**Different Memory Allocation Strategies**

**Introduction:**

In this lab, we explore the performance of four memory management techniques: OMAP, Best-fit, Worst-fit and paging. We use a C program including different kinds of CPU scheduling strategies, namely FCFS, SRTF, and RR. However, in this lab, what we need to do is executing those memory allocation strategies under the FCFS. We operate OMAP, Best-fit, Worst-fit, and paging with two different page sizes (256B and 8KB) under FCFS. We execute our program and collect the average turnaround time (TAT), the average response time (RT), the CPU Busy time(%) (CBT), the throughput (T), the average waiting time (AWT) and the average waiting time in job queue (AWTJQ).

**Workflow:**

At first, we do not have the memory limit because it was an infinite memory, for this time, our memory is finite, we have a variable called AvailableMemory contained in common2.h, and it was initialized a value by system, then we decrease it as we allocate memory and increase when the process is finished. Another variable called memorypolicy, which represents four memory allocation strategies. There is also a function getMemoryPolicy() which is to tell if the memorypolicy allocate memory successfully. Only when the memory is enough for process, then LongtermScheduler() function will move process from job queue to ready queue. To implement best-fit and worst-fit, we use double-linked list. Each node represents the memory of each process and there is also memory left in it. The smallest memory that is larger than MemoryRequest should be used when the strategy is best-fit, while the largest memory is used in worst-fit. When process is finished, we use releaseMemoryPolicy to add MemoryRequest back to AvailableMemory. In double linked list, we combine two nodes representing released memory together to make it one block of memory. Bookkeeping function is used for calculating all data and print them, including AWTJQ, which is the average waiting time in job queue. We modify the memorypolicy on code each time before executing. When memorypolicy is paging, we modify page size on code because two different page size(256 and 8kb) are needed.

**Data:**

Below is the data table from four different memory allocation strategies, the CPU scheduling is FCFS.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Memory Allocation Strategies** | **ATAT** | **ART** | **CBT** | **T** | **AWT** | **AWTJQ** | **Processes Completed** |
| OMAP | 17.193593 | 3.415846 | 0.971125 | 0.987527 | 11.611648 | 0.589544 | 229 |
| Paging(256) | 17.168550 | 3.435489 | 0.971173 | 0.987577 | 11.571321 | 0.624386 | 229 |
| Paging(8kb) | 16.634460 | 3.552252 | 0.971197 | 0.987601 | 10.902383 | 0.905195 | 229 |
| Best-fit | 15.488415 | 4.000599 | 0.970318 | 1.009409 | 9.493580 | 1.635629 | 234 |
| Worst-fit | 13.319381 | 5.473720 | 0.931331 | 1.000090 | 5.955621 | 3.923925 | 188 |

**Analysis:**

Above is the average turnaround time graph for the five allocation strategies. The Worst-fit has the best ATAT but the number of completed processes really drags it down. In this case, the Best-fit still did a good job in terms of average turnaround time and number of completed processes.

Above is the average response time graph for the five allocation strategies. As we can see, OMAP has the lowest response time. The reason is when a process is ready to run, there is no need to wait for others to complete. And OMAP is basically paging with page size 1 byte, as the page size growing, the response time is also increasing because more wasted space to store other data. This is also why the Worst-fit has the longest average response time.

Above is the CPU Busy Time and Throughput graph for the five allocation strategies. There are not noticeable differences among the five strategies. The reason may be since they are all running in the same environment and similar code file, CPU is working as expected and the overall amount of jobs are the same.

Above is the Average Waiting Time graph for the five allocation strategies. Again, the Worst-fit seems perform well but it cannot complete as many jobs as others do. The Best-fit strategies will create the smaller wasted space, thus resulting in shorter waiting time.

Above is the Average Waiting Time in the Job Queue. And this result is as expected. Since OMAP has enough memory, there is no need to wait for other processes to complete. And as the page size increasing, new processes will spend more time in finding sufficient space for them to fit in. And the Worst-fit strategy waste the most amount of time in the job queue on finding spots.

**Conclusion:**

In this lab assignment, we implement four different memory allocation strategies, with the CPU scheduling FCFS. There is no big difference among each data of different strategies, except that metric data in Worst-fit is relatively less than others. Compared to infinite memory, the performance might not be so perfect but it is also pretty good.