ( Res );
    };
}
```

EJERCICIO 3

Hacemos una pequeña modificación al ejercicio 2 donde ahora son 3 los empleados para atender los reportes.

En una empresa de software hay N personas que prueban un nuevo producto para encontrar errores, cuando encuentran uno generan un reporte para que alguno de los 3 empleados corrija el error y esperan la respuesta del mismo. Los empleados toman los reportes de acuerdo al orden de llegada, los evalúan, hacen las correcciones necesarias y le responden a la persona que hizo el reporte.

Partiendo de la solución del ejercicio 2, se debe analizar si hay que hacer alguna modificación.
Como las personas deben enviar sus reportes para que cualquiera de los empleados lo resuelva, se debe seguir usando un único canal para enviar los reportes. Sólo se debería usar un canal para cada empleado si la persona manda el reporte a UN empleado en particular (pero no es este el caso).

EJERCICIO 3

Por lo tanto la solución es la misma, sólo se tendrán 3 empleados en lugar de 1.

```
chan Reportes(int, texto);
chan Respuestas[N](texto);

Process Persona[id: 0..N-1]
{ texto R, Res;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( id , R );
      receive Respuestas[id] ( Res );
    };
}
```

```
Process Empleado [id: 0..2]
{ texto Rep, Res;
  int idP;
  while (true)
    { receive Reportes ( idP , Rep );
      Res = resolver (Rep);
      send Respuestas[idP] ( Res );
    };
}
```

EJERCICIO 4

Hacemos una modificación al ejercicio 1, la cual es marcada en verde.

En una empresa de software hay N personas que prueban un nuevo producto para encontrar errores, cuando encuentran uno generan un reporte para que un empleado corrija el error (las personas no deben reciben ninguna respuesta). El empleado toma los reportes de acuerdo al orden de llegada, los evalúan y hace las correcciones necesarias**; cuando no hay reportes para atender el empleado se dedica a leer durante 10 minutos**.

En este caso el empleado no puede quedarse bloqueado haciendo un *receive* sobre el canal *Reportes* cuando está vacío, sino no podrá "leer durante 10 minutos".

¿Cómo lo hacemos?

EJERCICIO 4

Los procesos *Persona* no tienen ninguna diferencia en su funcionamiento, el cambio es "invisible" para ellos.
El empleado debe chequear antes de hacer el *receive* si hay algo en el canal, y si está vacío se pone a leer. Para esto usamos la función *empty*.

```
chan Reportes(texto);

Process Persona[id: 0..N-1]
{ texto R;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( R );
    };
}
```

```
Process Empleado
{ texto Rep;
  while (true)
    { if ( not empty (Reportes) )
        { receive Reportes ( Rep );
          resolver (Rep);
        }
      else  delay (600); //lee 10 minutos
    };
}
```

EJERCICIO 5

Hacemos una modificación al ejercicio 4, donde habrá 3 empleados.

En una empresa de software hay N personas que prueban un nuevo producto para encontrar errores, cuando encuentran uno generan un reporte para que **uno de los 3 empleados** corrija el error (las personas no deben reciben ninguna respuesta). Los empleados toman los reportes de acuerdo al orden de llegada, los evalúan y hacen las correcciones necesarias; cuando no hay reportes para atender los empleados se dedican a leer durante 10 minutos.

Partimos de la solución del ejercicio 4 poniendo 3 procesos *Empleado* en lugar de 1.

EJERCICIO 5

```
chan Reportes(texto);

Process Persona[id: 0..N-1]
{ texto R;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( R );
    };
}
```

```
Process Empleado[id: 0..2]
{ texto Rep;
  while (true)
    { if ( not empty (Reportes) )
        { receive Reportes ( Rep );
          resolver (Rep);
        }
      else  delay (600); //lee 10 minutos
    };
}
```

Posible demora innecesaria

Puede haber demora innecesaria ya que si dos o más empleados chequean por si el canal está vacío (supongamos que hay un solo mensaje) con la función *empty*, a todos les devolverá que no está vacío, por lo que más de un empleado intentará hacer el *receive* → uno lo podrá hacer y el resto se bloqueará en el *receive* cuando en realidad deberían leer por 10 minutos

EJERCICIO 5

Este problema se da porque estamos usando la función *empty* sobre un canal con múltiples receptores. Para evitar la demora innecesaria debemos:

- **No usar el *empty***: NO se puede evitar porque sino el empleado se quedaría si o si dormido en el *receive* y nunca leería.
- **El canal no tenga múltiples receptores**: podría poner un canal Reportes para cada empleado, pero esto obligaría a que la persona le entregue su reporte a UN empleado en particular (que no es lo pedido).

La única opción que queda es poner un proceso intermedio *Coordinador* que se encargue de recibir los reportes por el único canal Reportes, y que los empleados le pidan a ese proceso un reporte para resolver.

Para las personas estos cambios son "invisibles", por lo que esos procesos no se modifican.

EJERCICIO 5

Un empleado le "pide" el siguiente reporte al *Coordinador* por medio de un canal *Pedido*, el cual devolverá el reporte a atender por un canal *Siguiente*.

```
chan Reportes(texto);
chan Pedido(int);
chan Siguiente(texto);

Process Empleado[id: 0..2]
{ texto Rep;
  while (true)
    { send Pedido(id);
      if ( not empty (Siguiente) )
          { receive Siguiente ( Rep );
            resolver (Rep);
          }
      else delay (600); //lee 10 minutos
    };
}
```

Se sigue teniendo demora innecesaria ahora con el canal *Siguiente*

```
Process Coordinador
{ texto Rep;
  int idE;
  while (true)
    { receive Pedido ( idE);
      if ( not empty (Reportes) )
          { receive Reportes ( Rep );
            send Siguiente ( Rep );
          }
    };
}
```

EJERCICIO 5

Debería usar un canal "privado" para que cada empleado reciba la respuesta que es sólo para él.

```
chan Reportes(texto);
chan Pedido(int);
chan Siguiente[3](texto);

Process Empleado[id: 0..2]
{ texto Rep;
  while (true)
    { send Pedido(id);
      if ( not empty (Siguiente[id]) )
          { receive Siguiente[id] ( Rep );
            resolver (Rep);
          }
      else delay (600); //lee 10 minutos
    };
}
```

Probablemente el *Coordinador* aún no alcanzó a atender su pedido, por lo que el canal estará vacío aunque haya reportes pendientes

```
Process Coordinador
{ texto Rep;
  int idE;
  while (true)
    { receive Pedido ( idE);
      if ( not empty (Reportes) )
          { receive Reportes ( Rep );
            send Siguiente[idE] ( Rep );
          }
    };
}
```

EJERCICIO 5

El Coordinador tendrá que responderle siempre al empleado, haya o no reportes pendientes. El empleado tendrá que esperar la respuesta del Coordinador para saber si efectivamente hay o no reportes pendientes.

```
chan Reportes(texto);
chan Pedido(int);
chan Siguiente[3](texto);

Process Empleado[id: 0..2]
{ texto Rep;
  while (true)
    { send Pedido(id);
      receive Siguiente[id] ( Rep );
      if (Rep <> "VACIO") resolver (Rep)
      else delay (600); //lee 10 minutos
    };
}
```

```
Process Coordinador
{ texto Rep;
  int idE;
  while (true)
    { receive Pedido ( idE);
      if (empty (Reportes) ) Rep = "VACIO";
      else  receive Reportes ( Rep )

      send Siguiente[idE] ( Rep );
    };
}
```

EJERCICIO 5

```
chan Reportes(texto);
chan Pedido(int);
chan Siguiente[3](texto);

Process Coordinador
{ texto Rep;
  int idE;
  while (true)
    { receive Pedido ( idE);
      if (empty (Reportes) ) Rep = "VACIO";
      else  receive Reportes ( Rep )

      send Siguiente[idE] ( Rep );
    };
}
```

```
Process Empleado[id: 0..2]
{ texto Rep;
  while (true)
    { send Pedido(id);
      receive Siguiente[id] ( Rep );
      if (Rep <> "VACIO") resolver (Rep)
      else delay (600); //lee 10 minutos
    };
}

Process Persona[id: 0..N-1]
{ texto R;
  while (true)
    { R = generarReporteConProblema;
      send Reportes ( R );
    };
}
```