Design of MP2

The objective of machine problem 2 is to get a demand-paging based memory allocation system. We use this system to allocate each frame which is 4 KB in virtual memory. Following the simple\_frame\_pool, I built cont\_frame\_pool for the allocation of consecutive frames. There are several key points deserving of discussion and highlighting.

The first important point is that we cannot use only one bit as the smallest unit to construct our bitmap which stores the allocation information of all frames in the frame pool. There are three states, FREE, HEAD\_OF\_SEQUENCE and ALLOCATED. In this condition, we have to use two bits to present a frame, where 00 means FREE, 01 means HEAD\_OF\_SEQUENCE and 10 means ALLOCATED. And we also use bit manipulation to handle the bitmap. However, there are several challenging scenarios. It’s really difficult to use bit manipulation to search the whole frame pool and to manage each frame with any fragmentations. How to handle them? And also whether our code could handle multiple process frame pools is also interesting.

For me, I do not use the specified code to handle these fragmentations in my project. But for the purpose of minimizing fragmentation and simplifying the algorithm, when I search for a char array of bitmaps, I always look at the first two digits of this byte because I make the frame of each application a multiple of 4 as the starting point. And I also round the frame number up to a multiple of 4 in order to simplify the searching process but with an accurate update to the bitmap. For example, if I apply for 22 frames to the system, the system will search for 24 consecutive frames, the starting point of which is a multiple of 4. And we only update the first 22 frames of these 24 frames in bitmap, so the left 2 frames are fragmentations. Thus, every time when we apply for a multiple of four frames, it does not cause fragmentations, and only several fragmentations less than 4 will be introduced in other conditions. We can define a ratio which is (allocated frames)/ (allocated frames +fragmentations). It seems that the more we apply for a small number of frames, the more fragments we have. So my program is more suitable for a large amount of frame allocation.

And for multiple process pools, I have not completed due to several debugging problems. So I have to make a small modification in kernel.C, changing the release\_frames function from static to public. And I also removed the comment of process pool to validate the robustness of my allocation system. It works well. The modified kernel.C is also attached in my ZIP file.