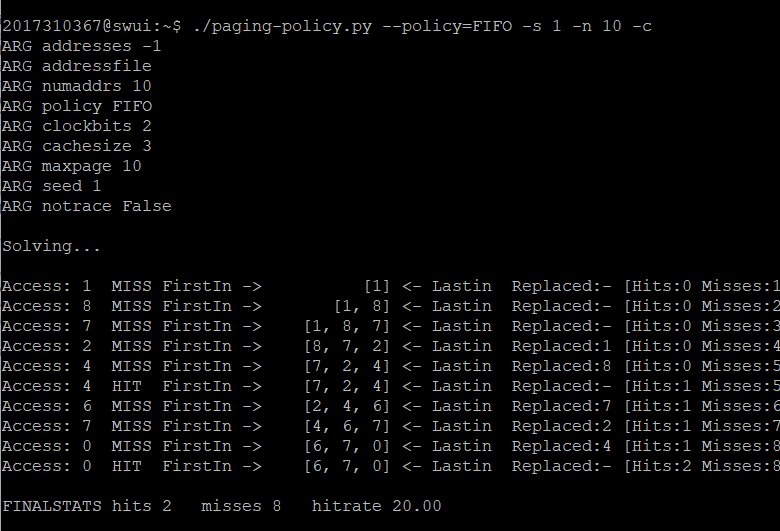
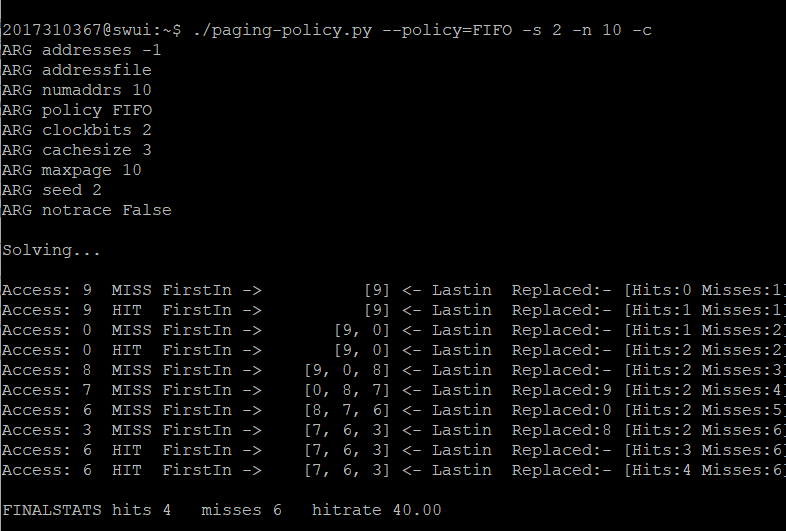
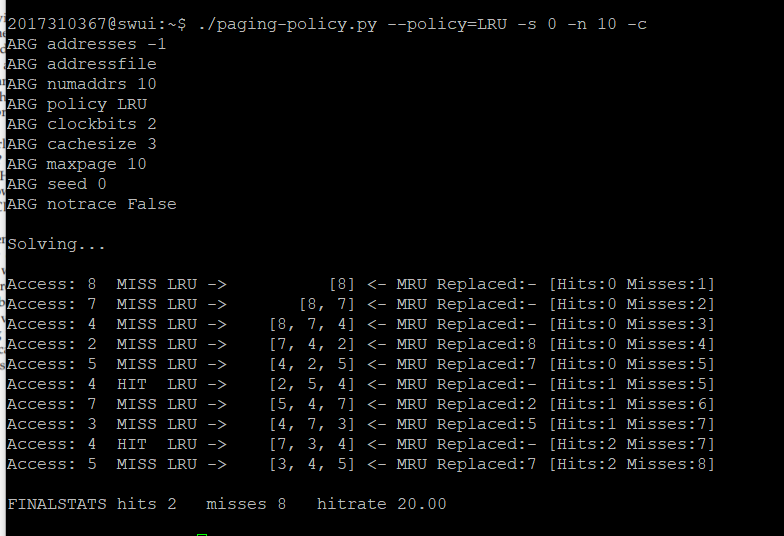
1. Generate random addresses with the following arguments: -s 0 -n 10, -s 1 -n 10, and -s 2 -n 10. Change the policy from FIFO, to LRU, to OPT. Compute whether each access in said address traces are hits or misses.

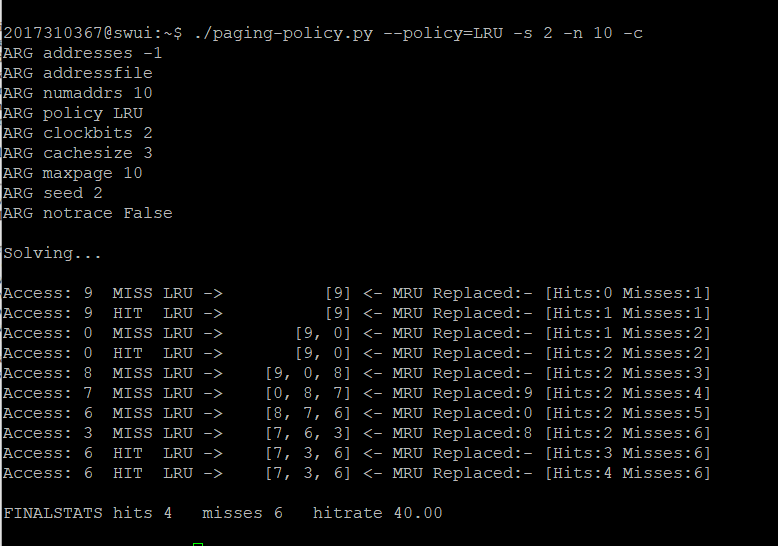
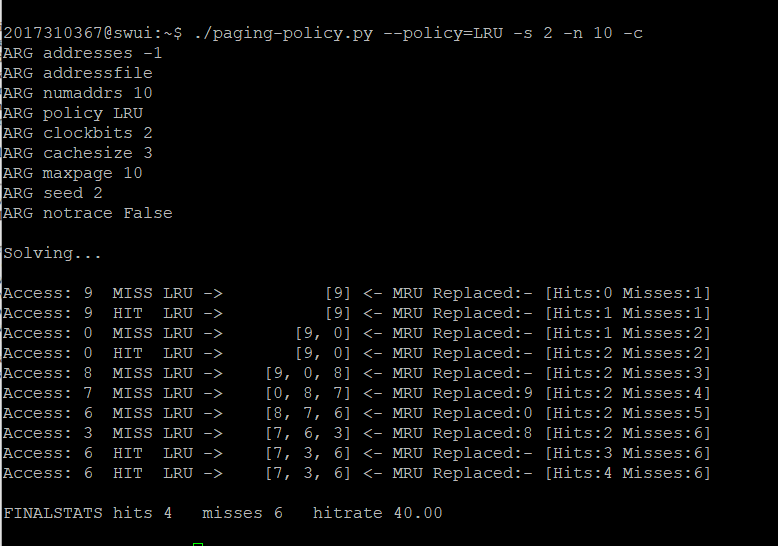
테이블이(가) 표시된 사진

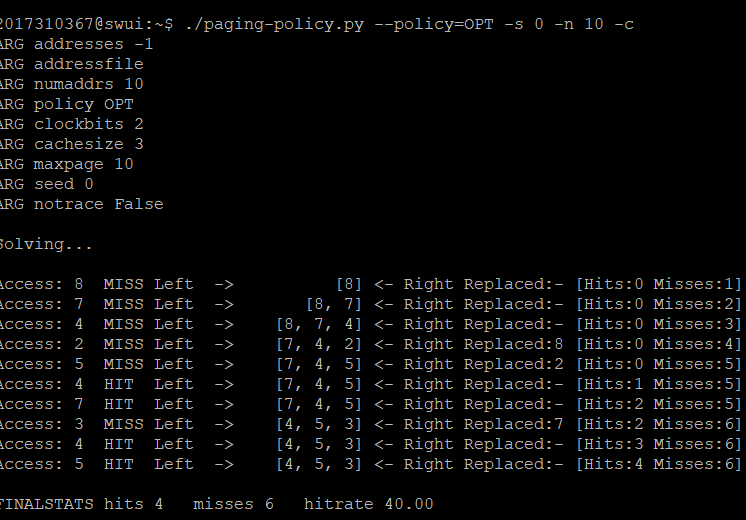
자동 생성된 설명

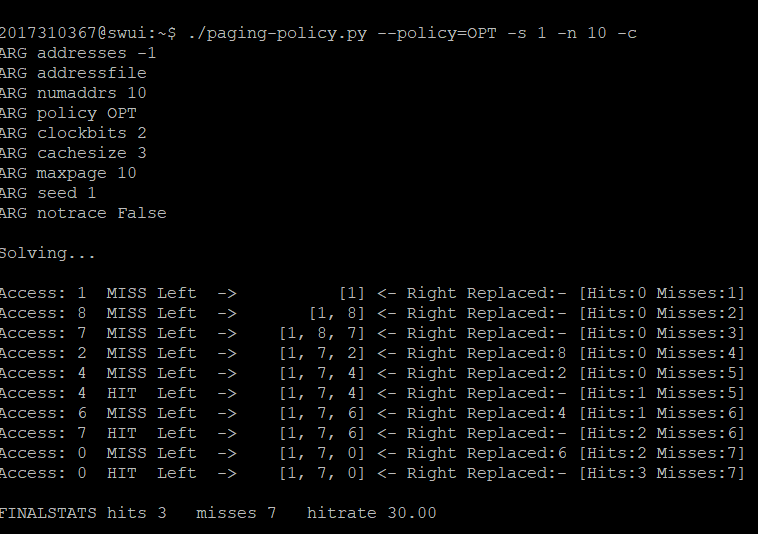


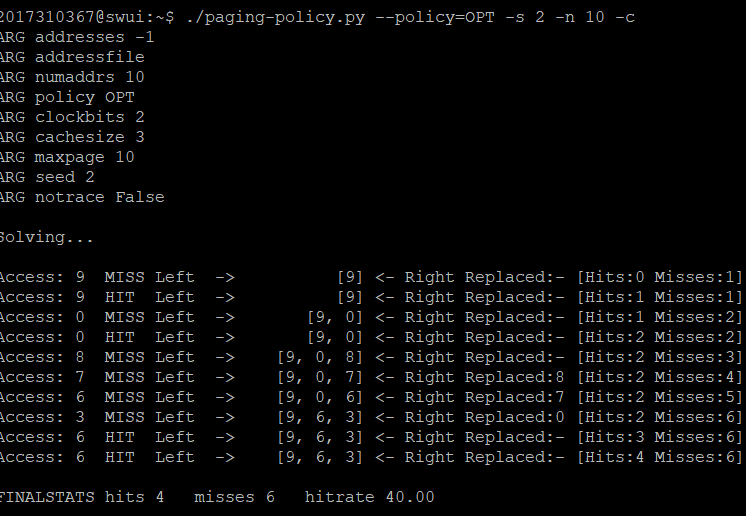












2. For a cache of size 5, generate worst-case address reference streams for each of the following policies: FIFO, LRU, and MRU (worst-case reference streams cause the most misses possible. For the worst case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?

3. Generate a random trace (use python or perl). How would you expect the different policies to perform on such a trace?

4. Now generate a trace with some locality. How can you generate such a trace? How does LRU perform on it? How much better than RAND is LRU? How does CLOCK do? How about CLOCK with different numbers of clock bits?

5. Use a program like valgrind to instrument a real application and generate a virtual page reference stream. For example, running valgrind --tool=lackey --trace-mem=yes ls will output a nearly-complete reference trace of every instruction and data reference made by the program ls. To make this useful for the simulator above, you’ll have to first transform each virtual memory reference into a virtual page-number reference (done by masking off the offset and shifting the resulting bits downward). How big of a cache is needed for your application trace in order to satisfy a large fraction of requests? Plot a graph of its working set as the size of the cache increases.