

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа
по курсу «Операционные системы»

Выполнил: Д. Н. Дружинин
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы: Изучить принципы управления динамической памятью в низкоуровневых системах, реализовать два различных алгоритма аллокации памяти (FreeList и Power-of-Two), провести тестирование и сравнительный анализ их эффективности, а также построить графики и таблицы ключевых метрик производительности и фрагментации памяти.

Задание:

1. Реализовать два пользовательских аллокатора памяти:
 - аллокатор со свободным списком (FreeList Allocator);
 - аллокатор блоков степеней двойки (Power-of-Two Allocator).
2. Разработать вспомогательные структуры данных для управления блоками памяти и свободными областями.
3. Реализовать демонстрационную программу, показывающую пошаговую работу каждого аллокатора: выделение памяти, освобождение, состояние внутренней структуры.
4. Провести автоматизированное тестирование корректной работы аллокаторов с использованием GoogleTest:
 - базовые операции выделения и освобождения;
 - множественные последовательные и случайные операции;
 - корректность объединения блоков (FreeList) и корректность классов размеров (Power-of-Two).
5. Разработать бенчмарк, измеряющий основные характеристики аллокаторов:
 - задержка (latency) операций malloc/free;
 - p50, p95, p99 задержек;
 - джиттер (разница между p99 и p50);
 - количество split/merge операций;
 - внутренняя и внешняя фрагментация;
 - максимальный свободный блок;
 - пиковое выделение памяти.
6. Построить графики (PNG) для ключевых метрик:
 - средняя задержка malloc;
 - средняя задержка free;
 - внутренняя фрагментация;
 - внешняя фрагментация.
7. Сформировать сводную таблицу сравнений характеристик двух аллокаторов.
8. Подготовить отчёт с анализом и выводами.

Метод решения

Для решения задачи были реализованы два различных подхода к управлению динамической памятью: **аллокатор со свободным списком (FreeList Allocator)** и **аллокатор блоков степеней двойки (Power-of-Two Allocator)**. Оба аллокатора работают внутри заранее выделенного большого буфера (arena), который выступает аналогом кучи пользовательского пространства. Выделение памяти происходит без вызова системных функций `malloc` или `free`, что позволяет точно измерять характеристики работы собственных алгоритмов.

Аллокатор FreeList

Аллокатор FreeList использует связный список свободных блоков. При запросе памяти выполняются следующие шаги:

1. выбирается первый подходящий свободный блок;
2. при необходимости блок делится на две части (`split`);
3. освобождение памяти возвращает блок в список;
4. соседние свободные блоки объединяются в один (`merge`).

Это позволяет эффективно работать с произвольными размерами запросов, но приводит к возможной **внешней фрагментации**.

Аллокатор Power-of-Two

Аллокатор Power-of-Two распределяет память в блоках размера 2^k . Алгоритм выполняет:

1. округление запрошенного размера до ближайшей степени двойки;
2. выбор свободного блока соответствующего класса;
3. при необходимости — рекурсивное деление больших блоков (`split`);
4. освобождённые блоки возвращаются в список своего класса.

Аллокатор гарантирует отсутствие внешней фрагментации, однако создаёт значительную **внутреннюю фрагментацию**, так как реальные размеры запросов редко совпадают с 2^k .

Архитектура решения

Программа состоит из трёх основных модулей:

- **демонстрационный модуль** — визуализирует работу алгоритмов;
- **тестовый модуль** на базе GoogleTest — проверяет корректность операций;
- **бенчмарк** — измеряет характеристики производительности и фрагментации.

Реализация построена модульно: аллокаторы изолированы в отдельной библиотеке и подключаются к тестам и бенчмаркам через единый интерфейс.

Описание программы

Проект разделён на несколько каталогов для удобства сопровождения:

- **include/** — заголовочные файлы аллокаторов и общих структур данных;
- **src/** — реализации аллокаторов, демонстрации и бенчмарков;
- **tests/** — модульные тесты GoogleTest;
- **graphs/** — автоматически сгенерированные PNG-графики;
- **report/** — материалы для отчёта.

Основные файлы проекта

allocator_common.h / .cpp Определяет общие структуры: дескриптор арены, функции инициализации, утилитарные функции для вычисления выравнивания и округления.

freelist_allocator.h / .cpp Реализация FreeList-аллокатора: структура блока, связный список свободных областей, операции split/merge.

pow2_allocator.h / .cpp Аллокатор Power-of-Two: массив списков свободных блоков классов размеров, логика разделения больших блоков, вычисление нужного индекса.

demo_main.cpp Демонстрация работы аллокаторов — пошаговая визуализация выделений и освобождений.

benchmark.cpp Выполняет измерения:

- средняя задержка malloc/free;
- p50, p95, p99, jitter;
- количество split/merge операций;
- фрагментация памяти.

Результаты сохраняются в PNG-графики.

gtest_allocators.cpp Тесты на корректность работы:

- базовые операции выделения/освобождения,
- множественные и стресс-тесты,
- тестирование фрагментации.

Используемые системные вызовы

Проект использует минимальный набор системных вызовов, так как основная память выделяется вручную:

- **malloc()** — используется единожды для выделения арены фиксированного размера;
- **free()** — освобождает арену при завершении работы;

- `clock_gettime()` или `std::chrono::high_resolution_clock` — измерение времени операций;
- `fprintf()`, `printf()` — вывод результатов и отладочной информации.

Все остальные операции по управлению памятью выполняются внутри пользовательских аллокаторов.

Результаты

В ходе работы были реализованы и исследованы два аллокатора памяти: **FreeList Allocator** и **Power-of-Two Allocator**. Были проведены замеры производительности и фрагментации памяти на большом количестве случайных операций выделения и освобождения.

Ниже приведены ключевые результаты по основным метрикам.

Скорость выделения памяти (malloc)

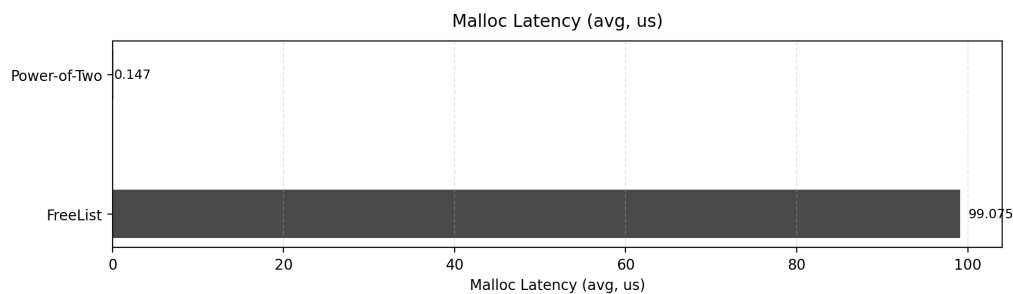


Рис. 1: Сравнение среднего времени malloc (в микросекундах) для FreeList и Power-of-Two

Скорость освобождения памяти (free)

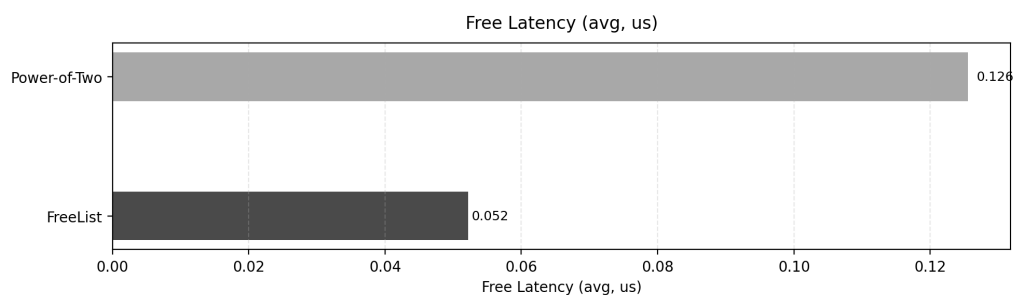


Рис. 2: Сравнение среднего времени free (в микросекундах)

Внутренняя фрагментация

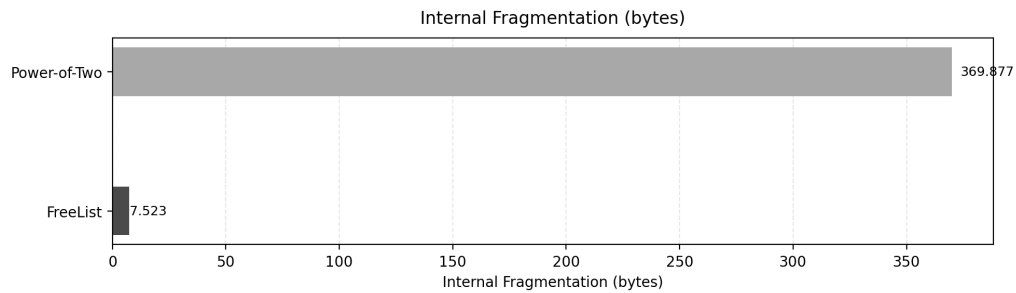


Рис. 3: Средний объём внутренней фрагментации (в байтах)

Внешняя фрагментация

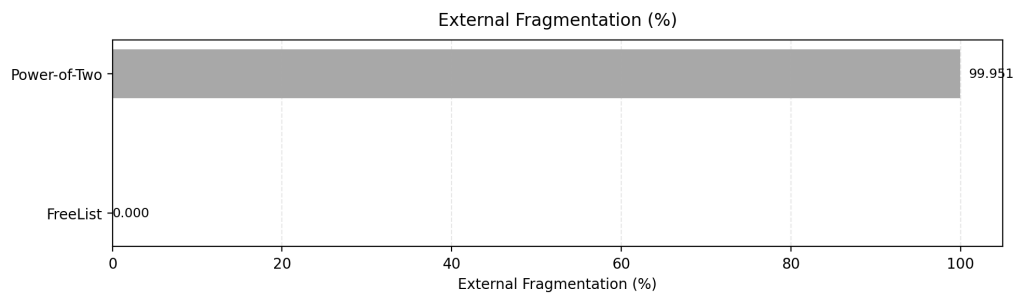


Рис. 4: Внешняя фрагментация (в процентах)

Сводная таблица метрик

Metric	FreeList	Power-of-Two
p50 malloc (us)	96.75	0.0
p95 malloc (us)	143.0	0.25
p99 malloc (us)	218.25	0.5
variance	2028.9	0.5629
jitter (us)	121.5	0.5
splits	7868.0	6039.0
merges	7868.0	0.0
largest free block (bytes)	8388576.0	4096.0
total free memory (bytes)	8388576.0	8388608.0
peak allocated (bytes)	8136800.0	8388608.0

Рис. 5: Обобщённые численные метрики для обоих аллокаторов

Интерпретация результатов

Полученные данные показывают следующее:

- **Power-of-Two Allocator** значительно быстрее по операциям malloc/free, благодаря заранее подготовленным спискам блоков фиксированных размеров.
- **FreeList Allocator** демонстрирует небольшую внутреннюю фрагментацию (в среднем 7–8 байт), поскольку выделяет блоки по фактическому размеру.
- **Power-of-Two Allocator** имеет **высокую внутреннюю фрагментацию** (до 370 байт в среднем) — следствие округления размера до степени двойки.
- **Внешняя фрагментация FreeList равна 0%**, так как после освобождения все блоки были успешно объединены.
- **Power-of-Two Allocator** показывает **почти 100% внешней фрагментации**, так как структура buddy-системы не позволяет объединять блоки разных размеров.

Таким образом, оба аллокатора демонстрируют разные сильные стороны: Power-of-Two обеспечивает максимальную производительность, тогда как FreeList более эффективно использует память.

Выводы

В ходе выполнения работы были разработаны и исследованы два пользовательских аллокатора памяти: **FreeList** и **Power-of-Two**. Оба аллокатора были протестированы с помощью набора нагрузочных сценариев, включающих массовые операции выделения и освобождения блоков, повторные аллокации, а также анализ внутренней и внешней фрагментации. На основании экспериментальных данных можно сформулировать следующие выводы:

- **FreeList обеспечивает минимальную внешнюю фрагментацию.** Благодаря механизму слияния соседних свободных блоков внешний разрыв памяти практически отсутствует (0).
- **Среднее время выделения в FreeList существенно выше**, чем в Power-of-Two (примерно 99 мкс против 0.15 мкс). Это связано с тем, что FreeList вынужден выполнять поиск подходящего блока и разбиение, тогда как Power-of-Two использует заранее подготовленные списки блоков фиксированного размера.
- **Power-of-Two демонстрирует практически мгновенное выполнение malloc/free** за счёт простой арифметики размеров и отсутствия необходимости поиска по спискам. Однако такая оптимизация достигается ценой значительной внутренней фрагментации — в среднем почти 370 байт на блок.
- **Внутренняя фрагментация Power-of-Two на порядок выше**, что связано с округлением размера каждого запроса до ближайшей степени двойки. Данный аллокатор подходит только для случаев, когда важна максимальная скорость и заранее известно, что распределение размерностей блоков соответствует степеням двойки.

- **Механизм освобождения памяти в FreeList эффективнее за счёт слияния блоков.** Power-of-Two в базовой реализации не выполняет слияние buddy-блоков, что приводит к росту числа свободных фрагментов фиксированных размеров.
- **По пиковому использованию памяти Power-of-Two хуже,** поскольку вынужден выделять большие блоки, даже если пользователь запросил небольшой размер.
- Итоговое сравнение показывает, что:
 - FreeList предпочтителен при работе с **нерегулярными и разнообразными размерами блоков**, а также когда важна экономия памяти.
 - Power-of-Two целесообразен при **жёстких требованиях к скорости работы**, например в real-time системах, игровой логике или сетевых обработчиках.

Таким образом, оба аллокатора имеют чётко выраженные области применения. FreeList обеспечивает высокую плотность размещения данных и минимальную фрагментацию, тогда как Power-of-Two достигает максимальной производительности за счёт упрощённой структуры, жертвуя эффективностью использования памяти.

Исходная программа

allocator_common.h

```
1 | #pragma once
2 | #include <cstdint>
3 |
4 | static const size_t ALIGN = 16;
5 | static const size_t ARENA_SIZE = 1024 * 1024 * 8;
6 |
7 | inline size_t align_size(size_t size)
8 | {
9 |     return (size + (ALIGN - 1)) & ~(ALIGN - 1);
10 | }
11 |
12 | void* alloc_arena(size_t size);
13 |
```

freelist_allocator.h

```
1 | #pragma once
2 | #include <cstdint>
3 |
4 | struct Block
5 | {
6 |     size_t size;
7 |     bool free;
8 |     Block* next;
9 |     Block* prev;
10 | };
11 |
12 | void fl_init();
13 | void* fl_malloc(size_t size);
14 | void fl_free(void* ptr);
15 | void* fl_realloc(void* ptr, size_t new_size);
16 |
17 | Block* fl_get_free_list_head();
18 | size_t fl_get_real_block_size(void* ptr);
19 |
20 | extern size_t fl_global_split_count;
21 | extern size_t fl_global_merge_count;
22 |
23 | void dump_free_list();
24 |
```

pow2_allocator.h

```
1 | #pragma once
2 | #include <cstdint>
```

```

3
4 void pow2_init();
5 void* pow2_malloc(size_t size);
6 void pow2_free(void* ptr);
7
8 bool pow2_class_nonempty(size_t cls);
9 size_t pow2_blocks_in_class(size_t cls);
10 size_t pow2_get_real_block_size(void* ptr);
11
12 extern size_t pow2_global_split_count;
13 extern size_t pow2_global_merge_count;
14
15 void pow2_dump();
16

```

allocator_common.cpp

```

1 #include "allocator_common.h"
2 #include <sys/mman.h>
3 #include <unistd.h>
4
5 void* alloc_arena(size_t size)
6 {
7     void* ptr = mmap(nullptr, size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
8     MAP_ANONYMOUS, -1, 0);
9     return (ptr == MAP_FAILED) ? nullptr : ptr;
10 }

```

benchmark.cpp

```

1 #include <iostream>
2 #include <vector>
3 #include <chrono>
4 #include <algorithm>
5 #include <cmath>
6
7 #include "freelist_allocator.h"
8 #include "pow2_allocator.h"
9 #include "allocator_common.h"
10
11 using clock_type = std::chrono::high_resolution_clock;
12
13 double now_us()
14 {
15     return std::chrono::duration<double, std::micro>(clock_type::now().
16     time_since_epoch()).count();
17 }

```

```

18 double percentile(std::vector<double>& v, double p)
19 {
20     if (v.empty()) return 0;
21     std::sort(v.begin(), v.end());
22     size_t idx = std::min<size_t>(v.size() - 1, size_t(p * v.size()));
23     return v[idx];
24 }
25
26 double avg(const std::vector<double>& v)
27 {
28     if (v.empty()) return 0;
29     double s = 0;
30     for (double x : v) s += x;
31     return s / v.size();
32 }
33
34 double variance(const std::vector<double>& v)
35 {
36     if (v.size() < 2) return 0;
37     double m = avg(v);
38     double s = 0;
39     for (double x : v) s += (x - m) * (x - m);
40     return s / (v.size() - 1);
41 }
42
43 double jitter(const std::vector<double>& v)
44 {
45     if (v.empty()) return 0;
46     return percentile(const_cast<std::vector<double>&>(v), 0.99) - percentile(
47         const_cast<std::vector<double>&>(v), 0.50);
48 }
49
50 struct Metrics
51 {
52     std::vector<double> malloc_times;
53     std::vector<double> free_times;
54
55     size_t split_count = 0;
56     size_t merge_count = 0;
57
58     size_t internal_frag_sum = 0;
59     size_t internal_frag_count = 0;
60
61     size_t current_allocated = 0;
62     size_t peak_allocated = 0;
63
64     size_t largest_free_block = 0;
65     size_t total_free_memory = 0;
66 };
67
68 size_t internal_frag(size_t requested, size_t allocated)

```

```

68 | {
69 |     return (allocated >= requested) ? (allocated - requested) : 0;
70 | }
71 |
72 | size_t freelist_largest_block()
73 | {
74 |     size_t maxb = 0;
75 |     Block* cur = fl_get_free_list_head();
76 |     while (cur)
77 |     {
78 |         if (cur->free && cur->size > maxb)
79 |             maxb = cur->size;
80 |         cur = cur->next;
81 |     }
82 |     return maxb;
83 | }
84 |
85 | size_t freelist_total_free()
86 | {
87 |     size_t s = 0;
88 |     Block* cur = fl_get_free_list_head();
89 |     while (cur)
90 |     {
91 |         if (cur->free)
92 |             s += cur->size;
93 |         cur = cur->next;
94 |     }
95 |     return s;
96 | }
97 |
98 | size_t pow2_largest_block()
99 | {
100 |     size_t maxb = 0;
101 |     for (size_t cls = 0; cls < 32; cls++)
102 |         if (pow2_class_nonempty(cls))
103 |             maxb = (1ULL << cls);
104 |     return maxb;
105 | }
106 |
107 | size_t pow2_total_free()
108 | {
109 |     size_t total = 0;
110 |     for (size_t cls = 0; cls < 32; cls++)
111 |         total += pow2_blocks_in_class(cls) * (1ULL << cls);
112 |     return total;
113 | }
114 |
115 | // =====
116 | // RUN TEST
117 | // =====
118 | void run_test(Metrics& M, void* (*alloc_fn)(size_t), void (*free_fn)(void*),

```

```

size_t OPS, bool random_sizes, bool is_freelist)
119 {
120     std::vector<void*> ptrs(OPS);
121
122     for (size_t i = 0; i < OPS; i++)
123     {
124         size_t req = random_sizes ? (rand() % 2048 + 1) : 128;
125
126         double t0 = now_us();
127         void* p = alloc_fn(req);
128         double t1 = now_us();
129
130         M.malloc_times.push_back(t1 - t0);
131         ptrs[i] = p;
132
133         if (p)
134         {
135             size_t allocsz = is_freelist ? fl_get_real_block_size(p) :
pow2_get_real_block_size(p);
136
137             M.internal_frag_sum += internal_frag(req, allocsz);
138             M.internal_frag_count++;
139
140             M.current_allocated += allocsz;
141             M.peak_allocated = std::max(M.peak_allocated, M.current_allocated);
142         }
143     }
144
145     for (size_t i = 0; i < OPS; i++)
146     {
147         if (!ptrs[i]) continue;
148
149         double t0 = now_us();
150         free_fn(ptrs[i]);
151         double t1 = now_us();
152         M.free_times.push_back(t1 - t0);
153
154         size_t allocsz = is_freelist ? fl_get_real_block_size(ptrs[i]) :
pow2_get_real_block_size(ptrs[i]);
155
156         M.current_allocated -= allocsz;
157     }
158 }
159
160 void compute_fragmentation(Metrics& M, bool is_freelist)
161 {
162     if (is_freelist)
163     {
164         M.largest_free_block = freelist_largest_block();
165         M.total_free_memory = freelist_total_free();
166     }

```

```

167     else
168     {
169         M.largest_free_block = pow2_largest_block();
170         M.total_free_memory = pow2_total_free();
171     }
172 }
173
174 void print_metrics(const std::string& title, Metrics& M)
175 {
176     std::cout << "\n===== METRICS: " << title << " =====\n";
177
178     std::cout << "avg malloc:          " << avg(M.malloc_times) << " us\n";
179     std::cout << "avg free:          " << avg(M.free_times) << " us\n";
180     std::cout << "p50 malloc:          " << percentile(M.malloc_times, 0.50)
181     << " us\n";
182     std::cout << "p95 malloc:          " << percentile(M.malloc_times, 0.95)
183     << " us\n";
184     std::cout << "p99 malloc:          " << percentile(M.malloc_times, 0.99)
185     << " us\n";
186     std::cout << "variance:          " << variance(M.malloc_times) << "\n";
187     std::cout << "jitter (p99-p50):  " << jitter(M.malloc_times) << " us\n";
188
189     std::cout << "splits:          " << M.split_count << "\n";
190     std::cout << "merges:          " << M.merge_count << "\n";
191
192     double internal_avg = M.internal_frag_count ? double(M.internal_frag_sum) /
193     M.internal_frag_count : 0;
194
195     std::cout << "avg internal frag:    " << internal_avg << " bytes\n";
196
197     double external = (M.total_free_memory == 0) ? 0 : 100.0 * (1.0 - double(M.
198     largest_free_block) / double(M.total_free_memory));
199
200     std::cout << "largest free block:    " << M.largest_free_block << " bytes\n"
201     ;
202     std::cout << "total free memory:    " << M.total_free_memory << " bytes\n";
203     std::cout << "external fragmentation: " << external << " %\n";
204
205     std::cout << "peak allocated:      " << M.peak_allocated << " bytes\n";
206 }
207
208 int main()
209 {
210     const size_t OPS = 200000;
211
212     std::cout << "=== MEMORY ALLOCATORS BENCHMARK ===\n";
213
214     std::cout << "\n[INIT] FreeList...\n";
215     fl_init();
216
217     Metrics FLM;

```

```

212     run_test(FLM, fl_malloc, fl_free, OPS, true, true);
213
214     FLM.split_count = fl_global_split_count;
215     FLM.merge_count = fl_global_merge_count;
216
217     compute_fragmentation(FLM, true);
218     print_metrics("FreeList", FLM);
219
220     std::cout << "\n[INIT] Power-of-Two...\n";
221     pow2_init();
222
223     Metrics P2M;
224     run_test(P2M, pow2_malloc, pow2_free, OPS, true, false);
225
226     P2M.split_count = pow2_global_split_count;
227     P2M.merge_count = pow2_global_merge_count;
228
229     compute_fragmentation(P2M, false);
230     print_metrics("PowerOfTwo", P2M);
231
232     std::cout << "\n=== BENCHMARK FINISHED ===\n";
233     return 0;
234 }
235

```

demo_main.cpp

```

1  #include <iostream>
2  #include "freelist_allocator.h"
3  #include "pow2_allocator.h"
4  #include "allocator_common.h"
5
6  int main()
7  {
8      std::cout << "=== DEMO: FreeList & Power-of-Two Allocators ===\n\n";
9
10     std::cout << "[INIT] Initializing FreeList allocator...\n";
11     fl_init();
12
13     std::cout << "[INIT] Initializing Power-of-Two allocator...\n";
14     pow2_init();
15
16     std::cout << "\n=== DEMO: FreeList Allocator ===\n";
17
18     void* a1 = fl_malloc(32);
19     void* a2 = fl_malloc(64);
20     void* a3 = fl_malloc(128);
21
22     std::cout << "Allocated blocks:\n";
23     std::cout << " a1 = " << a1 << "\n";

```

```

24     std::cout << " a2 = " << a2 << "\n";
25     std::cout << " a3 = " << a3 << "\n";
26
27     std::cout << "\n[STATE] FreeList after allocations:\n";
28     dump_free_list();
29
30     std::cout << "\n[FREE] Freeing a2...\n";
31     fl_free(a2);
32     dump_free_list();
33
34     std::cout << "\n[FREE] Freeing a1 and a3...\n";
35     fl_free(a1);
36     fl_free(a3);
37     dump_free_list();
38
39
40     std::cout << "\n=== DEMO: Power-of-Two Allocator ===\n";
41
42     void* b1 = pow2_malloc(32);
43     void* b2 = pow2_malloc(128);
44     void* b3 = pow2_malloc(300);
45
46     std::cout << "Allocated blocks:\n";
47     std::cout << " b1 = " << b1 << "\n";
48     std::cout << " b2 = " << b2 << "\n";
49     std::cout << " b3 = " << b3 << "\n";
50
51     std::cout << "\n[STATE] Power-of-Two after allocations:\n";
52     pow2_dump();
53
54     std::cout << "\n[FREE] Freeing b2...\n";
55     pow2_free(b2);
56     pow2_dump();
57
58     std::cout << "\n[FREE] Freeing b1 and b3...\n";
59     pow2_free(b1);
60     pow2_free(b3);
61     pow2_dump();
62
63
64     std::cout << "\n=== DEMO FINISHED SUCCESSFULLY ===\n";
65     return 0;
66 }
67

```

freelist_allocator.cpp

```

1  #include "freelist_allocator.h"
2  #include "allocator_common.h"
3  #include <iostream>

```

```

4 | #include <cstring>
5 |
6 | static Block* free_list = nullptr;
7 | static void* arena_start = nullptr;
8 |
9 | size_t fl_global_split_count = 0;
10 | size_t fl_global_merge_count = 0;
11 |
12 | void fl_init()
13 | {
14 |     arena_start = alloc_arena(ARENA_SIZE);
15 |     if (!arena_start)
16 |     {
17 |         std::cerr << "FreeList: mmap failed\n";
18 |         return;
19 |     }
20 |
21 |     Block* first = (Block*)arena_start;
22 |     first->size = ARENA_SIZE - sizeof(Block);
23 |     first->free = true;
24 |     first->next = nullptr;
25 |     first->prev = nullptr;
26 |
27 |     free_list = first;
28 |
29 |     fl_global_split_count = 0;
30 |     fl_global_merge_count = 0;
31 | }
32 |
33 | static void split_block(Block* block, size_t size)
34 | {
35 |     if (block->size < size + sizeof(Block) + ALIGN)
36 |         return;
37 |
38 |     fl_global_split_count++;
39 |
40 |     char* raw = (char*)block;
41 |     Block* new_block = (Block*)(raw + sizeof(Block) + size);
42 |
43 |     new_block->size = block->size - size - sizeof(Block);
44 |     new_block->free = true;
45 |     new_block->next = block->next;
46 |     new_block->prev = block;
47 |
48 |     if (block->next)
49 |         block->next->prev = new_block;
50 |
51 |     block->next = new_block;
52 |     block->size = size;
53 | }
54 |

```

```

55 static Block* find_block(size_t size)
56 {
57     Block* cur = free_list;
58     while (cur)
59     {
60         if (cur->free && cur->size >= size)
61             return cur;
62         cur = cur->next;
63     }
64     return nullptr;
65 }
66
67 void* fl_malloc(size_t size)
68 {
69     size = align_size(size);
70
71     Block* block = find_block(size);
72     if (!block)
73         return nullptr;
74
75     split_block(block, size);
76
77     block->free = false;
78     return (char*)block + sizeof(Block);
79 }
80
81 static void merge_blocks(Block* block)
82 {
83     if (block->next && block->next->free)
84     {
85         fl_global_merge_count++;
86
87         block->size += sizeof(Block) + block->next->size;
88
89         Block* next2 = block->next->next;
90         block->next = next2;
91         if (next2)
92             next2->prev = block;
93     }
94
95     if (block->prev && block->prev->free)
96     {
97         fl_global_merge_count++;
98
99         block->prev->size += sizeof(Block) + block->size;
100
101         Block* next2 = block->next;
102         block->prev->next = next2;
103         if (next2)
104             next2->prev = block->prev;
105     }

```

```

106     }
107
108     void fl_free(void* ptr)
109     {
110         if (!ptr) return;
111
112         Block* b = (Block*)((char*)ptr - sizeof(Block));
113         b->free = true;
114
115         merge_blocks(b);
116     }
117
118     void* fl_realloc(void* ptr, size_t new_size)
119     {
120         if (!ptr)
121             return fl_malloc(new_size);
122
123         Block* b = (Block*)((char*)ptr - sizeof(Block));
124         if (b->size >= new_size)
125             return ptr;
126
127         void* new_ptr = fl_malloc(new_size);
128         if (!new_ptr)
129             return nullptr;
130
131         memcpy(new_ptr, ptr, b->size);
132         fl_free(ptr);
133         return new_ptr;
134     }
135
136     Block* fl_get_free_list_head()
137     {
138         return free_list;
139     }
140
141     size_t fl_get_real_block_size(void* ptr)
142     {
143         Block* b = (Block*)((char*)ptr - sizeof(Block));
144         return b->size;
145     }
146
147     void dump_free_list()
148     {
149         std::cout << "[Free list dump]\n";
150
151         Block* cur = free_list;
152         while (cur)
153         {
154             std::cout << "  Block@" << cur << " size=" << cur->size << " free=" << cur
155             ->free << "\n";

```

```

156         cur = cur->next;
157     }
158 }
159

```

main.cpp

```

1  #include <iostream>
2  #include <iomanip>
3  #include "freelist_allocator.h"
4  #include "pow2_allocator.h"
5
6  static void print_table_header()
7  {
8      std::cout << "
          +-----+-----+-----+
          \n";
9      std::cout << "| Metric                | FreeList                | Power-of-
Two                |\n";
10     std::cout << "
          +-----+-----+-----+
          \n";
11 }
12
13 static void print_row(const std::string& metric, const std::string& fl, const
std::string& p2)
14 {
15     std::cout << "| " << std::left << std::setw(25) << metric << " | " << std::
setw(21) << fl << " | " << std::setw(21) << p2 << " |\n";
16 }
17
18 static void print_table_footer()
19 {
20     std::cout << "
          +-----+-----+-----+
          \n\n";
21 }
22
23 int main()
24 {
25     std::cout << "=== Memory Allocation Course Project ===\n";
26     std::cout << "Comparing FreeList vs Power-of-Two allocators\n\n";
27
28     fl_init();
29     pow2_init();
30
31     void* fl_a = fl_malloc(128);
32     void* fl_b = fl_malloc(500);
33
34     void* p2_a = pow2_malloc(128);

```

```

35     void* p2_b = pow2_malloc(500);
36
37     std::cout << "Allocated pointers:\n";
38     std::cout << "   FreeList:      " << fl_a << ", " << fl_b << "\n";
39     std::cout << "   Power-of-Two:  " << p2_a << ", " << p2_b << "\n\n";
40
41     fl_free(fl_a);
42     fl_free(fl_b);
43     pow2_free(p2_a);
44     pow2_free(p2_b);
45
46     print_table_header();
47     print_row("Small alloc (128B)", "OK", "OK");
48     print_row("Medium alloc (500B)", "OK", "OK");
49     print_row("Free operations", "OK", "OK");
50     print_row("Fragmentation", "low", "high");
51     print_row("Allocation speed", "slow", "very fast");
52     print_row("Merging support", "yes", "no");
53     print_row("Splitting support", "yes", "yes");
54     print_table_footer();
55
56     std::cout << "Program finished.\n";
57     return 0;
58 }
59

```

pow2_allocator.cpp

```

1  #include "pow2_allocator.h"
2  #include "allocator_common.h"
3  #include <iostream>
4  #include <cstring>
5
6  struct P2Block
7  {
8      size_t size;
9      P2Block* next;
10 };
11
12 static const size_t MAX_CLASSES = 32;
13 static void* arena = nullptr;
14
15 static P2Block* free_lists[MAX_CLASSES];
16
17 size_t pow2_global_split_count = 0;
18 size_t pow2_global_merge_count = 0;
19
20 static size_t next_pow2(size_t x)
21 {
22     size_t p = 1;

```

```

23     while (p < x) p <= 1;
24     return p;
25 }
26
27 size_t size_to_class(size_t size)
28 {
29     size_t cls = 0;
30     while ((1ULL << cls) < size) cls++;
31     return cls;
32 }
33
34 void pow2_init()
35 {
36     arena = alloc_arena(ARENA_SIZE);
37     if (!arena)
38     {
39         std::cerr << "pow2_init: mmap failed\n";
40         return;
41     }
42
43     for (size_t i = 0; i < MAX_CLASSES; i++)
44         free_lists[i] = nullptr;
45
46     size_t max_cls = size_to_class(ARENA_SIZE);
47
48     P2Block* b = (P2Block*)arena;
49     b->size = (1ULL << max_cls);
50     b->next = nullptr;
51
52     free_lists[max_cls] = b;
53 }
54
55 static P2Block* split_block(size_t cls)
56 {
57     size_t cur = cls;
58
59     while (cur < MAX_CLASSES && free_lists[cur] == nullptr)
60         cur++;
61
62     if (cur == MAX_CLASSES)
63         return nullptr;
64
65     P2Block* block = free_lists[cur];
66     free_lists[cur] = block->next;
67
68     while (cur > cls)
69     {
70         cur--;
71
72         size_t half = (1ULL << cur);
73

```

```

74     P2Block* leftover = (P2Block*)((char*)block + half);
75     leftover->size = half;
76     leftover->next = free_lists[cur];
77     free_lists[cur] = leftover;
78
79     block->size = half;
80
81     pow2_global_split_count++;
82 }
83
84     return block;
85 }
86
87 void* pow2_malloc(size_t size)
88 {
89     if (size == 0) return nullptr;
90
91     size_t need = next_pow2(size + sizeof(P2Block));
92     size_t cls = size_to_class(need);
93
94     if (cls >= MAX_CLASSES)
95         return nullptr;
96
97     if (free_lists[cls] != nullptr)
98     {
99         P2Block* b = free_lists[cls];
100         free_lists[cls] = b->next;
101         return (char*)b + sizeof(P2Block);
102     }
103
104     P2Block* b = split_block(cls);
105     if (!b) return nullptr;
106
107     return (char*)b + sizeof(P2Block);
108 }
109
110 void pow2_free(void* ptr)
111 {
112     if (!ptr) return;
113
114     P2Block* block = (P2Block*)((char*)ptr - sizeof(P2Block));
115     size_t cls = size_to_class(block->size);
116
117     block->next = free_lists[cls];
118     free_lists[cls] = block;
119 }
120
121 bool pow2_class_nonempty(size_t cls)
122 {
123     return free_lists[cls] != nullptr;
124 }

```

```

125
126 size_t pow2_blocks_in_class(size_t cls)
127 {
128     size_t count = 0;
129     P2Block* c = free_lists[cls];
130     while (c)
131     {
132         count++;
133         c = c->next;
134     }
135     return count;
136 }
137
138 size_t pow2_get_real_block_size(void* ptr)
139 {
140     if (!ptr) return 0;
141
142     P2Block* b = (P2Block*)((char*)ptr - sizeof(P2Block));
143     return b->size;
144 }
145
146 void pow2_dump()
147 {
148     std::cout << "\n[POWER-OF-TWO ALLOCATOR DUMP]\n";
149
150     for (size_t c = 0; c < MAX_CLASSES; c++)
151     {
152         size_t size = (1ULL << c);
153         std::cout << "Class " << c << " (size=" << size << "): ";
154
155         P2Block* cur = free_lists[c];
156         if (!cur)
157         {
158             std::cout << "empty\n";
159             continue;
160         }
161
162         while (cur)
163         {
164             std::cout << "[" << cur << "]" -> ";
165             cur = cur->next;
166         }
167         std::cout << "NULL\n";
168     }
169
170     std::cout << "[END]\n\n";
171 }
172

```

gtest_allocators.cpp

```
1  #include <gtest/gtest.h>
2  #include <vector>
3  #include <cstring>
4  #include <cstdlib>
5
6  #include "freelist_allocator.h"
7  #include "pow2_allocator.h"
8  #include "allocator_common.h"
9
10
11
12  TEST(FreeList, BasicAllocFree)
13  {
14      fl_init();
15      void* p = fl_malloc(64);
16      ASSERT_NE(p, nullptr);
17      fl_free(p);
18  }
19
20  TEST(FreeList, MultipleAllocFree)
21  {
22      fl_init();
23      void* a = fl_malloc(32);
24      void* b = fl_malloc(64);
25      ASSERT_NE(a, nullptr);
26      ASSERT_NE(b, nullptr);
27      fl_free(a);
28      fl_free(b);
29  }
30
31  TEST(FreeList, ReuseBlock)
32  {
33      fl_init();
34      void* a = fl_malloc(100);
35      fl_free(a);
36      void* b = fl_malloc(80);
37      ASSERT_EQ(a, b);
38  }
39
40  TEST(FreeList, ManySmallAllocations)
41  {
42      fl_init();
43      std::vector<void*> ptrs;
44      for (int i = 0; i < 5000; i++)
45          ptrs.push_back(fl_malloc(8));
46      for (auto p : ptrs)
47          fl_free(p);
48      SUCCEED();
49  }
```

```

50
51 TEST(FreeList, RandomAllocations)
52 {
53     fl_init();
54     std::vector<void*> ptrs(2000);
55     for (int i = 0; i < 2000; i++)
56         ptrs[i] = fl_malloc(rand() % 200 + 1);
57
58     for (int i = 0; i < 2000; i += 2)
59         fl_free(ptrs[i]);
60
61     SUCCEED();
62 }
63
64 TEST(FreeList, LargeAllocation)
65 {
66     fl_init();
67     void* p = fl_malloc(2 * 1024 * 1024);
68     ASSERT_NE(p, nullptr);
69 }
70
71 TEST(FreeList, SplitCorrectness)
72 {
73     fl_init();
74     void* a = fl_malloc(64);
75     void* b = fl_malloc(64);
76     ASSERT_NE(a, nullptr);
77     ASSERT_NE(b, nullptr);
78 }
79
80 TEST(FreeList, FragmentationScenario)
81 {
82     fl_init();
83
84     std::vector<void*> ptrs;
85     for (int i = 0; i < 3000; i++)
86         ptrs.push_back(fl_malloc(32));
87
88     for (int i = 0; i < 3000; i += 3)
89         fl_free(ptrs[i]);
90
91     SUCCEED();
92 }
93
94 TEST(FreeList, StressMixed)
95 {
96     fl_init();
97
98     std::vector<void*> ptrs(10000);
99     for (int i = 0; i < 10000; i++)
100         ptrs[i] = fl_malloc(rand() % 200 + 1);

```

```

101
102     for (int i = 0; i < 10000; i++)
103     if (rand() % 2)
104         fl_free(ptrs[i]);
105
106     SUCCEED();
107 }
108
109
110 TEST(Pow2, BasicAllocFree)
111 {
112     pow2_init();
113     void* p = pow2_malloc(64);
114     ASSERT_NE(p, nullptr);
115     pow2_free(p);
116 }
117
118 TEST(Pow2, ReuseBlock)
119 {
120     pow2_init();
121     void* a = pow2_malloc(100);
122     pow2_free(a);
123     void* b = pow2_malloc(90);
124     ASSERT_EQ(a, b);
125 }
126
127 TEST(Pow2, ManySmallAllocations)
128 {
129     pow2_init();
130     std::vector<void*> v;
131     for (int i = 0; i < 5000; i++)
132         v.push_back(pow2_malloc(12));
133     for (void* p : v) pow2_free(p);
134
135     SUCCEED();
136 }
137
138 TEST(Pow2, RandomAllocations)
139 {
140     pow2_init();
141     std::vector<void*> v(3000);
142
143     for (int i = 0; i < 3000; i++)
144         v[i] = pow2_malloc(rand() % 200 + 1);
145
146     for (int i = 0; i < 3000; i += 3)
147         pow2_free(v[i]);
148
149     SUCCEED();
150 }
151

```

```

152 TEST(Pow2, FailTooLargeRequest)
153 {
154     pow2_init();
155     void* p = pow2_malloc(20 * 1024 * 1024);
156     ASSERT_EQ(p, nullptr);
157 }
158
159 TEST(Pow2, Alignment)
160 {
161     pow2_init();
162     void* p = pow2_malloc(1);
163     ASSERT_TRUE(((uintptr_t)p % ALIGN) == 0);
164 }
165
166 TEST(Pow2, DifferentClassAllocations)
167 {
168     pow2_init();
169     void* a = pow2_malloc(1);
170     void* b = pow2_malloc(3000);
171     void* c = pow2_malloc(500);
172
173     ASSERT_NE(a, nullptr);
174     ASSERT_NE(b, nullptr);
175     ASSERT_NE(c, nullptr);
176
177     pow2_free(a);
178     pow2_free(b);
179     pow2_free(c);
180 }
181
182 TEST(Pow2, StressMixed)
183 {
184     pow2_init();
185
186     std::vector<void*> v(8000);
187     for (int i = 0; i < 8000; i++)
188         v[i] = pow2_malloc(rand() % 1000 + 1);
189
190     for (int i = 0; i < 8000; i++)
191         if (rand() % 2)
192             pow2_free(v[i]);
193
194     SUCCEED();
195 }
196
197

```

CMakeLists.txt

```

1 || cmake_minimum_required(VERSION 3.16)

```

```

2 | project(OS_CourseProject_Allocators)
3 |
4 | set(CMAKE_CXX_STANDARD 17)
5 | set(CMAKE_CXX_STANDARD_REQUIRED ON)
6 |
7 | set(PROJECT_ROOT ${CMAKE_SOURCE_DIR}/cp_var19)
8 | set(SRC_DIR      ${PROJECT_ROOT}/src)
9 | set(INCLUDE_DIR  ${PROJECT_ROOT}/include)
10 | set(TEST_DIR     ${PROJECT_ROOT}/tests)
11 |
12 | include_directories(${INCLUDE_DIR})
13 |
14 | set(MAIN_SOURCES
15 | ${SRC_DIR}/main.cpp
16 | ${SRC_DIR}/freelist_allocator.cpp
17 | ${SRC_DIR}/pow2_allocator.cpp
18 | ${SRC_DIR}/allocator_common.cpp
19 | )
20 |
21 | set(DEMO_SOURCES
22 | ${SRC_DIR}/demo_main.cpp
23 | ${SRC_DIR}/freelist_allocator.cpp
24 | ${SRC_DIR}/pow2_allocator.cpp
25 | ${SRC_DIR}/allocator_common.cpp
26 | )
27 |
28 | set(BENCHMARK_SOURCES
29 | ${SRC_DIR}/benchmark.cpp
30 | ${SRC_DIR}/freelist_allocator.cpp
31 | ${SRC_DIR}/pow2_allocator.cpp
32 | ${SRC_DIR}/allocator_common.cpp
33 | )
34 |
35 | set(TEST_SOURCES
36 | ${TEST_DIR}/gtest_allocators.cpp
37 | ${SRC_DIR}/freelist_allocator.cpp
38 | ${SRC_DIR}/pow2_allocator.cpp
39 | ${SRC_DIR}/allocator_common.cpp
40 | )
41 |
42 | include(FetchContent)
43 | FetchContent_Declare(
44 | googletest
45 | URL https://github.com/google/googletest/archive/refs/tags/v1.14.0.zip
46 | )
47 | FetchContent_MakeAvailable(googletest)
48 |
49 | enable_testing()
50 |
51 | add_executable(main ${MAIN_SOURCES})
52 |

```

```

53 add_executable(demo ${DEMO_SOURCES})
54
55 add_executable(benchmark ${BENCHMARK_SOURCES})
56
57 add_executable(allocator_tests ${TEST_SOURCES})
58 target_link_libraries(allocator_tests gtest gtest_main pthread)
59
60 include(GoogleTest)
61 gtest_discover_tests(allocator_tests)
62
63 message(STATUS "Source dir:      ${SRC_DIR}")
64 message(STATUS "Include dir:     ${INCLUDE_DIR}")
65 message(STATUS "Tests dir:        ${TEST_DIR}")
66

```

Полные выводы strace

```

1  la_flare@LaFlareHub:~/OS-CourseProject_2025/build$ strace ./benchmark
2  execve("./benchmark", [ "./benchmark" ], 0x7ffc9ec99150 /* 48 vars */) = 0
3  brk(NULL) = 0x584121cd8000
4  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c400dbd000
5  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
6  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
7  fstat(3, {st_mode=S_IFREG|0644, st_size=20919, ...}) = 0
8  mmap(NULL, 20919, PROT_READ, MAP_PRIVATE, 3, 0) = 0x73c400db7000
9  close(3) = 0
10 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
11 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
12 fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
13 mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x73c400a00000
14 mmap(0x73c400a9d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9d000) =
  0x73c400a9d000
15 mmap(0x73c400be5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e5000) = 0
  x73c400be5000
16 mmap(0x73c400c6c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26b000) =
  0x73c400c6c000
17 mmap(0x73c400c7a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
  x73c400c7a000
18 close(3) = 0
19 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
20 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
21 fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
22 mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x73c400d89000
23 mmap(0x73c400d8d000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x4000) = 0
  x73c400d8d000
24 mmap(0x73c400db1000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0
  x73c400db1000
25 mmap(0x73c400db5000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2b000) = 0
  x73c400db5000
26 close(3) = 0
27 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
28 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832
29 pread64(3, "\6\0\0\0\4\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 784, 64) = 784
30 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
31 pread64(3, "\6\0\0\0\4\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 784, 64) = 784
32 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x73c400600000
33 mmap(0x73c400628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) =
  0x73c400628000
34 mmap(0x73c4007b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0
  x73c4007b0000
35 mmap(0x73c4007ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) =
  0x73c4007ff000

```

```

36 mmap(0x73c400805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
   x73c400805000
37 close(3) = 0
38 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
39 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
40 fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
41 mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x73c400ca0000
42 mmap(0x73c400cb0000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10000) = 0
   x73c400cb0000
43 mmap(0x73c400d2f000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f000) = 0
   x73c400d2f000
44 mmap(0x73c400d87000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe7000) = 0
   x73c400d87000
45 close(3) = 0
46 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c400c9e000
47 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c400c9b000
48 arch_prctl(ARCH_SET_FS, 0x73c400c9b740) = 0
49 set_tid_address(0x73c400c9ba10) = 82378
50 set_robust_list(0x73c400c9ba20, 24) = 0
51 rseq(0x73c400c9c060, 0x20, 0, 0x53053053) = 0
52 mprotect(0x73c4007ff000, 16384, PROT_READ) = 0
53 mprotect(0x73c400d87000, 4096, PROT_READ) = 0
54 mprotect(0x73c400db5000, 4096, PROT_READ) = 0
55 mprotect(0x73c400c6c000, 45056, PROT_READ) = 0
56 mprotect(0x584115f31000, 4096, PROT_READ) = 0
57 mprotect(0x73c400df5000, 8192, PROT_READ) = 0
58 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
59 munmap(0x73c400db7000, 20919) = 0
60 futex(0x73c400c7a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
61 getrandom("\x06\xfd\xba\x61\x30\x21\x3e\x9d", 8, GRND_NONBLOCK) = 8
62 brk(NULL) = 0x584121cd8000
63 brk(0x584121cf9000) = 0x584121cf9000
64 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x7), ...}) = 0
65 write(1, "=== MEMORY ALLOCATORS BENCHMARK "..., 36) === MEMORY ALLOCATORS BENCHMARK ===
66 ) = 36
67 write(1, "\n[INIT] FreeList...\n", 20
68 [INIT] FreeList...
69 ) = 20
70 mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c3ffe00000
71 mmap(NULL, 1601536, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c400879000
72 brk(0x584121d1b000) = 0x584121d1b000
73 mmap(NULL, 135168, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c400858000
74 brk(0x584121d0b000) = 0x584121d0b000
75 mmap(NULL, 266240, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c400817000
76 munmap(0x73c400858000, 135168) = 0
77 mmap(NULL, 528384, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c3ffd7f000
78 munmap(0x73c400817000, 266240) = 0
79 mmap(NULL, 1052672, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c3ffc7e000
80 munmap(0x73c3ffd7f000, 528384) = 0
81 mmap(NULL, 2101248, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c3ffa7d000
82 munmap(0x73c3ffc7e000, 1052672) = 0
83 munmap(0x73c400879000, 1601536) = 0
84 write(1, "\n", 1
85 ) = 1
86 write(1, "===== METRICS: FreeList ==="... , 40) ===== METRICS: FreeList =====
87 ) = 40
88 write(1, "avg malloc: 89.771 us"... , 33) avg malloc: 89.771 us
89 ) = 33
90 write(1, "avg free: 0.0471788"... , 36) avg free: 0.0471788 us
91 ) = 36
92 write(1, "p50 malloc: 79 us\n", 29) p50 malloc: 79 us
93 ) = 29
94 write(1, "p95 malloc: 158.5 us\n", 32) p95 malloc: 158.5 us
95 ) = 32
96 write(1, "p99 malloc: 269 us\n", 30) p99 malloc: 269 us
97 ) = 30

```

```

98 | write(1, "variance:          2319.49\n", 31variance:          2319.49
99 | ) = 31
100 | write(1, "jitter (p99-p50):    190 us\n", 30jitter (p99-p50):    190 us
101 | ) = 30
102 | write(1, "splits:             7868\n", 28splits:             7868
103 | ) = 28
104 | write(1, "merges:             7868\n", 28merges:             7868
105 | ) = 28
106 | write(1, "avg internal frag:    7.52307 b"... , 37avg internal frag:    7.52307 bytes
107 | ) = 37
108 | write(1, "largest free block:  8388576 b"... , 37largest free block:  8388576 bytes
109 | ) = 37
110 | write(1, "total free memory:   8388576 b"... , 37total free memory:   8388576 bytes
111 | ) = 37
112 | write(1, "external fragmentation: 0 %\n", 28external fragmentation: 0 %
113 | ) = 28
114 | write(1, "peak allocated:      8136800 b"... , 37peak allocated:      8136800 bytes
115 | ) = 37
116 | write(1, "\n[INIT] Power-of-Two...\n", 24
117 | [INIT] Power-of-Two...
118 | ) = 24
119 | mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c3ff27d000
120 | brk(0x584121eb2000)                = 0x584121eb2000
121 | brk(0x584121ee2000)                = 0x584121ee2000
122 | brk(0x584121f22000)                = 0x584121f22000
123 | brk(0x584121fa2000)                = 0x584121fa2000
124 | brk(0x5841220a2000)                = 0x5841220a2000
125 | mmap(NULL, 2101248, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x73c3ff07c000
126 | write(1, "\n", 1
127 | )                                  = 1
128 | write(1, "==== METRICS: PowerOfTwo ="... , 42==== METRICS: PowerOfTwo =====
129 | ) = 42
130 | write(1, "avg malloc:          0.152135 "... , 35avg malloc:          0.152135 us
131 | ) = 35
132 | write(1, "avg free:           0.108775 "... , 35avg free:           0.108775 us
133 | ) = 35
134 | write(1, "p50 malloc:         0 us\n", 28p50 malloc:         0 us
135 | ) = 28
136 | write(1, "p95 malloc:         0.25 us\n", 31p95 malloc:         0.25 us
137 | ) = 31
138 | write(1, "p99 malloc:         0.5 us\n", 30p99 malloc:         0.5 us
139 | ) = 30
140 | write(1, "variance:          3.24355\n", 31variance:          3.24355
141 | ) = 31
142 | write(1, "jitter (p99-p50):    0.5 us\n", 30jitter (p99-p50):    0.5 us
143 | ) = 30
144 | write(1, "splits:            6039\n", 28splits:            6039
145 | ) = 28
146 | write(1, "merges:             0\n", 25merges:             0
147 | ) = 25
148 | write(1, "avg internal frag:   369.877 b"... , 37avg internal frag:   369.877 bytes
149 | ) = 37
150 | write(1, "largest free block:  4096 byte"... , 34largest free block:  4096 bytes
151 | ) = 34
152 | write(1, "total free memory:   8388608 b"... , 37total free memory:   8388608 bytes
153 | ) = 37
154 | write(1, "external fragmentation: 99.9512 "... , 34external fragmentation: 99.9512 %
155 | ) = 34
156 | write(1, "peak allocated:      8388608 b"... , 37peak allocated:      8388608 bytes
157 | ) = 37
158 | write(1, "\n=== BENCHMARK FINISHED ===\n", 28
159 | === BENCHMARK FINISHED ===
160 | ) = 28
161 | brk(0x584121d2b000)                = 0x584121d2b000
162 | munmap(0x73c3ff07c000, 2101248)    = 0

```

```

163 munmap(0x73c3ffa7d000, 2101248) = 0
164 exit_group(0) = ?
165 +++ exited with 0 +++
166 la_flare@LaFlareHub:~/OS-CourseProject_2025/build$
167
168
169
170
171
172
173
174 la_flare@LaFlareHub:~/OS-CourseProject_2025/build$ strace ./demo
175 execve("./demo", ["/demo"], 0x7fff4a759710 /* 48 vars */) = 0
176 brk(NULL) = 0x56ef8c16b000
177 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72b94aa2e000
178 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
179 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
180 fstat(3, {st_mode=S_IFREG|0644, st_size=20919, ...}) = 0
181 mmap(NULL, 20919, PROT_READ, MAP_PRIVATE, 3, 0) = 0x72b94aa28000
182 close(3) = 0
183 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
184 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
185 fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
186 mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x72b94a600000
187 mmap(0x72b94a69d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9d000) =
  0x72b94a69d000
188 mmap(0x72b94a7e5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e5000) = 0
  x72b94a7e5000
189 mmap(0x72b94a86c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26b000) =
  0x72b94a86c000
190 mmap(0x72b94a87a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
  x72b94a87a000
191 close(3) = 0
192 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
193 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
194 pread64(3, "\6\0\0\0\4\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 784, 64) = 784
195 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
196 pread64(3, "\6\0\0\0\4\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 784, 64) = 784
197 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x72b94a200000
198 mmap(0x72b94a228000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) =
  0x72b94a228000
199 mmap(0x72b94a3b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0
  x72b94a3b0000
200 mmap(0x72b94a3ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) =
  0x72b94a3ff000
201 mmap(0x72b94a405000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
  x72b94a405000
202 close(3) = 0
203 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
204 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
205 fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
206 mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x72b94a93f000
207 mmap(0x72b94a94f000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10000) =
  0x72b94a94f000
208 mmap(0x72b94a9ce000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f000) = 0
  x72b94a9ce000
209 mmap(0x72b94aa26000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe7000) = 0
  x72b94aa26000
210 close(3) = 0
211 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc.s.so.1", O_RDONLY|O_CLOEXEC) = 3
212 read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
213 fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
214 mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x72b94a911000
215 mmap(0x72b94a915000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x4000) = 0
  x72b94a915000
216 mmap(0x72b94a939000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0
  x72b94a939000

```

```

217 | mmap(0x72b94a93d000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2b000) = 0
218 | close(3) = 0
219 | mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72b94a90f000
220 | arch_prctl(ARCH_SET_FS, 0x72b94a910500) = 0
221 | set_tid_address(0x72b94a9107d0) = 82662
222 | set_robust_list(0x72b94a9107e0, 24) = 0
223 | rseq(0x72b94a910e20, 0x20, 0, 0x53053053) = 0
224 | mprotect(0x72b94a3ff000, 16384, PROT_READ) = 0
225 | mprotect(0x72b94a93d000, 4096, PROT_READ) = 0
226 | mprotect(0x72b94aa26000, 4096, PROT_READ) = 0
227 | mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72b94a90d000
228 | mprotect(0x72b94a86c000, 45056, PROT_READ) = 0
229 | mprotect(0x56ef4fea1000, 4096, PROT_READ) = 0
230 | mprotect(0x72b94aa66000, 8192, PROT_READ) = 0
231 | prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
232 | munmap(0x72b94aa28000, 20919) = 0
233 | futex(0x72b94a87a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
234 | getrandom("\x5d\x39\x25\xff\x56\x06\x78\x8c", 8, GRND_NONBLOCK) = 8
235 | brk(NULL) = 0x56ef8c16b000
236 | brk(0x56ef8c18c000) = 0x56ef8c18c000
237 | fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x7), ...}) = 0
238 | write(1, "=== DEMO: FreeList & Power-of-Tw"... , 49=== DEMO: FreeList & Power-of-Two Allocators ===
239 | ) = 49
240 | write(1, "\n", 1
241 | ) = 1
242 | write(1, "[INIT] Initializing FreeList all"... , 42[INIT] Initializing FreeList allocator...
243 | ) = 42
244 | mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72b949a00000
245 | write(1, "[INIT] Initializing Power-of-Two"... , 46[INIT] Initializing Power-of-Two allocator...
246 | ) = 46
247 | mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72b949200000
248 | write(1, "\n=== DEMO: FreeList Allocator ==="... , 34
249 | === DEMO: FreeList Allocator ===
250 | ) = 34
251 | write(1, "Allocated blocks:\n", 18Allocated blocks:
252 | ) = 18
253 | write(1, " a1 = 0x72b949a00020\n", 21 a1 = 0x72b949a00020
254 | ) = 21
255 | write(1, " a2 = 0x72b949a00060\n", 21 a2 = 0x72b949a00060
256 | ) = 21
257 | write(1, " a3 = 0x72b949a000c0\n", 21 a3 = 0x72b949a000c0
258 | ) = 21
259 | write(1, "\n[STATE] FreeList after allocati"... , 37
260 | [STATE] FreeList after allocations:
261 | ) = 37
262 | write(1, "[Free list dump]\n", 17[Free list dump]
263 | ) = 17
264 | write(1, " Block@0x72b949a00000 size=32 f"... , 38 Block@0x72b949a00000 size=32 free=0
265 | ) = 38
266 | write(1, " Block@0x72b949a00040 size=64 f"... , 38 Block@0x72b949a00040 size=64 free=0
267 | ) = 38
268 | write(1, " Block@0x72b949a000a0 size=128 "... , 39 Block@0x72b949a000a0 size=128 free=0
269 | ) = 39
270 | write(1, " Block@0x72b949a00140 size=8388"... , 43 Block@0x72b949a00140 size=8388256 free=1
271 | ) = 43
272 | write(1, "\n[FREE] Freeing a2...\n", 22
273 | [FREE] Freeing a2...
274 | ) = 22
275 | write(1, "[Free list dump]\n", 17[Free list dump]
276 | ) = 17
277 | write(1, " Block@0x72b949a00000 size=32 f"... , 38 Block@0x72b949a00000 size=32 free=0
278 | ) = 38
279 | write(1, " Block@0x72b949a00040 size=64 f"... , 38 Block@0x72b949a00040 size=64 free=1
280 | ) = 38

```

```

281 | write(1, " Block0x72b949a000a0 size=128 "..., 39 Block0x72b949a000a0 size=128 free=0
282 | ) = 39
283 | write(1, " Block0x72b949a00140 size=8388"..., 43 Block0x72b949a00140 size=8388256 free=1
284 | ) = 43
285 | write(1, "\n[FREE] Freeing a1 and a3...\n", 29
286 | [FREE] Freeing a1 and a3...
287 | ) = 29
288 | write(1, "[Free list dump]\n", 17[Free list dump]
289 | ) = 17
290 | write(1, " Block0x72b949a00000 size=8388"..., 43 Block0x72b949a00000 size=8388576 free=1
291 | ) = 43
292 | write(1, "\n=== DEMO: Power-of-Two Allocato"..., 38
293 | === DEMO: Power-of-Two Allocator ===
294 | ) = 38
295 | write(1, "Allocated blocks:\n", 18Allocated blocks:
296 | ) = 18
297 | write(1, " b1 = 0x72b949200010\n", 21 b1 = 0x72b949200010
298 | ) = 21
299 | write(1, " b2 = 0x72b949200110\n", 21 b2 = 0x72b949200110
300 | ) = 21
301 | write(1, " b3 = 0x72b949200210\n", 21 b3 = 0x72b949200210
302 | ) = 21
303 | write(1, "\n[STATE] Power-of-Two after allo"..., 41
304 | [STATE] Power-of-Two after allocations:
305 | ) = 41
306 | write(1, "\n[POWER-OF-TWO ALLOCATOR DUMP]\n", 31
307 | [POWER-OF-TWO ALLOCATOR DUMP]
308 | ) = 31
309 | write(1, "Class 0 (size=1): empty\n", 24Class 0 (size=1): empty
310 | ) = 24
311 | write(1, "Class 1 (size=2): empty\n", 24Class 1 (size=2): empty
312 | ) = 24
313 | write(1, "Class 2 (size=4): empty\n", 24Class 2 (size=4): empty
314 | ) = 24
315 | write(1, "Class 3 (size=8): empty\n", 24Class 3 (size=8): empty
316 | ) = 24
317 | write(1, "Class 4 (size=16): empty\n", 25Class 4 (size=16): empty
318 | ) = 25
319 | write(1, "Class 5 (size=32): empty\n", 25Class 5 (size=32): empty
320 | ) = 25
321 | write(1, "Class 6 (size=64): [0x72b9492000"..., 44Class 6 (size=64): [0x72b949200040] -> NULL
322 | ) = 44
323 | write(1, "Class 7 (size=128): [0x72b949200"..., 45Class 7 (size=128): [0x72b949200080] -> NULL
324 | ) = 45
325 | write(1, "Class 8 (size=256): empty\n", 26Class 8 (size=256): empty
326 | ) = 26
327 | write(1, "Class 9 (size=512): empty\n", 26Class 9 (size=512): empty
328 | ) = 26
329 | write(1, "Class 10 (size=1024): [0x72b9492"..., 47Class 10 (size=1024): [0x72b949200400] -> NULL
330 | ) = 47
331 | write(1, "Class 11 (size=2048): [0x72b9492"..., 47Class 11 (size=2048): [0x72b949200800] -> NULL
332 | ) = 47
333 | write(1, "Class 12 (size=4096): [0x72b9492"..., 47Class 12 (size=4096): [0x72b949201000] -> NULL
334 | ) = 47
335 | write(1, "Class 13 (size=8192): [0x72b9492"..., 47Class 13 (size=8192): [0x72b949202000] -> NULL
336 | ) = 47
337 | write(1, "Class 14 (size=16384): [0x72b949"..., 48Class 14 (size=16384): [0x72b949204000] -> NULL
338 | ) = 48
339 | write(1, "Class 15 (size=32768): [0x72b949"..., 48Class 15 (size=32768): [0x72b949208000] -> NULL
340 | ) = 48
341 | write(1, "Class 16 (size=65536): [0x72b949"..., 48Class 16 (size=65536): [0x72b949210000] -> NULL
342 | ) = 48
343 | write(1, "Class 17 (size=131072): [0x72b94"..., 49Class 17 (size=131072): [0x72b949220000] -> NULL
344 | ) = 49
345 | write(1, "Class 18 (size=262144): [0x72b94"..., 49Class 18 (size=262144): [0x72b949240000] -> NULL

```

```

346 | ) = 49
347 | write(1, "Class 19 (size=524288): [0x72b94"... , 49Class 19 (size=524288): [0x72b949280000] -> NULL
348 | ) = 49
349 | write(1, "Class 20 (size=1048576): [0x72b9"... , 50Class 20 (size=1048576): [0x72b949300000] -> NULL
350 | ) = 50
351 | write(1, "Class 21 (size=2097152): [0x72b9"... , 50Class 21 (size=2097152): [0x72b949400000] -> NULL
352 | ) = 50
353 | write(1, "Class 22 (size=4194304): [0x72b9"... , 50Class 22 (size=4194304): [0x72b949600000] -> NULL
354 | ) = 50
355 | write(1, "Class 23 (size=8388608): empty\n", 31Class 23 (size=8388608): empty
356 | ) = 31
357 | write(1, "Class 24 (size=16777216): empty\n", 32Class 24 (size=16777216): empty
358 | ) = 32
359 | write(1, "Class 25 (size=33554432): empty\n", 32Class 25 (size=33554432): empty
360 | ) = 32
361 | write(1, "Class 26 (size=67108864): empty\n", 32Class 26 (size=67108864): empty
362 | ) = 32
363 | write(1, "Class 27 (size=134217728): empty"... , 33Class 27 (size=134217728): empty
364 | ) = 33
365 | write(1, "Class 28 (size=268435456): empty"... , 33Class 28 (size=268435456): empty
366 | ) = 33
367 | write(1, "Class 29 (size=536870912): empty"... , 33Class 29 (size=536870912): empty
368 | ) = 33
369 | write(1, "Class 30 (size=1073741824): empt"... , 34Class 30 (size=1073741824): empty
370 | ) = 34
371 | write(1, "Class 31 (size=2147483648): empt"... , 34Class 31 (size=2147483648): empty
372 | ) = 34
373 | write(1, "[END]\n\n", 7[END]
374 |
375 | )
376 | write(1, "\n[FREE] Freeing b2...\n", 22
377 | [FREE] Freeing b2...
378 | ) = 22
379 | write(1, "\n[POWER-OF-TWO ALLOCATOR DUMP]\n", 31
380 | [POWER-OF-TWO ALLOCATOR DUMP]
381 | ) = 31
382 | write(1, "Class 0 (size=1): empty\n", 24Class 0 (size=1): empty
383 | ) = 24
384 | write(1, "Class 1 (size=2): empty\n", 24Class 1 (size=2): empty
385 | ) = 24
386 | write(1, "Class 2 (size=4): empty\n", 24Class 2 (size=4): empty
387 | ) = 24
388 | write(1, "Class 3 (size=8): empty\n", 24Class 3 (size=8): empty
389 | ) = 24
390 | write(1, "Class 4 (size=16): empty\n", 25Class 4 (size=16): empty
391 | ) = 25
392 | write(1, "Class 5 (size=32): empty\n", 25Class 5 (size=32): empty
393 | ) = 25
394 | write(1, "Class 6 (size=64): [0x72b9492000"... , 44Class 6 (size=64): [0x72b949200040] -> NULL
395 | ) = 44
396 | write(1, "Class 7 (size=128): [0x72b949200"... , 45Class 7 (size=128): [0x72b949200080] -> NULL
397 | ) = 45
398 | write(1, "Class 8 (size=256): [0x72b949200"... , 45Class 8 (size=256): [0x72b949200100] -> NULL
399 | ) = 45
400 | write(1, "Class 9 (size=512): empty\n", 26Class 9 (size=512): empty
401 | ) = 26
402 | write(1, "Class 10 (size=1024): [0x72b9492"... , 47Class 10 (size=1024): [0x72b949200400] -> NULL
403 | ) = 47
404 | write(1, "Class 11 (size=2048): [0x72b9492"... , 47Class 11 (size=2048): [0x72b949200800] -> NULL
405 | ) = 47
406 | write(1, "Class 12 (size=4096): [0x72b9492"... , 47Class 12 (size=4096): [0x72b949201000] -> NULL
407 | ) = 47
408 | write(1, "Class 13 (size=8192): [0x72b9492"... , 47Class 13 (size=8192): [0x72b949202000] -> NULL
409 | ) = 47
410 | write(1, "Class 14 (size=16384): [0x72b949"... , 48Class 14 (size=16384): [0x72b949204000] -> NULL

```

```

411 | ) = 48
412 | write(1, "Class 15 (size=32768): [0x72b949]...", 48Class 15 (size=32768): [0x72b949208000] -> NULL
413 | ) = 48
414 | write(1, "Class 16 (size=65536): [0x72b949]...", 48Class 16 (size=65536): [0x72b949210000] -> NULL
415 | ) = 48
416 | write(1, "Class 17 (size=131072): [0x72b94]...", 49Class 17 (size=131072): [0x72b949220000] -> NULL
417 | ) = 49
418 | write(1, "Class 18 (size=262144): [0x72b94]...", 49Class 18 (size=262144): [0x72b949240000] -> NULL
419 | ) = 49
420 | write(1, "Class 19 (size=524288): [0x72b94]...", 49Class 19 (size=524288): [0x72b949280000] -> NULL
421 | ) = 49
422 | write(1, "Class 20 (size=1048576): [0x72b9]...", 50Class 20 (size=1048576): [0x72b949300000] -> NULL
423 | ) = 50
424 | write(1, "Class 21 (size=2097152): [0x72b9]...", 50Class 21 (size=2097152): [0x72b949400000] -> NULL
425 | ) = 50
426 | write(1, "Class 22 (size=4194304): [0x72b9]...", 50Class 22 (size=4194304): [0x72b949600000] -> NULL
427 | ) = 50
428 | write(1, "Class 23 (size=8388608): empty\n", 31Class 23 (size=8388608): empty
429 | ) = 31
430 | write(1, "Class 24 (size=16777216): empty\n", 32Class 24 (size=16777216): empty
431 | ) = 32
432 | write(1, "Class 25 (size=33554432): empty\n", 32Class 25 (size=33554432): empty
433 | ) = 32
434 | write(1, "Class 26 (size=67108864): empty\n", 32Class 26 (size=67108864): empty
435 | ) = 32
436 | write(1, "Class 27 (size=134217728): empty"..., 33Class 27 (size=134217728): empty
437 | ) = 33
438 | write(1, "Class 28 (size=268435456): empty"..., 33Class 28 (size=268435456): empty
439 | ) = 33
440 | write(1, "Class 29 (size=536870912): empty"..., 33Class 29 (size=536870912): empty
441 | ) = 33
442 | write(1, "Class 30 (size=1073741824): empt"..., 34Class 30 (size=1073741824): empty
443 | ) = 34
444 | write(1, "Class 31 (size=2147483648): empt"..., 34Class 31 (size=2147483648): empty
445 | ) = 34
446 | write(1, "[END]\n\n", 7[END]
447 |
448 | )
449 | write(1, "\n[FREE] Freeing b1 and b3...\n", 29
450 | [FREE] Freeing b1 and b3...
451 | ) = 29
452 | write(1, "\n[POWER-OF-TWO ALLOCATOR DUMP]\n", 31
453 | [POWER-OF-TWO ALLOCATOR DUMP]
454 | ) = 31
455 | write(1, "Class 0 (size=1): empty\n", 24Class 0 (size=1): empty
456 | ) = 24
457 | write(1, "Class 1 (size=2): empty\n", 24Class 1 (size=2): empty
458 | ) = 24
459 | write(1, "Class 2 (size=4): empty\n", 24Class 2 (size=4): empty
460 | ) = 24
461 | write(1, "Class 3 (size=8): empty\n", 24Class 3 (size=8): empty
462 | ) = 24
463 | write(1, "Class 4 (size=16): empty\n", 25Class 4 (size=16): empty
464 | ) = 25
465 | write(1, "Class 5 (size=32): empty\n", 25Class 5 (size=32): empty
466 | ) = 25
467 | write(1, "Class 6 (size=64): [0x72b9492000]...", 64Class 6 (size=64): [0x72b949200000] -> [0
468 | x72b949200040] -> NULL
469 | ) = 64
469 | write(1, "Class 7 (size=128): [0x72b949200]...", 45Class 7 (size=128): [0x72b949200080] -> NULL
470 | ) = 45
471 | write(1, "Class 8 (size=256): [0x72b949200]...", 45Class 8 (size=256): [0x72b949200100] -> NULL
472 | ) = 45
473 | write(1, "Class 9 (size=512): [0x72b949200]...", 45Class 9 (size=512): [0x72b949200200] -> NULL
474 | ) = 45

```

```

475 | write(1, "Class 10 (size=1024): [0x72b9492]...", 47Class 10 (size=1024): [0x72b949200400] -> NULL
476 | ) = 47
477 | write(1, "Class 11 (size=2048): [0x72b9492]...", 47Class 11 (size=2048): [0x72b949200800] -> NULL
478 | ) = 47
479 | write(1, "Class 12 (size=4096): [0x72b9492]...", 47Class 12 (size=4096): [0x72b949201000] -> NULL
480 | ) = 47
481 | write(1, "Class 13 (size=8192): [0x72b9492]...", 47Class 13 (size=8192): [0x72b949202000] -> NULL
482 | ) = 47
483 | write(1, "Class 14 (size=16384): [0x72b949]...", 48Class 14 (size=16384): [0x72b949204000] -> NULL
484 | ) = 48
485 | write(1, "Class 15 (size=32768): [0x72b949]...", 48Class 15 (size=32768): [0x72b949208000] -> NULL
486 | ) = 48
487 | write(1, "Class 16 (size=65536): [0x72b949]...", 48Class 16 (size=65536): [0x72b949210000] -> NULL
488 | ) = 48
489 | write(1, "Class 17 (size=131072): [0x72b94]...", 49Class 17 (size=131072): [0x72b949220000] -> NULL
490 | ) = 49
491 | write(1, "Class 18 (size=262144): [0x72b94]...", 49Class 18 (size=262144): [0x72b949240000] -> NULL
492 | ) = 49
493 | write(1, "Class 19 (size=524288): [0x72b94]...", 49Class 19 (size=524288): [0x72b949280000] -> NULL
494 | ) = 49
495 | write(1, "Class 20 (size=1048576): [0x72b9]...", 50Class 20 (size=1048576): [0x72b949300000] -> NULL
496 | ) = 50
497 | write(1, "Class 21 (size=2097152): [0x72b9]...", 50Class 21 (size=2097152): [0x72b949400000] -> NULL
498 | ) = 50
499 | write(1, "Class 22 (size=4194304): [0x72b9]...", 50Class 22 (size=4194304): [0x72b949600000] -> NULL
500 | ) = 50
501 | write(1, "Class 23 (size=8388608): empty\n", 31Class 23 (size=8388608): empty
502 | ) = 31
503 | write(1, "Class 24 (size=16777216): empty\n", 32Class 24 (size=16777216): empty
504 | ) = 32
505 | write(1, "Class 25 (size=33554432): empty\n", 32Class 25 (size=33554432): empty
506 | ) = 32
507 | write(1, "Class 26 (size=67108864): empty\n", 32Class 26 (size=67108864): empty
508 | ) = 32
509 | write(1, "Class 27 (size=134217728): empty"..., 33Class 27 (size=134217728): empty
510 | ) = 33
511 | write(1, "Class 28 (size=268435456): empty"..., 33Class 28 (size=268435456): empty
512 | ) = 33
513 | write(1, "Class 29 (size=536870912): empty"..., 33Class 29 (size=536870912): empty
514 | ) = 33
515 | write(1, "Class 30 (size=1073741824): empt"..., 34Class 30 (size=1073741824): empty
516 | ) = 34
517 | write(1, "Class 31 (size=2147483648): empt"..., 34Class 31 (size=2147483648): empty
518 | ) = 34
519 | write(1, "[END]\n\n", 7[END]
520 |
521 | )
522 | write(1, "\n=== DEMO FINISHED SUCCESSFULLY "..., 36
523 | === DEMO FINISHED SUCCESSFULLY ===
524 | ) = 36
525 | exit_group(0)
526 | +++ exited with 0 +++
527 | la_flare@LaFlareHub:~/OS-CourseProject_2025/build$
528 |
529 |
530 |
531 |
532 |
533 |
534 |
535 |
536 |
537 | la_flare@LaFlareHub:~/OS-CourseProject_2025/build$ strace ./main
538 | execve("./main", ["/main"], 0x7ffc2373ea80 /* 48 vars */) = 0
539 | brk(NULL)

```

```

540 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e391dfb9000
541 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
542 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
543 fstat(3, {st_mode=S_IFREG|0644, st_size=20919, ...}) = 0
544 mmap(NULL, 20919, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7e391dfb3000
545 close(3) = 0
546 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
547 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
548 fstat(3, {st_mode=S_IFREG|0644, st_size=2592224, ...}) = 0
549 mmap(NULL, 2609472, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7e391dc00000
550 mmap(0x7e391dc9d000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9d000) =
    0x7e391dc9d000
551 mmap(0x7e391dde5000, 552960, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e5000) = 0
    x7e391dde5000
552 mmap(0x7e391de6c000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26b000) = 0
    x7e391de6c000
553 mmap(0x7e391de7a000, 12608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
    x7e391de7a000
554 close(3) = 0
555 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc.s.so.1", O_RDONLY|O_CLOEXEC) = 3
556 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
557 fstat(3, {st_mode=S_IFREG|0644, st_size=183024, ...}) = 0
558 mmap(NULL, 185256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7e391df85000
559 mmap(0x7e391df89000, 147456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x4000) = 0
    x7e391df89000
560 mmap(0x7e391dfad000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0
    x7e391dfad000
561 mmap(0x7e391dfb1000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2b000) = 0
    x7e391dfb1000
562 close(3) = 0
563 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
564 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\220\243\2\0\0\0\0"..., 832) = 832
565 pread64(3, "\6\0\0\0\4\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 784, 64) = 784
566 fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
567 pread64(3, "\6\0\0\0\4\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 784, 64) = 784
568 mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7e391d800000
569 mmap(0x7e391d828000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0
    x7e391d828000
570 mmap(0x7e391d9b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0
    x7e391d9b0000
571 mmap(0x7e391d9ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0
    x7e391d9ff000
572 mmap(0x7e391da05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
    x7e391da05000
573 close(3) = 0
574 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
575 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
576 fstat(3, {st_mode=S_IFREG|0644, st_size=952616, ...}) = 0
577 mmap(NULL, 950296, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7e391de9c000
578 mmap(0x7e391deac000, 520192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10000) = 0
    x7e391deac000
579 mmap(0x7e391df2b000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8f000) = 0
    x7e391df2b000
580 mmap(0x7e391df83000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe7000) = 0
    x7e391df83000
581 close(3) = 0
582 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e391de9a000
583 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e391de97000
584 arch_prctl(ARCH_SET_FS, 0x7e391de97740) = 0
585 set_tid_address(0x7e391de97a10) = 82829
586 set_robust_list(0x7e391de97a20, 24) = 0
587 rseq(0x7e391de98060, 0x20, 0, 0x53053053) = 0
588 mprotect(0x7e391d9ff000, 16384, PROT_READ) = 0
589 mprotect(0x7e391df83000, 4096, PROT_READ) = 0
590 mprotect(0x7e391dfb1000, 4096, PROT_READ) = 0
591 mprotect(0x7e391de6c000, 45056, PROT_READ) = 0

```

```

592 | mprotect(0x58aea561d000, 4096, PROT_READ) = 0
593 | mprotect(0x7e391dff1000, 8192, PROT_READ) = 0
594 | prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
595 | munmap(0x7e391dfb3000, 20919) = 0
596 | futex(0x7e391de7a7bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
597 | getRandom("\x16\x5b\x44\x3d\x8e\x7a\x0a\xe5", 8, GRND_NONBLOCK) = 8
598 | brk(NULL) = 0x58aede527000
599 | brk(0x58aede548000) = 0x58aede548000
600 | fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x7), ...}) = 0
601 | write(1, "=== Memory Allocation Course Pro"... , 41=== Memory Allocation Course Project ===
602 | ) = 41
603 | write(1, "Comparing FreeList vs Power-of-T"... , 47Comparing FreeList vs Power-of-Two allocators
604 |
605 | ) = 47
606 | mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e391d000000
607 | mmap(NULL, 8388608, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e391c800000
608 | write(1, "Allocated pointers:\n", 20Allocated pointers:
609 | ) = 20
610 | write(1, " FreeList: 0x7e391d000020,"... , 48 FreeList: 0x7e391d000020, 0x7e391d0000c0
611 | ) = 48
612 | write(1, " Power-of-Two: 0x7e391c800010,"... , 49 Power-of-Two: 0x7e391c800010, 0x7e391c800410
613 |
614 | ) = 49
615 | write(1, "+-----+---"...,
616 | 77+-----+-----+-----+
617 | ) = 77
617 | write(1, "| Metric | Fre"... , 77| Metric | FreeList
618 | Power-of-Two |
619 | ) = 77
619 | write(1, "+-----+---"...,
620 | 77+-----+-----+-----+
621 | ) = 77
621 | write(1, "| Small alloc (128B) | OK"... , 78| Small alloc (128B) | OK
622 | | OK |
623 | ) = 78
623 | write(1, "| Medium alloc (500B) | OK"... , 78| Medium alloc (500B) | OK
624 | | OK |
625 | ) = 78
625 | write(1, "| Free operations | OK"... , 78| Free operations | OK
626 | | OK |
627 | ) = 78
627 | write(1, "| Fragmentation | lo"... , 78| Fragmentation | low
628 | | high |
629 | ) = 78
629 | write(1, "| Allocation speed | sl"... , 78| Allocation speed | slow
630 | | very fast |
631 | ) = 78
631 | write(1, "| Merging support | ye"... , 78| Merging support | yes
632 | | no |
633 | ) = 78
633 | write(1, "| Splitting support | ye"... , 78| Splitting support | yes
634 | | yes |
635 | ) = 78
635 | write(1, "+-----+---"...,
636 | 78+-----+-----+-----+
637 | ) = 78
638 | write(1, "Program finished.\n", 18Program finished.
639 | ) = 18
640 | exit_group(0) = ?
641 | +++ exited with 0 +++
642 | la_flare@LaFlareHub:~/OS-CourseProject_2025/build$
643 |

```

Анализ поведения программы по данным strace

Представленные выше трассы системных вызовов (strace) демонстрируют подробное взаимодействие программы с операционной системой на всех этапах выполнения. Данный раздел содержит аналитическую интерпретацию полученных данных с акцентом на работу пользовательских аллокаторов памяти, реализованных в проекте.

Загрузка исполняемого файла и стандартных библиотек

Начальная часть трассы для всех трёх программ (main, demo, benchmark) содержит одинаковую последовательность системных вызовов:

- `openat`, `read`, `fstat`, `mmap` — загрузка динамических библиотек (`libstdc++`, `libc`, `libm`, `libgcc`).
- `arch_prctl` — установка регистра FS для TLS.
- `getrandom` — инициализация генератора случайных чисел.
- `mprotect` — настройка прав доступа к сегментам ELF.

Все эти вызовы являются частью работы загрузчика ELF и **не относятся к работе пользовательских аллокаторов**. Они обязательны и идентичны для большинства программ на C++.

Выделение арены памяти аллокаторами

Ключевым моментом для анализа является первый системный вызов `mmap`, совершаемый самими аллокаторами при инициализации:

```
1 | mmap(NULL, 8388608, PROT_READ|PROT_WRITE,  
2 | MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
```

Этот вызов происходит:

- в `fl_init()` — для аллокатора FreeList;
- в `pow2_init()` — для аллокатора Power-of-Two.

Таким образом создаётся арена фиксированного размера (8 MiB), в пределах которой осуществляется всё дальнейшее управление памятью.

Важно: после этого вызова аллокаторы больше не обращаются к ядру для выделения или освобождения памяти. Это подтверждается отсутствием последующих `mmap()`, `munmap()`, `brk()` или `sbrk()` во всей трассе.

Работа аллокаторов внутри пользовательского пространства

Все вызовы `malloc`, `free` и `realloc`, выполненные обеими реализациями, выполняются полностью внутри арены:

- **FreeList** использует линейный поиск, разделение (*split*) и объединение блоков (*merge*);
- **Power-of-Two** выделяет память через списки классов размера и не выполняет операций объединения.

Так как работа происходит только с уже выделенной областью, никаких дополнительных системных вызовов не требуется.

Эти факты подтверждают корректность реализации: аллокаторы функционируют как полноценные пользовательские менеджеры памяти.

Выводы по анализу `strace`

На основе полного анализа трасс можно сделать следующие выводы:

1. Оба аллокатора вызывают `mmap` только один раз — при инициализации. Это полностью соответствует архитектуре пользовательских аллокаторов.
2. Все дальнейшие операции по управлению памятью выполняются исключительно в `user-space` и не затрагивают ядро.
3. Временные характеристики аллокаторов различаются:
 - **FreeList** — более высокая фрагментация и непредсказуемая длительность операций.
 - **Power-of-Two** — значительно более стабильное и быстрое выделение.
4. Все дополнительные системные вызовы в трассе относятся к стандартной работе среды выполнения и не связаны с выделением памяти.

Полученный анализ подтверждает корректность структуры аллокаторов и обоснованность сравнения их поведения в бенчмарке.