UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
SCHOOL OF COMPUTER SCIENCE & APPLIED MATHEMATICS

COMS3010A
Operating Systems

Project 1B - Buddy

Instructors

COMS3010A Lecturer:

Mr. Branden Ingram	branden.ingram@wits.ac.za

COMS3010A Tutors:

Mr. Joshua Bruton	1699140@students.wits.ac.za
Mr. Mark Alence	1676701@students.wits.ac.za
Mr. Tshepo Nkambule	1611821@students.wits.ac.za
Mr. Dylan Brandt	1352906@students.wits.ac.za
Mr. Skhumbuzo Dube	1608194@students.wits.ac.za

Consultation Times

Questions should be firstly posted on the moodle question forum, for the Lecturer and the Tutors to answer. If further explanation is required consultation times can be organised.

1 Introduction

This is an assignment where you will implement your own malloc using the buddy algorithm to manage the splitting and coalescing of free blocks. You should be familiar with the role of the allocator and how to implement and benchmark a simple version of it like we did last week. In this project you will not be given all details on how to do the implementation, but the general strategy will be outlined. In PartA of the project you implemented the functions that allow us to perform operations on the blocks and in PartB you will now implement the actual Buddy algorithm. A Project Template has been provided on moodle to use with the correctly defined functions, however, you may still use your own. You are required to finish the implementation as well as answer a few questions related to your project. This will be in the form of a moodle quiz.

2 The buddy algorithm

The idea of the buddy algorithm is that given a block to free, we could quickly find its sibling and determine if we can combine the two into one larger block. The benefit of the buddy algorithm is that the amortized cost of the allocate and free operations are constant. The slight down side is that we will have some internal fragmentation since blocks only come in powers of two. We will keep the implementation as simple as possible, and therefore we will not do the uttermost to save space or reduce the number of operations. The idea is that you should have a working implementation and do some benchmarking which you can later optimise upon. It is important to note that you are allowed to create any additional functions or files in order to solve the problem, just ensure that they along with a makefile are included in your submission.

I encourage you to first write down the pseudo code, what needs to be done in each step. Draw pictures that describes how the double linked list is manipulated. Do small tests as you proceed, make sure that simple things work before you continue.

2.1 the implementation 2 marks

So given that the primitive operations work as intended, it is time to implement the algorithm. Your task is now to implement two procedures balloc() and bfree() that will be the API of the memory allocator. We do not replace the regular malloc() and free() since we want to use them when you write your benchmark program. You will use one global structure that holds the double linked free lists of each layer.

```
struct head * flists[LEVELS] = {NULL};
```

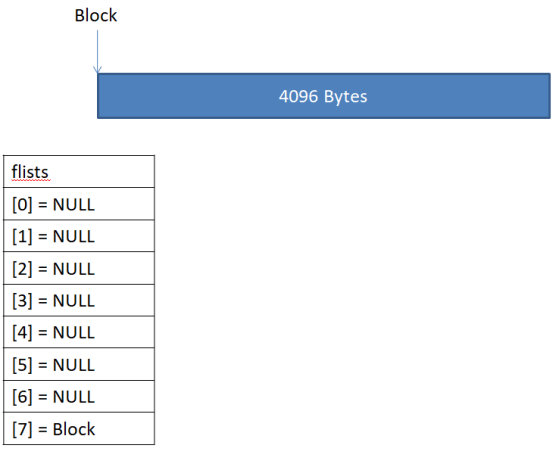

Figure 1: Flists Example

Our flists is basically an array of linked lists where each track the blocks of a specific size. Here since we have only initialised our new block we have one entry in the list at index 7.

2.2 balloc 12 marks

The first part of our buddy memory allocator will be the allocation component. Below is the function structure as well as some psuedocode

```
void *balloc(size_t size) {
        //Check if size is 0
        //Determine required level based upon size
        //Based on that level do the necessary splitting
        //   if required remembering to update flists and
        //   return a pointer to the allocated memory block
        //Finally hide the header and return the pointer
}
```

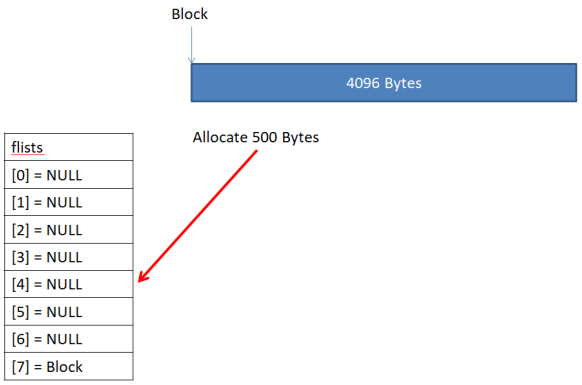

Figure 2: Allocation Example

If we were to allocate 500 Bytes we would need to first determine the index corresponding to the smallest block required to store this amount of data. In this case it would be 4. Since flists[4] is Null we need to split in order to divide our memory into the required block size. If there was already a free block in this list we would only need to return a pointer to it.

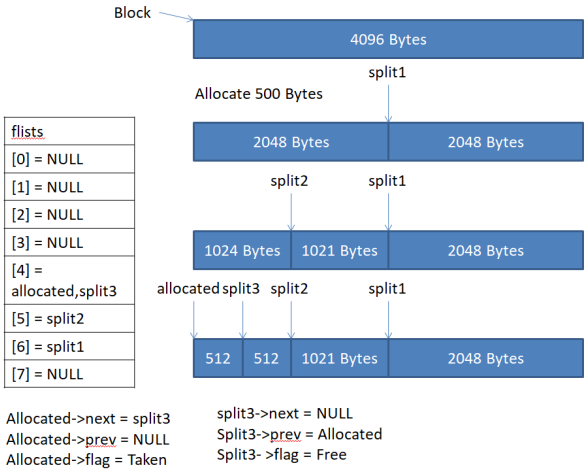

Figure 3: Allocation Example

Here you can see some information demonstrating the process of splitting that would be required to find a pointer to a block of the right size. Additionally you can see the state of flists. Note how it no longer contains an item in flists[7] since that block is no longer available. It is up to you to decide in which way you add or remove from the linked lists.

2.3 bfree 12 marks

The second part of our buddy memory allocator will be the free component. Below is the function structure as well as some psuedocode

```
void bfree(void *memory) {
        //Check memory is a null pointer
        //Get the header of the pointer
        //Update flists and free the block which the pointer points to
}
```

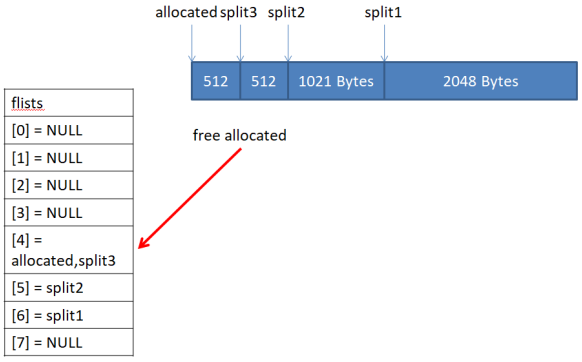

Figure 4: Free Example

To free some memory we need to identify which part of flists it belongs to.

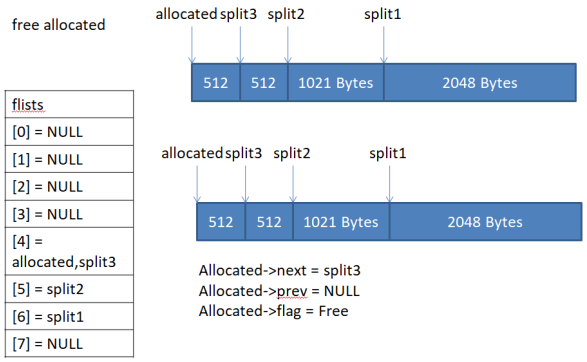

Figure 5: Free Example

We then have to update that block in terms of its status.

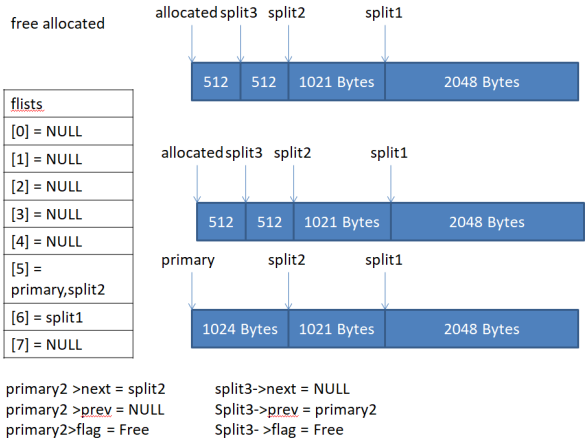

Figure 6: Free Example

lastly if merging is possible, i.e. if its buddy is also free, we need to combine these two blocks and update flists accordingly. If this results in the possibility to merge further up the tree you need to do so.

3 submission

Please upload a zip file containing all files to the moodle submission.

- implementation of flists : **2 marks**
- implementation of balloc : **12 marks**
- implementation of bfree : **12 marks**
- Moodle Quiz Questions(Write-up) : **10 marks**
- **total 36**

Academic Integrity

There is a zero-tolerance policy regarding plagiarism in the School. Refer to the General Undergraduate Course Outline for Computer Science for more information. Failure to adhere to this policy will have severe repercussions.

During assessments:

- You may not use any materials that aren't explicitly allowed, including the Internet and your own/other people's source code.
- You may not access anyone else's Sakai, Moodle or MSL account.

Offenders will receive 0 for that component, may receive FCM for the course, and/or may be taken to the legal office.