Assessment 2 – Testing Document

The following table lists several tests carried out on the physical memory manager to ensure it meets the given requirements and is robust. Testing will be done on the individual functions of the physical memory manager ensuring they do their expected operations correctly and without issue.

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test description** | **Test results** |
| 1 | When initialising the physical memory manager, the bitmap should be placed at the address 0x106000. | When running the OS, the console correctly outputs “Physical Memory Map: Address: 0x106000”. |
| 2 | The initialise function should determine the optimum size of the bitmap. | The console prints out the correct size of the bitmap which should be 1023 bytes. This is worked out from dividing the number of blocks per byte (8) by the total number of blocks (8184). |
| 3 | Each memory region inside the memory map from the boot info structure should be marked as used. | The number of blocks that are free is listed as 0 and used blocks set to total number of blocks as expected. Memory is set as used through memset. |
| 4 | Once all memory has been marked as used, each region marked as available should be then marked as unused so that memory can be allocated. | Using the boot info structure, going through each region that was marked as available, calling the Free Blocks function with the start address of the region and the size of the region multiplied by the size of a block (4096 bytes), the correct number of blocks are marked as free and used. |
| 5 | FreeBlock should take an address of a block of memory and mark it as used. The number of free blocks should also be decremented by 1. | Passing a memory address to FreeBlock first checks if the bit for the given address is set, if so then clears that bit and decrements used blocks by 1, ensuring that block counts are changed only if bits are changed. |
| 6 | FreeBlocks should take an address and number of blocks to free, and unset each bit in the bitmap from start address + size, then decrement the number of blocks as used by the given size. | Passing a memory address and number of blocks to free will test if the bit at address is set, and then attempt to clear each bit after it and decrement the number of used blocks by 1 only if a bit was cleared. |
| 7 | AllocateBlock should allocate a single bit in the bitmap for a single block, then return the address for that block. | Allocating a single block finds the first available bit that is available in the bitmap, sets the bit to used, incremented used blocks count and returns the address of that block. |
| 8 | AllocateBlocks should take several blocks to allocate then return the address of the 1st block that was allocated. | Allocating multiple blocks finds the first available bit that has continuous free bits available after it equal to the given size, then sets each bit to used, increments the used block count for each bit that gets set, and returns the address of the first bit. |
| 9 | Testing allocation of a single block and checking its address. | Allocating a single block, will print out to the console that a block was allocate, and the address the block was allocated too. The first block as expected is allocated at address 0x1000. |
| 10 | Testing allocation of 100 blocks and checking its address. | Allocating a chunk of 100 blocks successfully allocates at address 0x2000. The number of blocks and the address are also output to the console. |
| 11 | Testing allocating a single block again, to ensure it is positioned correctly after the 100 blocks. The expected address is 0x66000. | Allocating a single block after the 100 blocks successfully allocates at address 0x66000 and outputs the address to the console. |
| 12 | Test allocating a larger chunk of 500 blocks. Since region 0 (1st available memory region) is only 159 blocks, and the memory map is located at address 0x106000, the expected start for these blocks is 0x107000. | Allocating 500 blocks will correctly allocate at start address 0x107000. As region 3 (next available memory region) starts at address 0x100000, and the memory map is allocated at 0x106000, the next available address with 500 free bits in the bitmap is 0x107000. |
| 13 | Testing deallocating the first block (address 0x1000). Expected results are the console to output a single block at address 0x1000 was deallocated. | Deallocating a single block at address 0x1000 correctly outputs that a block at address 0x1000 was deallocated. |
| 14 | Test allocating a single block again, to ensure that the bit for the previously deallocated block was cleared and the block being allocated now is placed at address 0x1000. | Allocating a single block after deallocating a single block at address 0x1000, correctly allocates a new single block at address 0x1000 showing that the bit in the bitmap is cleared correctly and the address is made available. |
| 15 | Test deallocating a chunk of 100 blocks at address 0x2000. | Deallocating a chunk of 100 blocks at address 0x2000 correctly prints to the console that 100 blocks were unallocated at address 0x2000. This is guaranteed to ensure that 100 blocks were deallocated as the FreeBlocks function only prints out the number of deallocated blocks based on how many bits in the bitmap were actually cleared (set beforehand). |