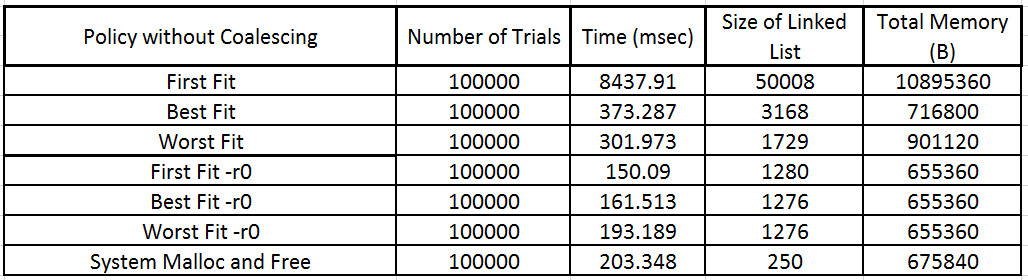
Austin Johnson

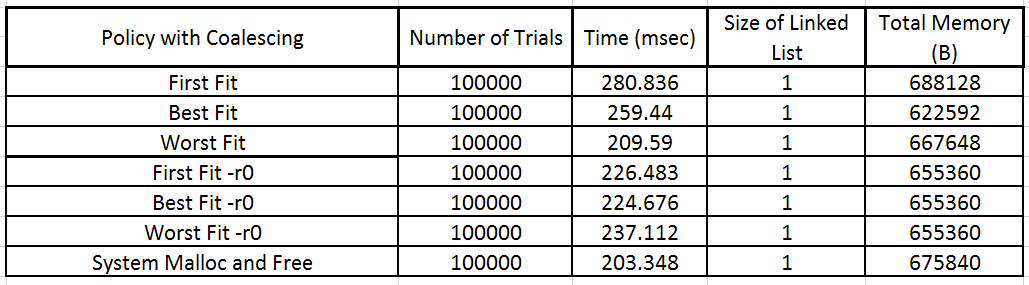
12 October 2016

C03603497

MP4

Performance Evaluation





I chose to implement a chunk\_t structure that contained a struct chunk\_tag next pointer and an integer that contained the size of the memory block. The size of this structure was 16 bytes on a 64-bit machine. I chose to not implement a prev pointer in my structure for a couple of reasons. First, it changed the size of my structure to 24 bytes which does not evenly divide into the PAGESIZE of 4096 bytes. Therefore, I would have to implement a dummy member in order to bring the size up to 32 bytes. Second, implementing a prev pointer would have made me have to set twice as many pointers in places I wouldn’t even need a prev pointer. I think that the use of a one-way linked list efficiently searches for memory without the need of implementing a previous pointer. It was not difficult implementing a previous pointer whenever I needed one and therefore think a one-way linked list works better than a two-way in this situation.

The use of a global roving pointer can easily fragment memory at specific locations. This is especially true with best fit without coalescing. This is due to the fact that best fit leaves behind a tiny segment of memory that likely will not accommodate any new processes. The worst fit search policy appears to be the fastest on average out of the search policies I implemented. I would assume this is due to the fact that it leaves behind bigger segments of memory whenever allocating and allows for freeing bigger blocks of memory versus the other search policies. First fit is the clear loser out of the three implementations. Although it allocates memory much faster than the other two, it frees memory at a much slower rate due to how many small blocks it leaves in the list that need to be freed. However, when using the –r0 flag first fit was the front runner in allocating and deallocating memory. Coalescing the memory reduces the fragmentation that the search policies produce. First fit’s disadvantage is that it can leave small unused memory that cannot accommodate larger memory requests. If you combine first fit with coalescing, it reduces these useless areas in memory drastically but is still the slowest algorithm out of the three implemented. The same can be said for best fit which gets much faster when used with coalescing versus without it. The standard library’s malloc and free functions appear to be the most efficient due to the more sophisticated data structures. However, the three search policies have clear advantages and disadvantages depending on what flag is used. This can be an advantage or disadvantage depending on what one is looking for. If one expects best case / average scenarios then the use of another algorithm over the standard malloc and free might be more efficient (such as the case when the –r0 flag is used). If unpredictable scenarios are the case then the use of the standard malloc and free will probably be the most efficient.