Report of Homework02

Chaoyi Jiang

1、Circular queues are used quite a bit in operating systems and high performance systems, especially when performance matters. Do a little outside research, and edit this section of the readme answering specifically: Why is a ring buffer useful and/or when should it be used?

Circular queues, also known as ring buffers, are a type of queue data structure where the last position is connected back to the first. This makes them particularly useful in situations where the data being stored is constantly overwritten, as it allows for efficient memory utilization. They are commonly used in real-time systems, such as audio and video processing, where data must be processed in a timely manner and limited buffer size is available.
Circular queues have the advantage of constant time complexity for insertion and deletion operations, making them well suited for high performance systems. They also have the added benefit of not suffering from overflow issues, as the oldest data can be overwritten once the buffer is full. However, they can be more complex to implement than traditional linear queues and care must be taken when overwriting data to avoid any potential corruption or lost data. Overall, circular queues are a useful tool for managing real-time data in high performance systems where efficient memory utilization and constant time complexity are a priority.

2、We are going to talk about stacks quite a lot in this course, so it will be important to understand them. Do a little outside research, and edit this section of the README answering specifically: Why is a stack useful and/or when should it be used?

A stack is a linear data structure that operates on a last-in-first-out (LIFO) principle. It's useful because it enables a fast, efficient, and controlled way to manage data. The stack can be implemented using an array or linked list, with the main operations being push (add an element to the top of the stack), pop (remove the top element), and peek (get the value of the top element without removing it).
One common use of stacks is to keep track of function calls in a program, where each function call is pushed onto the stack and popped off when it returns. This is known as the call stack. Other use cases for stacks include solving mathematical expressions, managing memory allocation, and searching for a path in a graph. The key advantage of stacks is their simplicity, as the only operations required are push, pop, and peek. However, the LIFO nature of stacks can also limit their functionality, as you cannot access elements in the middle of the stack without first removing elements from the top.