University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

Lab 4 Report
(Group 68)

Prepared by
Zhidong Zhang
20619543
z498zhan@uwaterloo.ca
2B Computer Engineering
and
Shuyu Lyu
20622527
s88lu@edu.uwaterloo.ca
2B Computer Engineering

23 November 2017

In this report, I am going to illustrate the implementation of the two memory allocation algorithms: best fit and worst fit. Then, I will show the testing scenario and use experimental data to analyze the advantage and disadvantage of the two different allocation algorithms.

Statement of the problem:

Between best fit allocation and worst fit allocation, which allocation algorithm has less external fragmentation?

The data structure and algorithms to implement the allocation algorithms:

- **Data structure**
  We used double linked list to store the information of memory blocks. Each node represents a memory block. The size of each node is 32 bytes.
  Each node is defined using struct with four fields:
  1. an integer called `allocated` to indicate if a node is allocated or not (1 is allocated, 0 is not allocated)
  2. a size_t called `block_size` to store the size of the memory block which is node size(32 bytes) + allocated memory.
  3. a pointer `*prev` points to the previous node
  4. a pointer `*next` points to the next node
- **Algorithms**
  - **Best fit allocation algorithms :**
    1. Increase the input called `size` to a multiple of 4
    2. update `size` with `size` + node size(32 bytes)
    3. Loop through the linked list and look for the **smallest** free memory block whose `block_size` is greater than `size` + node size(32 bytes) or equal to `size`.
    4. If the required memory block is found and its `block_size` is equal to `size`, the found block will be marked as allocated.
    5. If the required memory block is found and its `block_size` is greater than `size` + node size(32 bytes), the found block will be split into two memory blocks: one is allocated, the other one is a new free memory block.
    6. Return the pointer to the allocated memory which is the allocated block pointer + node size(32 bytes)
  - **Worst fit allocation algorithms :**
    1. Increase the input called `size` to a multiple of 4
    2. update `size` with `size` + node size(32 bytes)

3. Loop through the linked list and look for the **largest** free memory block whose `block_size` is greater than `size` + node size(32 bytes) or equal to `size`.
4. If the required memory block is found and its `block_size` is equal to `size`, the found block will be marked as allocated.
5. If the required memory block is found and its `block_size` is greater than `size` + node size(32 bytes), the found block will be split into two memory blocks: one is allocated, the other one is a new free memory block.
6. Return the pointer to the allocated memory which is the allocated block pointer + node size(32 bytes)

- **Pseudocode of both Algorithms** :

Best fit:

void *best_fit_alloc(size_t size):

```
        Increase the size to a multiple of 4
        size ← size + node size(32 bytes)
        current_block ← head of the linked list
        best_block ← NULL
        new_block ← NULL

        //Find the smallest block which is not allocated and the size is greater than
size + node size
        while current_block is not NULL do
                if current_block is not allocated then
                        if current_block->block_size > size + node size(32 bytes) OR
                        current_block->block_size = size then
                                if best_block = NULL then
                                        best_block ← current_block
                                end if
                                if current_block->block_size < best_block->block_size then
                                        best_block ← current_block
                                end if
                        end if
                end if
                current_block ← current_block->next
        end while

        //Cannot find the required block, return NULL
        if best_block = NULL then
                return NULL
        end if

        if best_block->block_size = size then
                change best_block to allocated
        else
                //split best block into best block and new block
                //create new_block
```

```
                new_block ⟵ best_block + size
                set new_block to not allocated
                new_block->block_size ⟵ best_block->block_size - size;

                insert the new_block to the linked list after the best_block
                //update best_block
                set best_block to allocated
                best_block->block_size ⟵ size
        end if

        return best_block + node size(32 bytes)
```

worst fit:

void *worst_fit_alloc(size_t size):

```
        Increase the size to a multiple of 4
        size ⟵ size + node size(32 bytes)
        current_block ⟵ head of the linked list
        worst_block ⟵ NULL
        new_block ⟵ NULL

        //Find the largest block which is not allocated and the size is greater than
size + node size
        while current_block is not NULL do
                if current_block is not allocated then
                        if current_block->block_size > size + node size(32 bytes) OR
                        current_block->block_size = size then
                                if worst_block = NULL then
                                        worst_block ⟵ current_block
                                end if
                                if current_block->block_size > worst_block->block_size
                                then
                                        worst_block ⟵ current_block
                                end if
                        end if
                end if
                current_block ⟵ current_block->next
        end while

        //Cannot find the required block, return NULL
        if worst_block = NULL then
                return NULL
        end if

        if worst_block->block_size = size then
                change worst_block to allocated
        else
                //split worst_block into worst_block and new_block
                //create new_block
                new_block ⟵ worst_block + size
```

```
            set new_block to not allocated
            new_block->block_size ⟵ worst_block->block_size - size;

            insert the new_block to the linked list after the worst_block
            //update worst_block
            set worst_block to allocated
            worst_block->block_size ⟵ size
        end if

        return worst_block + node size(32 bytes)
```

Testing scenario description:

Best fit Tests:

1. best_fit_memory_init failed --- input is smaller than (node size(32 bytes) + 4),
2. best_fit_alloc failed --- Input is too big (greater than memory space)
3. Allocate succeed, deallocate succeed (with correct input)
4. Will allocate in the smallest available block
5. best_fit_dealloc failed --- input is not returned by function best_fit_alloc

,

Worst fit Tests:

1. worst_fit_memory_init failed --- input is smaller than (node size(32 bytes) + 4),
2. worst_fit_alloc failed --- Input is too big (greater than memory space)
3. Allocate succeed, deallocate succeed (with correct input)
4. Will allocate in the largest available block
5. worst_fit_dealloc failed --- input is not returned by function worst_fit_alloc

External fragmentation test for both best fit and worst fit:

1. Initialize 102400 bytes memory space.
2. Using a while loop to allocate and deallocate memory blocks repeatedly. More specifically, in each iteration, we allocate 8 memory blocks with a random size from 1 - 512 bytes and deallocate one every two blocks of memory. The while loop will break when one allocation failed.
3. Check the external fragmentation for memory block size less than 4 bytes, 8 bytes, 16 bytes, 32 bytes, 64 bytes, 128 bytes, 256 bytes, 512 bytes. These will be the experimental data.

Experimental data:

Table 1. External fragmentation table

External fragmentation memory block size (upper limits)	Best fit	Worst fit
4 bytes	0	0
8 bytes	10	0
16 bytes	17	4
32 bytes	28	11
64 bytes	48	21
128 bytes	53	40
256 bytes	55	81
512 bytes	57	170

Time analysis:

Best:
real 0m0.003s
user 0m0.000s
sys 0m0.002s

Worst:
real 0m0.003s
user 0m0.000s
sys 0m0.002s

The comparison conclusion:

According to the Experimental data, the external fragmentation for best fit algorithm is greater for block size less than or equal to 128 bytes, while the external fragmentation for worst fit algorithm is significantly greater for block size greater than 128 bytes. In total, external fragmentation of best fit is 57 and external fragmentation of worst fit is 170. Therefore, best fit algorithm has less external fragmentation for larger memory blocks than worst fit algorithm.

In terms of runtime, best fit and worst fit has the same system time which is 0.002s. And due to the similar implementation, both best fit and worst fit has O(n) time complexity.

To conclude, for small memory block, worst fit algorithm has less external fragmentation. For large memory block, however, best fit algorithm has less external fragmentation.