INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

Memory Management Simulator

Design and Implementation Report

Name: Abhinav Shresth
Enrollment No.: 24115008
Branch: Computer Science and Engineering (II Year)

Contents

1 Introduction **3**

2 Objectives **3**

3 Simulator Interface **3**
3.1 Command-Line Interface . . . 3
3.2 Command Description . . . 3

4 Metrics and Formulas **3**
4.1 Memory Utilization . . . 3
4.2 External Fragmentation . . . 4
4.3 Internal Fragmentation . . . 4
4.4 Allocation Failure Rate . . . 4

5 Allocation Strategies **4**
5.1 First Fit Allocation . . . 4
5.1.1 Logic . . . 4
5.1.2 Data Structure . . . 4
5.1.3 Test Case . . . 4
5.1.4 Observation . . . 5
5.2 Best Fit Allocation . . . 5
5.2.1 Logic . . . 5
5.2.2 Data Structure . . . 5
5.2.3 Test Case . . . 5
5.2.4 Observation . . . 6
5.3 Worst Fit Allocation . . . 6
5.3.1 Logic . . . 6
5.3.2 Data Structure . . . 6
5.3.3 Test Case . . . 7
5.3.4 Observation . . . 7
5.4 Buddy Allocation . . . 8
5.4.1 Logic . . . 8
5.4.2 Data Structure . . . 8
5.4.3 Test Case . . . 8
5.4.4 Observation . . . 9

6 Comparative Analysis **10**

7 Cache Memory Simulator **10**
7.1 Introduction . . . 10

8 Cache Simulator Interface **10**
8.1 Command-Line Interface . . . 10
8.2 Command Description . . . 11

9 Assumptions and Performance Model **11**
9.1 Block and Addressing Assumptions . . . 11
9.2 Cache Hit Times and Penalties . . . 11
9.3 Access Time Calculation . . . 11
9.4 Performance Metrics . . . 12

10 Cache Replacement Policies **12**
10.1 Least Recently Used (LRU) . . . 12
10.1.1 Overview . . . 12
10.1.2 Data Structures Used . . . 12
10.1.3 Test Case . . . 12
10.1.4 Observation . . . 13
10.2 First In First Out (FIFO) . . . 13
10.2.1 Overview . . . 13
10.2.2 Data Structures Used . . . 13
10.2.3 Test Case . . . 13
10.2.4 Observation . . . 14
10.3 Least Frequently Used (LFU) . . . 14
10.3.1 Overview . . . 14
10.3.2 Data Structures Used . . . 14
10.3.3 Test Case . . . 14
10.3.4 Observation . . . 15

11 Comparison of Replacement Policies **16**

12 Multi-Level Caching **16**
12.1 Theory . . . 16
12.2 Hierarchy Diagram . . . 17
12.3 L1 Cache Behavior . . . 17
12.3.1 Test Case . . . 17
12.3.2 Observation . . . 17
12.4 L2 Cache Behavior . . . 17
12.4.1 Test Case . . . 17
12.4.2 Observation . . . 18
12.5 L3 Cache Behavior . . . 19
12.5.1 Test Case . . . 19
12.5.2 Observation . . . 19

13 Conclusion **20**

14 Future Work and Suggestions **20**
14.1 Feature Enhancements . . . 20
14.2 Performance Improvements . . . 20

1 Introduction

Memory management is a fundamental responsibility of an operating system. Efficient allocation and deallocation of memory directly affect system performance, memory utilization, and fragmentation.

This project implements a **Memory Management Simulator** that supports multiple dynamic memory allocation strategies and provides detailed statistics such as fragmentation, utilization, and allocation failure rate. The simulator is designed as an interactive command-line interface (CLI) to visualize memory behavior under different allocation policies.

2 Objectives

The objectives of this allocator module are:

- To simulate common dynamic memory allocation strategies
- To visualize memory layouts after allocation and deallocation
- To compute internal and external fragmentation
- To compare allocation strategies under identical workloads

3 Simulator Interface

3.1 Command-Line Interface

The simulator provides the following commands:

```
init memory <size>
set allocator <first|best|worst|buddy>
malloc <size>
free <id>
dump memory
stats memory
exit
```

3.2 Command Description

Command	Description
`init memory N`	Initialize memory of size N bytes
`set allocator X`	Select allocation strategy
`malloc S`	Allocate S bytes of memory
`free ID`	Free allocated block
`dump memory`	Display memory layout
`stats`	Display allocator statistics
`exit`	Terminate simulator

4 Metrics and Formulas

4.1 Memory Utilization

$$\text{Memory Utilization} = \frac{\text{Used Memory}}{\text{Total Memory}} \times 100$$

This metric indicates how efficiently the memory is being used.

4.2 External Fragmentation

External fragmentation occurs when free memory exists but is split into non-contiguous blocks.

$$\text{External Fragmentation} = \frac{\text{Total Free Memory} - \text{Largest Free Block}}{\text{Total Free Memory}} \times 100$$

4.3 Internal Fragmentation

Internal fragmentation occurs when allocated blocks contain unused memory. This is common in Buddy Allocation due to power-of-two rounding.

$$\text{Internal Fragmentation} = \frac{\text{Wasted Allocated Memory}}{\text{Total Allocated Memory}} \times 100$$

4.4 Allocation Failure Rate

$$\text{Failure Rate} = \frac{\text{Failed Allocation Requests}}{\text{Total Allocation Requests}} \times 100$$

5 Allocation Strategies

5.1 First Fit Allocation

5.1.1 Logic

First Fit scans memory from the beginning and allocates the first free block large enough to satisfy the request.

5.1.2 Data Structure

- Doubly linked list of memory blocks
- Blocks ordered by starting address

5.1.3 Test Case

```
init memory 256
set allocator first
malloc 40
malloc 60
malloc 30
malloc 50
malloc 20
free 2
free 4
malloc 45
dump memory
stats memory
```

5.1.4 Observation

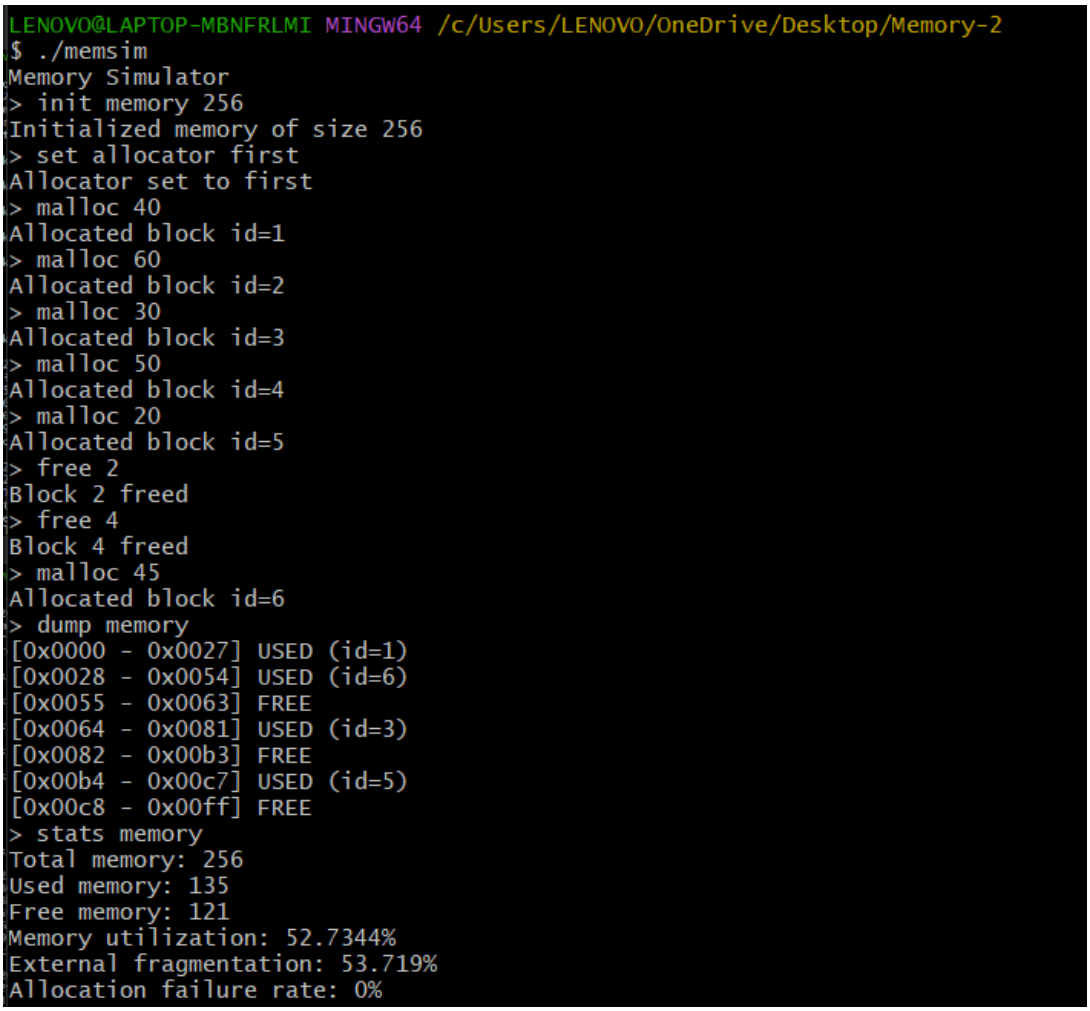

Figure 1: First Fit allocation: memory dump and statistics

First Fit provides fast allocation but leads to moderate external fragmentation over time.

5.2 Best Fit Allocation

5.2.1 Logic

Best Fit allocates the smallest free block that satisfies the request, thereby reducing leftover space.

5.2.2 Data Structure

- Doubly linked list
- Full scan to find smallest suitable block

5.2.3 Test Case

```
init memory 256
set allocator best
malloc 40
```

```
malloc 60
malloc 30
malloc 50
malloc 20
free 2
free 4
malloc 45
dump memory
stats memory
```

5.2.4 Observation

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ ./memsim
Memory Simulator
> init memory 256
Initialized memory of size 256
> set allocator best
Allocator set to best
> malloc 40
Allocated block id=1
> malloc 60
Allocated block id=2
> malloc 30
Allocated block id=3
> malloc 50
Allocated block id=4
> malloc 20
Allocated block id=5
> free 2
Block 2 freed
> free 4
Block 4 freed
> malloc 45
Allocated block id=6
> dump memory
[0x0000 - 0x0027] USED (id=1)
[0x0028 - 0x0063] FREE
[0x0064 - 0x0081] USED (id=3)
[0x0082 - 0x00ae] USED (id=6)
[0x00af - 0x00b3] FREE
[0x00b4 - 0x00c7] USED (id=5)
[0x00c8 - 0x00ff] FREE
> stats memory
Total memory: 256
Used memory: 135
Free memory: 121
Memory utilization: 52.7344%
External fragmentation: 50.4132%
Allocation failure rate: 0%
```

Figure 2: Best Fit allocation: memory dump and statistics

Best Fit reduces external fragmentation but has higher allocation overhead.

5.3 Worst Fit Allocation

5.3.1 Logic

Worst Fit allocates memory from the largest available free block.

5.3.2 Data Structure

- Doubly linked list

- Scan to locate the largest free block

5.3.3 Test Case

```
init memory 256
set allocator worst
malloc 40
malloc 60
malloc 30
malloc 50
malloc 20
free 2
free 4
malloc 45
dump memory
stats memory
```

5.3.4 Observation

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ ./memsim
Memory Simulator
> init memory 256
Initialized memory of size 256
> set allocator worst
Allocator set to worst
> malloc 40
Allocated block id=1
> malloc 60
Allocated block id=2
> malloc 30
Allocated block id=3
> malloc 50
Allocated block id=4
> malloc 20
Allocated block id=5
> free 2
Block 2 freed
> free 4
Block 4 freed
> malloc 45
Allocated block id=6
> dump memory
[0x0000 - 0x0027] USED (id=1)
[0x0028 - 0x0054] USED (id=6)
[0x0055 - 0x0063] FREE
[0x0064 - 0x0081] USED (id=3)
[0x0082 - 0x00b3] FREE
[0x00b4 - 0x00c7] USED (id=5)
[0x00c8 - 0x00ff] FREE
> stats memory
Total memory: 256
Used memory: 135
Free memory: 121
Memory utilization: 52.7344%
External fragmentation: 53.719%
Allocation failure rate: 0%
```

Figure 3: Worst Fit allocation: memory dump and statistics

Worst Fit often leads to high external fragmentation and poor memory utilization.

5.4 Buddy Allocation

5.4.1 Logic

Buddy Allocation divides memory into blocks of size 2^k. Allocation requests are rounded up to the nearest power of two. Adjacent free blocks (buddies) are merged during deallocation.

5.4.2 Data Structure

- Array of free lists indexed by block size

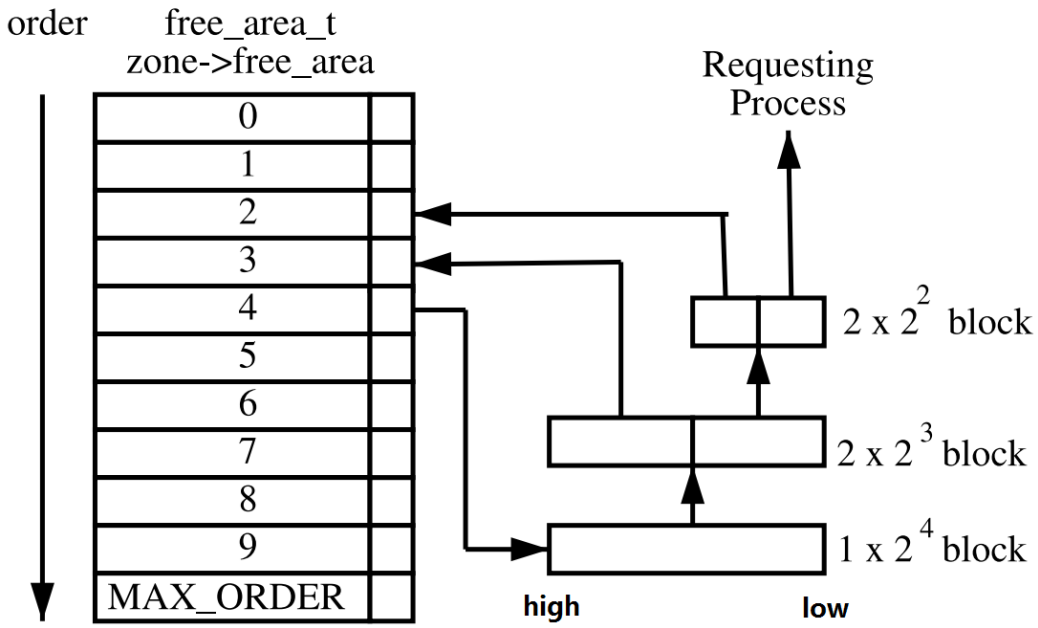

Figure 4: Structure of Buddy Allocator

- Each free list stores blocks of size 2^k

The buddy address is computed using:

$$\text{Buddy}(addr) = addr \oplus 2^k$$

5.4.3 Test Case

```
init memory 256
set allocator buddy
malloc 40
malloc 60
malloc 30
malloc 50
malloc 20
free 2
free 4
malloc 45
dump memory
stats memory
```

5.4.4 Observation

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ ./memsim
Memory Simulator
> init memory 256
Initialized memory of size 256
> set allocator buddy
Allocator set to buddy
> malloc 40
Allocated block id=1
> malloc 60
Allocated block id=2
> malloc 30
Allocated block id=3
> malloc 50
Allocated block id=4
> malloc 20
Allocated block id=5
> free 2
Block 2 freed
> free 4
Block 4 freed
> malloc 45
Allocated block id=6
> dump memory
Buddy Free Lists:
  size 64: 192
Allocated Blocks:
  id=6 addr=64 size=64 requested=45
  id=5 addr=160 size=32 requested=20
  id=3 addr=128 size=32 requested=30
  id=1 addr=0 size=64 requested=40
> stats memory
Total memory: 256
Used memory: 192
Memory utilization: 75%
External fragmentation: 0%
Internal fragmentation: 29.6875%
Allocation failure rate: 0%
```

Figure 5: Buddy allocation: memory dump and statistics

Buddy allocation minimizes external fragmentation but introduces internal fragmentation due to block rounding.

6 Comparative Analysis

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ make random
g++ -std=c++17 -Wall -Wextra -I include tests/random_test.cpp src/allocator/list_allocator.cpp src/a
llocator/buddy_allocator.cpp -o random_test
./random_test
Strategy: First Fit
Avg External Fragmentation: 82.8073%
Avg Utilization: 84.1309%
Avg Failure Rate: 49.076%

Strategy: Best Fit
Avg External Fragmentation: 75.9197%
Avg Utilization: 88.5156%
Avg Failure Rate: 50.4993%

Strategy: Worst Fit
Avg External Fragmentation: 86.48%
Avg Utilization: 75.1562%
Avg Failure Rate: 49.553%

Strategy: Buddy
Avg External Fragmentation: 63.8668%
Avg Internal Fragmentation: 22.4463%
Avg Utilization: 90.9375%
Avg Failure Rate: 51.2251%
```

Figure 6: Average Allocation Metrics

Strategy	Ext. Frag.	Int. Frag.	Utilization	Speed
First Fit	Medium	Low	Medium	Fast
Best Fit	Low	Low	High	Slow
Worst Fit	High	Low	Low	Slow
Buddy	Very Low	High	Medium	Very Fast

Best Fit achieves the best memory utilization, while First Fit provides a good trade-off between speed and fragmentation. Worst Fit performs poorly under most workloads. Buddy Allocation enables fast merging and predictable behavior at the cost of internal fragmentation.

7 Cache Memory Simulator

7.1 Introduction

Cache memory is a small, fast storage layer placed between the CPU and main memory. Its primary purpose is to reduce average memory access time by storing frequently accessed data closer to the processor.

This project implements a **multi-level cache simulator** with configurable replacement policies and detailed logging to study cache behavior, hit/miss patterns, and access penalties.

The simulator supports:

- Single and multi-level caching (L1, L2, L3)
- Multiple replacement policies
- Access-time and penalty-based performance analysis
- Detailed cache access logs

8 Cache Simulator Interface

8.1 Command-Line Interface

The following commands are supported:

```
init cache
set policy <lru|fifo|lfu>
access <address>
dump cache
stats cache
enable logs
disable logs
enable filelog
disable filelog
exit
```

8.2 Command Description

Command	Description
`init cache`	Initialize cache system (L1, L2, L3)
`set policy X`	Set replacement policy
`access A`	Access memory address A
`dump cache`	Display cache contents
`stats cache`	Display cache statistics
`enable logs`	Print cache logs to terminal
`enable filelog`	Save logs to `cache_log.txt`
`exit`	Exit simulator

9 Assumptions and Performance Model

9.1 Block and Addressing Assumptions

- Block size = 16 bytes
- Address is mapped to block number using:

$$\text{Block Number} = \left\lfloor \frac{\text{Address}}{16} \right\rfloor$$

9.2 Cache Hit Times and Penalties

The following access times are assumed:

Component	Access Time (cycles)
L1 Cache	1
L2 Cache	5
L3 Cache	20
Main Memory	100

9.3 Access Time Calculation

$$T_{\text{access}} = \begin{cases} 1 & \text{L1 hit} \\ 1 + 5 & \text{L2 hit} \\ 1 + 5 + 20 & \text{L3 hit} \\ 1 + 5 + 20 + 100 & \text{Memory access} \end{cases}$$

9.4 Performance Metrics

Hit Rate

$$\text{Hit Rate} = \frac{\text{Total Cache Hits}}{\text{Total Accesses}} \times 100$$

Miss Rate

$$\text{Miss Rate} = 100 - \text{Hit Rate}$$

Average Access Time

$$\text{Average Time} = \frac{\sum \text{Access Cycles}}{\text{Total Accesses}}$$

10 Cache Replacement Policies

10.1 Least Recently Used (LRU)

10.1.1 Overview

LRU replaces the block that has not been accessed for the longest time. It leverages temporal locality and is widely used in modern processors.

10.1.2 Data Structures Used

- Hash map for block lookup
- Timestamp or counter for last-used tracking

10.1.3 Test Case

```
init cache
set policy lru
access 0
access 16
access 32
access 0
dump cache
stats cache
```

10.1.4 Observation

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ ./memsim
Memory Simulator
> init cache
Cache initialized
> set policy lru
Cache policy set to lru
> access 0
> access 16
> access 32
> access 0
> dump cache
L1 Cache:
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
  block=0 freq=2 last_used=7
L2 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
L3 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
> stats cache
Total accesses: 4

L1 hits: 1  misses: 3  hit rate: 25%
L2 hits: 0  misses: 3  hit rate: 0%
L3 hits: 0  misses: 3  hit rate: 0%
Memory accesses: 3

Overall hit rate: 25%
Average access time: 94.75 cycles

Miss penalties:
  L1 -> L2: 5 cycles
  L2 -> L3: 20 cycles
  L3 -> Memory: 100 cycles
```

Figure 7: LRU : memory dump and statistics

LRU achieves a high hit rate for workloads with repeated access patterns.

10.2 First In First Out (FIFO)

10.2.1 Overview

FIFO evicts the block that was inserted earliest, regardless of access history.

10.2.2 Data Structures Used

- Queue for insertion order

10.2.3 Test Case

```
init cache
set policy fifo
access 0
access 16
access 32
access 48
dump cache
stats cache
```

10.2.4 Observation

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ ./memsim
Memory Simulator
> init cache
Cache initialized
> set policy fifo
Cache policy set to fifo
> access 0
> access 16
> access 32
> access 48
> dump cache
L1 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
  block=3 freq=1 last_used=8
L2 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
  block=3 freq=1 last_used=8
L3 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
  block=3 freq=1 last_used=8
> stats cache
Total accesses: 4

L1 hits: 0  misses: 4  hit rate: 0%
L2 hits: 0  misses: 4  hit rate: 0%
L3 hits: 0  misses: 4  hit rate: 0%
Memory accesses: 4

Overall hit rate: 0%
Average access time: 126 cycles

Miss penalties:
  L1 -> L2: 5 cycles
  L2 -> L3: 20 cycles
  L3 -> Memory: 100 cycles
```

Figure 8: FIFO : memory dump and statistics

FIFO is simple but may evict frequently used blocks, leading to lower hit rates.

10.3 Least Frequently Used (LFU)

10.3.1 Overview

LFU replaces the block with the lowest access frequency.

10.3.2 Data Structures Used

- Hash map with frequency counters

10.3.3 Test Case

```
init cache
set policy lfu
access 0
```

```
access 0
access 16
access 32
dump cache
stats cache
```

10.3.4 Observation

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ ./memsim
Memory Simulator
> init cache
Cache initialized
> set policy lfu
Cache policy set to lfu
> access 0
> access 0
> access 16
> access 32
> dump cache
L1 Cache:
  block=0 freq=2 last_used=3
  block=1 freq=1 last_used=5
  block=2 freq=1 last_used=7
L2 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
L3 Cache:
  block=0 freq=1 last_used=2
  block=1 freq=1 last_used=4
  block=2 freq=1 last_used=6
> stats cache
Total accesses: 4

L1 hits: 1  misses: 3  hit rate: 25%
L2 hits: 0  misses: 3  hit rate: 0%
L3 hits: 0  misses: 3  hit rate: 0%
Memory accesses: 3

Overall hit rate: 25%
Average access time: 94.75 cycles

Miss penalties:
  L1 -> L2: 5 cycles
  L2 -> L3: 20 cycles
  L3 -> Memory: 100 cycles
```

Figure 9: LFU : memory dump and statistics

LFU performs well when access frequency is stable but adapts slowly to changes.

11 Comparison of Replacement Policies

```
LENOVO@LAPTOP-MBNFRLMI MINGW64 /c/Users/LENOVO/OneDrive/Desktop/Memory-2
$ make cache_random
g++ -std=c++17 -Wall -Wextra -I include tests/cache_random_test.cpp src/cache/cache_level.cpp src/cache/cache_simulator.cpp -o cache_random_test
./cache_random_test
Policy: FIFO
Avg Overall Hit Rate: 25.59%
Avg Access Time: 97.569 cycles
Avg L1 Hit Rate: 6.18%
Avg L2 Hit Rate: 6.48%
Avg L3 Hit Rate: 12.93%
Avg Memory Accesses: 744.1

Policy: LRU
Avg Overall Hit Rate: 24.07%
Avg Access Time: 99.1935 cycles
Avg L1 Hit Rate: 6.53%
Avg L2 Hit Rate: 5.52%
Avg L3 Hit Rate: 12.02%
Avg Memory Accesses: 759.3

Policy: LFU
Avg Overall Hit Rate: 36.11%
Avg Access Time: 86.36 cycles
Avg L1 Hit Rate: 6.2%
Avg L2 Hit Rate: 9.9%
Avg L3 Hit Rate: 20.01%
Avg Memory Accesses: 638.9
```

Figure 10: Average Cache Replacement Policy Metrics

Policy	Hit Rate	Adaptability	Complexity
LRU	Medium to High	High	Medium
FIFO	Low to Medium	Low	Low
LFU	Medium to High	Low	High

LRU consistently provides the best balance between performance and adaptability, while FIFO trades performance for simplicity. LFU is effective only for stable workloads.

12 Multi-Level Caching

12.1 Theory

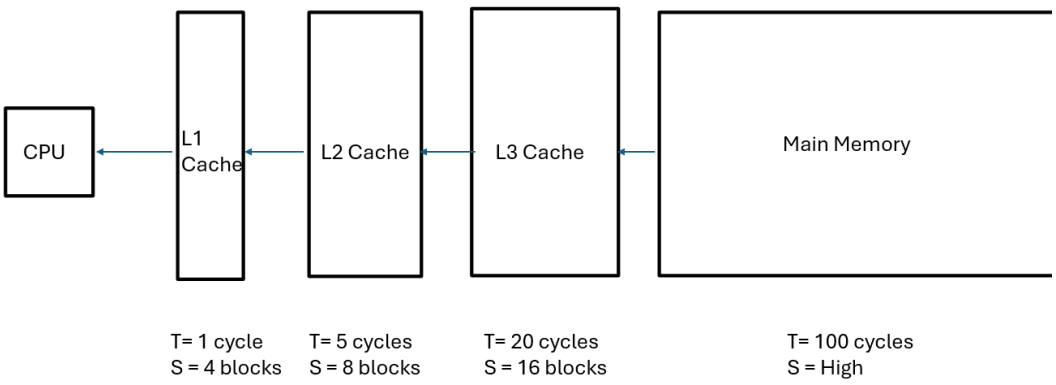

Figure 11: Cache levels

Modern processors use multiple cache levels:

- **L1**: Smallest and fastest
- **L2**: Larger, moderately fast
- **L3**: Largest, shared cache

This simulator implements an **inclusive cache hierarchy**, meaning all blocks in L1 are guaranteed to exist in L2 and L3.

12.2 Hierarchy Diagram

$$\text{CPU} \rightarrow \text{L1} \rightarrow \text{L2} \rightarrow \text{L3} \rightarrow \text{Main Memory}$$

12.3 L1 Cache Behavior

12.3.1 Test Case

```
init cache
enable filelog
access 0
access 0
access 1
```

12.3.2 Observation

cache_log.txt

```
ACCESS 0 (block 0):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 0 (block 0):
  L1 HIT
  Access time: 1 cycles
ACCESS 1 (block 0):
  L1 HIT
  Access time: 1 cycles

```

Figure 12: L1 Cache HIT Case

Repeated access leads to L1 hits with minimal access time (1 cycle).

12.4 L2 Cache Behavior

12.4.1 Test Case

```
init cache
enable filelog
access 0
access 16
access 32
access 48
access 64
access 0
```

12.4.2 Observation

cache_log.txt

```
ACCESS 0 (block 0):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 16 (block 1):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 32 (block 2):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 48 (block 3):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 64 (block 4):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 0 (block 0):
  L1 MISS
  L2 HIT
  Loaded into L1
  Access time: 6 cycles

```

Figure 13: L2 Cache HIT Case

Evicted L1 blocks may still reside in L2, reducing memory access penalties.

12.5 L3 Cache Behavior

12.5.1 Test Case

```
init cache
enable filelog
access 0
access 16
access 32
access 48
access 64
access 80
access 96
access 112
access 128
access 0
```

12.5.2 Observation

cache_log.txt

```
ACCESS 0 (block 0):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 16 (block 1):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 32 (block 2):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 48 (block 3):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 64 (block 4):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 80 (block 5):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
```

cache_log.txt

```
ACCESS 80 (block 5):
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 96 (block 6):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 112 (block 7):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 128 (block 8):
  L1 MISS
  L2 MISS
  L3 MISS
  MAIN MEMORY ACCESS
  Loaded into L3
  Loaded into L2
  Loaded into L1
  Access time: 126 cycles
ACCESS 0 (block 0):
  L1 MISS
  L2 MISS
  L3 HIT
  Loaded into L2
  Loaded into L1
  Access time: 26 cycles
```

Figure 14: L3 Cache

L3 acts as the final cache level before main memory, significantly lowering average access time.

13 Conclusion

The project successfully implements a comprehensive **Memory Management Simulator** that demonstrates both dynamic memory allocation strategies and multi-level cache simulation. The simulator provides an interactive command-line interface that allows users to experiment with different allocation algorithms (First Fit, Best Fit, Worst Fit, and Buddy Allocation) and cache replacement policies (LRU, FIFO, LFU).

Through detailed quantitative analysis, the project highlights the fundamental trade-offs in memory management:

- **Allocation Strategies**: The comparative analysis reveals that Best Fit achieves optimal memory utilization, while First Fit provides a balanced trade-off between speed and fragmentation. Buddy Allocation minimizes external fragmentation at the cost of internal fragmentation due to power-of-two rounding.
- **Cache Policies**: LRU consistently outperforms FIFO and LFU by leveraging temporal locality, making it the preferred choice for most modern processors. The multi-level cache hierarchy demonstrates how inclusive caching significantly reduces average memory access time.

The simulator's ability to visualize memory layouts and provide real-time statistics makes it an effective educational tool for understanding memory management concepts. The modular design allows for easy extension and integration of additional features.

Project Repository: The complete source code, documentation, and additional resources are available at: `https://github.com/AbhinavShresth/Memory-Management-Simulator`

14 Future Work and Suggestions

The following enhancements could further improve the simulator's capabilities:

14.1 Feature Enhancements

- **Graphical User Interface**: Develop a GUI to visualize memory allocation and cache behavior in real-time, making the simulator more intuitive for educational purposes.
- **Virtual Memory Simulation**: Implement virtual memory management with page tables, page replacement algorithms (FIFO, LRU, Optimal), and address translation mechanisms to simulate complete memory hierarchy from virtual addresses to physical memory.
- **Advanced Policies**: Implement additional replacement policies such as Clock, Random, and Adaptive Replacement Cache (ARC) for more comprehensive analysis.
- **Workload Analysis**: Add support for trace file input to simulate real-world memory access patterns and benchmark performance under realistic workloads.

14.2 Performance Improvements

- **Randomized Stress Testing**: Implement automated test suites with randomized allocation patterns to identify edge cases and validate robustness.
- **Parallel Simulation**: Support multi-threaded simulation for analyzing concurrent memory access patterns and cache coherence protocols.
- **Memory Profiling**: Add detailed profiling capabilities to track allocation patterns, identify memory leaks, and optimize allocation strategies.