Drake Tetreault
ID 35571095

CS 145B – Project 2

Final Document

Introduction

The goal of Project 2 is to evaluate the performance of various memory allocation strategies. To do so we must generate a large number of memory reservation requests according to varying Gaussian distribution and record two performance metrics. The first metric is the number of segments of contiguous free memory that each allocation strategy examines before returning an allocation address. Lower average numbers of segments examined are better because an allocation strategy which examines less segments executes quicker than one which examines more segments. The second metric is the memory utilization as a percent of the total memory available. Higher average memory utilization values are better because more of the memory is being put to useful work and less is wasted on external fragmentation.

The two memory allocation strategies I chose to evaluate are the Next Fit and Worst Fit strategies.

The deliverable requirements are:

- A preliminary document.
- This final document.
- A final report which details the raw data produced by my implementation and analyzes the results according to the two metrics.
- The source code and an executable program.

Data Structures

The primary data structure that I implemented for this project will be the linked list of free segments of memory, or the *free list*. The full set of main memory is represented by a simple linear array of integer elements. This array can be found in the MemoryManager class as the `int[] mainMemory` field. As recommended by the textbook, the free list is maintained in the main memory array, with each node's metadata located adjacent to user-accessible memory locations. There are two types of memory segments in main memory, free segments and reserved segments. Only free segments are part of the free list. The compositions of both types of segments are shown below,where red cells are segment metadata and green cells are usable memory.

Free Segment:

Reserved Segment:

Note that reserved segments do not have Prev and Next pointers, because they are not part of a linked list.

Drake Tetreault
ID 35571095

To simplify several sub-algorithms, the free list is implemented as a circular linked list.

System Architecture

My final program divides functionality among the following components:

- The entry point. This component submits a sequence of test cases to the driver and records the results of each test case to a file. This corresponds to the `Main()` function in the Program class.
- The test case driver. This component executes a single test case, which consists of submitting a large number of memory allocation and release requests to the memory manager according to a specified Gaussian distribution of request sizes and a specified memory allocation strategy. This corresponds to the `Driver()` function in the Program class.
- The memory manager. This component maintains both the array representing all available system memory and the free list which tracks unused memory available for reservation. When it receives a reservation request, it queries the allocation strategy for the free segment to use to satisfy the request. This is implemented by the MemoryManager class.
- The allocation strategy. This component is a function which given the current state of the free list and a memory request's size chooses a free segment to satisfy the request. For my implementation I will create two such allocation strategy functions, one for the Next Fit strategy and one for the Worst Fit strategy. In my implementation, allocation strategies are represented by a C# delegate (equivalent to C function pointers) called AllocationStrategy, with the following signature:
  `bool AllocationStrategy(MemoryManager memory,int minSegmentSize, out int segmentAddress, out int segmentsInspected)`.

Pseudocode

The pseudocode for my final implementation is identical to the pseudocode I provided in the preliminary document:

```
main():
        testCases := { sequence of (mean, st.dev., memSize) tuples }
        foreach(case in testCases)
                result := driver(case, NextFit);
                record_results_to_file(result, NextFit);
                result := driver(case, WorstFit);
                record_results_to_file(result, WorstFit);
driver(testCase, strategy):
        memory := new MemoryManager(strategy);
        for 100,000 times
                while(memory.reserve(Gaussian(testCase.Mean, testCase.StDev)))
                        record_allocation();
                memory.release(random_allocation());
        return statistics;
MemoryManager.reserve(int amount):
        int allocAddr = strategy(memoryState, amount);
        split_segment(allocAddr, amount);
        update_free_list();
        return allocAddr;
MemoryManager.release(int allocAddr):
        try_coalesce_left(allocAddr);
        try_coalesce_right(allocAddr);
```

```
update_free_list();
```