C Memory Manager

Jasper Mishra, Jack Coughlin
CSE 374 Winter 2013

Introduction

The free-list is a linked-list maintained as a series of "struct BlockHeader" interface where each interface holds

- The size variable that indicates allocable memory for the block
- a struct BlockHeader pointer that joins it to the next block.

Since we use pointer to a struct for deferencing and for call-by-value operations this "struct BlockHeader* " pointer has been typedef(ed) to "Node* " for coding convenience.

Getmem Description and Algorithm

a) A traverser pointer is used to scan th free-list and look for a block of the appropriate size.

b) If a block of the right size is found , a check is performed whether to give away the entire block or bifurcate it to conserve leftover memory for later use.

c) We calculate this through an estimated remaining_space which is = (current->size - request_size - HEADER_SIZE) where HEADER_SIZE is size of the struct Node

   a. If remaining_space is significantly large then the block is bifurcated. This is measured by the macro DIFF_THRESHOLD where if remaining_space >=DIFF_THRESHOLD then block is bifurcated

   b. If the requested size is larger than the size of the current block by only a few bytes then the entire block is given away. This holds TRUE only when remaining_space < DIFF_THRESHOLD

d) This is done by the following algorithm.

```
prev = root;

current = root->next;

while (current!=NULL){
    if(block is large enough){
      if((current->size - request - HEADER_SIZE) >=DIFF_THRESHOLD){ // Step
c) (i)

            // bifurcate the block
            //perform necessary castings, alignments, allocations
        }
    else{   // Step c) (ii) we give away the entire block since the remaining
space < DIFF_THRESHOLD

                //perform necessary castings, alignments, allocations
            prev->next = current->next
      }
   }
}
```

Freemem Algorithm

a) The pointer given as the argument is decremented by the HEADER_SIZE in order to get to the actual start location of this memory block.

b) A check is performed to see if this block is contiguous with any of the block on the free-list. THis is found out either TRUE or FALSE using a boolean on the arithmetic -- (top+ top->size + header->size) == (current) where top is the pointer obtained in (a) and bottom is the current block being scanned on the free-list.

c) If the condition in b) is TRUE then
    a. if top is contiguous with bottom such that were top to be on the free list and bottom follows immediately after, then top->size is incremented appropriately and the pointer to the "previously exisitng" block now points to the start of the top (top comes first in this joint block)

    b. if top is contiguous with the bottom such that were top to be on the free list and it follows immediatly after bottom, then bottom->size is simply incremented and no delinking is performed since bottom is already on the free list.

d) If the condition in a) is FALSE then we just keep moving and attach the block to the end of the free-list

e) Following is the algorithm:

```
Node* actualPtr = originalPtr - HEADER_SIZE;
  while(current!=NULL){
    if (actualPtr is contiguous with current){  //c) i)
      actualPtr->size = actualPtr->size + HEADER_SIZE + current->size
      actualPtr->next = current->next;
      current=actualPtr
    }
    else if(current is contiguous with actualPtr){  //c) ii)
      current->size = current->size + HEADER_SIZE + actualPtr->size;
    }
    current=current->next;
 }

 if(current==NULL){  //Step d)
     current->next=actualPtr;
 }
```

Bench Program Results and Observations

On each execution, the underlying system is called frequently for allocating new chunks of memory depending on the size of the request. From this, the following can be observed:

a) Total acquired memory from the underlying is always increasing.

b) Total amount of free bytes on the free-list varies depending on the series of requests, hence increasing or decreasing.

c) From (ii) Total number of blocks on the free-List also varies on the series of requests, hence inreasing or decreasing.

d) When "pctget" is small, for instane 10 and "pctlarge" is large, for instance 90, the number of free blocks on the free-list is small, in this case under 10, but the size of blocks is significantly large. This is convincing since the number of calls to getmem is a small percentage of the total trials but the 90% of those 10% request get a large sized block.

e) When pctget is large, for instance 50 and "pctlarge" is large, for instance 90, the number of feee blocks on the free-list is significantly large, in this case more than 50, where size of each block is also sufficienlty large. Again, this properly coincides with the parameters for pctget and pctlarge.

f) In all the above trials, CPU time between in the intervals in short and negligible.

Source Code

In order to secure academic integrity, source will be provided at the request and successive permission of both of the authors. Please mail at jbloom1704@gmail.com with your affiliated organization/university and state the purpose of your project/experiment that requires this program.