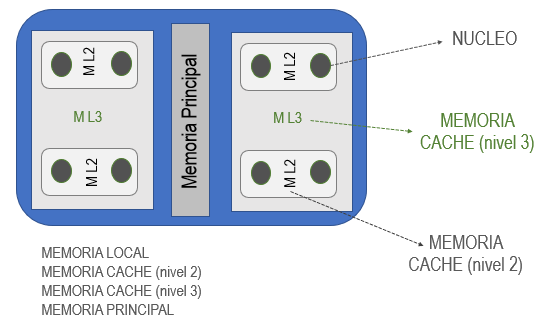
**Clase 1**

Conceptos de concurencia.



Si hablamos de capacidad la memoria principal es la mas grande, después sieguen las cache nivel 3, las cache nivel 2, y las locales. Si hablamos de velocidad de acceso a los datos, las memorias locales por se mas chicas son las mas rápidas, seguidas por las memorias cache nivel 2 , las de nivel 3, y por ultimo la memoria principal que es la mas lenta. Si bien nos da la ventaja de que todos los núcleos pueden acceder a esa memoria principal, la desventaja es que buscar algo en la memoria principal siempre es mucho mas lento que buscarlo en los otros tipos de memoria que existen dentro de nuestra computadora.

Programación secuencial: se ejecuta una instrucción atrás de la otra, y recién cuando termina la primera instrucción se puede ejecutar la segunda.

En un programa concurrente podemos tener mas de un proceso ejecutando un programa al mismo tiempo. Entonces para poder hacer eso los procesos deben comunicarse y coordinarse, y para poder escribir el programa correctamente tendremos que controlar esa comunicación y esa coordinación entre los procesos.

Definición:

Un programa concurrente se divide en tareas (2 o más), las cuales se ejecutan al mismo tiempo y realizan acciones para cumplir un objetivo común. Para esto pueden: compartir recursos, coordinarse y cooperar.

Caracteristicas: comunicación – sincronización.

Cualquier lenguaje que brinde concurrencia debe proveer mecanismos para comunicar y sincronizar procesos. En este caso quiero proteger el acceso a la variable compartida (dos procesos no accedan al mismo tiempo, sincronicen).

Van a existir variables en mi programa que querré proteger. Esas variables las podré proteger con el mecanismo que me provea el lenguaje. El mecanismo mas usado son los semáforos, que tienen una operación para proteger y otra para liberar. Mientras la variable está protegida ningún otro proceso la puede acceder, cuando la libero eso hace que otro proceso la pueda acceder.

Todo programa concurrente tiene un sistema operativo. Ese sistema operativo va a determinar que instrucción se ejecuta de cada uno de los procesos, utilizando diferentes políticas. Por ej.: round robin que es una instrucción para cada proceso; o primero ejecuto el proceso mas corto; o el que llego primero a la memoria. Lo que podría pasar, sin tener en cuenta estas políticas, es que las instrucciones, al estar formadas por instrucciones más pequeñas, pueden ser partidas en cualquier momento, puede ser suspendida y pasado el control a otro proceso.

Cada vez que vayamos a modificar una variable donde podemos suponer que va a ser accedida por otros procesos vamos a tener que protegerla.

Sin comunicación ni sincronización no hay programa concurrente.

Para darnos esa comunicación cualquier programa concurrente nos va a proveer de dos mecanismos: el envió de mensajes o la memoria compartidas. El entorno CMRE nos da los dos mecanismos.

Envió de mensajes: tengo un origen que es el proceso que lo envía, un destino que es el contacto que elijo para dar el mensaje y un contenido, que es el mensaje.

Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos. También el lenguaje debe proveer un protocolo adecuado (el orden de los parámetros del mensaje). Para que la comunicación sea efectiva los procesos deben “saber” cuando tienen mensajes para leer y cuando deben transmitir mensajes. Para eso todo lenguaje deberá proveer una instrucción para leer mensajes y para recibir mensajes.

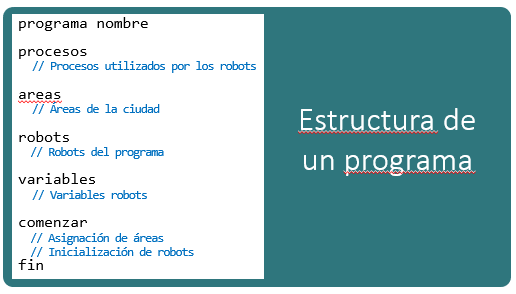
Memoria compartida: es una variable que puede ser accedida por más de un proceso (cache nivel 2 – se comparte entre 2 procesos - , nivel 3 – entre 4 procesos - y memoria principal - entre todos los procesos).

Para poder usar esa memoria compartida tendré que preguntar si el recurso esta libre, si esta libre tendré que bloquearla, la uso, y cuando termino de usarla lo antes posible la libero. En caso de que no este libre el proceso se quedara preguntando hasta que se libere el recurso.

Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre los datos residentes en ella. Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria. La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida.

El proceso queda bloqueado hasta que se libere el recurso que necesite usar. Si no se libera nunca se terminará de ejecutar.

El ambiente CMRE nos permite tener recursos compartidos, sincronizar robots, y nos permite trabajar con procesadores heterogéneos (no lo vemos). Se permite declarar mas de un robot. Las áreas pueden ser privadas, compartidas y parcialmente compartidas. Tendremos comunicación y sincronización con las operaciones de enviar y recibir mensajes, y bloquear y desbloquear esquina.



Area compartida (areaC): cualquier robot puede circular por la misma. (Representa la memoria principal).

Area privada (areaP): solo puede haber en ella un único robot. (Representa la memoria local).

Area parcialmente compartida (areaPC): se debe seleccionar que subconjunto de robots pueden circular por la misma (2 o más pero no todos). (Representa la memoria cache).

Las áreas deben ser disjuntas, no puedo tener esquinas que sean compartidas por 2 áreas.

Los robots pueden estar asignados a mas de un área.

Para cada una de las variables de tipo robot que están declaradas en el programa tengo que asignarle un área y tengo que decirle en que esquina comienza. Si me olvido cualquiera de esas dos opciones el programa va a compilar, pero va a andar mal.

Si tengo mas de un robot la esquina en donde quiero que comiencen deben ser diferentes, sino van a chocar de entrada.

Cuando se ejecute un programa, se cargarán los robots en memoria, y el administrador del programa va a ir eligiendo que instrucción se ejecuta de cada robot. Siempre en orden, pero no significa que se ejecute todo el robot1 y después todo el robot2.

**Clase 2**

Pasaje de mensajes.

Para dar la comunicación entre procesos en un lenguaje concurrente vamos a tener que brindar pasaje de mensajes y memoria compartida, ambos dos o al menos uno de estos mecanismos.

Pasaje de mensajes:

Dados dos procesos, uno quiere enviar a otro, para que esto se de tenemos que tener en el lenguaje una operación para enviar y otra para recibir. Ese envió y recepción pueden ser asincrónicos o sincrónicos.

El pasaje de mensajes asincrónico es cuando el que envía o recibe el mensaje no esta esperando que se dé la comunicación para continuar su ejecución.

En el pasaje de mensajes sincrónico, el que envía o recibe si se queda bloqueado y no puede seguir ejecutando las instrucciones que tiene abajo hasta que el otro robot o recibe el mensaje o me manda el mensaje que estoy esperando.

El ambiente CMRE tiene el envío de manera asincrónica (un robot después de enviar un mensaje puede continuar su ejecución), pero la recepción es de manera sincrónica (un robot que está esperando para recibir un mensaje no puede continuar su ejecución hasta que otro robot no le haya mandado el mensaje).

Sintaxis para envío de mensajes:

EnviarMensaje(variable,variableRobot)

EJ: EnviarMensaje(3,robot1)

El segundo parámetro del enviar mensaje tiene que ser una de las variables de tipo robot declaradas en el programa. (no del tipo que es el robot) Por ej.: robot1: tipo1, el mensaje será EnviarMensaje(1,robot1) y no EnviarMensaje(1,tipo1)

También puedo enviar el contenido de una variable de tipo numero

EnviarMensaje(flores,robot2)

Solo puedo enviar de tipo número. Los mensajes se mandan de a un solo valor.

Sintaxis para recepción de mensajes:

RecibirMensaje(variable,variableRobot)

Siempre recibiré el mensaje en una variable, y tengo que decir de que robot recibiré el mensaje.

Por ej.

RecibirMensaje(altura,robot3)

RecibirMensaje(ancho,robot3)

Para que el programa sea correcto todos los robots deben decir finalizado en su estado. Otra cosa a tener en cuenta, si un robot está haciendo un recibir mensaje y por algún motivo nos olvidamos de mandarle el mensaje que está esperando, entonces ese robot que esta esperando recibir va a quedar bloqueado y nunca va a decir finalizado. Controlar que para todo robot que este esperando recibir mensaje haya uno que le mande un mensaje. La cantidad de mensajes enviados tiene que ser igual a la cantidad de recibidos.

Sintaxis para recibir un mensaje de cualquier robot

RecibirMensaje (variable, \*)

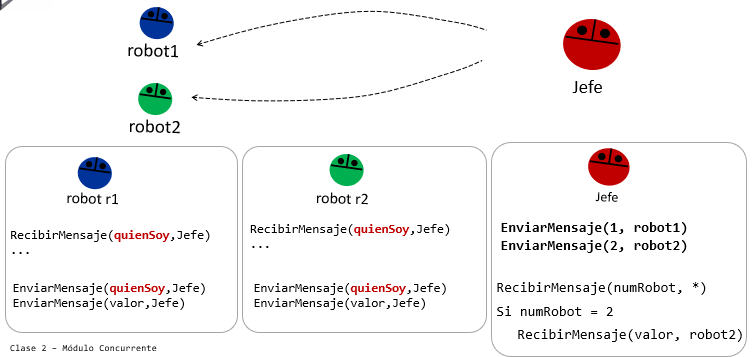
Esto hace que no fuerce a recibir mensaje de robot1 y después de robot2.

¿Como sabe el jefe quien le envió el mensaje?

Lo primero que va a tener que hacer es decirle a cada robot que numero les corresponde.

En cada robot, lo primero que hare es recibir en una variable que numero de robot soy.

Finalmente deberé mandarle al robot jefe que numero de robot soy y el dato a informar.



Random(variable,inf,sup)

**Clase 3**

Memoria compartida.

En nuestro caso particular nuestro recurso es una esquina en la que pueden pararse dos robots.

Bloquear un recurso:

Dado un recurso disponible el programador bloquea dicho recurso para que ningún otro proceso pueda acceder.

Yo voy a poder ejecutar una instrucción que es bloquear recurso. Si ese recurso esta libre lo va a bloquear, y sino lo va a dejar esperando hasta que ese recurso este libre, al proceso que ejecuto la instrucción bloquear.

Desbloquear un recurso:

Dado un recurso bloqueado el programador libera dicho recurso para que cualquier proceso puede bloquearlo.

Cuando el bloqueo esta a cargo del programador, este tiene que disponer de instrucciones para bloquear y desbloquear el recurso. Hay lenguajes que permiten que el programador haga el código y el sistema operativo se encarga de detectar cuales son los recursos que se están compartiendo y donde podría haber conflictos en ese código, y automáticamente generar los bloqueos y desbloqueos de recursos. Este tipo de soluciones, si bien parecen más fáciles y cómodas, son más ineficientes en cuanto al tiempo de desarrollo general de la solución planteada, porque el sistema operativo no sabe explícitamente determinar el lugar optimo para bloquear el recurso, quizás este bloqueando mucho mas tiempo que el que realmente se requiera bloquear el recurso. Entonces el tiempo de ejecución de nuestro programa seguramente será mas lento. Por eso la mayoría de los lenguajes lo dejan a cargo del programador.

BloquearEsquina(avenida,calle)

BloquearEsquina(10,15)

BloquearEsquina(posAv,posCa) bloquea la esquina donde estoy parado

BloquearEsquina(av,ca) según los valores que tengan previamente

Si la esquina llegara a estar bloqueada previamente por otro robot entonces el robot que ejecuto esta instrucción quedara esperando hasta que pueda ejecutarla, es como si se bloqueara. Por eso es tan importante bloquear solo en el momento preciso y desbloquear lo más rápido posible mientras el programa funcione.

LiberarEsquina(avenida,calle)

LiberarEsquina(10,15)

LiberarEsquina(posAv,posCa)

LiberarEsquina(av,ca)

Cuando tenga que hacer un programa en el cual 2 o mas robot podrían llegar a compartir una esquina, lo que debo hacer es bloquear la esquina en cuestión, posicionarme, realizar los cálculos, volver a mi esquina anterior o a una que no tenga problemas de colisión, y liberar la esquina para que otro robot pueda bloquearla.

No debo bloquear la esquina antes de un repetir. No es necesario bloquear recursos si no es necesario. Debo liberar el recurso lo mas pronto posible.