Segmentation in Memory

Calculating the number of time steps to reach equilibrium:

To be absolutely certain that the memory has reached equilibrium, i.e. process eviction has started to accommodate for newer processes, we can assume the worst case scenario.

In worst case:

Size of every process = 1 byte

Max number of processes in memory = 1MB / 1B = 2^20 = 1048576

Therefore, to be 100 percent certain that equilibrium has been reached, we need to run the simulation for 1048576 times before starting to record statistics. However this is unreasonably large number especially when the average size of process for the smallest process mode(2) is 50. We can assume maximum number of processes = 1MB / 50B = 2097.51. We can take some extra caution and set the equilibrium time to 30972(this is supported by experimental testing).

Results from the simulation

All the values given in the table are averaged over 3 consecutive runs. A better formatted table can be found in the attached spreadsheet.

process size	insertion policy	avg fragmentation	avg hole size	avg holes examined	max fragmentation	min fragmentation	avg process count	max process count	min process count	avg process size	avg hole count	max hole count	min hole count
1-1024	best fit	0.04825	63.179681	711.059021	0.076956	0.028868	1941.077677	1958.666666	1979	509.155334	800.205017	1915.666666	781.333333
	worst fit	0.186477	230.923269	846.682678	0.193364	0.180012	1688.302002	1709.333333	1655	509.047668	852.680684	861	831
	first fit	0.083917	95.123136	1361.335998	0.099028	0.071323	1844.026327	1921	1841.666666	502.408	925.529338	943	904
1-100	best fit	0.015099	8.498933	932.222005	0.019691	0.010803	20348.216797	20436.666666	20225.666666	50.561667	1896.762003	2008.333333	1797
	worst fit	0.157455	23.628845	6987.4813417	0.159521	0.155805	17531.791667	17577.666666	17484	49.633666	6987.459635	7090.333333	6993.333333
	first fit	0.073765	13.224954	2257.542969	0.075918	0.071589	19246.195311	19306.333333	19192.333333	50.497334	5849.337415	5938.333333	5745.666666
500-1000	best fit	0.054321	88.256482	637.649658	0.072135	0.045653	1322.246663	1335.333333	1295.666666	749.608663	645.120686	655.666666	635.333333
	worst fit	0.115167	195.094853	618.923665	0.127465	0.106193	1240.080997	1255	1219.666666	750.687011	618.92102	634	607
	first fit	0.0789	129.202283	312.778991	0.091699	0.06975	1284.88571	1300.333333	1262.333333	755.887654	640.369018	644.333333	626

Worst fit insertion policy

The results from average fragmentation show that across all the process sizes, fragmentation is highest in worst fit insertion policy followed by first fit followed by best fit. Process count is also the lowest for worst fit in all cases. Higher fragmentation and lower process count means that this insertion policy results in wastage of memory space.

First fit insertion policy

First fit insertion had overall better results from worst fit and worse than best fit.

Best fit insertion policy

Best fit insertion policy showed the best results in almost every field. It had the lowest fragmentation and highest number of processes running at a time. The average hole size of best is the smallest, which is in line with what was discussed in class. Even though best fit had the lowest fragmentation among all insertion policies, it also had the highest number of holes in mode 3(size 50-100). This is because, as written below, best fit leaves small holes in memory and in the case of mode 3, holes smaller than 50B cannot be filled at all, unless a process directly touching it is evicted from memory.

It can also be seen that in mode 2(size 1-100), best fit had a surprisingly large number of continuous evictions and insertions(95 and 110 respectively).

Variation with respect to process size

For insertion with best fit policy, processes with small sizes show the best result. This is expected because best fit tends to leave small holes in the memory, which processes as small as 1 byte can fill more easily. On the other hand, worst fit performs worse with smaller process sizes because it starts to fill the largest hole in memory first and when a larger sized process arrives, it cannot accommodate it in memory.

Unexpected observations

Looking at the average holes examined column, we see that best fit performed better than first fit in first 2 modes. Theoretically, first fit should have to examine the lowest number of holes before inserting a process, which is one of the big advantages it has over other insertion policies. This result can only be explained by looking at the code used to implement best fit. Unlike worst fit, we already know the most ideal hole size for best fit, which is the size of the process itself. Therefore as soon as we encounter a hole with `process_size == hole_size`, we exit the loop and do not need to search any further. This hypothesis is further supported by the fact that the difference between insertion policies is more prominent in mode 2(sizes 1-100). Because the number of available sizes to choose from is less, it is more likely that a process with the same size as that of a hole will be created. This phenomenon may not be observed if the program was not written to explicitly stop searching after encountering `process_size == hole_size`.

During class, it was discussed that around 1/3rd of memory is usually wasted due to fragmentation, however the maximum fragmentation we noticed in the simulation was 0.193364(around 20 percent).