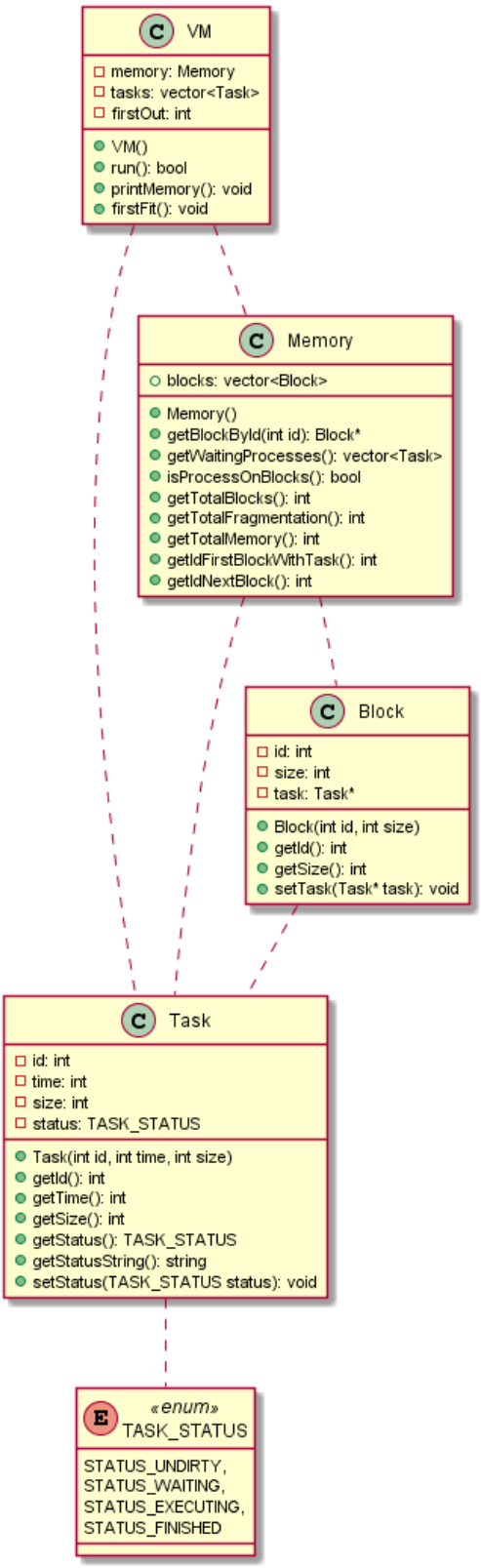
Diagrama UML del simulador de un sistema operativo



"Este proyecto es un fork del original IvyVM hecho por mí el semestre pasado, ahora tiene este simulador de sistema operativo. Puedes consultar el código original en mi Github."

Este simulador ha sido escrito en C++, requiere de Visual Studio versión 2017 en adelante para ser compilado, además es indispensable el Win32 API. En otras palabras, este proyecto solo funciona en sistemas operativos basados en Windows.

La ilustración anterior contiene el diagrama UML de las clases del proyecto que estaré desarrollando del simulador del sistema operativo.

VM

La clase VM (Virtual Machine) es la base del programa. Se encarga de mantener todo en control del simulador.

Atributos

- memory: atributo que contiene un objeto de tipo Memory. Este atributo es utilizado para controlar los bloques de memoria.
- tasks: atributo que contiene un vector de objetos tipo Task. Este atributo es utilizado para la lista de tareas pendientes.
- firstOut: atributo que contiene la siguiente posición del bloque donde se removerá la tarea.

Funciones

- VM(): Simple constructor, inicializa todos los valores.
- run(): Crea un ciclo que se encarga de ejecutar el simulador.
- printMemory(): Imprime el estado de la memoria, tareas, etc.
- firstFit(): Función que se encarga de hacer el primer ajuste.

Memory

La clase Memory es la encargada de manejar las particiones o bloques de memoria. Depende de la clase Block.

Atributos

- blocks: Vector de objetos tipo Block, contiene los bloques de memoria.

Funciones

- Memory(): Constructor que inicializa las variables.
- getBlockById(int id): Función que devuelve el puntero de un bloque según su Id, si no existe devuelve nullptr.

- getWaitingProcesses(): Devuelve un vector con todas las tareas que se encuentran en espera en el momento.
- isProcessOnBlocks(int id): Devuelve cierto o falso si el Id del proceso se encuentra en algún bloque según sea el caso.
- getTotalBlocks(): Devuelve la longitud total de los bloques.
- getTotalFragmentation(): Devuelve la cantidad total de fragmentación interna en la memoria.
- getTotalMemory(): Devuelve la cantidad máxima de memoria, independientemente de que esté en uso.
- getIdFirstBlockWithTask(): Devuelve el id del primer bloque que contenga una tarea. Si el resultado es cero quiere decir que no existe.
- getIdNextBlock(int pos): Devuelve el id del siguiente bloque según el id indicado. Si no se encuentra una posición a partir del id indicado, se devolverá el resultado de getIdFirstBlockWithTask().

Block

La clase Block se encarga de controlar la partición o bloque de memoria.

Si la tarea o task que tiene el bloque no es igual a nullptr, quiere decir que la tarea se encuentra en curso.

Atributos

- id: Id relativo al bloque.
- size: Tamaño del bloque.
- task: Puntero que contiene la tarea que tiene en curso, según sea el caso.

Funciones

- Block(int id, int size): Constructor que inicializa los valores del bloque.
- getId(): Devuelve el id relativo al bloque.
- getSize(): Devuelve el tamaño del bloque.
- setTask(Task* task): Recibe el puntero de una tarea y la asigna al bloque.

Task

La clase Task corresponde a las tareas, ya sea las utilizadas por los bloques o las que están en cola.

Atributos

- id: Id relativo a la tarea.
- time: Tiempo de ejecución de la tarea, informativo.

- size: Tamaño de la tarea.
- status: valor de TASK_STATUS según sea su estado.

Funciones

- Task(int id, int time, int size): Simple constructor que inicializa los valores.
- getId(): Devuelve el id relativo de la tarea.
- getTime(): Devuelve el tiempo de ejecución de la tarea.
- getSize(): Devuelve el tamaño de la tarea.
- getStatus(): Devuelve un valor del enumerador TASK_STATUS según sea el estado de la tarea.
- getStatusString(): Devuelve una cadena de texto con el estado de la tarea según sea el caso.
- setStatus(TASK_STATUS status): Establece el estado de la tarea, requiere un valor del enumerador TASK_STATUS.

TASK_STATUS

<< enum >>

Enumerador que contiene los diferentes estados de las tareas.

- **STATUS_UNDIRTY:** Estado por defecto, hasta este momento la tarea no ha sido utilizada ni se ha tomado en cuenta para los ajustes.
- **STATUS_WAITING**: Estado que indica que la tarea se encuentra en espera y por tanto, es prioritaria. Se le dará preferencia durante el ajuste.
- **STATUS_EXECUTING**: Estado que indica que la tarea se está ejecutando en ese momento, durante este estado la tarea no será considerada en los ajustes.
- **STATUS_FINISHED**: Estado que indica que la tarea ha finalizado. Al tener este estado la tarea no es considerada en los futuros ajustes.

Si desea consultar el estado del proyecto en tiempo real, puede consultar el repositorio de IvyVM en Github.

https://github.com/BrayanIribe/IvyVM