`

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

CS3.304.M21 Advanced Operating Systems

Project Final Report

Buddy Algorithm

Team Name:-Processors

Team Members:-

1. Aditya Sharma	2021201016
2. Anchal Jakhmola	2021201051
3. Utkarsh Tripathi	2021202007
4. Tej Prakash Mittal	2021202021

Instructor:- Prof Manish Shrivastava
Mentor:- Agrima Singh

BUDDY ALGORITHM

Introduction **:-** The buddy algorithm is a kind of memory allocation technique that divides memory into partitions to try to satisfy a memory request as suitably as possible. This system makes use of splitting memory into halves or merging free contiguous memory to try to give a best fit. It supports limited but efficient splitting and coalescing of memory blocks. So in our Project we will be using this concept to implement basic memory allocation functions as mentioned below:-

malloc():-The malloc() function allocates size bytes and returns a pointer to the allocated memory. The memory is not initialized if size is 0, then malloc() returns either NULL, or a unique pointer. In case of smaller request splitting of memory block is managed in this.

free():-The free() function frees the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc(). Undefined behaviour occurs if free(ptr) has already been called before. No operation is performed if ptr is NULL. Merging of contiguous free block is also managed here.

calloc():-The calloc() function allocates memory for an array of nmemb elements of size bytes each and returns a pointer to the allocated memory. All the value will be set to zero in this.

realloc():- The realloc() function changes the size of the memory block pointed to by ptr to size bytes. If the new size is larger than the old size, then we are simply freeing up the old memory and allocating a new memory of new size.If ptr is NULL,then the call is equivalent to malloc(size), for all values of size and if user asked for smaller size then we are not doing anything.

reallocarray():- The reallocarray() function changes the size of the memory block pointed to by ptr to be large enough for an array of nmemb elements, each of which is size bytes. However, unlike that realloc() call, reallocarray() fails safely in the case where the multiplication would overflow. If such an overflow occurs, reallocarray() returns NULL, sets errno to ENOMEM, and leaves the original block of memory unchanged.

posix_memalign():- This function allocates size bytes and places the address of the allocated memory in *memptr. The address of the allocated memory will be a multiple of alignment, which must be a power of two and a multiple of sizeof(void *). It will return 0 if successfully allocates else error will be return.

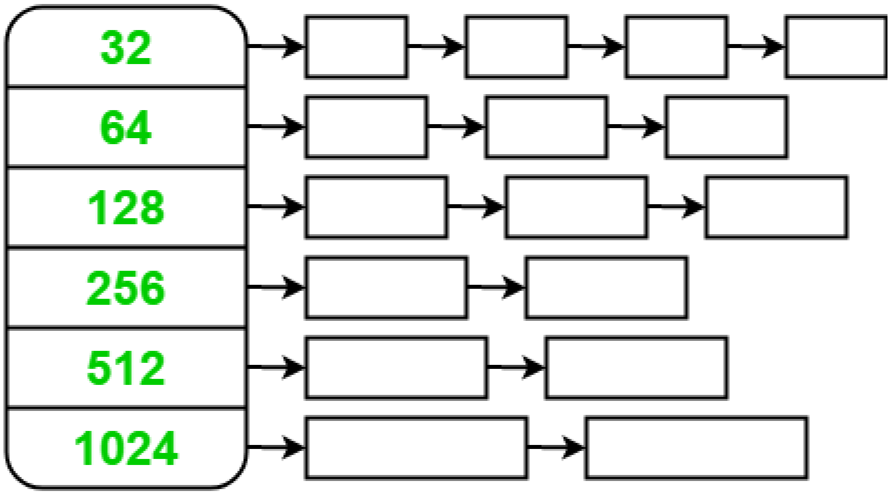
memalign():- The obsolete function memalign() allocates size bytes and returns a pointer to the allocated memory. The memory address will be a multiple of alignment, which must be a power of two.It memory returned will always be divisible by alignment size.

Problem Description:-To write a malloc library using buddy algorithm that provides the above described dynamic memory allocation routines with given constraints:

- If no blocks are available on the free-list, you must ask kernel for more memory (using sbrk() systemcall). The memory request must always be multiple of PAGE_SIZE.
- All the addresses returned by malloc, calloc and realloc should be 8-byte aligned.
- All the implementation should be done by considering of thread safety i.e multiple threads should be able to allocate the memory or free it using proper locks.
- Proper error handling should be done in all the cases.

Solution Approach:- To allocate the memory dynamically our custom malloc function makes the use of sbrk and mmap system calls. Above a certain threshold size the request is satisfied by mmap call bcz of the fragmentation issue due to contiguous heap memory allocation that sbrk does internally.

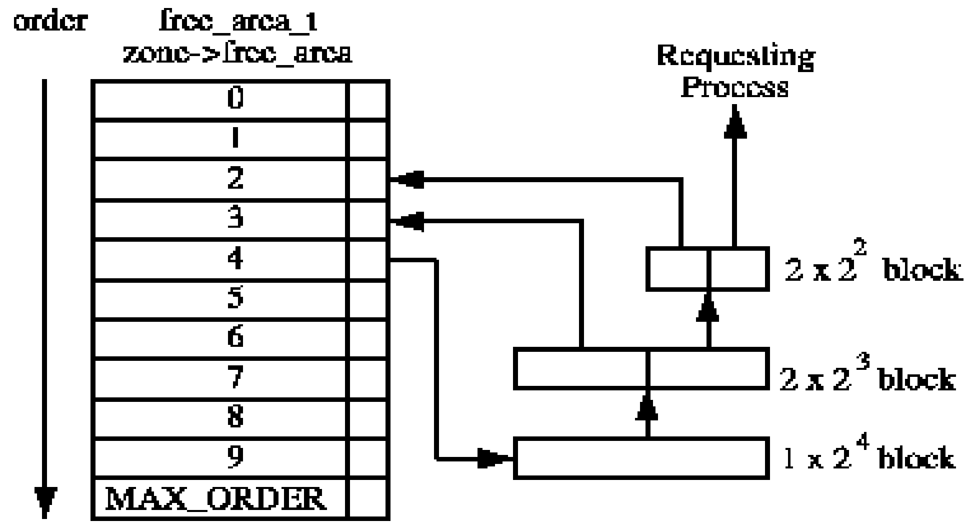
Due to overhead in switching between kernel mode and user mode when user asks memory for the first time a huge chunk of memory is requested (multiple of PAGE_SIZE) using sbrk call. This memory chunk is then processed according to the buddy approach and the user is provided with the requested size. For this purpose we maintain an array of linked list where each index of an array would hold a list of nodes of free memory of the same size (in powers of 2).



A header structure containing the metadata about the node is stored in each node to track the allocation and deallocation in our implementation.

When a user request for memory block we look for the availability of closest free block (in the array of free blocks) of order of 2 which can satisfy this request

If a free block cannot be found of the requested order, a higher order block is split into two buddies. One is allocated and the other is placed on the free list for the lower order. Figure shows where a 2^4 block is split and how the buddies are added to the free lists until a block for the process is available.



When the block is later freed, the buddy will be checked. If both are free, they are merged to form a higher order block and placed on the higher free list where its buddy is checked and so on. If the buddy is not free, the freed block is added to the free list at the current order.

Work Distribution:-

malloc, split function is written by Aditya Sharma
free, merge function is written by Utkarsh Tripathi
calloc, realloc, reallocarray written by Anchal Jakhmola
memalign, posix_memalign written by Tej Prakash

Threads, testing and report were contributed by all the members.

Problems Faced and Shortcomings:-

- Issue of Thread Contention
- The rounding of requests to the next power of two often leaves a lot of unused space in the buffer, resulting in poor memory utilization(Internal Fragmentation).
- The problem becomes worse due to the necessity to store the header in the allocated buffers. For example, a **512-byte** request would consume a **1024-byte** buffer.

- Managing the chunks of memory during malloc and free and appropriately splitting and merging them.
- While merging we have to check that both the memory is contiguous then only we can merge.

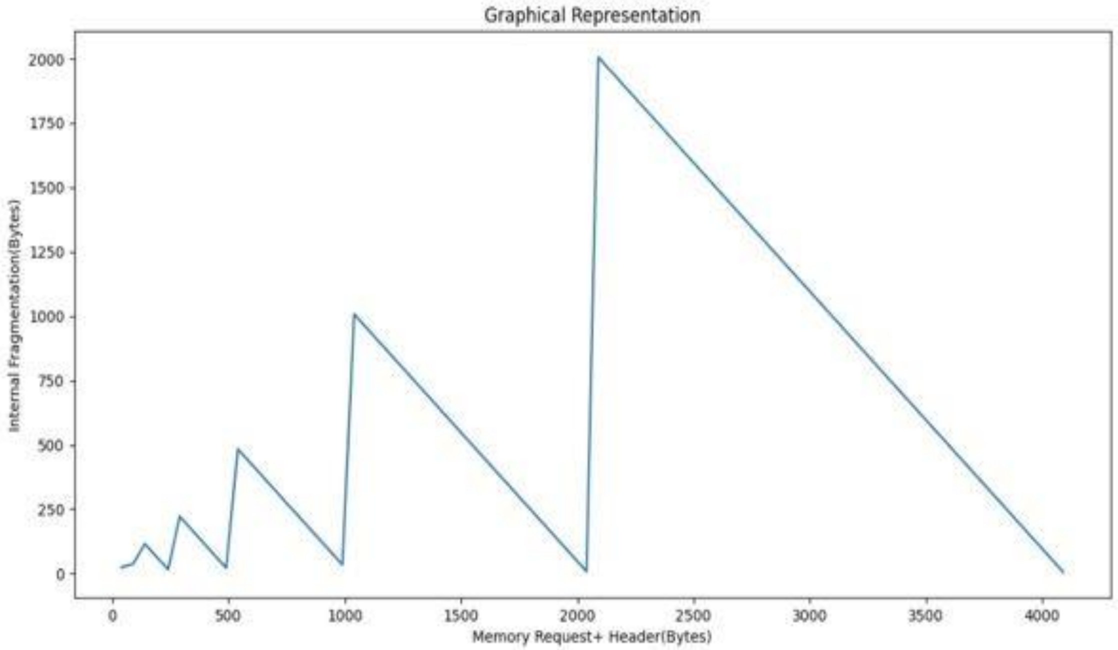
Learnings:-

- Learned about how the dynamic memory allocation works and how the OS gave shorter memory by splitting the bigger blocks to fulfil the requirement.
- How to fulfil the request of a memory block in the nearest power of two using Buddy Algorithm
- Learned about using pthreads for multi programming and using locks to satisfy the condition of mutual exclusion.

Result and conclusion:-

- Memory will be allocated in the next biggest power of two which leads to Internal Fragmentation.
- The returned address from malloc, calloc and realloc must be aligned in 8-byte boundary.
- Memory will be freed and will merged if it’s contiguous memory block is also free.

The below graph represents Internal Fragmentation(represented by Y-axis) in which X-axis represents size + header.


Graphical Representation
Internal Fragmentation(Bytes)
Memory Request+ Header(Bytes)