Seminar 2: Buddy

Sadok Dalin

2018-11-29

1 Problem

When implementing malloc and free one or more data structures and/or algorithms are necessary to track the number of free blocks in the system. The buddy algorithm solves this problem by dividing memory into fixed-size blocks in power of two increments. Each block has a single unique buddy which can be reached by flipping a bit in the blocks virtual address. An advantage of this approach is that the algorithm guarantees memory utilization between 50-100% with an average of 75%, i.e. the algorithm has some internal fragmentation. In this seminar, the buddy algorithm was implemented along with a benchmark procedure. The problem was divided into several smaller problems:

- Implement the find procedure which given a level of a block either finds a suitable block in the free list or maps and recursively splits a new block using mmap.
- Implement the insert procedure which given the header of a block inserts it into the free list and merges it recursively with its buddy until it finds a buddy which is not in the free list.
- Implement a benchmark, of which the output is used to plot and compare the amount of allocated memory using mmap with the total allocated size of blocks(external fragmentation), and the total allocated size of blocks with the allocated amount of memory using balloc/bfree(internal fragmentation).

2 Solution

struct head find(int index):
The find procedure is responsible for finding and returning a free block to balloc. At first the free list is checked at the index level to see if there are any free blocks. If no free block exists a new one is created using the new procedure. The algorithm then enters a for-loop where the block is split recursively until the required index or size, has been reached. When splitting the block the block's level and the level of the split blocks buddy is decremented. Each time the block is split the buddy of the split block is inserted into the free list. An if statement is used to handle the case where the list already has free blocks in it. After the required index has been reached the split block is marked as taken and returned to the caller.

If there is a free block a free block in the list the first block in the list is unlinked. The list is then reconnected by setting the next and previous pointers of the second block in the list. The unlinked block is then marked as taken and returned to the caller.

struct head insert(struct head* block):
The insert procedure is responsible for inserting an allocated block into the free list. Initially, the level of the block is saved and used to loop from the level to the max level of a block. In each iteration, the free status of the buddy is checked. If the buddy is free the buddy is unlinked from the free list and merged with the original block. Two different cases exist for unlinking the block from the middle of the list and the beginning of the list. This algorithm iterates until the buddy of the block is not free.

If the buddy is not free the block is inserted into the free list at the current iteration value. Depending on if the list is empty or if the list already has free blocks in it the pointers are set differently. The procedure then returns 0.

3 Benchmarks

The included split, merge and buddy procedures were tested extensively using a for-loop. Initially, a new block was created and split recursively to the lowest level. At each iteration, the level of the block, the buddy, the buddy of the buddy, the split, the buddy of the split, the merged buddy etc.. was printed and verified manually to gain an understanding of the procedure implementations. To test the implementation of the find and insert procedures 3 randomly sized blocks were allocated and freed with the

expected result of 1-3 level 7 free blocks. The output for the first test can be seen below.

```
adress of new block 0x7fd0c3d7c000
end of new block 0x7fd0c3d94000
level 6, block 0x7fd0c3d7c800, buddy 0x7fd0c3d7c000, budbud 0
    x7fd0c3d7c800, budbudbud 0x7fd0c3d7c000, split 0x7fd0c3d7cc00,
    buddy(split) 0x7fd0c3d7c800, merge(split) 0x7fd0c3d7c800, merge(
    buddy(split)) 0x7fd0c3d7c000
level 5, block 0x7fd0c3d7cc00, buddy 0x7fd0c3d7c800, budbud 0
    x7fd0c3d7cc00, budbudbud 0x7fd0c3d7c800, split 0x7fd0c3d7ce00,
    buddy(split) 0x7fd0c3d7cc00, merge(split) 0x7fd0c3d7cc00, merge(
    buddy(split)) 0x7fd0c3d7c800
level 4, block 0x7fd0c3d7ce00, buddy 0x7fd0c3d7cc00, budbud 0
    x7fd0c3d7ce00, budbudbud 0x7fd0c3d7cc00, split 0x7fd0c3d7cf00,
    buddy(split) 0x7fd0c3d7ce00, merge(split) 0x7fd0c3d7ce00, merge(
    buddy(split)) 0x7fd0c3d7cc00
level 3, block 0x7fd0c3d7cf00, buddy 0x7fd0c3d7ce00, budbud 0
    x7fd0c3d7cf00, budbudbud 0x7fd0c3d7ce00, split 0x7fd0c3d7cf80,
    buddy(split) 0x7fd0c3d7cf00, merge(split) 0x7fd0c3d7cf00, merge(
    buddy(split)) 0x7fd0c3d7ce00
```

The benchmark was implemented by randomly allocating a random amount of memory and then randomly deallocating the latest allocation for an x number of operations. Time was modeled as the total amount of calls to balloc and bfree. A stack(array) was used to save pointers to all allocated blocks along with the argument saved to balloc. The rand procedure with seeds based on the clock was used to generate a realistic plot. Figure 1 illustrates the output of the benchmark. The space between the mmap and the allocated blocks lines illustrate the external fragmentation while the space between the allocated memory and the allocated blocks illustrate the internal fragmentation.

4 Conclusion

Multiple problems and bugs were encountered and solved during the seminar:

- The split function was initially missing from the instructions.
- The documentation for the split and merge functions was not clear. Initially, I thought that the split and merge procedures decreased and increased the level of the block respectively. This resulted in an absurd amount of segmentation errors when trying to find the buddy of a

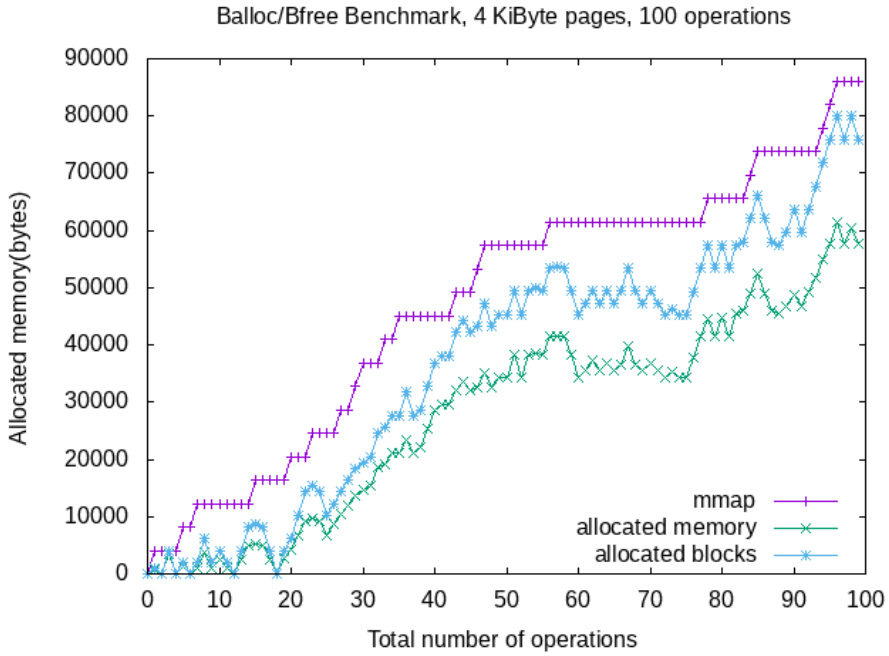

Figure 1: Benchmark of balloc/bfree

block. After extensive testing of both split, merge and buddy(after I had tried to implement find and insert) I discovered that level of the blocks had to be changed manually after each call for buddy to work properly. This solved 90% of all the issues I had.

- The instructions for how to test the procedures should have been more clear. One test, for example, could have been splitting a new block and checking that the buddy of the split block was the original block address.

- The instructions for implementing the benchmarks were inconsistent. In the walkthrough of the seminar, it was clear that a graph plotting allocated memory against time was necessary. This was not at all obvious after reading only the written seminar instructions. In the future, an example picture of a graph could be included. Some instructions on how to use Gnuplot could also have been copied over and adapted from the assignments.