**Memory Management**

Using generic linked list to keep track of free memory blocks

Memory management API provides an interface for processes to interact to when requesting or returning memory.

**Global Variables Used**:

**gHeap** - list to hold free memory blocks,

**MEM\_BLOCK\_SIZE**: 128 bytes,

**NUM\_MEM\_BLOCKS**: 30

**mem\_blocks:** holds all memory blocks

|  |
| --- |
| METHOD 1: void MEMORY\_INIT(void) |
| allocate memory for RTX\_IMAGE  allocate memory for PCB pointers  allocate memory for each process stack  update stack pointer  allocate memory for heap, size equals NUM\_MEM\_BLOCKS\*128 bytes |

|  |
| --- |
| METHOD 2: void\* K\_REQUEST\_MEMORY\_BLOCK(void) |
| while no memory available  add the calling process to the BLOCKED\_ON\_RESOURCE queue  k\_release\_processor();  once is memory available  pop the memory block from the heap  and return it to the calling process |

|  |
| --- |
| METHOD 3: K\_RELEASE\_MEMORY\_BLOCK(void\*) |
| Check if the block being freed is valid  If the block is valid  Add the block back the heap (gHeap List)  Follow preemption policy to assign the memory to processes |

|  |
| --- |
| METHOD 4: K\_RELEASE\_MEMORY\_BLOCK\_VALID(void\* p\_mem\_block) |
| ***// A helper function to check if the memory block being released is actually valid***  return RTX\_ERR if p\_mem\_block is invalid  return RTX\_ERR if address pointed by p\_mem\_block is outside the bounds  return RTX\_ERR if the p\_mem\_block size is not 128 bytes  return RTX\_ERR if trying to free a block that is already free  return RTX\_OK |