Operating Systems Principles Assignment 3: Memory Management

This experiment was to test the effects of different memory allocation methods on the runtime of the program as well as its memory efficiency. The allocation methods used were: First-Fit and Best-Fit, with these being tested with data files generated using the provided generator, sizes of 100, 250, 500, 750 and 1000.

For the First-Fit allocation method, when an allocation was called, it would first check the list of free chunks to see if there is any available. If there are free chunks, is will iterate through the list in order, and return the first chunk found that is large enough to store the required amount, if there there are no valid free chunks found, or the free chunks list is empty, the program will grow the heap using sbrk(). Similarly, the Best-Fit method also first checks whether there are free chunks available, however when searching for a suitable free chunk, it searches ALL free chunks, and returns the chunk (if there is a valid one) that fits the required job size the best.

Presented below is the raw data obtained from conducting the tests on the different allocation methods:

First Fit Data (File: 100)	1	2	3	4	5	Averages
Total Allocation	8928	7200	8352	14112	15328	10784
Total Fill	6620	5445	5795	9697	11356	7782.6
Percentage Fill	74.14874552	75.625	69.38457854	68.71456916	74.08663883	72.39190641
Total Free	2784	4544	2304	1792	1664	2617.6
Total Time Taken (µs)	68	71	66	65	64	66.8

Best Fit Data (File: 100)	1	2	3	4	5	Averages
Total Allocation	8928	6560	7904	14112	15072	10515.2
Total Fill	6620	5445	5795	9697	11356	7782.6
Percentage Fill	74.14874552	83.00304878	73.31730769	68.71456916	75.34501062	74.90573635
Total Free	2784	5184	2752	1792	1664	2835.2
Total Time Taken (µs)	67	80	64	74	62	69.4

First Fit Data (File: 250)	1	2	3	4	5	Averages
Total Allocation	25792	21952	31168	32128	23552	26918.4
Total Fill	20217	16554	22983	24712	17682	20429.6
Percentage Fill	78.38477047	75.40998542	73.73909138	76.91733068	75.07642663	75.90552092
Total Free	4768	6592	832	1536	8480	4441.6
Total Time Taken (µs)	203	207	203	199	209	204.2

Best Fit Datal (File: 250)	1	2	3	4	5	Averages
Total Allocation	25664	21952	30912	31616	23552	26739.2
Total Fill	20217	16554	22983	24712	17682	20429.6
Percentage Fill	78.77571696	75.40998542	74.34976708	78.16295547	75.07642663	76.35497031
Total Free	4896	6592	832	1536	8480	4467.2
Total Time Taken (µs)	223	243	214	216	229	225

First Fit Data (File: 500)	1	2	3	4	5	Averages
Total Allocation	63200	47712	55008	44736	54720	53075.2
Total Fill	47561	35743	39913	33123	40357	39339.4
Percentage Fill	75.25474684	74.91406774	72.55853694	74.04104077	73.75182749	74.10404395
Total Free	1088	10816	3840	11520	9568	7366.4
Total Time Taken (µs)	596	607	596	602	607	601.6

Best Fit Data(File: 500)	1	2	3	4	5	Averages
Total Allocation	62496	47712	55008	44480	54208	52780.8
Total Fill	47561	35743	39913	33123	40357	39339.4
Percentage Fill	76.10247056	74.91406774	72.55853694	74.46717626	74.4484209	74.49813448
Total Free	1792	10560	3840	11296	10080	7513.6
Total Time Taken (µs)	607	615	639	625	687	634.6

First Fit Data (File: 750)	1	2	3	4	5	Averages
Total Allocation	85696	73120	99968	97472	98016	90854.4
Total Fill	63942	53366	73716	69355	71758	66427.4
Percentage Fill	74.61491785	72.98413567	73.73959667	71.15376724	73.21049625	73.14058273
Total Free	2624	7680	1024	320	8160	3961.6
Total Time Taken (µs)	1285	1258	1274	1268	1287	1274.4

Best Fit Data (File: 750)	1	2	3	4	5	Averages
Total Allocation	84992	72480	99456	96960	97568	90291.2
Total Fill	63942	53366	73716	69355	71758	66427.4
Percentage Fill	75.2329631	73.6285872	74.11920849	71.5294967	73.54665464	73.61138203
Total Free	3136	8320	1024	320	8608	4281.6
Total Time Taken (µs)	1189	1190	1131	1127	1167	1160.8

First Fit Data (File: 1000)	1	2	3	4	5	Averages
Total Allocation	121824	125152	112096	111808	121376	118451.2
Total Fill	89138	92378	83833	83712	89356	87683.4
Percentage Fill	73.16949041	73.81264383	74.78678989	74.87120778	73.61916689	74.05185976
Total Free	6656	2400	6688	8512	1088	5068.8
Total Time Taken (µs)	2078	2065	2422	2155	2357	2215.4

Best Fit Data (File: 1000)	1	2	3	4	5	Averages
Total Allocation	121056	124832	111328	110528	121120	117772.8
Total Fill	89138	92378	83833	83712	89356	87683.4
Percentage Fill	73.63369019	74.0018585	75.30270911	75.73827446	73.77476882	74.49026022
Total Free	6944	1792	7200	9152	576	5132.8
Total Time Taken (µs)	2015	1885	1966	1954	1922	1948.4

Average Fill of Allocation (%)	100	250	500	750	1000
First Fit	72.39190641	75.90552092	74.10404395	73.14058273	74.05185976
Best Fit	74.90573635	76.35497031	74.49813448	73.61138203	74.49026022

Average Runtime (µs)	100	250	500	750	1000
First Fit	66.8	204.2	601.6	1274.4	2215.4

Best Fit	69.4	225	634.6	1160.8	1948.4

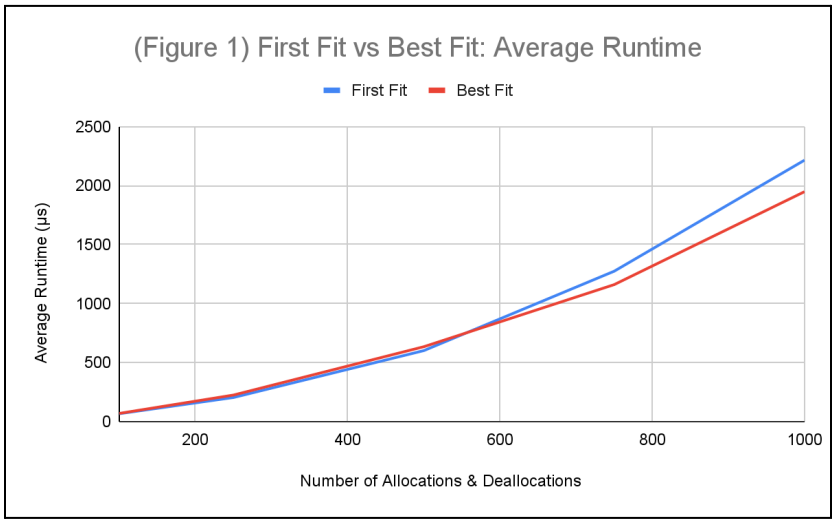

Figure 1: *Average Runtimes of First-Fit and Best-Fit*

Overall, the best scheduling method (based on the results obtained from the experiments) is the Best Fit allocation method. At the lower allocation and deallocation numbers, it runs faster than the First Fit method, however gets slower as the number of allocations / deallocations increases. However, overall the method retains a higher percentage of used allocated space, thus has higher memory efficiency.

Although the First-Fit method has a shorter run time for the 100, 250 and 500 datafiles, the best fit allocation method has a faster runtime for the 750 and 1000 datafiles (refer to figure 1). The first-fit method being faster results from the method not needing to search the entire free chunk list, with it allocating the first valid chunk, whereas best-fit must search the entire free chunk list for the most suitable chunk to allocate to. The Best-fit method being faster for the higher number datafiles, is likely due to an inefficiency in the implementation of the First fit method. Additionally, it was found that runtimes were highly inconsistent between runs, likely resulting from background operations in windows affecting the performance of the program.

Lastly, throughout the experiment, the Best Fit method maintained the higher allocated usage with the method staying roughly 1% better usage than the first-fit method. This results from the method are due to the method allocating the smallest valid and free memory chunk to the requested amount. Furthermore, this method also results in less wasted memory, and fragmentation.

Thus, for a more memory efficient but slower method, the Best Fit allocation method is recommended, as it minimizes wasted memory. However, if time efficiency is what is most needed, the First-Fit method is the recommended allocation method as it has a faster time efficiency, but sacrifices memory efficiency.