

Cmpsc 473 Programming Assignment 3- Report

Group members:

Vibhu Patel

Jon Dahl

Sanyukta Baluni

4.1 Workloads- refer to the .txt files on github

4.2 Measurements

Measure Malloc

For all the graphs: (x - axis represents processes and y-axis represents the time)

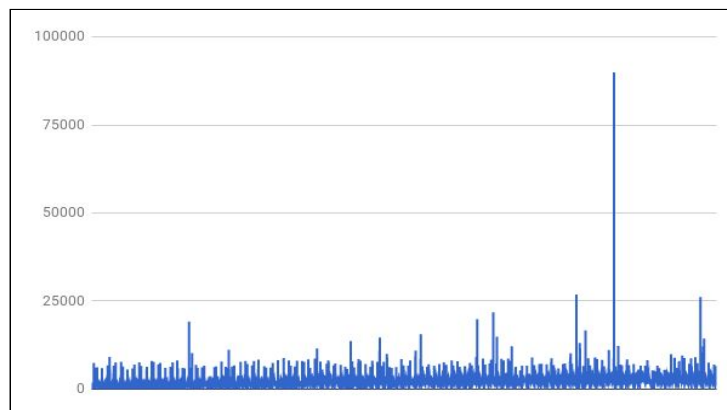
Small - Best

Average = 1991.7526 nanoseconds

Median = 1746 nanoseconds

25th percentile = 1257 nanoseconds

75th percentile = 2444 nanoseconds



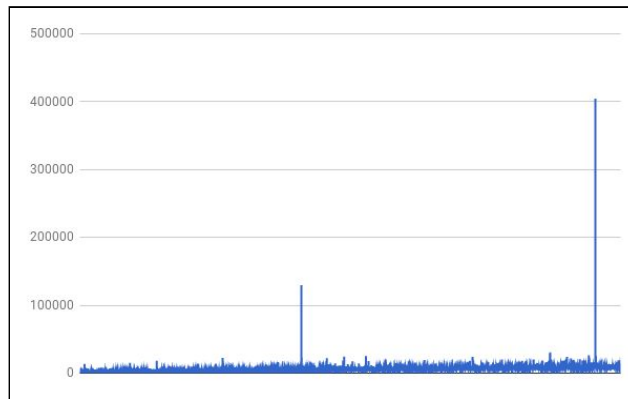
Small - Worst

Average = 5108.3269 nanoseconds

Median = 4819 nanoseconds

25th percentile = 2864 nanoseconds

75th percentile = 6845 nanoseconds



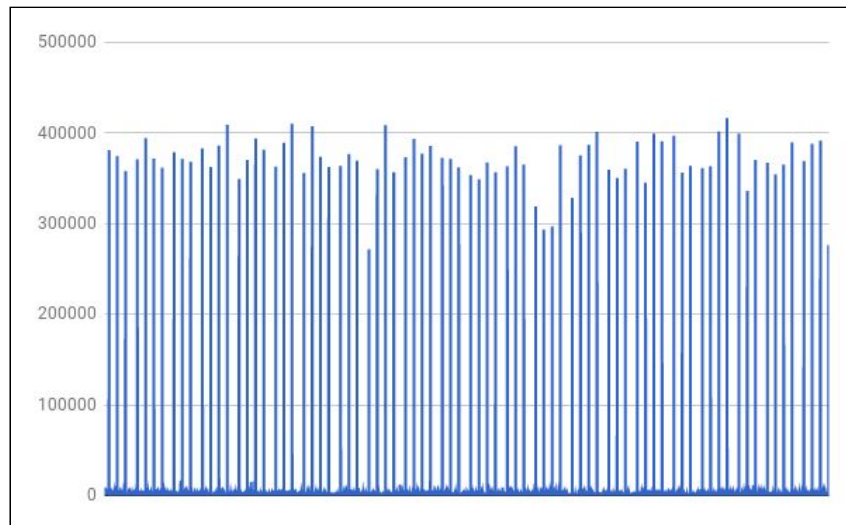
Mixed - Best

Average = 3838.8869 nanoseconds

Median = 908 nanoseconds

25th percentile = 768 nanoseconds

75th percentile = 978 nanoseconds



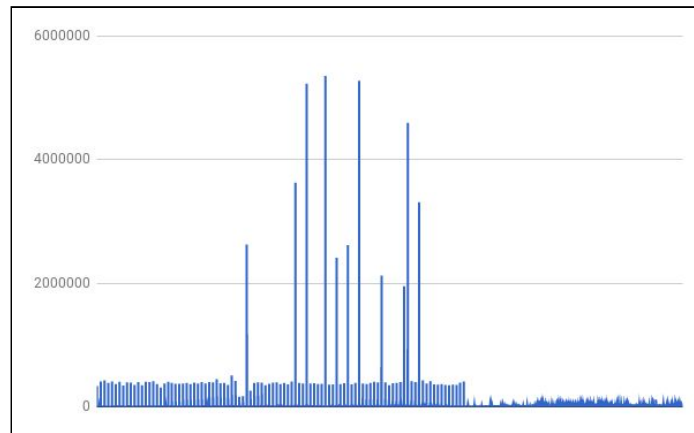
Mixed - Worst

Average = 13778.9167 nanoseconds

Median = 4400 nanoseconds

25th percentile = 2444 nanoseconds

75th percentile = 8939 nanoseconds



Measure Free

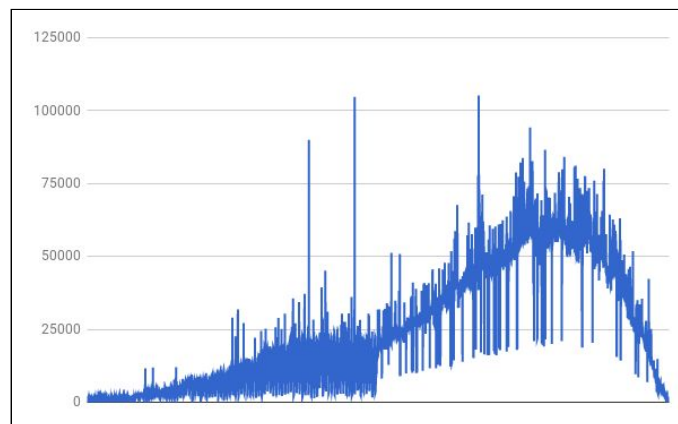
Small - Best

Average = 24153.8451 nanoseconds

Median = 18997 nanoseconds

25th percentile = 4959 nanoseconds

75th percentile = 42463 nanoseconds



Small - Worst

Average = -41293.5702 nanoseconds

Median = 39111 nanoseconds

25th percentile = 10965 nanoseconds

75th percentile = 99942.5 nanoseconds



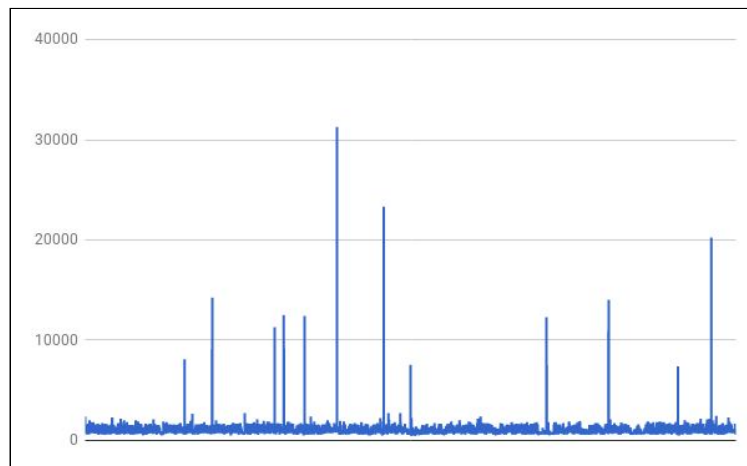
Mixed - Best

Average = 1044.3812 nanoseconds

Median = 1047 nanoseconds

25th percentile = 768 nanoseconds

75th percentile = 1188 nanoseconds



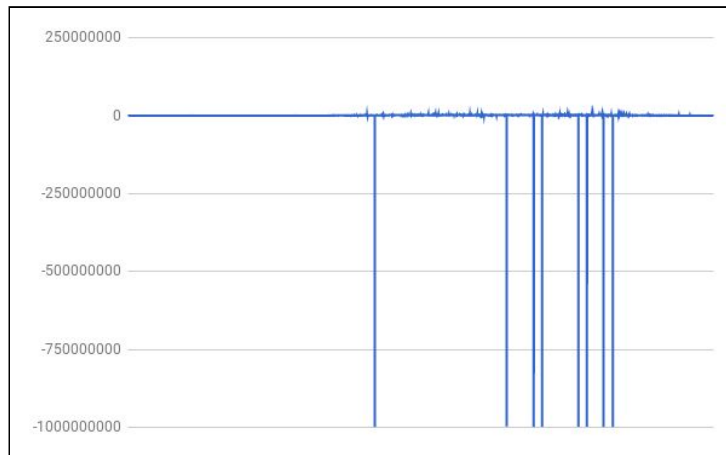
Mixed - Worst

Average = -144780.016 nanoseconds

Median = 342325.5 nanoseconds

25th percentile = 39442.5 nanoseconds

75th percentile = 1132053.5 nanoseconds



Table

(Times in nanoseconds)	Average	Median	25th percentile	75th percentile
Malloc(small-best)	1991.7526	1746	1257	2444
Malloc(small-worst)	5108.3269	4819	2864	6845
Malloc(mixed-best)	3838.8869	908	768	978
Malloc (mixed-worst)	13778.9167	4400	2444	8939
Free(small-best)	24153.8451	18997	4959	42463
Free(small-worst)	-41293.5702	39111	10965	99942.5
Free(mixed-best)	1044.3812	1047	768	1188
Free(mixed-worst)	-144780.016	342325.5	39442.5	1132053.5

The malloc_fail.txt is currently an empty file (not shown on Github currently as it is an empty file)
But if reset.sh is run, there will be a new malloc_fail.txt file will be generated (as referenced in the README)

-> Number of occasions when a psumalloc() could not be satisfied due to lack of memory- there will be a malloc_fail.txt and it will display "lack of memory has occurred".

-> Number of occasions when a psumalloc() could not be satisfied due to internal fragmentation - there will be a malloc_fail.txt and it will display " internal fragmentation has occurred" and it occurs inside the allocated memory for the process.

5.1

In this project we use a linked list to store data. A linked list is like an array that's bound together by memory addresses. With link lists we can allocate and deallocate memory when the program is running.

The possible data structures one could use are:

Array (Did not use)

Tree

Linked List

Our preference was using linked list was because you don't know how many items will be in the list and with an array you would have to redeclare and copy every time which we thought would be inconvenient and here we would like to insert items middle of the list.

The project contains a psumemory.c and a psumemory.h file and the psumemory.c has the functions:

int psumeminit(int algo, int sizeOfRegion) it is called one time by the process to use the memory allocator. And algo represents which memory allocation policy should be used -0 for best fit and 1-worst fit. sizeOfRegion is the bytes of memory you request from the OS using mmap

void* psumalloc(int size) it takes as input the size in bytes of the object to be allocated and returns a pointer to the start of that object.

void* splitting(node_t* p, node_t* op, int size) it splits larger memory into smaller ones and the requested space is returned while the leftover space remains on the free list

void coalesce() it looks for memory chunks that can be combined

int psufree(void* ptr) this function frees the memory that the pointer points to

void psumemdump() this function is simply for debugging purposes

We are using a function **clock_gettime(CLOCK_MONOTONIC, &s_begin)** to measure performance of malloc and free.

Ambiguities in the specification:

Refer to the README.md

There is no way we can measure internal fragmentation. We included a flag for external but not internal.

Internal fragmentation is the wasted space between each allocated block whereas external fragmentation is the various free spaced holes in either your disk space or memory

All the features in the project work correctly and there are no bugs

All the graphs and tables are included in 4.2

Free small-worst and free mixed-worst have a very high average time and it is giving a negative value due to an overflow