Abstract

This report presents the design and implementation of a memory management simulator to demonstrate various memory management techniques, including fixed-sized memory partitioning, unequal-sized fixed partitioning, dynamic memory allocation, buddy system, and paging. The simulator is developed in Python and includes a simple user interface for configuring memory parameters, adding and removing processes, and selecting different memory management strategies. The implementation aims to visualize memory allocation, track process memory usage, and handle fragmentation. Comparative analysis of the techniques is performed using a uniform set of processes, with results displayed in graphical form.

Introduction

Memory management is a critical component of operating systems, ensuring efficient allocation, utilization, and management of memory resources. Various techniques have been developed to handle memory allocation, each with its own advantages and drawbacks. This report explores five such techniques: fixed-sized memory partitioning, unequal-sized fixed partitioning, dynamic memory allocation, buddy system, and paging. The goal is to implement a simulator that demonstrates these techniques, providing insights into their functionality and performance through visualization and analysis.

Objectives

The primary objective of this project is to design and implement a memory management simulator that demonstrates the following techniques:

1. Fixed-sized memory partitioning
2. Unequal-sized fixed partitioning
3. Dynamic memory allocation
4. Buddy system
5. Paging

Methodology

Fixed-sized Memory Partitioning

Implementation:

1. Divide memory into fixed-size partitions.
2. Allocate processes to partitions using First Fit, Best Fit, or Worst Fit strategies.
3. Display the memory allocation status.

Unequal-sized Fixed Partitioning

Implementation:

1. Divide memory into partitions of different sizes.
2. Allocate processes based on partition sizes using appropriate strategies.
3. Display the memory allocation status.

Dynamic Memory Allocation

Implementation:

1. Use dynamic memory allocation techniques like First Fit, Best Fit, or Worst Fit.
2. Implement allocation and de-allocation of memory blocks.
3. Handle external fragmentation.
4. Display the memory allocation status.

Buddy System

Implementation:

1. Get a large memory chunk and allocate memory for processes of different sizes.
2. Implement allocation and de-allocation of memory blocks.
3. Display the memory allocation status.

Paging

Implementation:

1. Implement paging with a fixed page size.
2. Simulate the process of dividing the process memory into pages and the physical memory into frames.
3. Use a page table to keep track of the mapping between pages and frames.
4. Display the memory allocation status.

User Interface

A simple user interface is created to:

1. Define the total size of memory.
2. Select a memory management technique.
3. Add or remove processes with specific memory requirements.
4. View the current memory allocation status.

Process Management

Simulate the arrival and departure of processes. Each process has a unique identifier and specified memory requirement. The system allocates and deallocates memory based on the chosen memory management technique.

Memory Allocation Algorithms

Fixed-sized Partitioning

First Fit:

1. When a process arrives, scan the list of fixed-sized partitions and allocate the first partition that is sufficiently large to accommodate the process.
2. Deallocate the memory when the process departs and mark the partition as available.
3. Display the memory allocation status after each allocation and deallocation.

Best Fit:

1. When a process arrives, scan the list of fixed-sized partitions and find the smallest partition that is large enough to accommodate the process.
2. Allocate the process to this partition.
3. Deallocate the memory when the process departs and mark the partition as available.
4. Display the memory allocation status after each allocation and deallocation.

Dynamic Memory Allocation

First Fit:

1. When a process arrives, scan the list of available memory blocks and allocate the first block that is sufficiently large to accommodate the process.
2. Split the block if necessary, and update the list of available blocks.
3. Deallocate the memory when the process departs and merge adjacent free blocks to minimize fragmentation.
4. Display the memory allocation status after each allocation and deallocation.

Best Fit:

1. When a process arrives, scan the list of available memory blocks and find the smallest block that is large enough to accommodate the process.
2. Allocate the process to this block and split the block if necessary.
3. Update the list of available blocks.
4. Deallocate the memory when the process departs and merge adjacent free blocks to minimize fragmentation.
5. Display the memory allocation status after each allocation and deallocation.

Worst Fit:

1. When a process arrives, scan the list of available memory blocks and find the largest block available.
2. Allocate the process to this block and split the block if necessary.

3. Update the list of available blocks.
4. Deallocate the memory when the process departs and merge adjacent free blocks to minimize fragmentation.
5. Display the memory allocation status after each allocation and deallocation.

Output

Conclusion

This report demonstrates the implementation of various memory management techniques, highlighting their strengths and weaknesses through a practical simulation. By comparing fixed-sized partitioning, unequal-sized partitioning, dynamic allocation, buddy system, and paging, we gain insights into their effectiveness in managing memory resources. The visualand analytical results provide a comprehensive understanding of each technique's performance, aiding in the selection of appropriate memory management strategies for different scenarios.