Landon Moon
ID: 1001906270

Mavalloc Documentation and Results

Executive Summary

Mavalloc processes

Mavalloc is composed of two blocks of memory, the Node array and the useable memory. The Node array is split into to linked lists, used and not used. Nodes are ‘thrown’ and ‘caught’ between the two. Both linked lists and the memory block is initiated with mavalloc_init. Mavalloc_init also saves the algorithm type for use in mavalloc_alloc. Mavalloc_alloc is a switch statement to separate functions that implement each as expected. mavalloc_free searches the used linked list until it finds a matching pointer.

Benchmark processes and results

Each benchmark runs through each algorithm type along with malloc as a baseline. The benchmarks are done by first calling alloc, Then with a 50% chance a free call is made to a previous pointer. If a null pointer is returned, then it is logged, and the total is output. The testing range is withing 50 to 20,000. A ‘NUM’ variable is made that is generally the average of rand()%40 but because we randomly free memory it will actually be reduced to stress test the system. NUM is used to generate the amount of memory mavalloc gets. Testing the speed sets NUM as constant while testing memory sets LOOPS as constant. This kind of benchmark is able to simulate allocating different sizes of memory along with randomly freeing previous memory. This allows the benchmark to test allocating from the large block of memory or from a previously freed block.

The results are that malloc has a significantly better time complexity than mavalloc, but at sub-100 calls mavalloc beats malloc. Overall next fit is the fastest when compared to the other mavalloc algorithms. The time complexity for all of mavalloc is abysmal. This specific benchmark is a bad showcase of worst case which is explained later. First and Best fit perform the best in terms of memory in this benchmark while best-fit would outperform first fit in more real-life cases.

Benchmarks

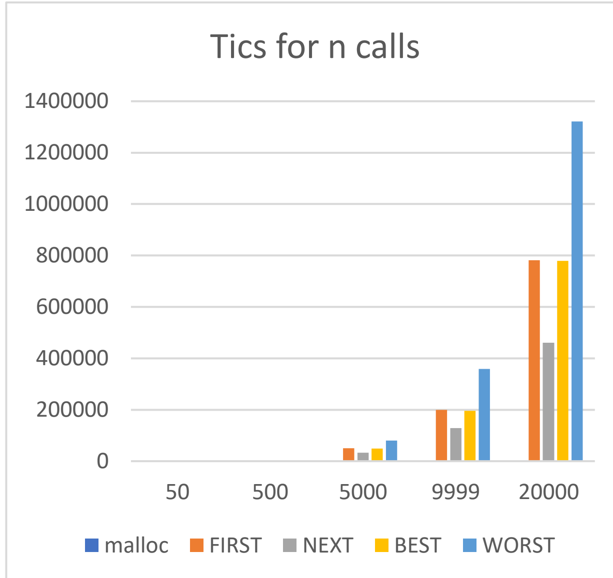
Speed

At a low number of calls, mavalloc is faster by about 50%-100% depending on the type of algorithm used. This means that the access time to the mavalloc functions are smaller than malloc with less overhead. This is probably due to malloc needing to be executed in kernel space, adding time to its execution. Mavalloc doesn’t need this, so its low-level execution time is faster. Much of this implementation of mavalloc has multiple inefficient sections such as searching the entire block of memory for a pointer or for the smallest or largest block available. This is why the time complexity for mavalloc is abysmal.

Next fit is the fastest for this benchmark because for most of the execution time the next block of memory is already free so it can exit almost immediately. Next fit is a simple algorithm, so it is expected to perform well for simple use cases but if memory wasn’t as available then the execution time is expected to decline until it has to search the entire Node linked list like Best and Worst Fit.

Landon Moon
ID: 1001906270

First and Best fit are expected to execute very similar due to the setup of the benchmark. Both need to search through the holes generated, if first fit finds it, it exit immediately, while best fit continues to find a better one. In both cases the holes will be filled as they come in so they both search the entire linked list.



Worst fit and best fit are very similar except they look for a larger or smaller block respectively. The difference in execution time is due to worst fit always choosing the largest memory block leaving the holes to be checked again. Because worst fit has to check all the holes that were generated and left, it has more nodes to go through. Best fit on the other hand fills in these holes as they come in because they are smaller than the main memory block. This way all best fit has to do is see if its open and carry on, and because the holes are being filled, far less are open.

NUM	20		NUM	20		NUM	20		NUM	20		NUM	20	
LOOPS	50		LOOPS	500		LOOPS	5000		LOOPS	9999		LOOPS	20000	
	time	nulls		time	nulls		time	nulls		time	nulls		time	nulls
malloc	33	0	malloc	60	0	malloc	256	0	malloc	509	0	malloc	1045	0
FIRST	26	0	FIRST	649	0	FIRST	49994	0	FIRST	199736	0	FIRST	782210	0
NEXT	11	0	NEXT	369	0	NEXT	32353	0	NEXT	128443	0	NEXT	461275	0
BEST	16	0	BEST	327	0	BEST	48836	0	BEST	196405	0	BEST	778580	0
WORST	19	0	WORST	813	0	WORST	80798	0	WORST	358367	0	WORST	1321503	0

Memory

The higher the value of NUM means that there is more space so less null returns are expected. In the benchmark there is one main block of memory will several small holes made along the way. The results show that in this benchmark worst fit is well… the worst. This is due to the fact that there is only one large block of memory and multiple smaller blocks. Its exactly what worst fit is not meant for. Worst fit fills the large block of memory and as holes are made, a lot of external fragmentation is made. when the large block of memory is gone all its left with is several blocks that are large but not large enough half the time. If the size of requested blocks was increased and the number of loops decreased then Worst fit would perform better because then those large pointers would be able to fit into the starting large memory block.

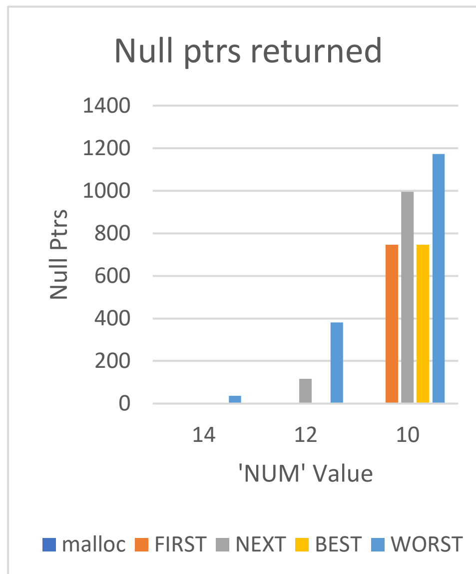
First and Best fit are expected to be almost exactly the same because when a hole is made, it will almost immediately be filled in both cases. Best fit would do better in more general use cases because it looks at the entire memory block instead of starting from the beginning. A better benchmark to split these results would be to increase the possible number of free calls per loop or having more large memory blocks to start. This would not be easy to implement because keeping track of all the pointers would be cumbersome.

Next fit is expected to be more similar to worst fit but not as bad. This is because during the first part of the execution when the main block of memory is being filled both execute exectly the same. When the holes are starting to be filled, worst fit leaves more holes to fill later while next fit simply choses the first open spot it finds. This way next fit is less likely to leave larger holes so it has more memory in the long run.

NUM	14	
LOOPS	9999	
	time	nulls
malloc	501	0
FIRST	198071	0
NEXT	115607	0
BEST	199296	0
WORST	324152	36

NUM	12	
LOOPS	9999	
	time	nulls
malloc	513	0
FIRST	195555	0
NEXT	110184	116
BEST	195897	0
WORST	314496	382

NUM	10	
LOOPS	9999	
	time	nulls
malloc	518	0
FIRST	196365	747
NEXT	124188	995
BEST	192685	747
WORST	280765	1173



Conclusion

Overall mavalloc is not optimal when compared to malloc. Malloc both outperforms mavalloc in terms of execution time and available memory when stress tested. Mavalloc is better when there is a small number of pointers (<100) but otherwise should not be used. Overall next fit or best fit would be good chosen algorithms because they are a balance between efficient memory utilization and execution time. Due to the relatively small, requested memory size and large number of requested blocks, worst fit performs worse than expected in the memory utilization section.

The benchmark could definitely be improved at the cost of extra complexity. Varying the requested pointer size more and switching between allocating multiple times and freeing multiple times would improve the accuracy of the tests.

I am currently making an optimized version of a static allocator with the lessons I've learned from doing this assignment, but it doesn't hold to all the requirements of mavalloc so it might not pass the unit tests.