Documentation for OS ch2 tasks

1. Function to implement FCFS algorithm

Parameters:

- requests: This parameter represents the list of disk requests, which are the track numbers that the disk arm needs to access.
- start: This parameter represents the starting position of the disk arm.

Function Purpose:

The purpose of this function is to simulate the First-Come, First-Served (FCFS) disk scheduling algorithm. In FCFS, the disk arm moves to each request in the order they arrive, without considering the distance or direction.

Algorithm Implementation:

- seek_sequence = [start]: Initialize the seek_sequence list with the starting position of the disk arm.
- seek_sequence.extend(requests): Extend the seek_sequence list with the disk requests. This represents the order in which the disk arm will visit the tracks.
- return seek_sequence, sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): Return two values:
    - seek_sequence: The sequence of tracks that the disk arm visits, including the starting position and all the requests.
    - sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): The total seek time, which is calculated as the sum of the absolute differences between consecutive tracks in the seek_sequence.

Function Output:

The function returns a tuple containing the seek_sequence list and the total seek time.

2. Function to implement SSTF algorithm

Parameters:

- **requests: This parameter represents the list of disk requests, which are the track numbers that the disk arm needs to access.**
- **start: This parameter represents the starting position of the disk arm.**

Function Purpose:

The purpose of this function is to simulate the Shortest Seek Time First (SSTF) disk scheduling algorithm. In SSTF, the disk arm moves to the request that is closest to its current position, minimizing seek time.

Algorithm Implementation:

- **seek_sequence = [start]: Initialize the seek_sequence list with the starting position of the disk arm.**
- **while requests:: Continue the loop until there are no more requests.**
- **closest_track = min(requests, key=lambda x: abs(x - seek_sequence[-1])): Find the request with the minimum absolute distance from the current position of the disk arm using the min function and a lambda function as the key.**
- **seek_sequence.append(closest_track): Append the closest track to the seek_sequence.**
- **requests.remove(closest_track): Remove the closest track from the list of requests.**
- **return seek_sequence, sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): Return two values:**
  - **seek_sequence: The sequence of tracks that the disk arm visits, including the starting position and all the requests.**
  - **sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): The total seek time, which is calculated as the sum of the absolute differences between consecutive tracks in the seek_sequence.**

Function Output:

The function returns a tuple containing the seek_sequence list and the total seek time.

3. Function to implement SCAN algorithm

Parameters:

- **requests:** This parameter represents the list of disk requests, which are the track numbers that the disk arm needs to access.
- **start:** This parameter represents the starting position of the disk arm.
- **direction:** This parameter represents the direction in which the disk arm should move, either "outward" or "inward".

Function Purpose:

The purpose of this function is to simulate the SCAN disk scheduling algorithm. In SCAN, also known as the elevator algorithm, the disk arm moves in one direction, servicing requests along the way until it reaches the end, then reverses direction and services requests in the opposite direction.

Algorithm Implementation:

- **total_tracks = 200:** Set the total number of tracks on the disk.
- **current_position = start:** Initialize the current position of the disk arm to the starting position.
- **seek_sequence = [start]:** Initialize the seek_sequence list with the starting position of the disk arm.
- **if direction == "outward"::** Check if the direction is outward.

In the outward direction:

- **Iterate over the tracks from the current position towards track 0.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **If there are no more requests, break out of the loop.**
- **After reaching track 0, append it to the seek_sequence.**
- **Move to the inward direction and iterate over the tracks from the current position towards the end of the disk.**
- **Append any requested tracks to the seek_sequence and remove them from the requests.**
- **elif direction == "inward"::** Check if the direction is inward.

In the inward direction:

- **Iterate over the tracks from the current position towards the end of the disk.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **If there are no more requests, break out of the loop.**
- **After reaching the end of the disk, append the last track to the seek_sequence.**
- **Move to the outward direction and iterate over the tracks from the current position towards track 0.**
- **Append any requested tracks to the seek_sequence and remove them from the requests.**
- **return seek_sequence, sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): Return two values:**
    - **seek_sequence: The sequence of tracks that the disk arm visits, including the starting position and all the requested tracks.**
    - **sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): The total seek time, which is calculated as the sum of the absolute differences between consecutive tracks in the seek_sequence.**

Function Output:

The function returns a tuple containing the seek_sequence list and the total seek time.

4. Function to implement C-SCAN algorithm

Parameters:

- **requests: This parameter represents the list of disk requests, which are the track numbers that the disk arm needs to access.**
- **start: This parameter represents the starting position of the disk arm.**
- **direction: This parameter represents the direction in which the disk arm should move, either "outward" or "inward".**

Function Purpose:

The purpose of this function is to simulate the C-SCAN disk scheduling algorithm. In C-SCAN, the disk arm moves in one direction, servicing requests along the way until it reaches the end, then jumps to the beginning of the disk and continues in the same direction.

Algorithm Implementation:

- **total_tracks = 200: Set the total number of tracks on the disk.**
- **seek_sequence = [start]: Initialize the seek_sequence list with the starting position of the disk arm.**
- **if direction == "outward":: Check if the direction is outward.**

In the outward direction:

- **Iterate over the tracks from the current position towards track 0.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **After reaching track 0, append it to the seek_sequence.**
- **Jump to the end of the disk (track total_tracks - 1).**
- **Iterate over the tracks from the end towards the start, servicing requests along the way.**
- **elif direction == "inward":: Check if the direction is inward.**

In the inward direction:

- **Iterate over the tracks from the current position towards the end of the disk.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **After reaching the end of the disk, append the last track to the seek_sequence.**
- **Jump to the beginning of the disk (track 0).**
- **Iterate over the tracks from the start towards the current position, servicing requests along the way.**
- **return seek_sequence, sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): Return two values:**
  - **seek_sequence: The sequence of tracks that the disk arm visits, including the starting position and all the requested tracks.**

    - **sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): The total seek time, which is calculated as the sum of the absolute differences between consecutive tracks in the seek_sequence.**

Function Output:

The function returns a tuple containing the seek_sequence list and the total seek time.

5. Function to implement LOOK algorithm

Parameters:

- **requests: This parameter represents the list of disk requests, which are the track numbers that the disk arm needs to access.**
- **start: This parameter represents the starting position of the disk arm.**
- **direction: This parameter represents the direction in which the disk arm should move, either "outward" or "inward".**

Function Purpose:

The purpose of this function is to simulate the LOOK disk scheduling algorithm. In LOOK, the disk arm moves only as far as necessary in the requested direction and then reverses its direction without going to the end of the disk.

Algorithm Implementation:

- **seek_sequence = [start]: Initialize the seek_sequence list with the starting position of the disk arm.**
- **if direction == "outward":: Check if the direction is outward.**

In the outward direction:

- **Iterate over the tracks from the starting position towards track 0.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **Iterate over the tracks from the starting position towards the end of the disk, servicing requests along the way.**

- **elif direction == "inward":: Check if the direction is inward.**

In the inward direction:

- **Iterate over the tracks from the starting position towards the end of the disk.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **Iterate over the tracks from the starting position towards track 0, servicing requests along the way.**
- **return seek_sequence, sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): Return two values:**
    - **seek_sequence: The sequence of tracks that the disk arm visits, including the starting position and all the requested tracks.**
    - **sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): The total seek time, which is calculated as the sum of the absolute differences between consecutive tracks in the seek_sequence.**

Function Output:

The function returns a tuple containing the seek_sequence list and the total seek time.

6. Function to implement C-LOOK algorithm

Parameters:

- **requests: A list of disk requests, representing the track numbers that the disk arm needs to access.**
- **start: The starting position of the disk arm.**
- **direction: The direction in which the disk arm should move, either "outward" or "inward".**

Function Purpose:

The purpose of this function is to simulate the behavior of the C-LOOK disk scheduling algorithm, which is an improved version of LOOK. C-LOOK only

scans the tracks that are needed, unlike LOOK which scans from the start to the end regardless of whether any requests are present.

Algorithm Implementation:

- **total_tracks = 200: The total number of tracks on the disk.**
- **seek_sequence = [start]: Initialize the seek_sequence list with the starting position of the disk arm.**
- **if direction == "outward":: Check if the direction is outward.**

In the outward direction:

- **Iterate over the tracks from the starting position towards track 0.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **Iterate over the tracks from the end of the disk towards the starting position, servicing requests along the way.**
- **elif direction == "inward":: Check if the direction is inward.**

In the inward direction:

- **Iterate over the tracks from the starting position towards the end of the disk.**
- **If a track is in the requests list, append it to the seek_sequence and remove it from the requests.**
- **Iterate over the tracks from track 0 towards the starting position, servicing requests along the way.**
- **return seek_sequence, sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): Return two values:**
    - **seek_sequence: The sequence of tracks that the disk arm visits, including the starting position and all the requested tracks.**
    - **sum(abs(seek_sequence[i] - seek_sequence[i-1]) for i in range(1, len(seek_sequence))): The total seek time, calculated as the sum of the absolute differences between consecutive tracks in the seek_sequence.**

Function Output:

The function returns a tuple containing the seek_sequence list and the total seek time.

7. GUI

Import Statements:

The script imports necessary modules including tkinter, ttk from tkinter, matplotlib.pyplot as plt, and various algorithm functions from separate files.

Function Definitions:

- **run_algorithm(): This function is called when the "Run Algorithm" button is clicked. It retrieves user input, selects the appropriate algorithm function based on the selected algorithm in the combobox, runs the selected algorithm function with the provided parameters, updates labels with the results, and calls plot_disk_movement() to display the disk movement plot.**
- **plot_disk_movement(seek_sequence, start): This function plots the disk head movement graph using matplotlib. It takes the sequence of disk track accesses (seek_sequence) and the starting position of the disk head (start) as parameters.**

GUI Creation:

- **The script creates a GUI window (root) using tk.Tk().**
- **It sets the title of the window to "Disk Scheduling Algorithms".**
- **It creates various GUI elements including labels, entry fields, comboboxes, and buttons using ttk.Label, ttk.Entry, ttk.Combobox, and ttk.Button.**
- **The entry fields allow users to input the list of disk requests (comma-separated) and the starting position of the disk head.**
- **The combobox allows users to select the disk scheduling algorithm to apply.**
- **Labels are provided to display the total seek time and the sequence of disk track accesses.**
- **A "Run Algorithm" button triggers the run_algorithm() function when clicked.**

Main Loop:

The root.mainloop() statement starts the main event loop, which listens for user interactions and updates the GUI accordingly.