Submitted by: Shubhi Mittal

Libmalloc.so: Malloc Library Implementation using Buddy algorithm and per thread memory arenas

Design Overview: The malloc library provides the following api's to the programs.

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
struct mallinfo mallinfo();
void malloc_stats();
```

Program structure:
There is a common header file "utils.h" which has declarations for memory allocation data structure and functions that are used across the malloc library. For each api, there is a separate .c file i.e. malloc.c, realloc.c, free.c, mallinfo.c, malloc_stats.c and each file includes utils.h the header file. For convenience, definitions for all the functions declared in the utils.h lies in malloc. c

Algorithm used: Buddy allocation. Please refer to the shared link for detailed specifications of the algorithm here.
https://en.wikipedia.org/wiki/Buddy_memory_allocation

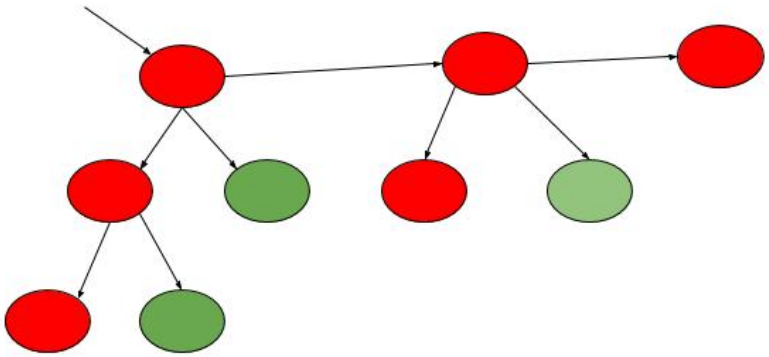
Multithreading support: Each thread allocates the memory from its own arena. There is a thread specific area and the contention happens in the system when memory is asked from the system itself I.e. at the time of sbrk or mmap. At the time of sbrk or mmap, a thread acquires the lock as this is the time when there is a manipulation in the global memory heap which is shared across different threads. One a thread is able to get the memory from the system then the thread releases the lock and works on its own arena. Thus, there is essentially one lock in the library for manipulation of the heap memory of the process.

DataStructures Overview:
1. The buddy allocation is done by representing each memory block as Binary Tree. The per thread arena will consist of many such trees (essentially a linked list of heaps). The thread arena is a pointer to the head of the linked list of memory trees.
2. Each time memory is requested, the linked list is traversed sequentially, each Binary tree is traversed to check if there is a free block for allocation.
3. if there is no free block in any of the trees then the thread asks for more memory from the system.
4. The smallest block is of 1PAGESIZE in memory and it is split further into smaller blocks recursively to allocate the most suitable block to fulfill requested memory.
5. Each memory node in the Binary Tree has a header attached to it.

Memory Header

```
typedef struct MemoryNodeHeader {
        unsigned int blockSize;
        unsigned int dataSize;
        struct MemoryNodeHeader *parent;
        struct MemoryNodeHeader *leftChild;
        struct MemoryNodeHeader *rightChild;
        unsigned int isNotFree;
        // used as padding bytes for data allignment
        unsigned int paddingBytes;
        struct MemoryNodeHeader* nextRoot;
} memoryNodeHeader;
```

A **memory arena** is represented as the following. All red nodes are the ones that are occupied and green ones are the ones that are free/available for allocation



Advantages:
It can be highly efficient for applications where memory is limited as it leads to very less memory wastage.

Limitations:
The Implementation asks for memory from system on demand basis and hence it can become slow when there are lots of threads making large memory requests and also for threads that sequentially make large requests.
The current implementation often goes out of memory when a large number of threads are requesting large memory (of the order of few GBs)

Improvements: There can better design by keeping a super block or allocating a very large arena for a thread the first time it requests memory so that we do not have to ask over and over again from the system. This can lead to better performance as it can save the cost of making a system call.