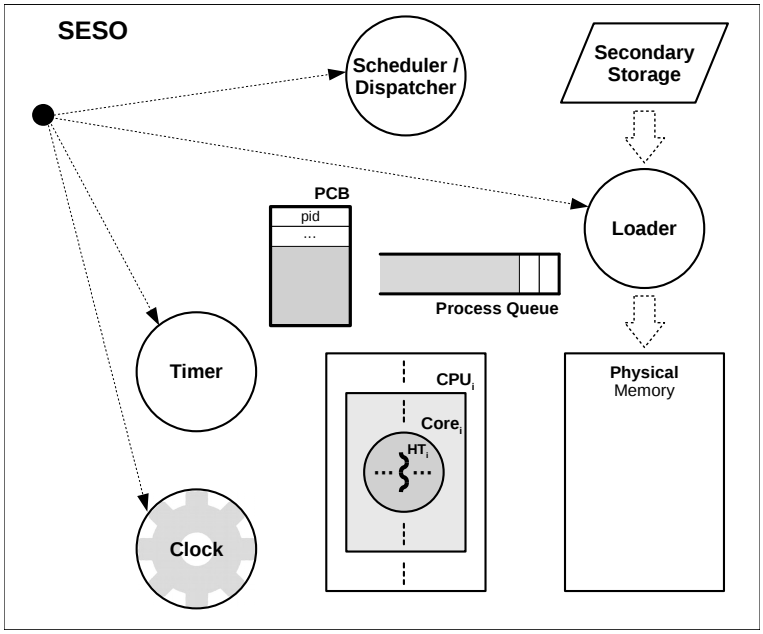
Parte 3. Gestor de Memoria
Fase a

Sobre la base del marco de funcionamiento construido en las dos primeras partes en la que habéis desarrollado un **Scheduler** (y **Dispatcher**) con unas determinadas políticas de planificación, queremos construir un sistema de gestión de memoria virtual. Esta parte la realizaremos en dos fases. En la primera fase (a), construiremos todo el entorno para poder simular el sistema. En la segunda fase (b), construiremos el propio sistema de gestión de memoria virtual. Esta segunda fase, la planteremos la semana que viene.

FASE a. AMPLIACIÓN DE LA ARQUITECTURA DEL SISTEMA

Para ello vamos a modificar parte de lo elaborado y vamos a crear nuevas estructuras de acuerdo al siguiente esquema:

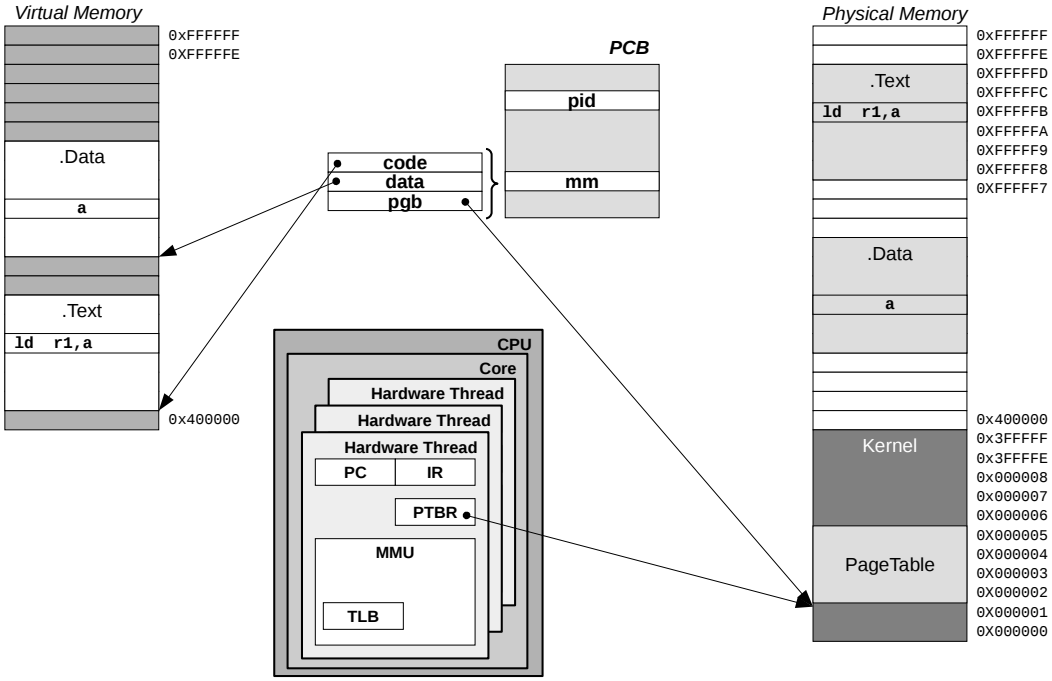


Por un lado vamos a transformar el **Process Generator** en un cargador, **Loader**. La máquina de cómputo, **CPUs**, compuesto por CPUs, cores e hilos hardware, va a ampliar su arquitectura. Por otro lado también crearemos una memoria física, **Physical**, (que la simularemos con un array de tamaño configurable).

El sistema de Memoria Virtual

El sistema de Memoria Virtual va a manejar instrucciones y datos. Para ello vamos a simular unos pequeños programas con unas instrucciones muy básicas, que definiremos en la segunda fase. Este programa se compondrá de sólo dos segmentos: instrucciones (**.Text**) y datos (**.Data**). Cada hilo hardware tendrá asociado, entre otras cosas, una **MMU**, una **TLB**, un registro apuntador a la tabla de páginas (**PTBR**), un registro de instrucción (**RI**) o un **PC**. La memoria física, **Physical**, tendrá un bus de direcciones de 24 bits y sus posiciones de tamaño de una palabra (4 Bytes). Debes tener

en cuenta que el Kernel tiene que tener reservado un tamaño fijo (común a cualquier proceso) que es donde se encontrarán físicamente las tablas de página.



Funcionamiento

El funcionamiento general debería ser el siguiente:

- En primer lugar se debe crear la memoria física y completar los cores.
- El **Loader**, en vez de generar procesos al azar, leerá programas de un fichero de texto que simulará la memoria secundaria (**Secondary Memory**). Deberá cargarlos en memoria y actualizar las estructuras de datos necesarias (p.e. la tabla de páginas). Ten en cuenta que tanto los segmentos **.Text** y **.Data**, como las tablas de páginas deben residir en memoria física.

  Para ello deberá proceder de la manera siguiente:
  - Crear e inicializar el **PCB**.
  - Crear la tabla de páginas del proceso en el espacio del Kernel y con un apuntador físico que se guardará en el campo **pgb** del **PCB**.
  - Cargar el ejecutable desde un fichero de texto. Copiar en memoria física ambos segmentos de código y de datos. Ten en cuenta que el Kernel tiene que tener reservado su propio espacio en memoria.

El **PCB** necesita un campos adicional:

- **mm**: Es una estructura que contiene los campos:
  - **code**: Puntero a la dirección virtual de comienzo del **segmento de código**.
  - **data**: Puntero a la dirección virtual de comienzo del **segmento de datos**.
  - **pgb**: Puntero a la dirección física de la correspondiente tabla de páginas.

Nota.- No vamos a tratar el *swap-out*, sólo la traducción de direcciones lógicas a físicas.