README

Names: Nate Webb, Noah Bakayou
Class Account Usernames: cssc1467, cssc1409
Class Information: CS480-02-Sum202
Assignment Information: Assignment #3 – Memory Allocation Simulation

---

File Manifest

- `Makefile`
- `sim.cpp` (Main)
- `sim.h`
- `memoryManager.cpp`
- `memoryManager.h`
- `requestGen.cpp`
- `requestGen.h`
- `README`
- `sim` (Generated on compile)
- `*.o` (Generated on compile)
- `results.csv` (Generated on run)
- `a3Analysis/` (CSV + graphs directory)
  - `results.csv`
  - `fragments_vs_request_id.png`
  - `cumulative_fragments.png`
  - `nodes_traversed.png`
  - `graphResults.py`

Compile Instructions

Run:

```
make
```

Generates an executable named `sim`.

Clean build artifacts:

```
make clean
```

Operating Instructions

Run:

```
./sim
```

- No command-line arguments required.
- Simulates 10,000 requests and prints final statistics for First Fit and Best Fit.
- Produces `results.csv` for potential graphing analysis.

Novel/Significant Design Decisions

- Separate `Policy` enum (`FirstFit`, `BestFit`) for algorithm selection instead of duplicate classes.
- Automatic coalescing after every deallocation.
- One-time-per-process allocation enforced through permanent PID consumption in request generator.
- Fixed random seed (`1`) for reproducible results.
- Structured statistics collection for both per-request and per-allocation metrics.

Extra Features

- CSV output for analysis and graphing.
- Python-based graph generation with three visualization types.
- Comprehensive error handling with `try-catch` blocks.
- Real-time fragment counting and accumulation tracking.
- Automatic CSV file closure to ensure data integrity.

Known Deficiencies or Bugs

None.
Compiles and runs successfully on both local and Edoras environments, producing output matching specifications.

Lessons Learned

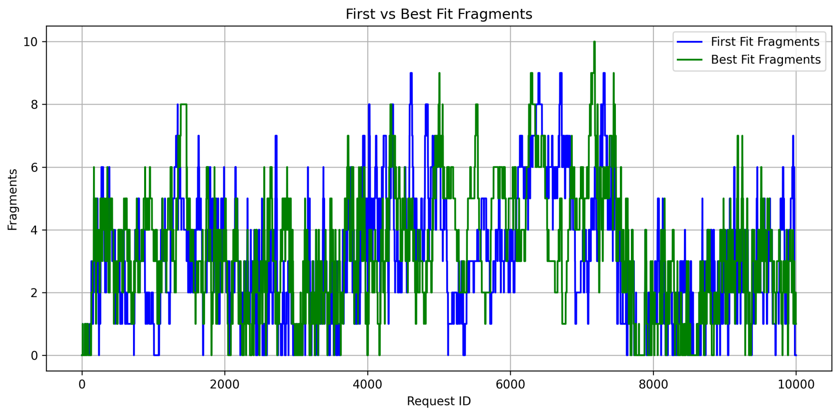
- Linked-list memory management requires careful pointer handling and coalescing logic.
- Performance precision depends on correct metric definitions.
- Theoretical advantages may not always translate to practical gains.
- Random workload generation significantly influences results.

- Exact specification compliance is critical for formatting and return semantics.

Graphs

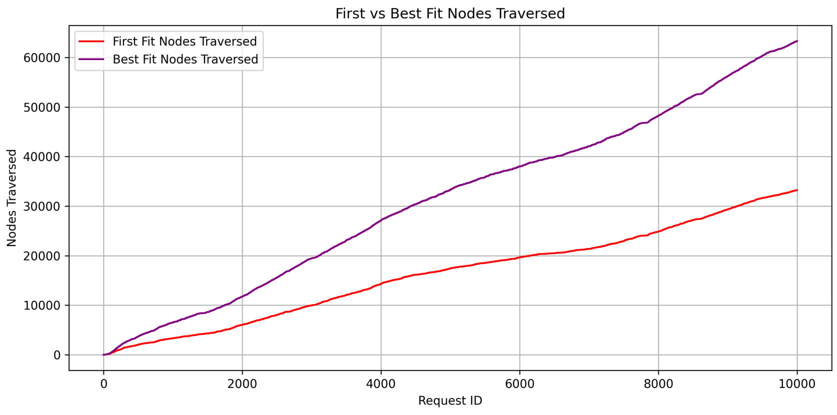
Located in `a3Analysis/`:

1. **Fragment Count Over Time** (`fragments_vs_request_id.png`)



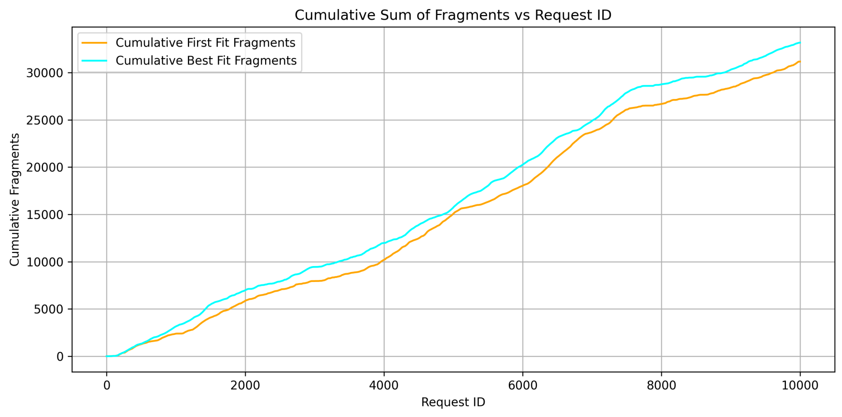
   Shows fragmentation evolution for both algorithms. Oscillation reflects allocation/deallocation cycles.

2. **Nodes Traversed Per Request** (`nodes_traversed.png`)



   Highlights computational cost. Best Fit traverses the entire free list; First Fit stops early.

3. **Cumulative Fragment Accumulation** (`cumulative_fragments.png`)



Shows net fragmentation trend despite coalescing.

Background

Algorithm Descriptions

- **First Fit:** Allocates in the first free block large enough. Splits if block is larger than needed.
- **Best Fit:** Chooses the smallest suitable free block. Minimizes waste but requires full list traversal.

Simulation System Design

- **Memory Model:** 256 KB total, divided into 128 × 2 KB units.
- **Data Structure:** Linked list of `memoryBlock` structures.
- **Coalescing:** Automatic after each deallocation.
- **Request Generation:** MT19937 RNG, fixed seed `1`.
- **Process ID Management:** Each PID allocated only once.

Configuration

- Total units: 128 (256KB ÷ 2KB)
- Requests: 10,000
- Request size: 3–10 units (uniform)
- Allocation/deallocation probability: 50/50
- Fragment: Free block of size 1 or 2 units
- Seed: `1`

Findings

Current Simulation Results

First Fit Allocation: - Average External Fragments Each Request: 3.116100 - Average Nodes Transversed Each Allocation: 6.631264 - Percentage Allocation Requests Denied Overall: 43.381913%

Best Fit Allocation: - Average External Fragments Each Request: 3.316900 - Average Nodes Transversed Each Allocation: 12.641046 - Percentage Allocation Requests Denied Overall: 42.862847%

What the Numbers Tell Us

The results were pretty surprising to me. I expected Best Fit to be way better at reducing fragmentation, but it actually created MORE fragments on average (3.32 vs 3.12). That was definitely not what I thought would happen based on what we learned in class about Best Fit being more "space efficient."

The performance difference is huge though. First Fit only needed to check about 6.6 nodes per allocation while Best Fit had to check 12.6 nodes. That's almost double the work. This makes sense though because Best Fit has to look at every single free block to find the "best" one, while First Fit just stops at the first block that works.

Both algorithms failed a decent amount, over 40% of allocation requests got denied. This shows our simulation is pretty harsh on memory. The 10,000 unique process IDs rule means once a process gets deallocated, that ID is gone forever, so we're constantly running out of available processes and memory gets really fragmented.

What This Means

Looking at the graphs, you can see the fragment count bouncing up and down throughout the simulation. This happens because we're constantly allocating and deallocating memory. The cumulative graph shows that even with coalescing (merging free blocks), fragments just keep building up over time.

The nodes traversed graph clearly shows Best Fit doing way more work. The way I thought about it is like the difference between grabbing the first parking spot you see versus driving around the whole parking lot to find the perfect spot. I learned that sometimes "good enough" is actually the better approach.

Why First Fit Won

I think First Fit performed better because our simulation creates a really stressful memory environment. When memory is tight and you're under pressure, speed matters more than finding the absolute perfect fit. Best Fit's strategy of being picky actually backfired because it creates tiny leftover fragments that are too small to be useful.

This taught me that theoretical “optimal” doesn’t always mean practical “better.” In real systems, you have to consider the whole system, not just space efficiency but also time efficiency and how the algorithm behaves under stress.

Conclusions Based on This Data Set

1. Under high memory pressure, First Fit’s speed advantage outweighs Best Fit’s theoretical space efficiency
2. The “better” algorithm depends heavily on workload characteristics and memory pressure levels
3. Simple metrics like “smallest fit” don’t always predict real-world fragmentation behavior under stress
4. Our one-time PID allocation rule creates an increasingly difficult environment that may not reflect typical OS behavior

---

Recommendations for Future Runs

1. **Variable Memory Pressure** – Test with 64, 256, 512 units.
2. **Request Size Distribution Studies** – Skewed, bimodal, real-world profiles.
3. **Alternative Strategies** – Next Fit, Worst Fit, Quick Fit.
4. **Dynamic Workloads** – Bursty patterns, process lifetimes, application-specific.
5. **Advanced Metrics** – Time-to-fragmentation, utilization efficiency, largest block tracking, request wait times.
6. **Policy Variations** – PID reuse, allocation priorities, periodic compaction.