DSA Assignment 4

March 24, 2022

Weighted Buddy System for Dynamic Storage Allocation

Mohit Kumar

SR No.- 04-01-03-10-51-21-1-19825

M.Tech in Artificial Intelligence

"Weighted Buddy Method" is an extension of the buddy method for dynamic storage allocation. The weighted buddy method allows block sizes of 2^k and 3.2^k, whereas the original buddy method allowed only block sizes of 2^k. Its performance is compared to the standard library malloc/free functions for dynamic memory allocation on the parameters of execution speed and efficiency of memory space utilization. The running time of the weighted buddy system is determined by the number of blocks which are split and combined. The effectiveness of the algorithm in managing memory is determined by measuring two types of losses in storage utilization: internal memory fragmentation and external memory fragmentation. External fragmentation is the result of breaking down memory into separate blocks which cannot be combined into a desired larger block. Internal fragmentation occurs as a result of allocating only blocks of predetermined sizes, so that a request for memory must be rounded up to the next larger block size. External fragmentation occurs in the buddy system because two empty blocks of size 2^k cannot be used to fill requests for blocks of size 2^{k+1} unless they are buddies. External fragmentation is not significant for the weighted buddy method, however internal fragmentation is a major concern as it results in a loss of one fourth to one third of memory.

The weighted buddy system decreases the amount of internal fragmentation by allowing more block sizes. Allowed block sizes are 1, 2, 3, 4, 6, 8, 12, . . . Block sizes available in the weighted buddy system are nearly twice as compared to the standard buddy system. The reduction in internal fragmentation is achieved at a cost of two extra bits of overhead in each block and a slight increase in running time, relative to the buddy method. The external fragmentation may increase slightly in the weighted buddy method as compared to the buddy method. If the distribution of requested block sizes is skewed towards small block sizes, the weighted buddy method can be a better allocation algorithm.

Analysis of the "Weighted Buddy" implementation

The simulation method used for the testing and analysis of the weighted buddy system is described as follows :

Step 1. Advance TIME by 1.

Step 2. Free all blocks in the system that are scheduled to be released at the current value of TIME.

Step 3. Calculate two quantities: S (a random size), and T (a random "lifetime") based on some probability distributions.

Step 4. Reserve a new block of size S, to be released at TIME+T. Return to Step 1.

Detailed statistics on parameters such as number of allocation requests, deallocation requests, SPheap area splits, buddy recombinations and internal fragmentation (quantified by the sum of

SPheap area allocated for all successful allocation requests minus sum of the request sizes of all successful allocation requests divided by sum of request sizes of all successful allocation requests) were printed whenever TIME was a multiple of 200. Two block size distributions for S were used in the experiments:

S1. An integer chosen uniformly between 100 and 2000.

S2. An integer chosen according to a (truncated) exponential distribution between 100 and 2000 with a mean of 1000.

The time(T) distribution was a random integer chosen uniformly between 1 and 100 for both block size distributions. Since both S1 and S2 limit block sizes requested to between 100 and 2000, the two buddy algorithms were limited to allocating blocks of size between 128 and 2048. The total memory size assumed available was 2^{28}.

The memory wasted for each allocated block is the difference between the size of the allocated block and the size of the request. The sum of this memory waste over all allocated blocks is the internal fragmentation. Memory overflow occurs when a request for memory cannot be satisfied because only smaller memory blocks are available. When overflow occurs, the ratio of the amount of unallocated memory to the total memory size is the external fragmentation. To measure external fragmentation, memory overflow was activated by ceasing to release blocks after a given time (2000 for this simulation).

We say that a steady state is reached, the average number of blocks which must be split into buddies to satisfy a request and the average number of blocks which are recombined with their buddies when a block is released are equal.

Results for generating block size requests of sizes according to the distribution of S1.

a. Statistics when the steady state is reached, i.e no. of area splits is approximately equal to no. of buddy recombinations which happens around time step 2000.

```
Time step 2000
Number of allocation requests : 2000
Number of deallocation requests : 1954
Number of area splits : 1164
Number of buddy recombinations : 1077
Internal fragmentation/total bytes requested : 0.168027
```

b. Statistics when memory overflow occurs

```
Time step 172200
Number of allocation requests : 172200
Number of deallocation requests : 1952
Number of area splits : 261926
Number of buddy recombinations : 1098
Internal fragmentation/total bytes requested : 0.173696
memory cannot be allocated,total available memory is 58753344 bytes
external fragmentation : 0.218873
allocation request was of size 1809 bytes
```

Results for generating block size requests of sizes according to the distribution of S2.

a. Statistics when the steady state is reached, i.e no. of area splits is approximately equal to no. of buddy recombinations which happens around time step 2000.

```
Time step 2000
Number of allocation requests : 2000
Number of deallocation requests : 1955
Number of area splits : 1073
Number of buddy recombinations : 974
Internal fragmentation/total bytes requested : 0.139036
```

b. Statistics when memory overflow occurs

```
Time step 222200
Number of allocation requests : 222200
Number of deallocation requests : 1951
Number of area splits : 294559
Number of buddy recombinations : 939
Internal fragmentation/total bytes requested : 0.136026
memory cannot be allocated,total available memory is 30566400 bytes
external fragmentation : 0.113869
allocation request was of size 1145 bytes
```

Distribution of block size	Uniform (S1)	Truncated Exponential (S2)
Internal fragmentation	17.37%	13.60%
External fragmentation	21.89%	11.39%
Total fragmentation	39.26%	24.99%

Comparison of SPheap performance with the standard library malloc/free functions in terms of memory space efficiency and speed on Assignment 2 (Large Integer Arithmetic)

a. *Simulation Results of SPheap on Assignment 2*

```
(base) ai-admin@ai-admin:~/Downloads/Mohit_Kumar_19825_Assignment_4/Assignment_2_SPheap$ /usr/bin/time -v ./a.out
For examples of input format see the README file
Enter = to terminate the program
Enter infix string for evaluation
111,041,548,411,111,011$ * 222,222$ * 222,222$ * 548,987,852,001,000$ =
3,10,387,122,183,681,911,615,690,972,861,472,962,124,0$

Number of allocation requests : 57
Number of deallocation requests : 9
Number of area splits : 117
Number of buddy recombinations : 11
=
        Command being timed: "./a.out"
        User time (seconds): 0.00
        System time (seconds): 0.00
        Percent of CPU this job got: 0%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:14.37
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 2136
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 100
        Voluntary context switches: 3
        Involuntary context switches: 0
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

b. *Simulation results for standard library functions malloc/free*

```
(base) ai-admin@ai-admin:~/Downloads/Mohit_Kumar_19825_Assignment_4/Assignment_2_malloc$ /usr/bin/time -v ./a.out
For examples of input format see the README file
Enter = to terminate the program
Enter infix string for evaluation
111,041,548,411,111,011$ * 222,222$ * 222,222$ * 548,987,852,001,000$ =
3,10,387,122,183,681,911,615,690,972,861,472,962,124,0$
=
        Command being timed: "./a.out"
        User time (seconds): 0.00
        System time (seconds): 0.00
        Percent of CPU this job got: 0%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:43.72
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 1408
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 61
        Voluntary context switches: 3
        Involuntary context switches: 0
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

We see that Assignment 2(Large Integer Arithmetic) runs faster with SPheap("Weighted Buddy" system for Dynamic Memory Allocation).