Design of MM (Memory Manager)

Introduction

This project's memory management (MM) system is designed to simulate paging in a memory-constrained environment. The MM handles the allocation and deallocation of memory for processes based on predefined page sizes. The simulation evaluates how different page sizes affect memory utilization and process turnaround time.

Author:

Victoria Guzman

Mark Ohlwine

Adam Kaci

This document provides an overview of the design, including the data structures, algorithms, and key functions used in the simulation.

Design Overview

Objective

- Simulate memory management using paging.
- Support three different page sizes: 100, 200, and 400.
- Efficiently allocate and deallocate memory while handling processes in a First-Come-First-Serve (FCFS) order.

Key Features

1. **Paging**:
    - Memory is divided into fixed-size pages.
    - Processes are allocated memory in chunks of these pages.
2. **Memory Map**:

    - Tracks allocated and free memory regions.
    - Updated dynamically as processes are admitted or complete.
3. **Input Queue**:
    - Processes wait in an input queue until memory becomes available.
    - The queue is managed in FCFS order.
4. **Simulation Clock**:
    - Tracks process arrivals, completions, and memory updates.
5. **Output**:
    - Detailed event logs.
    - Memory map updates after each event.
    - Average turnaround time at the end.

System Components

1. Data Structures

- **`Process` Structure**:
    - Holds the process details such as:
        - `id`: Unique identifier.
        - `arrivalTime`: The time the process enters the system.
        - `lifetime`: Time the process stays in memory after admission.
        - `memoryRequirement`: Total memory needed by the process.
- **`MemoryBlock` Structure**:
    - Represents a block of memory (allocated or free).
        - `start` and `end`: Define the memory range.
        - `processId`: Indicates the process occupying the block (`-1` for free).
- **Memory Map**:
    - A list of `MemoryBlock` structures tracking memory usage.
- **Input Queue**:
    - A `queue` of processes waiting for memory allocation.

---

2. Algorithms

Memory Allocation Algorithm

1. Scan the memory map to find the first free block large enough to accommodate the process.
2. If a suitable block is found:
    - Allocate memory for the process.
    - Split the block if there is excess memory.
3. If no block is available:
    - The process remains in the input queue.

Memory Deallocation Algorithm

1. Identify memory blocks allocated to the process.
2. Mark these blocks as free.
3. Merge adjacent free blocks to reduce fragmentation.

Process Handling

1. **Arrival**:
    - Add the process to the input queue.
    - Attempt to allocate memory for the process at the head of the queue.
2. **Completion**:
    - Free the memory occupied by the process.
    - Re-evaluate the input queue to admit waiting processes.

---

3. Key Functions

`allocate(Process &process)`

- Allocates memory for a process.
- Check for sufficient free memory and update the memory map.
- Returns `true` if the allocation succeeds; otherwise, `false`.

`release(int processId)`

- Frees memory blocks allocated to the specified process.
- Merges adjacent free blocks to reduce fragmentation.

`display(std::ofstream &outfile)`

- Outputs the current memory map to the specified file.

Simulation Functions

- **Arrival Handling**:
  - Detect new arrivals and add them to the input queue.
- **Memory Manager Invocation**:
  - Trigger memory allocation when processes arrive or when memory is freed.
- **Completion Handling**:
  - Detect and handle process completions.

Workflow

1. **Initialization**:
   - Read the input file and initialize the process list.
   - Set memory size to 2000 and page size as specified (100, 200, 400).
   - Initialize an empty memory map with a single free block.
2. **Simulation Loop**:
   - Increment the simulation clock.
   - Handle process arrivals, completions, and memory allocations.
   - Update memory map and log events.
3. **Output**:
   - Generate detailed logs for each event (arrival, admission, completion).
   - Display the memory map after each significant event.
   - Calculate and output the average turnaround time.

Assumptions

- Memory size is always a multiple of the page size.
- A process is admitted to memory only if all its pages can be allocated at once.
- Processes that exceed total memory size are ignored.

Sample Execution

Input File (in1.txt)

```
8
1
0 1000
1 400
2
0 2000
2 200 400
...
```

Output File (out1.txt)

```
t = 0: Process 1 arrives
        Input Queue:[1]
        MM moves Process 1 to memory
        Input Queue:[]
        Memory Map: 0-99: Process 1, Page 1
                100-199: Process 1, Page 2
                ...
...
Average Turnaround Time: 1175.00
```

Testing and Validation

1. **Test Cases**:
    - Small inputs with varying page sizes.
    - Edge cases with processes larger than memory.
    - High process arrival rates to test queue handling.
2. **Validation**:
    - Ensure memory map updates match expectations.
    - Verify turnaround time calculations.

Conclusion

The memory manager simulates paging for different configurations, providing insights into memory utilization and process turnaround time. This design can include advanced techniques such as demand paging or page replacement policies.