Práctica 4 – Pasaje de Mensajes

CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS DE PMA:

- Los canales son compartidos por todos los procesos.
- Cada canal es una cola de mensajes, por lo tanto, el primer mensaje encolado es el primero en ser atendido.
- Por ser pasaje de mensajes asincrónico el ***send*** no bloquea al emisor.
- Se puede usar la sentencia ***empty*** para saber si hay algún mensaje en el canal, pero no se puede consultar por la cantidad de mensajes encolados.
- Se puede utilizar el ***if/do*** no determinístico donde cada opción es una condición boolena donde se puede preguntar por variables locales y/o por ***empty*** de canales.

  if (cond 1) -> Acciones 1;
  □ (cond 2) -> Acciones 2;
  ….
  □ (cond N) -> Acciones N;
  end if

  De todas las opciones cuya condición sea Verdadera elige una en forma no determinística y ejecuta las acciones correspondientes. Si ninguna es verdadera sale del if/do si hacer nada.
- Se debe tratar de evitar hacer ***busy waiting*** (sólo hacerlo si no hay otra opción).
- En todos los ejercicios el tiempo debe representarse con la función ***delay***.

1. Suponga que N personas llegan a la cola de un banco. Para atender a las personas existen 2 empleados que van atendiendo de a una y por orden de llegada a las personas.

2. Se desea modelar el funcionamiento de un banco en el cual existen 5 cajas para realizar pagos. Existen P personas que desean pagar. Para esto cada una selecciona la caja donde hay menos personas esperando, una vez seleccionada espera a ser atendido. ***Nota:*** maximizando la concurrencia.

3. Resolver la administración de las impresoras de una oficina. Hay 3 impresoras, N usuarios y 1 director. Los usuarios y el director están continuamente trabajando y cada tanto envían documentos a imprimir. Cada impresora, cuando está libre, toma un documento y lo imprime, de acuerdo al orden de llegada, pero siempre dando prioridad a los pedidos del director. ***Nota:*** los usuarios y el director no deben esperar a que se imprima el documento.

4. En una mesa de exámenes hay 3 profesores que deben corregir los exámenes de 30 alumnos de acuerdo al orden en que terminaron. Cada examen es corregido por un único profesor. Cuando un alumno termino su examen lo deja y espera hasta que alguno de los profesores (cualquiera) lo corrija y le dé la nota. Cuando un profesor está libre toma el siguiente examen para corregir y al terminar le da la nota al alumno correspondiente.

5. En una oficina hay ***un empleado*** para atender a ***N personas***. Las personas pueden tener prioridad ALTA o BAJA (cada uno conoce su prioridad). El empleado atiende a las personas de acuerdo a la prioridad (primero los de ALTA y luego los de BAJA). Cada persona espera hasta que el empleado lo termina de atender y se retira. ***Nota:*** existe la función *atender()* que simula que el empleado está atendiendo a una persona; no debe hacerse *Busy Waiting*.

6. En un consultorio hay un médico que debe atender a 15 pacientes de acuerdo al turno de cada uno de ellos (no puede atender al paciente con turno i+1 si aún no atendió al que tiene turno i). Cada paciente ya conoce su turno al comenzar (valor entero entre 0 y 14, o entre 1 y 15, como les resulte más cómodo), al llegar espera hasta que el médico lo llame para ser atendido y luego espera hasta que el médico lo termine de atender. ***Nota:*** los únicos procesos que se pueden usar son los que representen a los pacientes y al médico; se debe evitar hacer *Busy Waiting*.

7. En una empresa de software hay 3 programadores que deben arreglar errores informados por N clientes. Los clientes continuamente están trabajando, y cuando encuentran un error envían un reporte a la empresa para que lo corrija (no tienen que esperar a que se resuelva). Los programadores resuelven los reclamos de acuerdo al orden de llegada, y si no hay reclamos pendientes trabajan durante una hora en otros programas. ***Nota:*** los procesos no deben terminar (trabajan en un loop infinito); suponga que hay una función *ResolverError* que simula que un programador está resolviendo un reporte de un cliente, y otra *Programar* que simula que está trabajando en otro programa.

8. Simular la atención en un locutorio con 10 cabinas telefónicas, que tiene un empleado que se encarga de atender a los clientes. Hay N clientes que al llegar esperan hasta que el empleado les indica a que cabina ir, la usan y luego se dirigen al empleado para pagarle. El empleado atiende a los clientes en el orden en que hacen los pedidos, pero siempre dando prioridad a los que terminaron de usar la cabina. ***Nota:*** maximizar la concurrencia; suponga que hay una función *Cobrar()* llamada por el empleado que simula que el empleado le cobra al cliente.

CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS DE PMS:

- Los canales son punto a punto y no deben declararse.
- No se puede usar la sentencia ***empty*** para saber si hay algún mensaje en un canal.
- Tanto el envío como la recepción de mensajes es bloqueante.
- Sintaxis de las sentencias de envío y recepción:

  Envío: nombreProcesoReceptor!port (datos a enviar)

  Recepción: nombreProcesoEmisor?port (datos a recibir)

  El port (o etiqueta) puede no ir. Se utiliza para diferenciar los tipos de mensajes que se podrían comunicarse entre dos procesos.
- En la sentencia de comunicación de recepción se puede usar el comodín * si el origen es un proceso dentro de un arreglo de procesos. Ejemplo: Clientes[*]?port(datos).
- Sintaxis de la Comunicación guardada:

  Guarda: (condición booleana); sentencia de recepción → sentencia a realizar

  Si no se especifica la condición booleana se considera verdadera (la condición booleana sólo puede hacer referencia a variables locales al proceso).

  Cada guarda tiene tres posibles estados:

  Elegible: la condición booleana es verdadera y la sentencia de comunicación se puede resolver inmediatamente.

  No elegible: la condición booleana es falsa.

  Bloqueada: la condición booleana es verdadera y la sentencia de comunicación no se puede resolver inmediatamente.

  Sólo se puede usar dentro de un ***if*** o un ***do*** guardado:

  El ***IF*** funciona de la siguiente manera: de todas las guardas ***elegibles*** se selecciona una en forma no determinística, se realiza la sentencia de comunicación correspondiente, y luego las acciones asociadas a esa guarda. Si todas las guardas tienen el estado de ***no elegibles***, se sale sin hacer nada. Si no hay ninguna guarda elegible, pero algunas están en estado ***bloqueado***, se queda esperando en el if hasta que alguna se vuelva elegible.

  El ***DO*** funciona de la siguiente manera: sigue iterando de la misma manera que el ***if*** hasta que todas las guardas hasta que todas las guardas sean ***no elegibles***.

9. En un laboratorio de genética veterinaria hay 3 empleados. El primero de ellos se encarga de preparar las muestras de ADN lo más rápido posible; el segundo, toma cada muestra de ADN preparada y arma el set de análisis que se deben realizar con ella y espera el resultado para archivarlo y continuar trabajando; el tercer empleado se encarga de realizar el análisis y devolverle el resultado al segundo empleado.

10. Suponga que existe un antivirus distribuido en él hay R procesos robots que continuamente están buscando posibles sitios web infectados; cada vez que encuentran uno avisan la dirección y continúan buscando. Hay un proceso analizador que se encargue de hacer todas las pruebas necesarias con cada uno de los sitios encontrados por los robots para determinar si están o no infectados.

11. En un banco hay un ***empleado*** para cobrar a ***N personas***. Las N personas forman una única fila y el empleado los atiende de acuerdo al orden en que llegaron. Cuando una persona es llamada por el cajero, le entrega la boleta a pagar y este le devuelve la boleta sellada. ***Notas:*** cada persona paga una sola boleta.

12. En un examen final hay P alumnos y 3 profesores. Cuando todos los alumnos han llegado comienza el examen. Cada alumno resuelve su examen, lo entrega y espera a que alguno de los profesores lo corrija y le indique la nota. Los profesores corrigen los exámenes respectando el orden en que los alumnos van entregando. ***Nota:*** maximizar la concurrencia y no generar demora innecesaria.

13. En un estadio de fútbol hay una máquina expendedora de gaseosas que debe ser usada por E Espectadores de acuerdo al orden de llegada. Cuando el espectador accede a la máquina, la usa y luego se retira para dejar al siguiente. ***Nota:*** cada Espectador una sólo una vez la máquina.

14. Hay ***N personas*** que deben usar un teléfono público de a una a la vez y de acuerdo al orden de llegada, pero dando prioridad a las que tienen que usarlo con urgencia (cada persona ya sabe al comenzar si es de tipo urgente o no). ***Nota:*** el teléfono NO ES un proceso, es un recurso compartido usado por las personas por medio de la función *Usar_Teléfono*().

15. En una mesa de exámenes hay 3 profesores que deben corregir los exámenes de 30 alumnos de acuerdo al orden en que terminaron. Cada examen es corregido por un único profesor. Cuando un alumno termino su examen lo deja y espera hasta que alguno de los profesores (cualquiera) lo corrija y le dé la nota. Cuando un profesor está libre toma el siguiente examen para corregir y al terminar le da la nota al alumno correspondiente. Cuando los 30 exámenes se han corregido los profesores se retiran. ***Nota:*** maximizar la concurrencia.

16. En una empresa de software hay 3 programadores que deben arreglar errores informados por N clientes. Los clientes continuamente están trabajando, y cuando encuentran un error envían un reporte a la empresa para que lo corrija (no tienen que esperar a que se resuelva). Los programadores resuelven los reclamos de acuerdo al orden de llegada, y si no hay reclamos pendientes trabajan durante una hora en otros programas. ***Nota:*** los procesos no deben terminar (trabajan en un loop infinito); suponga que hay una función *ResolverError* que simula que un programador está resolviendo un reporte de un cliente, y otra *Programar* que simula que está trabajando en otro programa.