KING ABDULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

CS240 - Assignment 2

We're done with the Counter . . . now we just need to find the right Bit.

Maruan Al-Shedivat (ID: 129096)

October 19, 2013

SOLUTION DESCRIPTION

In this homework assignment we were to reimplement xv6 memory allocation procedures using bitmap approach instead of the originally implemented linked list approach.

In my solution, I had to edit only the *kalloc.c* file. The solution is based on several additions to the $kmem$ structure:

- $freelist$ pointer was substituted with an array $kmem.freebitmap$,
- $working\_byte$ index was added to keep track of the current index of non zero byte of the $freebitmap$ array; $ceiling$ pointer holds the currently maximum available virtual memory address.
- $kinit1$, $kinit2$, $kalloc$, $kfree$ were edited as well as some new axillary functions were introduced to make the work with the bitmap as clear as it is possible.

The $freebitmap$ size ($BMSIZE$) was chosen according to the QEMU parameters: the total number of bits in the bitmap was chosen to be roughly[1] equal to $512MB - (v2p(end) + BMSIZE)$. These many bits were more than enough for covering all the memory from $end + BMSIZE$[2] to $PHYSTOP$ (= 224MB). The exact formula to compute $BMSIZE$ is

[1] Here I say roughly, because the used granularity for the bitmap was 8 bits (1 byte).

[2] Since the $end + BMSIZE$ pointer could point to a middle of some page, in the code we used $PGROUNDUP$ to start allocating from the first completely free page right after the $end + BMSIZE$.

$$BMSIZE = (512MB - v2p(end))/(8 * PGSIZE + 1).$$

The bitmap is initialised in $kinit1$. It is placed exactly at the end. The memory from $end + BMSIZE$ to $vend$ is allowed to be allocated. $kmem.freebitmap$ gets 1 values for all its bits which denotes that all the pages are currently available for allocation (i.e. free). When a page is allocated by $kalloc$, the corresponding bit in the $kmem.freebitmap$ is flipped to 0. An example of the bits indexing in the $kmem.freebitmap$ is presented in the table below:

Value:	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	...
Index:	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	...

Table 1: In this example of the $kmem.freebitmap$, only two pages are allocated.

In $kinit2$ we extend the allocation memory limit up to $PHYSTOP$ and also turn on kmem locking.

The most time consuming procedure in the bitmap approach for memory allocation seems to be free page finding. The simple approach is just to scan the whole bitmap from left to right until we find a 1 bit. The more clever approach was implemented for this assignment, though. In my implementation, I create a special pointer in the $kmem$ structure called $working\_byte$. This pointer is kept to always point to the lowest index byte in $kmem.freelist$ that is non zero (i.e. it has some non zero bits). Then, the process of finding a free page for allocation requested by $kalloc$ consumes a constant time: we look only at the $working\_byte$ and find the index of non zero bit in it. Also, since we keep $kmem.working\_byte$ being the left most and non zero, when it becomes zero we need to scan from its current position to the right and find a new $working\_byte$. This solution should significantly outperform the simple scan, since new free page finding will have constant amortized time cost.

Correctness Check

To check the correctness of the bitmap allocation implementation, I performed a couple of tests. First, it compiles and boots smoothly; user programs work well in the xv6 shell after the patching of $kalloc.c$; all the usertests pass. Second, regardless the early initialization, any extra memory allocation between calls $kinit1$ and $kinit2$ will cause a kernel panic during the runtime, as it will with the original free list implementation. As an example, the following test was performed: the allocation limit extension in $kinit2$ was turned off, then the $kinit1$ call in *main.c* was changed to "$kinit1(end, P2V(4 * 1024 * 1024/2 - 512 * 1024))$;" just to decrease a bit more the allocation limit. It eventually caused the following behaviour of the system:

```
qemu -nographic -hdb fs.img xv6.img -smp 2 -m 512
xv6...
cpu1: starting
cpu0: panic: userinit: out of memory?
 80104385 801036e2 0 0 0 0 0 0 0 0
```