PART B

Implementation of different page replacement algorithms

Introduction:

A virtual memory page replacement simulator requires the implementation of various page replacement algorithms. In this section, I'll show you how to implement three popular page replacement algorithms: FIFO (First-In-First-Out), LRU (Least Recently Used), and OPT (Optimal). The simulator is not included in these examples because they are simplified for demonstration purposes.

1. **FIFO (First-In-First-Out):**
   The OS maintains a list of all pages currently in memory:
   - Page at the head of the list is the oldest one
   - Page at the tail of the list is the most recent arrival
   - On a page fault, the page at the head is removed and the new page is added to the tail of the list
   - Algorithm Explanation: FIFO is one of the simplest page replacement algorithms. It replaces the oldest page in memory when a page fault occurs.
   - Code Implementation:

```
class FIFOPageReplacement(PageReplacementAlgorithm):
  def page_in(self, page, access_pattern):
    page_fault = 0
    eviction = 0
    replacement = 0
    if page not in self.frames:
      page_fault = 1
      if len(self.frames) < self.frame_count:
        self.frames.append(page)
      else:
        eviction = 1
        self.frames.pop(0)
        self.frames.append(page)
        replacement = 1
    return page_fault, eviction, replacement
```

2. **LRU (Least Recently Used):**

Pages that have been heavily used in the last few instructions will probably be heavily used in the next few instructions.

- Conversely, pages that have not been used for a long time will probably remain unused for a long time.
- In the LRU strategy, when a page fault occurs, swap out the page that has been unused for the longest time.
- To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear.
- List is updated on every memory reference.
- Algorithm Explanation: LRU replaces the page that has not been used for the longest time. It requires tracking and maintaining the order of page accesses.
- Code Implementation:

```
class LRUPageReplacement(PageReplacementAlgorithm):
  def page_in(self, page, access_pattern):
    page_fault = 0
    eviction = 0
    replacement = 0
    if page not in self.frames:
      page_fault = 1
      if len(self.frames) >= self.frame_count:
        eviction = 1
        # Find and remove the least recently used page
        lru_page_index = self.find_lru_page(access_pattern)
        lru_page = self.frames.pop(lru_page_index)
      self.frames.append(page)
      return page_fault, eviction, replacement
```

3. **Optimal OPT ():**
    - At the moment that a page fault occurs, some set of pages is in memory.
    - One of these pages will be referenced on the very next instruction.
    - Other pages may not be referenced until several instructions later.
    - The OPT algorithm says that the page which will be referenced furthest in the future should be replaced.
    - It is unrealizable, since at the time of the page fault, the OS has no way of knowing when each of the pages will be referenced next.

    - Algorithm Explanation: OPT replaces the page that will not be used for the longest time in the future, which is theoretically optimal but challenging to implement in practice.
    - Code Implementation:

```
class OPTPageReplacement(PageReplacementAlgorithm):
  def page_in(self, page, access_pattern):
    page_fault = 0
    eviction = 0
    replacement = 0
    if page not in self.frames:
      page_fault = 1
      if len(self.frames) >= self.frame_count:
        eviction = 1
        # Find the page in the frames that won't be used for the longest
          time (Optimal replacement)
        farthest_used = -1
        page_to_replace = None
        for i, frame in enumerate(self.frames):
          if frame not in access_pattern[i:]:
            page_to_replace = frame
            break
          distance = access_pattern[i:].index(frame)
          if distance > farthest_used:
            farthest_used = distance
            page_to_replace = frame
        self.frames.remove(page_to_replace)
      self.frames.append(page)
    return page_fault, eviction, replacement
```