CS240 Assignment 2

Fall 2013

Emaad Ahmed Manzoor (129110)

Goal. To manage physical memory allocation using a bitmap.

Files Modified. The following files were modified to implement and initialize the bitmap allocator.

➔ `param.h:`	The definitions of `RAMSIZEMB` and `BITMAPSIZE`.
➔ `main.c:`	The modified call to `kinit1.`
➔ `kalloc.c:`	The bitmap initialization and implementation.

Tests. The bitmap allocator was verified using `usertests`; a user program provided by xv6 that tests various kernel functions. For this assignment, the important tests are `sbrktest` and `mem`, both of which pass on the implemented allocator.

Bitmap Size. I assumed the amount of physical memory the QEMU machine has, and then used that to calculate the number of bytes in the bitmap. The bitmap size is calculated as:

```
bitmap size (bytes) = physical memory size (bytes) / page size (bytes) / 8
```

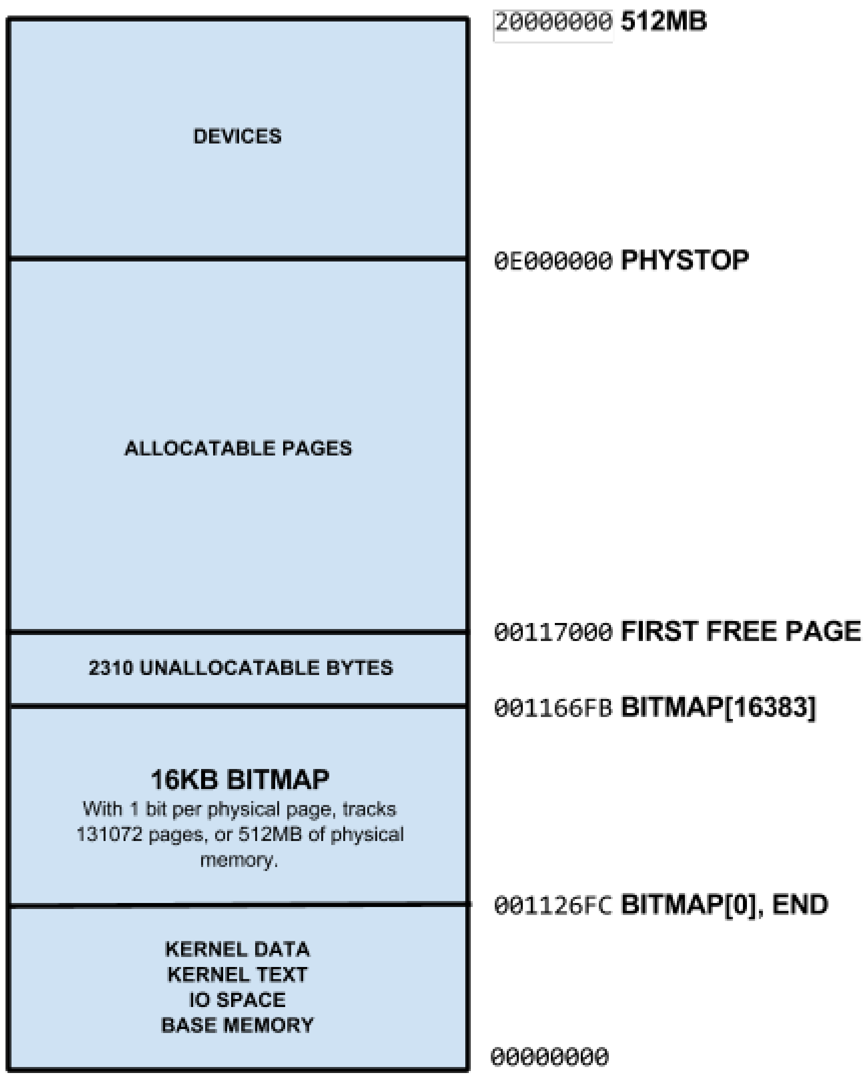
Initialization. The bitmap occupies the space from `end` to (`end + BITMAPSIZE)`. The call to `kinit1` in `main` is modified to free the initial pool of memory beginning at (`end + BITMAPSIZE)`. Before the call to the first `freerange`, `kinit1` initializes all the bitmap bits to 0 (indicating that they're allocated). Some bytes will remain unallocatable between `(end + BITMAPSIZE)` and the first free page address, because `(end + BITMAPSIZE`) is not page-aligned.

On the following page is a physical memory map of the situation for 512MB of physical memory, showing the locations of the bitmap, unallocatable bytes and allocatable pages.

Bitmap interface. The bitmap interface is composed of the following 3 functions:

`bitmap_set(uchar* const bitmap, uint idx):` Sets the bit at index `idx`.
`bitmap_clear(uchar* const bitmap, uint idx):` Clears the bit at index `idx`.
`bitmap_ffz(uchar* const bitmap, uint startIdx, uint endIdx):`
Returns the first cleared bit in the `bitmap` between `startIdx` and `endIdx`, excluding `endIdx`.

The functions take a constant first argument, and hence guarantee not to modify the bitmap. Since these functions are not thread-safe, the calls to these functions in `kfree` and `kalloc` lock the bitmap while operating on it.



Free page discovery. `kalloc` searches for the next free page using the *next-fit* algorithm. Searches for free pages are linear scans of the bitmap beginning after the index of the last freed page.

> *Hardware first-bit-set:* x86 provides hardware instructions to locate the first bit set in a word. These are exploited to speed up the allocation scan when looking for free pages within a word.
>
> To evaluate whether this optimization was required, I profiled the memory allocation time. The system call `uptime` was used to measure the number of clock ticks to complete the `mem` test (that allocates and de-allocates all available memory). The results tabulated below indicate that the performance improvement is unnoticeable. Either `uptime` is an inaccurate profiling tool, or there may be other factors that dominate the memory allocation time.

	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Mean Ticks**
Hardware	7829	7891	4651	8065	8101	7307.4
Software	7320	7308	4200	7204	7114	6629.2