CS3.304.M21 Advanced Operating Systems

Project Final Report

Buddy Algorithm

Team name: hello_world

Team members:

- Vineet Agrawal (2021201049)
- Kamal Phoolwani (2021201054)
- Rahul Katyal (2021201083)
- Shravan Sharma (2021201058)

Instructor: Prof Manish Shrivastava
Mentor: Agrima Singh

Introduction

The project is about implementing a memory management library that makes use of the Buddy algorithm for memory allocation. The algorithm used in the project helps to allocate space in shared memory replacing the malloc function which allocates spaces dynamically in the common standard c libraries. The Buddy algorithm requires all memory blocks to be power of two and tracks the empty memory blocks via a vector data structure.

Problem Description

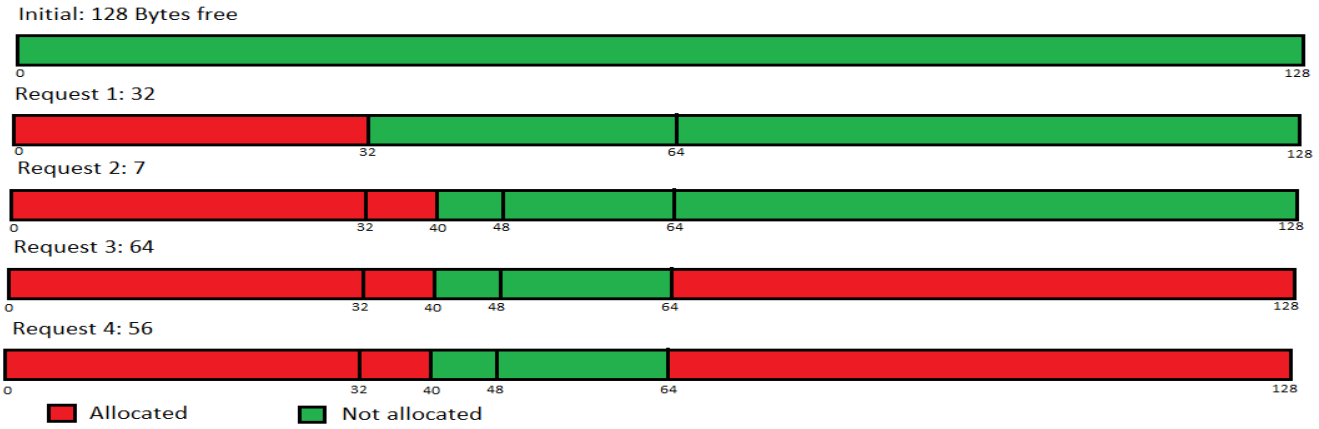
To write a malloc library that provides the following dynamic memory allocation routines (as defined in man 3 malloc):

- void *malloc(size_t size);
- void free(void *ptr);
- void *calloc(size_t nmemb, size_t size);
- void *realloc(void *ptr, size_t size);
- void *reallocarray(void *ptr, size_t nmemb, size_t size);
- int posix_memalign(void **memptr,size_t alignment, size_t size);
- void *memalign(size_t alignment, size_t size);

Also, other relevant functions that are typically part of a malloc library also implemented as per the requirements.

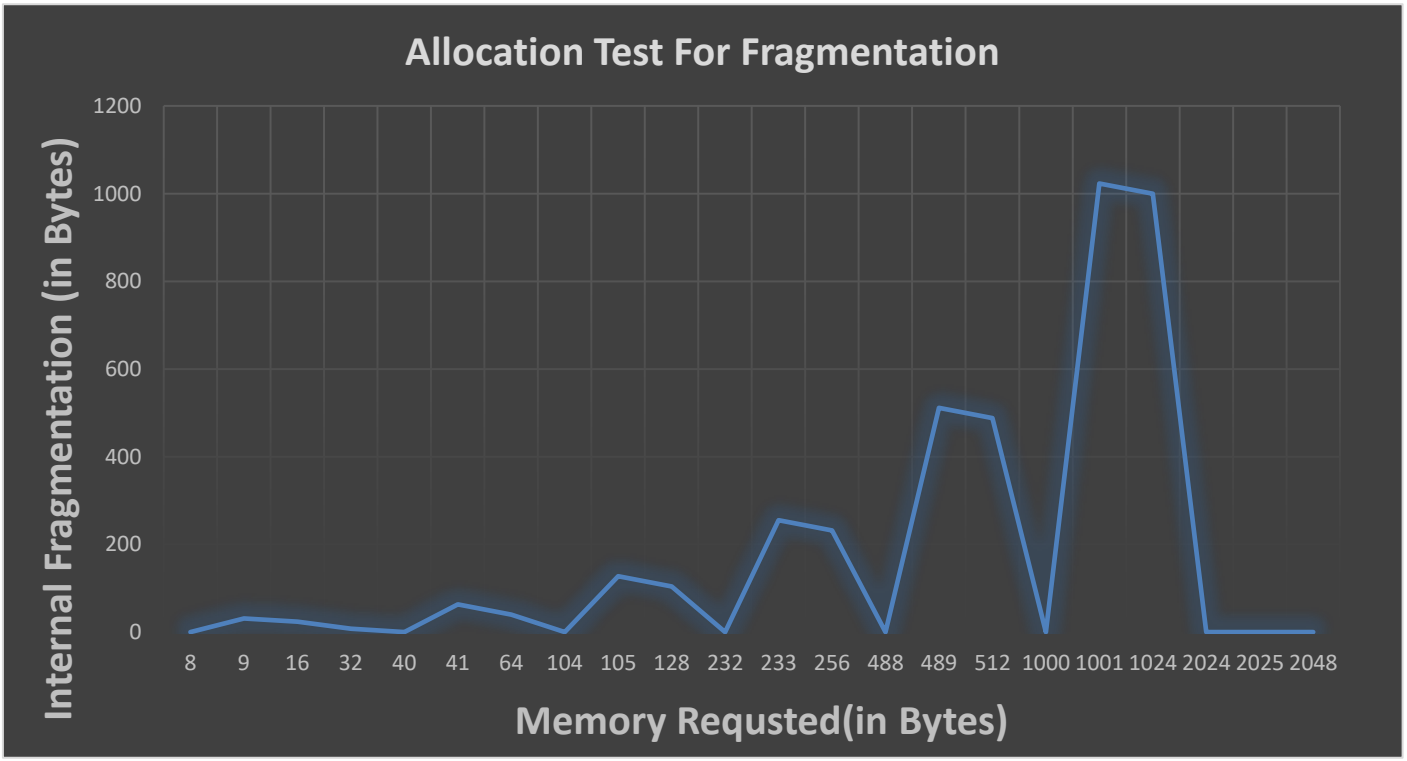
Solution Approach

- Our approach included a list of vector which keeps track of free blocks of different memory sizes up to 2KB.
- To begin with, we created a new block of 2KB using the sbrk() system call and added it to our vector of free blocks.
- We took a buddy Size variable to maintain the maximum size of the memory block present in the free list vector. So, if the user requested the block within the limits of BUDDY SIZE, we used buddy algorithm to provide it with the best fit memory block as per requirement otherwise If user requires memory > BUDDY SIZE, then we used mmap() system call to provide required memory.
- For free, we used the approach of Buddy deallocation i.e. search for buddy of free block, if that also free then we just marge then to form a bigger block.
- If the size to free block is larger than the BUDDY SIZE, we returned it to OS by calling munmap() system call.



Experiment(s) and Results

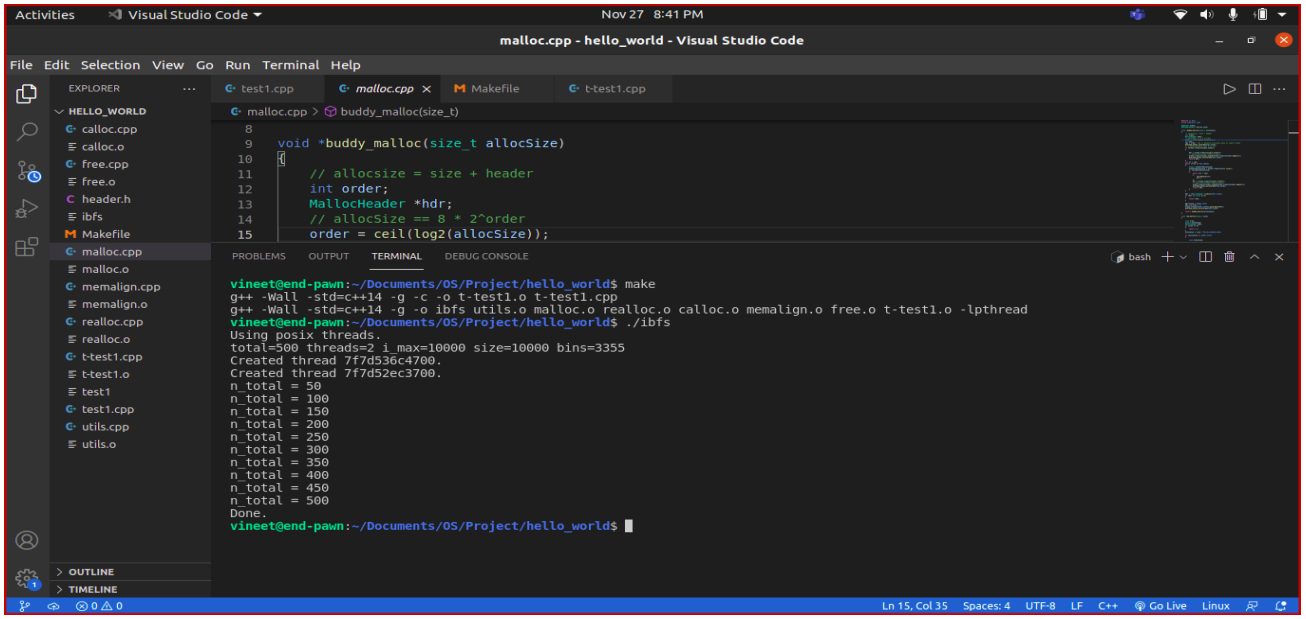
We wanted to measure the internal fragmentation. We initialized a shared memory with the segment size 2KB, on each new request for memory (let say R Bytes) from user we allocated 2^k bytes where $2^{(k-1)} < R <= 2^k$ bytes. We got this plot for internal fragmentation.

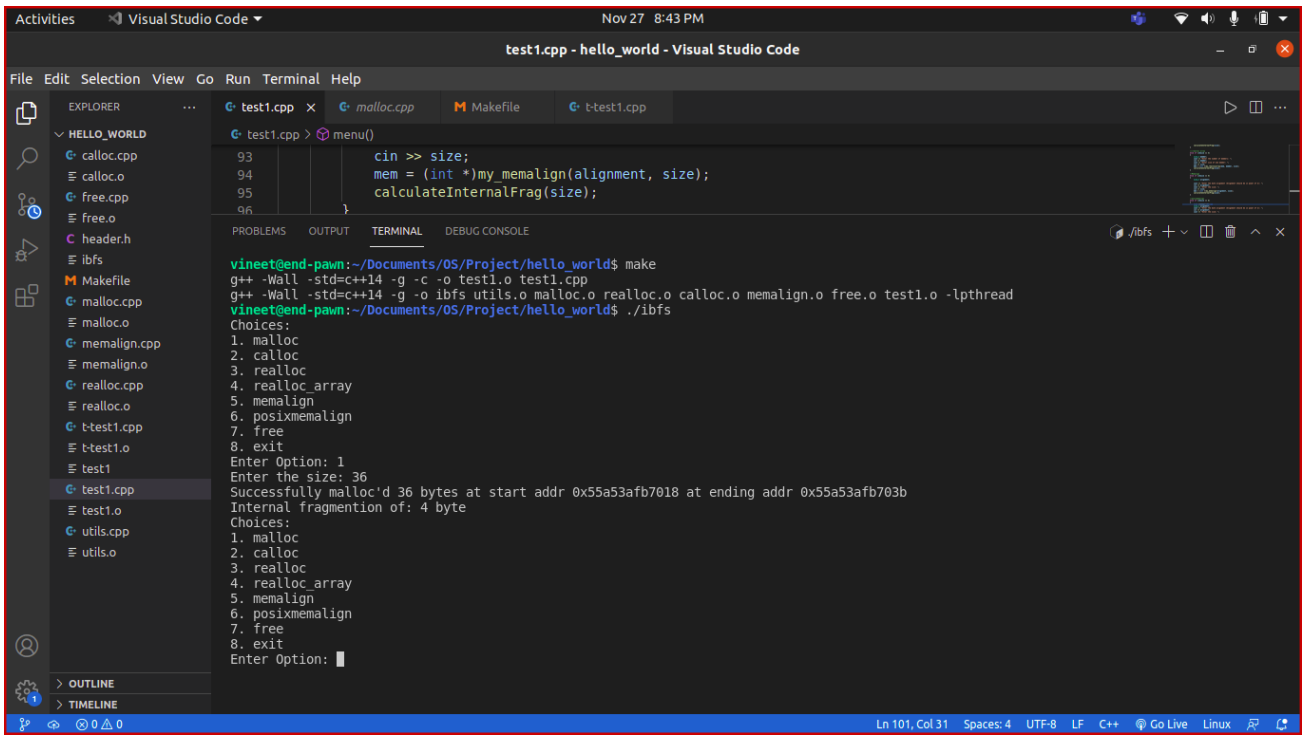


Graph start with 8 bytes since we assume the smallest request to be of 8 bytes. Whenever there is requirement of size is close to 2^k and less than 2^k we get less internal fragmentation.

On observation we got that as we increase Buddy size above 2KB, we are getting more internal fragmentation and normally that many requirements are very less. So, for that reason if user requires memory of size more than 2KB we choose to provide memory as per the user request and not handling that request by Buddy allocation but by system mmap() system call.

Screenshots of Implementation





Problem faced

- Up to how much memory requests should be allocated according to Buddy algorithm so that we can minimize internal fragmentation.
- How to keep track of how much memory allocated to user so that on calling of free() we can marge that block to it's Buddy
- How to make our memory allocation thread safe in case of multiple memory allocation calls at a time by different threads of same program.

Work Distribution

We studied the core concept of buddy allocation and deallocation. Then we distributed our work, learned respective functions, and implemented those functions. All the given functions were somehow interlinked, so it was a cumulative contribution to merge the functions.

malloc: Shravan **calloc:** Vineet **realloc:** Kamal
reallocarray: Kamal & Rahul **memalign:** Rahul & Vineet
posixmemalign: Rahul & Vineet **free:** Shravan & Kamal

Shortcomings

- For memory request of higher size, which is just more than some power of 2 will cause high internal fragmentation.
- Memory cannot be freed if it's Buddy block is not free, there may be blocks which are free and has same size but cannot be merge because they are not Buddy to each other.

Learnings

- In Buddy System, the cost to allocate and free a block of memory is low compared to that of best fit or first fit algorithm
- Search for a block of right size is cheaper than best fit because we need only find the first available block on the block list for blocks of size $2^{k.}$

Conclusion

This assignment helped us to understand handling memory management using the B algorithm. After completing the project, we can see the importance of how various memory management algorithms made tradeoffs in design in our case the way the algorithm handles fragmentation. Additionally, we learned how to allocate our memory allocation thread-safe in case of multiple memory allocation calls at a time by different threads of the same program.

References

[1] Man page
https://man7.org/linux/man-pages/man3/malloc.3.html
https://linux.die.net/man/3/memalign
https://man7.org/linux/man-pages/man3/posix_memalign.3.html
https://man7.org/linux/man-pages/man3/free.3p.html

[2] Wikipedia.
https://en.wikipedia.org/wiki/Buddy_memory_allocation