PRÁCTICO 5

Objetivos

- Familiarizarse con el uso de semáforos
- Familiarizarse con el uso de monitores

Ejercicio 1 (básico)

Implementar utilizando semáforos una solución al problema de concurrencia planteado en el ejercicio 4 del práctico 4.

Ejercicio 2 (medio)

Explicar por qué las construcciones *P(s) y V(s)* deben ejecutarse en forma indivisible en un semáforo no binario. Comentar el caso de semáforo binario.

Ejercicio 3 (medio)

(a) ¿Cuál es el resultado de intercambiar las instrucciones `P` del procedimiento `consumer_process` en la solución al problema productor-consumidor que sigue?

(b) ¿Cuál es el resultado de intercambiar las instrucciones `V` del procedimiento `producer_process` en la solución al problema productor-consumidor que sigue?

```
program producer_consumer_relationship;
  var  exclusive_access : semaphore;
       number_deposited : semaphore;
       buff : buffer;

  procedure producer_process;
    var next_result : integer;
  begin
    while true do
    begin
      calculate_next_result;
      P(exclusive_access);
      agrego_buffer(buffer, next_result);
      V(exclusive_access);
      V(number_deposited);
    end while;
  end procedure;

  procedure consumer_process;
    var next_result : integer;
  begin
    while true do
    begin
      P(number_deposited);
      P(exclusive_access);
      next_result := saco_buffer(buffer);
      V(exclusive_access);
      write(next_result);
```

```
    end while;
  end procedure;

begin {programa principal}
  init(exclusive_access, 1);
  init(number_deposited, 0);
  init_buffer(buffer);
  cobegin
    producer_process;
    consumer_process;
  coend
end program.
```

Ejercicio 4 (en clase)

Se desea controlar el proceso de llenado y tapado de envases de la Compañía Algarrobo Cola. Existe una sola cinta transportadora y una maquina para cada acción. Se dispone de los siguientes procedimientos:

- `avanzo_cinta` Avanza la cinta un paso (muy pequeño).
- `llenar_botella` Llena la botella de líquido (finaliza cuando está llena).
- `pongo_tapa` Comienza la acción de tapado (finaliza cuando la tapa queda colocada).

Y de las siguientes funciones:

- `puedo_llenar` Devuelve TRUE si hay un envase bajo la estación de llenado.
- `puedo_tapar` Ídem para la estación de tapado.

Se pide:

(a) Resolver utilizando semáforos.

(b) Resolver utilizando monitores.

Nota:

Las botellas vienen a intervalos irregulares y no siempre se está en situación de accionar las **dos** máquinas a la vez. Se desea conseguir los algoritmos mas sencillos posibles que resuelvan correctamente el problema.

Ejercicio 5 (medio)

(a) Implementar semáforos binarios utilizando semáforos no binarios.

(b) Implementar semáforos no binarios utilizando semáforos binarios.

Ejercicio 6 (medio)

(a) Implementar semáforos no binarios utilizando monitores.

(b) Implementar monitores utilizando semáforos no binarios.

Ejercicio 7 (avanzado)

Se considera un puente levadizo de además una única vía, con las siguientes características:

En cuanto al cruce por parte de vehículos:

- Pueden haber varios vehículos a la vez en el puente.
- La interacción entre las tareas `autos` y el `puente` se supondrá de la siguiente forma:

```
.............
P(auto_entrada)  -- Para entrar al puente
delay(X)         -- Demora arbitraria en realizar el cruce
V(auto_salida)   -- Fin de cruce del puente
.............
```

En cuanto al cruce por parte de barcos:

- Los barcos tendrán prioridad para el cruce, debiendo poder hacer el cruce tan pronto como se haya vaciado de vehículos y elevado el puente.
- Solamente se admitirá el cruce de un único barco a la vez.
- En caso de presentarse cola de barcos, se desea evitar que el puente se eleve y descienda para cada uno de ellos.
- La interacción entre las tareas Barcos y Puente se supondrá de la siguiente forma:

```
.............
P(barco_entrada) -- Comienza el cruce
delay(Y)         -- Demora arbitraria en realizar el cruce
V(barco_salida)  -- Fin de cruce de puente
.............
```

Se pide:

Implementar los procedimientos necesarios que simulen el puente.

Notas:

(a) Se pueden agregar todos los semáforos que se deseen a los procedimientos barco o auto.

(b) Puede necesitarse indicar alguna prioridad entre los procedimientos.

Ejercicio 8 (en clase)

Una tribu de N caníbales come de una gran marmita común con capacidad para 6 comensales simultáneos.

Cuando un comensal quiere comer, come de la marmita, a menos que no haya suficiente comida para él. Si no hay suficiente comida en la marmita, el caníbal despierta al cocinero y espera a que el cocinero haya rellenado la marmita con la carne de los misioneros capturados (no debe haber notificaciones repetidas). Para rellenar la marmita el cocinero debe esperar a que todos los comensales que se encuentran actualmente comiendo terminen. El cocinero, por su parte, vuelve a dormir cuando ha rellenado la marmita.

Consideraciones:

- No podrán entrar nuevos comensales a la marmita cuando el cocinero está rellenando o esperando para rellenar.
- Se supone que la marmita llena dispone de comida para más de seis caníbales.

Se dispone de las siguientes funciones:

- Hay_suficiente_comida (): boolean

  Esta función es ejecutada por los comensales y retorna si hay suficiente comida en la marmita. No puede ser ejecutada por dos o más comensales a la vez.

- Rellenar ()

  Esta función es ejecutada por el cocinero.

- Comer ()

  Es ejecutado por los comensales.

- Ocio ()

  Ejecutada por los caníbales cuando no están comiendo.

Se pide:

Implementar los procedimientos caníbal y cocinero utilizando monitores. Especifique la semántica de los mismos en caso de ser necesario.

Ejercicio 9 **(medio)**

Se desea implementar una nueva primitiva de sincronización que denominaremos **sincronizador múltiple,** que provee la facilidad de intercambio de información.

Esta primitiva brindará las siguientes operaciones:

- enviar_sinc_mult(cant_destinos: in integer, mensaje: in TMensaje);
- recibir_sinc_mult (mensaje: out TMensaje);

Ambas operaciones son bloqueantes, hasta que la cantidad de procesos que esperan en *recibir_sinc_mult* coincida con el valor *cant_destinos* de *enviar_sinc_mult*.

De esta forma, tanto *recibir_sinc_mult* como *enviar_sinc_mult* culminan sincrónicamente en ocasión de la aceptación del mensaje por parte de todos los receptores.

Las invocaciones a *enviar_sinc_mult* se deben resolver de forma FIFO.

Implementar la primitiva **sincronizador múltiple** utilizando monitores.

Ejercicio 10 **(avanzado)**

Se desea modelar con semáforos las tareas VENDEDOR, CLIENTE y SUPERVISOR de una cafetería. Esta cafetería dispone de una caja, 8 vendedores y 2 supervisores.

Los vendedores toman el pedido, cobran y elaboran el pedido de cada cliente. Tienen orden de no dejar la caja sola, por lo que para poder ir a elaborar el pedido debe haber otro vendedor en el área de la caja. Por razones de espacio no puede haber más de 2 vendedores en el área de caja a la vez, de los cuales solo uno de ellos puede estar atendiendo. Los vendedores solo toman pedidos cuando están en la caja. Los vendedores deben atender a los clientes lo antes posible, no se debe hacer esperar a un cliente si la caja esta libre.

Los vendedores le avisarán al grupo de supervisores cada 10 pedidos cobrados por ese vendedor. Los supervisores estarán esperando ser avisados para llenar la planilla. El primer supervisor libre recibirá el número de vendedor y con ese dato llenará la planilla.

Se dispone de los siguientes procedimientos:

- recibir_pedido(), cobrar_pedido(), elaborar_pedido(). Ejecutados por el vendedor
- enviar_pedido_pagar_y_recibir_pedido(). Ejecutado por el cliente, retorna cuando el vendedor termina de elaborar el pedido.
- llenar_planilla(int nro_vendedor). Ejecutado por el supervisor actualiza la planilla de ventas.

Aclaraciones:

- No se permite la implementación de tareas auxiliares
- Todos los procedimientos disponibles pueden demorar tiempos variables.

Ejercicio 11 (avanzado)

Se desea modelar con monitores el siguiente juego de mesa:

Del juego forman parte N jugadores y un jefe de mesa. Los jugadores pueden mover sus fichas todos al mismo tiempo siempre y cuando el jefe de mesa no este reordenando el tablero, lo cual hará cada vez que termine de pensar sus cambios. Esto implica que el jefe de mesa debe esperar a que la mesa quede libre de jugadores antes de reordenar el tablero.

El jefe de mesa además de la función de reordenar el tablero, dejará cartas con instrucciones respecto al comportamiento que debe tomar el jugador en caso que el jefe de mesa este esperando para reordenar el tablero. Hay 2 tipos de cartas, una que indica que se debe esperar a que el jefe de mesa reordene el tablero para poder jugar, y otra que le permite jugar antes del reordenamiento de fichas.

El jugador debe sacar una carta cada vez que quiera jugar y el jefe de mesa este esperando para reordenar el tablero. Solamente una persona (jugador/jefe de mesa) podrá tener acceso al mazo de cartas en cada momento.

En caso que no queden cartas en la mesa el jugador deberá esperar que el jefe de mesa reordene el tablero. Asimismo, el mazo nunca puede tener más de 10 cartas apiladas.

El jefe de mesa deberá poder hacer las 2 funciones a la vez (pensar reordenamiento y reordenar con elegir carta y colocarla en mazo).

Se dispone de las siguiente funciones que son ejecutadas por los jugadores:

- pensar_jugada()
- sacar_carta_de_mazo():Carta
- jugar()

Se dispone de las siguientes funciones que serán ejecutadas por el Jefe de Mesa:

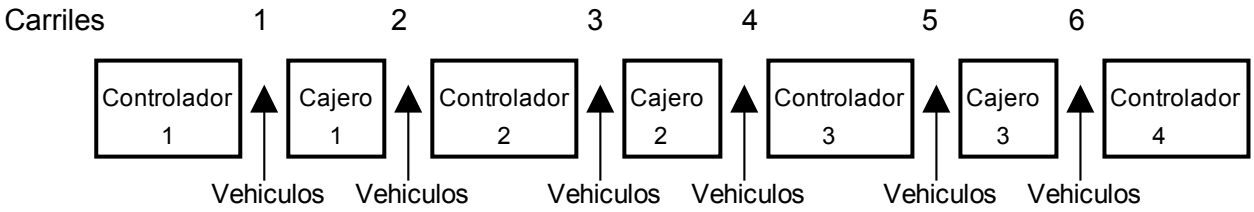
- pensar_reordenamiento()
- reordenar_tablero()
- elegir_proxima_carta():Carta
- colocar_carta_en_mazo(Carta)

Nota:

- Carta se modela con un enumerado [ESPERAR,JUGAR]
- No se pueden usar tareas auxiliares pero si se puede representar a los actores con más de un proceso

Ejercicio 12 (avanzado)

Se desea modelar un peaje de una carretera de una sola vía. El peaje además de cobrar el ticket controla los documentos de los autos y camiones. Los puestos están distribuidos de esta manera:



Las cajas y los controladores atenderán para los dos carriles que tienen a su costado, salvo las de las puntas que solamente atienden autos del carril que tienen al lado.

Una vez que un cajero atendió a un vehículo, no podrá atender a otro hasta que el controlador termine de controlar dicho vehículo, y viceversa.

Por otro lado los camiones solo podrán pasar por los carriles de las puntas. Un inspector controlara cada cierto tiempo a estos controladores (los de las puntas), los que no podrán atender ningún vehículo mientras están siendo controlados y viceversa.

El control por parte del inspector a estos controladores se hará con los 2 a la vez, es decir que ninguno de los 2 deberá estar controlando vehículos, debiendo obtener primero la atención del controlador 4 antes que el controlador 1.

Por razones de ordenamiento y comodidad, los vehículos siempre pedirán ser atendidos primero por el puesto de su izquierda y luego por el de la derecha.

Se pide:

Modelar este sistema usando monitores.

Se dispone de las siguientes funciones o procedimientos:

- pagar() paga o cobra al vehículo al cajero.
- controlar_documentacion() control de documentación del controlador al vehículo.
- inspeccion() control del inspector a los controladores de las puntas.
- dame_carril(tipo_vehículo: Integer) que retorna a que carril debe ir el vehículo, dependiendo si es auto o camión, el cual será ejecutado por el vehículo.
- que_soy() retorna 0 si es un auto o 1 si es un camión.

Nota: Se podrá utilizar como máximo una tarea auxiliar.