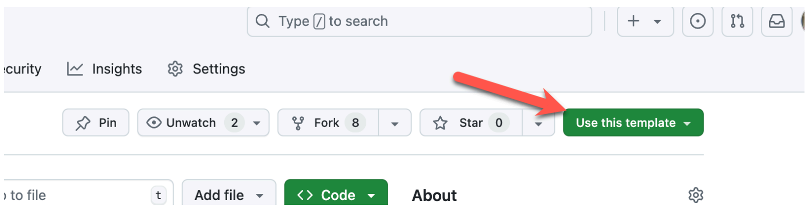
P3

Start Assignment

- Due Tuesday by 11:59pm
- Points 100
- Submitting a website url
- Available until Apr 9 at 11:59pm

Overview

- Fork (**use this template**) the **starter repository (https://github.com/shanep/makefile-project-starter)** into your own GitHub account
- Complete the lab as detailed below



Learning Outcomes

- 2.1 Demonstrate how low level memory is managed in user space
- 2.2 Explore the system call interface
- 3.1 Analyze a complex computing problem and apply principles of computing and other relevant disciplines to identify solutions.

Tasks

Task 1 - Prepare your repository

The starter repository is a bare bones template that you will need to update with the starter code below.

- **lab.h (https://boisestatecanvas.instructure.com/courses/38319/files/18935026?wrap=1) (https://boisestatecanvas.instructure.com/courses/38319/files/18935026/download?download_frd=1)**
- **test-lab.c (https://boisestatecanvas.instructure.com/courses/38319/files/18935027?wrap=1) (https://boisestatecanvas.instructure.com/courses/38319/files/18935027/download?download_frd=1)**
- **lab.c (https://boisestatecanvas.instructure.com/courses/38319/files/18947349?wrap=1) (https://boisestatecanvas.instructure.com/courses/38319/files/18947349/download?download_frd=1)**

The testing code is **NOT** comprehensive. You will need to add additional tests to get better coverage. This project will be 100% unit test. You will not need to use the file `app/main.c` all your work will be done int `src/lab.c` and `tests/test-lab.c`

Task 2 - Implement Buddy

Implement the **Buddy Algorithm (https://boisestatecanvas.instructure.com/courses/38319/files/18935101?wrap=1)** .

WARNING

You are NOT allowed to use **any functions** from the math library! You are required to use bit shifting and masking for your calculations. So this means no `pow()` or `log2()` functions should be found in your code.

Task 3 - Write Unit tests

Add enough tests beyond what the professor has given you to get as close to 100% coverage as possible.

Additional Resources

From the example in Knuth's book we can calculate the buddy of a block of size 16. With the table above we know that a block of size 16 is K=4 so we will have 4 zeros at the right. Using the **XOR operator (https://en.wikipedia.org/wiki/Exclusive_or)** we can calculate the buddy as follows:

```
   101110010110000
 ^ 000000000010000
-------------------
   101110010100000
```

Knuth notation to C syntax quick reference

```
struct buddy_pool *pool;
struct avail *L;
```

Knuth notation	**C syntax**
AVAILF[k]	pool->avail[k].next
AVAILB[k]	pool->avail[k].prev
LOC(AVAIL[k])	&pool->avail[k]
LINKF(L)	L->next
LINKB(L)	L->prev
TAG(L)	L->tag
KVAL(L)	L->kval
buddyk(L)	buddy_calc(pool, L);

Power of 2 quick reference

As specified in the algorithm in order for everything to work you must work in powers of two! This includes the address that you get back from mmap. You will need to scale the address appropriately or you will calculate invalid buddy values. You can use the table below for a quick reference for up to K=40.

Power (k value)	Bytes	KiB/MiB/GiB/TiB
2^0	1	
1	2	
2	4	
3	8	
4	16	
5	32	
6	64	
7	128	
8	256	
9	512	
10	1,024	1 KiB
11	2,048	2 KiB
12	4,096	4 KiB
13	8,192	8 KiB
14	16,384	16 KiB
15	32,768	32 KiB
16	65,536	64 KiB
17	131,072	128 KiB
18	262,144	256 KiB
19	524,288	512 KiB
20	1,048,576	1 MiB
21	2,097,152	2 MiB
22	4,194,304	4 MiB
23	8,388,608	8 MiB
24	16,777,216	16 MiB
25	33,554,432	32 MiB
26	67,108,864	64 MiB
27	134,217,728	128 MiB
28	268,435,456	256 MiB

Power (k value)	Bytes	KiB/MiB/GiB/TiB
29	536,870,912	512 MiB
30	1,073,714,824	1 GiB
31	2,147,483,648	2 GiB
32	4,294,967,296	4 GiB
33	8,589,934,592	8 GiB
34	17,179,869,184	16 GiB
35	34,359,738,368	32 GiB
36	68,719,476,736	64 GiB
37	137,438,953,472	128 GiB
38	274,877,906,944	256 GiB
39	549,755,813,888	512 GiB
40	1,099,511,627,776	1 TiB

Hints

- There should be no hard coded constants in your code. You will need to use the macros defined in the **C99 standard** (https://en.cppreference.com/w/c/types/integer) to get the correct width. For example to calculate a K value 64bit system you would write `UINT64_C(1) << kval` NOT `1 << kval`
- Remember to take into account the size of your header when calculating your K value. If a user asked for 32 bytes you will need to use K=6 not K=5 to account for the header size.
- If the memory cannot be allocated, then your buddy_calloc or buddy_malloc functions should set errno to ENOMEM (which is defined in errno.h header file) and return NULL.
- You can’t use malloc/free/realloc/calloc from the C standard library.
- You will need to make extensive use of pointer arithmetic! When you are dealing with raw (untyped) memory make sure and cast the pointer to the correct type so it is moved the correct amount.
- Be mindful of the differences between `sizeof(struct foo)` and `sizeof(struct foo*)`. The first instance give you the size of the struct while the second gives you the size of a pointer. If the size of the struct just **happens** to be the same as a pointer your code will appear to work until you either add a new member to `foo` OR you move to a system where the sizes of pointers are different. For example moving from a 64bit system to a 32 bit system or vice versa!
- Read **Chapter 14 - Interlude: Memory API** (http://pages.cs.wisc.edu/~remzi/OSTEP/vm-api.pdf)

Extra Fun Reading

- **Appendix B. Index to Notations** (https://www.oreilly.com/library/view/art-of-computer/9780321635754/app03.html)
- **C99 Standard** (http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf)

Debugging Raw memory

In the C programming language we can allocate a chunk of memory on the heap and treat that chunk of memory as an array. If you are working on debugging a problem and want to inspect the contents of the array using the GUI debugger interface in VSCode you may have to tell the debugger (with a cast) that a pointer is actually pointing to a dynamically allocated array not a single variable. This example walks through how to display a pointer as an array that is embedded within a struct.

More reading about C and GDB.

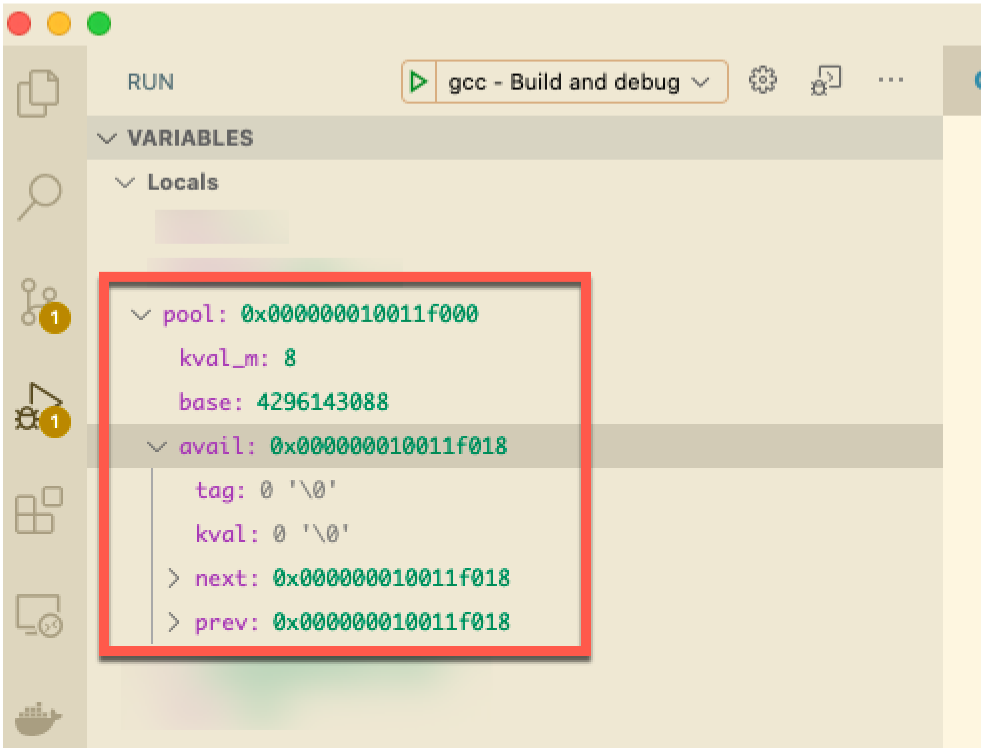
- **Reading C Declarations (http://unixwiz.net/techtips/reading-cdecl.html)**
- **GDB Artificial Arrays (https://sourceware.org/gdb/current/onlinedocs/gdb/Arrays.html)**
- **GDB to LLDB usage (http://lldb.llvm.org/use/map.html)**
- **VSCode Data inspection (https://code.visualstudio.com/docs/editor/debugging#_data-inspection)**

Consider the struct declaration `buddy_pool` shown below. The `avail` member is a pointer that we must dynamically allocate and want to display in the debugger as an array. We can allocate the a `buddy_pool` struct (in the stack or data segment) and then dynamically allocate the `avail` array using `malloc`.

c

```c
struct avail
{
    int tag;
    int kval;
    struct avail *next;
    struct avail *prev;
}
struct buddy_pool
{
    size_t kval_m;
    uintptr_t base;
    struct avail *avail; /*pointer to the start of the avail array*/
};
struct buddy_pool pool;
pool.kval = 9;
pool.base = 0;
pool.avail = malloc(sizeof(struct avail) * 9);
```

If we run the debugger we will see the variable `pool` with the element `avail` is displayed as a single variable not an array of 9 structs as we expected.



```
pool: 0x000000010011f000
  kval_m: 8
  base: 4296143088
  avail: 0x000000010011f018
    tag: 0 '\0'
    kval: 0 '\0'
    next: 0x000000010011f018
    prev: 0x000000010011f018
```

The element `avail` is just a pointer to the memory address of element and the debugger can't determine the size of the array and thus will display it as a single struct instead of an array as expected.

Fortunately, all is not lost! Most debuggers allow you to set a watch on a memory location and you can force the debugger to cast the memory to a certain type. Both gdb and lldb have specific commands to display a memory block as an array. However, using casting works regardless of what debugger you are using.

If we add a new **watch** (https://code.visualstudio.com/docs/editor/debugging#_data-inspection) on a variable and then force the debugger to display the memory block as an array instead of a single variable we can easily inspect the data and track down any issues you are experiencing.

`(struct avail(*) [9])pool->avail`

For a plain old dynamic array you can add a watch expression that is set to to the desired type.

`*(int(*)[10])A`

Task 4 - GRAD Students

Grad students are required to implement realloc as noted in lab.c.

Submitting

Make sure all your code is pushed to GitHub, and then submit the URL to your public repository.