CACHE OPTIMIZATION

Logic and Implementation:

To optimize our caching mechanism, we implemented a priority queue that arranges accessed sets, with the least frequently accessed set positioned at the top. This ensures that when all ways of a set are full, the least frequently accessed set can be identified and used for replacement according to the Least Recently Used (LRU) policy.

When a way needs to be replaced, it is remapped to a way in the least accessed set, which is determined using the priority queue. This remapping is facilitated by defining two maps: one for keys and one for values. The key map stores the index and tag of the victim set, while the value map stores the set and way of the new set. Whenever a set that exists in the map is accessed, it is redirected to the corresponding set and way, thereby reducing latency.

File Implementations:

1. **pq.cc**: This file in the src folder includes the following functions:
    - **insertSet(int set)**: Initializes every new set.
    - **findLeastUsedSet()**: Identifies the least accessed set in the priority queue.
    - **incrementAccessCount(int set)**: Tracks the number of accesses for each set.
2. **pq.h**: This header file in the inc folder includes the necessary declarations for pq.cc and defines the priority queue.

Modifications in cache.cc:

- **handle_fill()**: This function finds the least used set using findLeastUsedSet(), determines the corresponding way using least_set, and stores the new set and way in the map.
- **handle_read()**: For caches of type LLC, this function checks if the given index and set have a value in the map. If so, it retrieves the corresponding new set and way.

By implementing these changes, we have successfully decreased the latency of cache accesses.