Cristian Rivera López - 2013008650
Ismael Madrigal Campos - 2013084063
Francisco Alvarado Ferllini - 2014080338
Albin Mora Valverde – 2014084952

Explicación del diagrama de clase

El proyecto era muy simple des su inicio, el profesor agrego las clases necesarias para implementarlo, aunque se agregaron mas, como los son las clases de control de errores, el CrazyThread, fHeap (physhical Memory) y MinimalismBitVector. Estas clases extra a las mencionadas en la especificación de proyecto tienen una explicación de estar en el Lightweight Dynamic Memory Manager, en la siguiente enumeración se explicará la aparición de cada una en el proyecto:

1. CrazyThread: Es unas de las clases ya empleadas en el proyecto pasado (Crazy River Ride), esta clase da a paso a la creación de threads hijos, como es el caso de GarbageCollectorThread y del memoryCompactor. La razón de aparecer es que se usan threads y esta clase es ideal para manejarlos.
2. Fheap: Es la clase que se encarga de manejar la paginación, al menos con los métodos de escritura y lectura de las paginas. Lo ideal es que se separan métodos complejos para el vHeap y se colocan en el fHeap, cual si se encarga únicamente de leer del disco y escribir en el disco.
3. MinimalismBitVector: Esta si fue mencionada en la especificación, pero no con este nombre aun así su aparición es importante, pues se cuentan con las siguientes banderas y variables, _Id es el identificador del bitVector, _Weight es el tamaño del objeto a guardar, _Type el tipo del objeto a guardar, _ReferenceCounter es el contador de referencias, _IsUseFlag es la bandera que representa que el objeto esta en uso, _OnMemoryFlag es la bandera que indica si el objeto esta en memoria o en disco, _Offset es el corrimiento desde el Chuck de memoria solicitado, Serial es una variable para hacer seriales para que los id no se repitan.
4. vHeapException y sus clases hijas: Esto es para controlar las acciones incorrectas que tome el programador para ingresar al vHeap. Estas son clases que mejoran la vida al programador, pues controlan acciones corruptas como los punteros nulos.

Cristian Rivera López - 2013008650
Ismael Madrigal Campos - 2013084063
Francisco Alvarado Ferllini - 2014080338
Albin Mora Valverde – 2014084952

Se puede observar que hay algunas funcionalidades que no aparecen como es el caso del visor de memoria, la idea es que el proyecto sea escalar por lo cual solamente contamos con funcionalidades básicas, la implementación de los sockets y visor de memoria además del visor de los dumps, que son funcionalidades que no afectan directamente el uso del vHeap no aparezcan todavía. Debido a que son funcionalidades toscas que nos agrega complejidad al proyecto (Needless Complexity). Las funcionalidades que se consideran importantes y son parte de la etapa de desarrollo son el agregar y quitar los objeto de memoria, también los son la consulta y escritura de los mismos, además los conceptos de paginación, compactación de memoria y garbage collector son los verdaderamente necesarios para que el proyecto pueda usarse. La forma en que se manejan las etapas de desarrollo en el grupo de trabajo es realizar en diagrama UML los funcionalidades a trabajar en un cierto color (las clases son únicamente del color previamente establecido en este caso gris claro), la siguiente etapa en otro color. No se agregan objetos al UML hasta no completar las funcionalidades acordadas en la etapa de desarrollo. Por ello cualquier programador que conociera del proyecto diría que el diagrama esta incompleto pero con cierta razón, pues se quiere hacer un balance con las técnicas ágiles de desarrollo en este caso scrum. Agregando las funcionalidades básicas al inicio y luego agregar funcionalidades terciarias y secundarias. Además por recomendación del profesor (citados en clase), se realizó lo básico dejando de lado lo que son el visor de memoria, la aplicación de prueba y el visor de dumps lo cual se presentará (si no hay retrasos) el día 11 de abril en la defensa del proyecto.