CSCI 3232

Final Project: Memory Management Algorithms

Group 5: Joshua Joseph, Scott Maner, and Todd Twiggs
12-3-2015

OVERVIEW

Our team's task was to successfully simulate memory allocation algorithms. For our project we used the first fit, best fit, worst fit and next fit memory allocation algorithms. While running these algorithms we compared the performance of each algorithm and analyzed their differences.

HYPOTHESIS

If memory request allocation speed is the primary concern, then the Next Fit memory management algorithm will have the fastest execution time since it will not be required to reset to the beginning of the memory slot array for each process. The First Fit algorithm will be the 2nd fastest, as it is slowed by its requirement of starting at the beginning during each iteration.

If the most efficient use of space is the primary concern, the Best Fit algorithm will make the best use of the free memory slots since it locates the slot closest to the size of the request. The Worst Fit algorithm will be the 2nd most efficient since it leaves as much space as possible in each slot for future process requests.

EXPERIMENTAL SETUP

To test our hypothesis, we will use a Linux (Ubuntu) virtual machine within VMware Player on Windows 7 (in the classroom) to run each algorithm in the C programming language. We will use 3 arrays containing the same random data – the first array will contain completely random requests, the second will contain the same data sorted from lowest to highest, and the third array will contain the data sorted from highest to lowest. We will use 3 separate array sizes: 10, 100, and 1000. In other words, each algorithm will be executed 5 times for each of the combinations of array size and order, for a total of 45 executions of each algorithm.

The data will be generated randomly using **rand**() % 500 + 1 for a range of 1-500 per integer. Once the program has begun for each array size, all tests must be completed in succession before exiting the program in order to maintain consistency across all results. The same exact arrays (random, increasing, and decreasing) will be used for each of the 4 algorithms.

To assess the results, we will use 4 different measurements:

1. Time (the speed at which all process requests are fulfilled).
2. The number of processes it leaves unallocated.
3. The number of memory slots that are left unused.
4. The amount of free memory remaining **in used memory slots** upon completion.

All results will be compiled into an Excel spreadsheet in order to assess the fastest and most efficient algorithms with respect to process request order.

RESULTS (Full spreadsheet submitted)

- **Random Requests**

ARRAY SIZE	TEST	FIRST FIT				NEXT FIT				BEST FIT				WORST FIT			
		Time	Unalloc. Process	Unused Memory Slots	Fragments	Time	Unalloc. Process	Unused Memory Slots	Fragments	Time	Unalloc. Process	Unused Memory Slots	Fragments	Time	Unalloc. Process	Unused Memory Slots	Fragments
10	1	19	1	5	174	19	1	2	1040	20	1	3	638	19	1	5	1784
	2	19	1	5	174	19	1	2	1040	20	1	3	638	19	1	5	1784
	3	19	1	5	174	19	1	2	1040	19	1	3	638	19	1	5	1784
	4	19	1	5	174	19	1	2	1040	19	1	3	638	19	1	5	1784
	5	19	1	5	174	19	1	2	1040	20	1	3	638	19	1	5	1784
AVERAGE:		19	1	5	174	19	1	2	1040	19.6	1	3	638	19	1	5	1784
100	1	188	17	0	3807	182	24	38	4580	262	11	1	2201	199	17	36	19606
	2	187	17	0	3807	223	24	38	4580	214	11	1	2201	196	17	36	19606
	3	187	17	0	3807	211	24	38	4580	214	11	1	2201	199	17	36	19606
	4	228	17	0	3807	182	24	38	4580	214	11	1	2201	207	17	36	19606
	5	221	17	0	3807	183	24	38	4580	214	11	1	2201	231	17	36	19606
AVERAGE:		202.2	17	0	3807	196.2	24	38	4580	223.6	11	1	2201	206.4	17	36	19606
1000	1	3429	70	1	23599	2568	185	66	56489	6401	22	2	6752	4039	70	77	224448
	2	3259	70	1	23599	2425	185	66	56489	7537	22	2	6752	4279	70	77	224448
	3	3083	70	1	23599	2364	185	66	56489	6744	22	2	6752	4121	70	77	224448
	4	3082	70	1	23599	2553	185	66	56489	7497	22	2	6752	4276	70	77	224448
	5	3045	70	1	23599	2580	185	66	56489	7846	22	2	6752	3951	70	77	224448
AVERAGE:		3179.6	70	1	23599	2498	185	66	56489	7205	22	2	6752	4133.2	70	77	224448

- **Increasingly Larger Requests**

ARRAY SIZE	TEST	FIRST FIT				NEXT FIT				BEST FIT				WORST FIT			
		Time	Unalloc. Process	Unused Memory Slots	Fragments	Time	Unalloc. Process	Unused Memory Slots	Fragments	Time	Unalloc. Process	Unused Memory Slots	Fragments	Time	Unalloc. Process	Unused Memory Slots	Fragments
10	1	19	1	4	561	19	2	2	1618	19	1	3	538	19	1	4	1784
	2	19	1	4	561	19	2	2	1618	19	1	3	538	20	1	4	1784
	3	18	1	4	561	18	2	2	1618	20	1	3	538	19	1	4	1784
	4	18	1	4	561	19	2	2	1618	20	1	3	538	20	1	4	1784
	5	19	1	4	561	19	2	2	1618	20	1	3	538	20	1	4	1784
AVERAGE		18.6	1	4	561	18.8	2	2	1618	19.6	1	3	538	19.6	1	4	1784
100	1	186	30	52	4101	182	34	40	8329	235	12	1	3232	194	30	52	14108
	2	186	30	52	4101	183	34	40	8329	205	12	1	3232	194	30	52	14108
	3	186	30	52	4101	182	34	40	8329	206	12	1	3232	193	30	52	14108
	4	185	30	52	4101	182	34	40	8329	237	12	1	3232	194	30	52	14108
	5	185	30	52	4101	182	34	40	8329	215	12	1	3232	194	30	52	14108
AVERAGE		185.6	30	52	4101	182.2	34	40	8329	219.6	12	1	3232	193.8	30	52	14108
1000	1	3810	271	105	73243	2755	313	28	95535	5949	22	2	6884	4104	271	542	133980
	2	3069	271	105	73243	2717	313	28	95535	5296	22	2	6884	4116	271	542	133980
	3	3099	271	105	73243	3027	313	28	95535	5674	22	2	6884	4027	271	542	133980
	4	3141	271	105	73243	2688	313	28	95535	6395	22	2	6884	4078	271	542	133980
	5	3272	271	105	73243	2749	313	28	95535	6362	22	2	6884	3992	271	542	133980
AVERAGE		3278.2	271	105	73243	2787.2	313	28	95535	5935.2	22	2	6884	4063.4	271	542	133980

- **Decreasingly Smaller Requests**

ARRAY SIZE	TEST	FIRST FIT				NEXT FIT				BEST FIT				WORST FIT			
		Time (micro seconds)	Unalloc. Process	Unused Memory Slots	Fragments	Time (micro seconds)	Unalloc. Process	Unused Memory Slots	Fragments	Time (micro seconds)	Unalloc. Process	Unused Memory Slots	Fragments	Time (micro seconds)	Unalloc. Process	Unused Memory Slots	Fragments
10	1	19	1	5	174	19	1	1	1613	19	1	4	388	19	1	5	1784
	2	19	1	5	174	19	1	1	1613	20	1	4	388	19	1	5	1784
	3	19	1	5	174	20	1	1	1613	20	1	4	388	19	1	5	1784
	4	19	1	5	174	19	1	1	1613	19	1	4	388	19	1	5	1784
	5	19	1	5	174	20	1	1	1613	19	1	4	388	20	1	5	1784
AVERAGE		19	1	5	174	19.4	1	1	1613	19.4	1	4	388	19.2	1	5	1784
100	1	189	11	0	2209	181	12	1	2917	246	11	0	1484	201	11	9	22411
	2	188	11	0	2209	181	12	1	2917	205	11	0	1484	195	11	9	22411
	3	189	11	0	2209	181	12	1	2917	205	11	0	1484	237	11	9	22411
	4	188	11	0	2209	181	12	1	2917	204	11	0	1484	197	11	9	22411
	5	189	11	0	2209	80	12	1	2917	204	11	0	1484	197	11	9	22411
AVERAGE		188.6	11	0	2209	160.8	12	1	2917	212.8	11	0	1484	205.4	11	9	22411
1000	1	3617	22	1	7313	2469	22	6	9760	5550	22	0	6889	4801	22	1	234964
	2	3366	22	1	7313	2343	22	6	9760	6456	22	0	6889	4792	22	1	234964
	3	3844	22	1	7313	2400	22	6	9760	4605	22	0	6889	4124	22	1	234964
	4	4160	22	1	7313	2272	22	6	9760	4717	22	0	6889	4600	22	1	234964
	5	5023	22	1	7313	2394	22	6	9760	4625	22	0	6889	4631	22	1	234964
AVERAGE		4002	22	1	7313	2375.6	22	6	9760	5190.6	22	0	6889	4589.6	22	1	234964

ANALYSIS

The results were analyzed based on two main factors:

- Speed
- Efficiency of memory use

Speed analysis is very straightforward. As you can see in the chart below, the 4 algorithms consistently ranked the same across the 3 different array orders.

	SPEED	
	Rank	Algorithm
Random Requests	1	**Next Fit**
	2	First Fit
	3	Worst Fit
	4	Best Fit
Increasingly Larger Requests	1	**Next Fit**
	2	First Fit
	3	Worst Fit
	4	Best Fit
Decreasingly Smaller Requests	1	**Next Fit**
	2	First Fit
	3	Worst Fit
	4	Best Fit

The Next Fit algorithm always performed with the greatest speed, presumably due to the fact that the search always begins at where the last search stopped. The algorithm does not need to reset to the beginning of the memory array each time.

The First Fit algorithm always ranked 2nd in speed, presumably because the algorithm always finds the very first available location in which the process request fits. It does, however, reset to the beginning of the memory array each time, which makes it a bit slower than the Next Fit algorithm.

Consistently in 3rd place is the Worst Fit algorithm, which finds the first memory slot in which the process request will fit, then checks for the largest slot available after that slot.

The Best Fit algorithm was always the slowest. This algorithm checks for the first available memory slot in which the process request will fit, calculates the difference between the size of that slot and the size of the process request, then checks for a slot that will fit the process request more tightly (leaving less leftover memory in the slot). This algorithm performs the most checks and calculations, therefore takes the longest time to execute.

In terms of speed, our hypothesis was indeed correct.

The second main factor, efficiency of memory use, is far more difficult to interpret since the analysis depends heavily on what the system designer considers to be most important. We used three variables for our analysis: the number of unallocated processes after execution, the number of memory slots left completely unused, and amount of unused memory in each memory slot in which a process was allocated.

	EFFICIENCY OF MEMORY USE					
	UNALLOC. PROCESS		UNUSED MEM SLOT		FRAGMENTS	
	Rank	Algorithm	Rank	Algorithm	Rank	Algorithm
Random Requests	1	**Best Fit**	1	**Worst Fit**	1	**Best Fit**
	2	First Fit	2	Next Fit	2	First Fit
	3	Worst Fit	3	Best Fit	3	Next Fit
	4	Next Fit	4	First Fit	4	Worst Fit
Increasingly Larger Requests	1	**Best Fit**	1	**Worst Fit**	1	**Best Fit**
	2	First Fit	2	First Fit	2	First Fit
	3	Worst Fit	3	Next Fit	3	Next Fit
	4	Next Fit	4	Best Fit	4	Worst Fit
Decreasingly Smaller Requests	1	**FIrst Fit**	1	**Worst Fit**	1	**Best Fit**
	1	**Best Fit**	2	Next Fit	2	First Fit
	1	**Worst Fit**	3	First Fit	3	Next Fit
	4	Next Fit	4	Best Fit	4	Worst Fit

If the allocation of the most processes is a high priority, it is easy to see that the Best Fit algorithm best achieves this goal across all 3 array orders. When you consider the fact that it also leaves the least amount of fragmentation, this makes perfect sense; the Best Fit algorithm makes the most of each available memory slot.

It should be noted that when decreasingly smaller requests were used, the First Fit, Best Fit, and Worst Fit algorithms all left the same amount of unallocated processes.

If the system designer wants to make sure that plenty of memory is available at all times without concern for process allocation, the Worst Fit algorithm is by far the best choice.

In terms of efficiency, we were correct to hypothesize that the Best Fit algorithm makes the best use of the available memory slots. However, we were incorrect in our assumption that the Worst Fit would be the 2nd most efficient – unless, that is, always leaving a high amount of available memory is a high priority.

Potential Threats to Validity of Results

If the code is compiled and run on a computer with a fast enough processor clock speed, the differences in speed on all fits will be the same leading you to conclude that all fits runs at the same speed. If the code is compiled and run on a computer with a processor already busy with a previous process, this will output a speed that is not a clear representation of the actually speed of that fit.

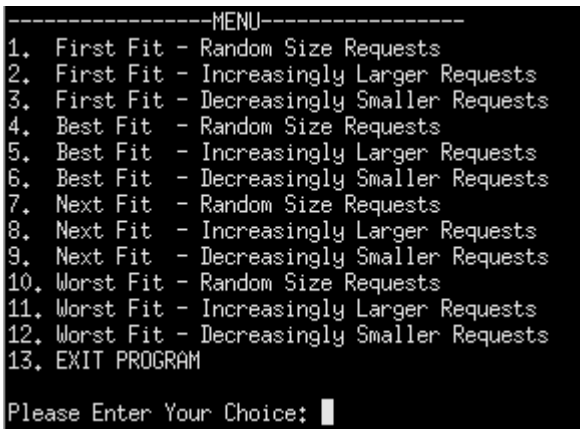
INSTRUCTIONS

When creating this program, we used a Linux (Ubuntu) virtual machine running through VMware Player on a Windows 7 operating system.

To run this program follow these steps

- Open Virtual Machine
- Open and run Ubuntu on your virtual machine
- Open CODE::BLOCK
- Click file, navigate to the source folder
- Click group5project.c then open
- Click run and debug to compile code.

When the program starts running successfully, you will be presented with a menu:



Here you select any number between 1 and 13 with each number performing a different task. 1 through 3 uses the First fit algorithm, giving you options for a random size, a larger size and smaller size request. 4 through 6 uses the Best fit algorithm giving you options for a random size, a larger size and smaller size request. 7 through 9 uses the next fit algorithm giving you options for a random size, a larger size and smaller size request, and 10 through 12 uses the worst fit algorithm giving you options for a random size, a larger size and smaller size request. The last option allows you to exit the program.