Departamento de Arquitectura

Sistemas Operativos
Práctico 4

Curso 2025

Objetivos

- Familiarizarse con el uso de monitores.
- Familiarizarse con el uso de colas de mensajes.

Duración

- 1 semana.

Ejercicio 1 (OpenFing) 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 2 (medio) Se quiere desarrollar un software que permita distribuir el trabajo entre dos grupos de TI, desarrollo y verificación. Los integrantes del grupo de desarrollo implementan módulos de software y en la medida que culminan dejan los módulos para su verificación. El grupo de verificación valida, en la medida que tenga módulos disponible, los módulos. Por reglas de la empresa no se puede tener más de 20 módulos disponibles para verificar. En caso de llegar a esta situación, el jefe pasará todos los módulos disponibles a un centro dedicado a la temática y esta es su única tarea. El software además, debe llevar los indicadores de producción, cuántos módulos se implementaron, cuántos se validaron positivamente, cuántos negativamente y cuantos se pasaron al centro especializado.

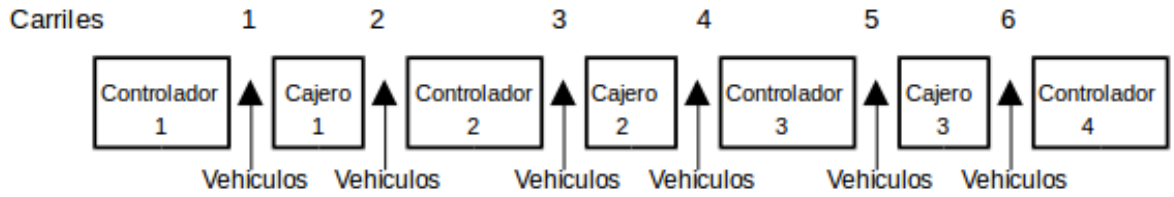
Se dispone de los siguientes procedimientos y funciones:

- **implementar()** : Función de los desarrolladores, devuelve un módulo.

- **MLPV(m)** : Procedimiento que inserta el módulo `m` en la estructura de módulos listos para verificar.
- **verificar(m)** : Función que verifica el módulo `m`. Devuelve verdadero si se valido el módulo y falso en caso contrario.
- **obtener()** : Función que devuelve el primer módulo disponible para verificar. Si no hay módulos la función falla.
- **a_centro(m)** : Procedimiento invocado por el jefe que entrega al centro especializado el módulo `m` a verificar.

Se pide: implemente con **monitores** la realidad anterior. Debe incluir los procedimientos Desarrollador, Tester y Jefe. Además debe implementar una función que retorne cada uno de los indicadores en el orden definido en la letra.

Ejercicio 3 (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)** — 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.

Ejercicio 4 (avanzado) Se quiere definir un sistema que permita modelar el ingreso de docentes y estudiantes al local de una muestra de parciales. Por la situación sanitaria no pueden ingresar más de 20 personas al local. Las personas entrarán en orden de llegada al local. Pero:

1. para que ingresen estudiantes tiene que haber al menos un docente dentro del local
2. en caso de tener personas esperando fuera tendrán prioridad para el ingreso a la sala los docentes
3. en caso que un docente se quiera retirar debe considerar que mientas haya estudiantes en el salón debe quedar al menos un docente.

Implemente con **monitores** la realidad anterior. Dispone de las funciones:

- `Muestro_Parciales()`: Procedimiento de los docentes que modela la tarea de mostrar parciales. Una vez concluida la ejecución de la función el docente se retira del salón.
- `Reviso_Parcial()`: Procedimiento de los estudiantes, para revisar el parcial. Una vez concluida la ejecución de la función el estudiante se retira del salón.

Nota: no es necesario modelar las interacciones docentes-estudiante durante la muestra.

Ejercicio 5 (OpenFing) Se desea modelar utilizando **mailboxes** el siguiente problema:
El taller mecánico "Boxes" tiene capacidad para 20 vehículos en sus instalaciones. La entrada de los usuarios a las instalaciones debe ser estrictamente en orden de llegada (la solución no debe permitir bajo ningún concepto "colados").
El usuario debe ser atendido por el primer mecánico libre de los 5 con que cuenta el taller. El mecánico, una vez terminado el arreglo, le indicará a la caja el monto a cobrar por el arreglo. El cliente recibirá de la caja el monto a pagar.

Se dispone de las siguientes funciones:

- `arreglar_auto():integer`
  Invocada por un mecánico para arreglar el auto. Retorna el costo del arreglo.
- `pagar_arreglo(Monto)`
  Invocada por el usuario para pagar el monto que le fue indicado por la caja.

Nota:

- Se prohíbe expresamente el busy-waiting.
- Se debe explicitar la semántica de las primitivas de mailbox utilizadas.
- Tener cuidado de que el socio pague el importe correcto del arreglo a la caja.

Ejercicio 6 (medio) Se desea tener un sistema para simplificar el orden de atención en las muestras de exámenes. Cada examen tiene un número que lo identifica y consta de tres ejercicios. Para ver los exámenes los estudiantes entran al salón en orden de llegada. El salón tiene espacio para seis estudiantes. Una vez dentro del salón los ejercicios se muestran en orden, es decir, primero se responden las dudas del ejercicio 1, luego del ejercicio 2 y por último del ejercicio 3. Los docentes que muestran cada ejercicio atenderán a los estudiantes en el orden en que solicitaron ver ese ejercicio. La cantidad de docentes no está determinada.

Si es necesario cambiar la nota del examen el docente deberá modificar la planilla de notas. Esta solo puede ser accedida por un docente a la vez.

Se pide: implementar los docentes y estudiantes usando **mailboxes**. Debe especificar la semántica de los mailboxes utilizados. Puede asumir que existe al menos un docente para cada ejercicio.

Dispone de los siguientes procedimientos y funciones:

- `bool quiero_ver(ex,ej)` : Función ejecutada por los estudiantes que recibe un examen `ex` y un ejercicio `ej` y retorna si el estudiante quiere que se muestre el ejercicio.
- `bool muestra(ex,ej)` : Función ejecutada por los docentes que muestra el ejercicio `ej` del examen `ex`. Retorna verdadero si es necesario modificar la nota en la planilla.

- `void modificar_planilla()` : Ejecutado por los docentes para cambiar la planilla de notas.
- `int mi_examen()` : Ejecutado por los estudiantes para obtener su número de examen.
- `int mi_ejercicio()` : Ejecutado por los docentes para ver que ejercicio contestan.

Ejercicio 7 (avanzado)

Se desea modelar el funcionamiento de una mesa de votación y su respectivo cuarto secreto para las elecciones nacionales.

Cuando un votante llega a la mesa deberá presentar su credencial la cual será validada por la mesa. Luego de ser autorizado el votante tomará un sobre de la pila y se dirigirá al cuarto secreto. Si el cuarto está ocupado deberá esperar a que se libere.

Una vez en el cuarto secreto, el votante pondrá la hoja de votación en el sobre. Cuando esta tarea finalice retornará a la mesa para depositar el voto. La mesa puede atender más votantes mientras el cuarto secreto esté ocupado. En el momento que el votante salga del cuarto secreto a depositar su voto puede tener que esperar a que la mesa termine la atención de otro votante que se encuentra en la mesa haciendo los trámites previos a la entrada al cuarto secreto, pero tendrá prioridad frente a los demás votantes y delegados esperando que la mesa los atienda.

Cada cierto tiempo, los delegados de los diferentes partidos llegan a la mesa de votación y solicitan acceder al cuarto secreto para reponer listas. Un delegado partidario debe presentar a la mesa su identificación de delegado. Los delegados harán cola para ser atendidos por la mesa junto a los votantes esperando para presentar su credencial. Ambos serán atendidos por orden de llegada.

En ningún momento podrán haber colados en las filas de la mesa de votación y del cuarto secreto. Se puede asumir que nunca se acaban las listas en el cuarto secreto. Si el documento que presenta no es válido la persona se retira inmediatamente.

Se pide: modelar utilizando **mailboxes**, los procesos Mesa, Votante y Delegado.

Se dispone de los siguientes procedimientos auxiliares:

- `obtener_documento():Documento` Ejecutado por los votantes y delegados para obtener su documento de identificación
- `validar_documento(Documento):boolean` Ejecutado por la mesa para validar la credencial del votante o la identificación del delegado
- `tomar_sobre():Sobre` Ejecutado por los votantes para obtener un sobre
- `poner_lista(Sobre):Sobre` Ejecutado por los votantes para poner una lista en el sobre
- `votar(Sobre)` Ejecutado por los votantes para poner el sobre en la urna
- `recargar()` Ejecutado por los delegados para recargar las listas de votación

Ejercicio 8 (avanzado)

Con motivo de las fiestas, Papá Noel y los duendes se disponen a entregar regalos a los niños. Los niños eligen el regalo que quieren y mandan una carta al Polo Norte. En el Polo Norte trabajan 6 Joulutonttu (duendes). Cada carta se asigna a un duende que esté esperando para confeccionar el regalo. Si no hay duendes esperando cuando se recibe una carta, el pedido no se considera y el niño se queda sin regalo. Los regalos se van agregando a una bolsa, que solo puede ser accedida por un duende a la vez. Cuando la bolsa contiene 100 regalos, se despierta a Papá Noel para que los reparta. Dependiendo de que tan cansado esté, Papá Noel puede solicitar la ayuda de uno de los duendes. Mientras Papá Noel reparte los regalos, los duendes pueden seguir trabajando.

Implementar los procesos niño, PapaNoel y duende usando **mailboxes**. Se pueden usar procesos auxiliares

Dispone de los siguientes procedimientos auxiliares:

- `escribir_carta(): Carta`. Ejecutado por un niño para escribir la carta a enviar al servidor.
- `confeccionar_regalo(c: Carta): Regalo`. Ejecutado por un duende para crear un regalo.
- `repartir_regalos(regalos: array[100] of Regalo)`. Ejecutado por Papá Noel para repartir los regalos. Recibe un arreglo con 100 regalos.
- `estoy_cansado(): Boolean`. Ejecutada por Papá Noel. Retorna True si necesita la ayuda de un duende.
- `ayudar()`: Ejecutada por un duende para ayudar a Papá Noel. Puede asumir que esta función termina al mismo tiempo que `repartir_regalos`.
- `recibir_regalo()`. Ejecutada por el Niño. Esta función se bloquea y termina cuando Papá Noel le entrega el regalo.