МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Програмної інженерії

Звіт
з лабораторної роботи №5
з дисципліни: «Операційні системи»
з теми: «Керування пам'яттю. Частина 1»

Виконали:
ст. гр. ПЗПІ-23-2
Ситник Є. С.
Малишкін. А. С.

Перевірила:
доц. каф. ПІ,
Мельнікова Р. В.,

Харків – 2025

5 КЕРУВАННЯ ПАМ'ЯТТЮ. ЧАСТИНА 1

5.1 Мета роботи

Метою даної лабораторної роботи є вивчення особливостей використання та функції для роботи з віртуальною та фізичною пам’яттю.

5.2 Хід роботи

Оскільки в якості операційних систем ми використовуємо дистрибутиви Linux, дану лабораторну роботу буде виконано саме для цієї ОС.

5.2.1 Розробка програми «Info». Виконання пунктів 1 – 7 методичних вказівок.

Код складеної програми розміщено в додатку Б.

5.2.1.1 Скласти програму для формування системної інформації про різні типи пам’яті

В рамках цієї програми ми зосередимося на дослідженні та розробці механізму для збору системної інформації про різні типи пам’яті, використовуючи доступні системні виклики та утиліти Linux, що є аналогами функціональності GetSystemInfo та GlobalMemoryStatusEx у Windows. Зокрема, для отримання необхідних даних будуть задіяні системні виклики «sysinfo» та «sysconf», а також проаналізовано вміст файлів «/proc/meminfo» та «/proc/self/maps».

Розглянемо детальніше згадані вище засоби:

- «sysinfo»: системний виклик, що надає загальну інформацію про систему, таку як час роботи, кількість процесів, використання оперативної та swap-пам’яті;
- «sysconf»: системний виклик, який дозволяє отримати різні системні конфігураційні значення, включаючи розміри сторінок пам’яті;
- «/proc/meminfo»: віртуальний файл у файловій системі /proc, який містить детальну інформацію про використання оперативної пам’яті, swap та інші параметри, зібрані ядром;

– «/proc/self/maps»: віртуальний файл, що відображає адресний простір поточного процесу, включаючи інформацію про завантажені бібліотеки, сегменти пам’яті (heap, stack) та їхні права доступу;

5.2.1.2 Скомпілюйте 2 екземпляри програми, відкрийте їх за допомогою налагоджувача та перегляньте адреси початку та кінця цих програм.

Для відлагодження буде використано програму «gf», яка є графічною обгорткою для налагоджувача GNU – «gdb».

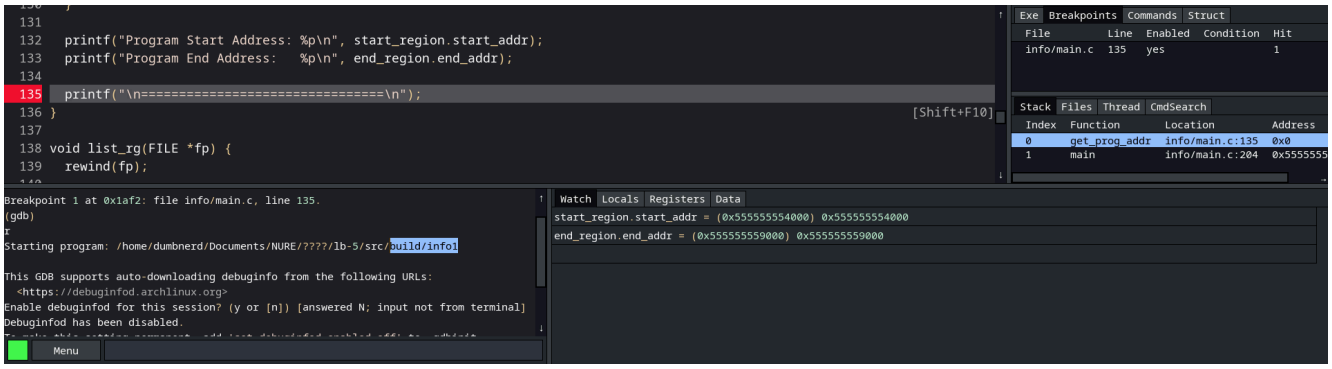

Рисунок 5.1 – Адреси початку та кінця програми 1

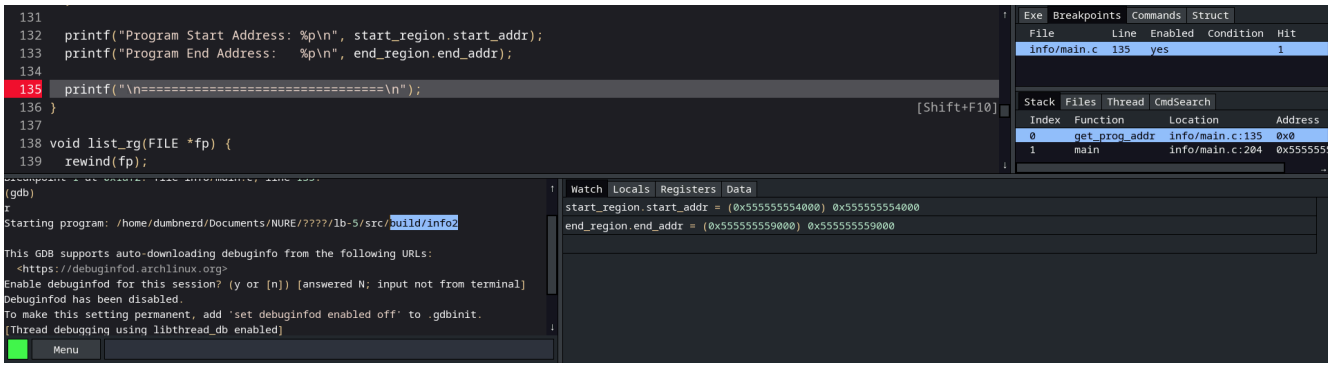

Рисунок 5.2 – Адреси початку та кінця програми 2

Можемо побачити, що адреси початку та кінця в обох програмах однакові. Така поведінка була очікуваною, адже налагоджувач «gdb» за замовчуванням вимикає ASLR[1] для зручності налагоджування. Увімкнемо ASLR в налагоджувачі за допомогою команди «set disable-randomisation off» та повторно виконаємо програми.

[1] ASLR (Address Space Layout Randomization) – технологія, при використанні якої розташування важливих структур даних випадковим чином змінюється в адресному просторі процесу.

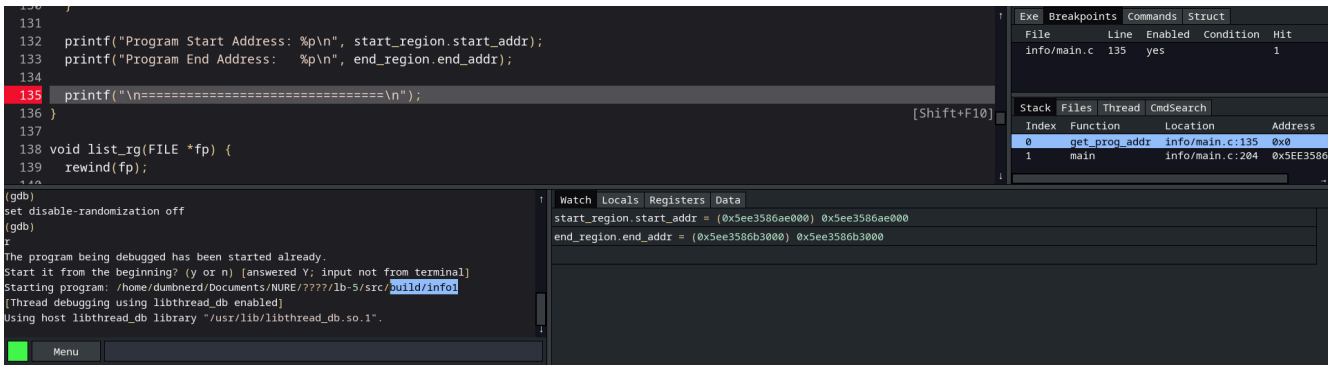

Рисунок 5.3 – Адреси початку та кінця програми 1 після увімкнення ASLR

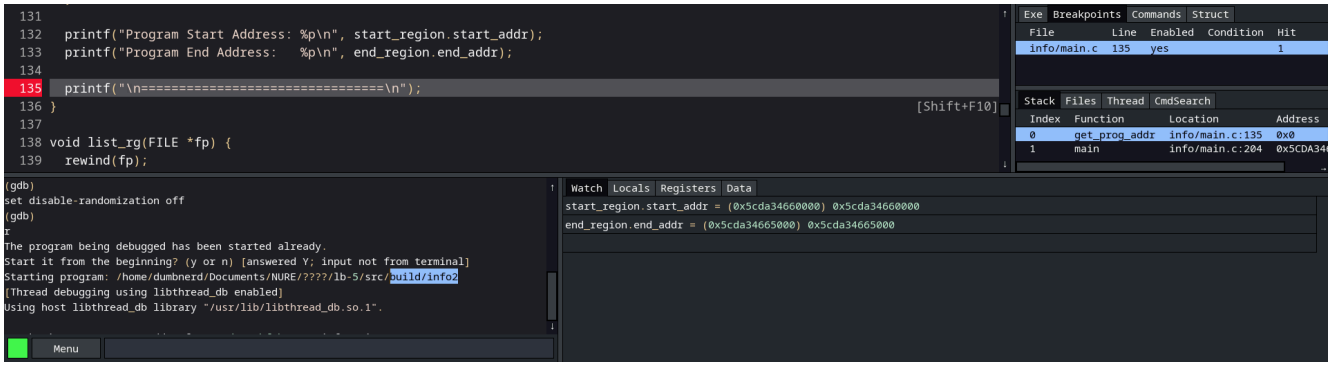

Рисунок 5.4 – Адреси початку та кінця програми 2 після увімкнення ASLR

Можемо побачити, що тепер в налагоджувачі відображаються різні адреси програм.

5.2.1.3 Побудуйте список блоків пам’яті доступних програмі, виділіть пам’ять, та побудуйте список блоків пам’яті знову. Дослідіть зміни в списку вільних блоків.

Linux, як і інші POSIX сумісні операційні системи, на відміну від Windows, не має окремого системного виклику для отримання інформації про доступні програмі блоки пам’яті, натомість цю інформацію можна отримати із файлу «/proc/self/maps» динамічної віртуальної файлової системи процесів. Оскільки програмі доступно доволі багато блоків пам’яті, наприклад блоки пам’яті, що належать завантаженим бібліотекам на кшталт «libc», виведемо лише блоки пам’яті, що належать безпосередньо програмі, та спеціальні блоки.

```
── Memory Region № 5 ──
Start Address: 0x607a079c0000
End Address:   0x607a079c1000
Size:          4096 bytes (0.00 MB)
Permissions:   rw-p
Offset:        0x3000
Device:        103:03
Inode:         3801557
Pathname:      /home/dumbnerd/Documents/NURE/OC/lb-5/src/build/info1
──────────────────────────
── Memory Region № 6 ──
Start Address: 0x607a3542e000
End Address:   0x607a3544f000
Size:          135168 bytes (0.13 MB)
Permissions:   rw-p
Offset:        0x0
Device:        00:00
Inode:         0
Pathname:      [heap]
──────────────────────────
── Memory Region № 7 ──
Start Address: 0x7a8380769000
End Address:   0x7a838076c000
Size:          12288 bytes (0.01 MB)
Permissions:   rw-p
Offset:        0x0
Device:        00:00
Inode:         0
Pathname:      [anonymous]
──────────────────────────
```

Рисунок 5.5 – Частина блоків пам’яті, доступних програмі, до виділення

```
── Memory Region № 5 ──
Start Address: 0x607a079c0000
End Address:   0x607a079c1000
Size:          4096 bytes (0.00 MB)
Permissions:   rw-p
Offset:        0x3000
Device:        103:03
Inode:         3801557
Pathname:      /home/dumbnerd/Documents/NURE/OC/lb-5/src/build/info1
────────────────────────
── Memory Region № 6 ──
Start Address: 0x607a3542e000
End Address:   0x607a3544f000
Size:          135168 bytes (0.13 MB)
Permissions:   rw-p
Offset:        0x0
Device:        00:00
Inode:         0
Pathname:      [heap]
────────────────────────
── Memory Region № 7 ──
Start Address: 0x7a837fc00000
End Address:   0x7a8380600000
Size:          10485760 bytes (10.00 MB)
Permissions:   rw-p
Offset:        0x0
Device:        00:00
Inode:         0
Pathname:      [anonymous]
────────────────────────
```

Рисунок 5.6 – Частина блоків пам’яті, доступних програмі, після виділення

Після виділення 10MB пам’яті можемо побачити, що в списку з’явився новий блок.

5.2.2 Розробка програми «MemAlloc». Виконання пункту 8 методичних вказівок.

Метою даного завдання є створення аллокатора, який буде виділяти пам’ять за стратегією «Найменший достатній».

Код складеної програми розміщено в додатку В.

5.2.2.1 Принцип роботи алгоритму

Даний код реалізує механізм керування пам’яттю за стратегією «Найменший достатній» (Best-Fit). Суть цієї стратегії полягає у виділенні для запиту такого вільного блоку пам’яті, розмір якого найближчий до запитуваного (але не менший).

- Ініціалізація. При створенні аллокатора за допомогою функції «bf_allocator_create» виділяється буфер пам’яті заданого розміру, який вирівнюється за вимогами системи. Спочатку цей буфер представляється як один великий вільний блок.
- Виділення пам’яті. Функція «bf_allocator_alloc» реалізує стратегію Best-Fit:
    а) проходить весь список блоків і шукає найменший вільний блок, розмір якого достатній для задоволення запиту;
    б) якщо такий блок знайдено, перевіряється, чи достатньо в ньому місця для розділення на два блоки (виділений і залишок);
    в) якщо розділення доцільне, створюється новий блок для залишку пам’яті та оновлюються зв’язки в списку;
    г) обраний блок позначається як зайнятий і повертається покажчик на його дані.
- Звільнення пам’яті. Функція «bf_allocator_free» звільняє раніше виділений блок пам’яті:
    а) за адресою знаходить відповідний блок і позначає його як вільний;
    б) перевіряє, чи можливе злиття з сусідніми вільними блоками, щоб запобігти фрагментації;
    в) якщо наступний блок вільний, він об’єднується з поточним;

г) якщо попередній блок вільний, поточний об’єднується з попереднім.

– Знищення аллокатора. Після завершення роботи з аллокатором необхідно викликати функцію «bf_allocator_destroy» яка знищує аллокатор і звільняє всю пам’ять, пов’язану з ним.

5.2.2.2 Особливості реалізації

а) Вирівнювання пам’яті забезпечується для коректної роботи на різних архітектурах;
б) реалізоване злиття суміжних вільних блоків для боротьби з фрагментацією;
в) використовується мінімальний поріг розміру для розділення блоків, щоб уникнути створення занадто малих блоків;
г) додана функція відображення стану пам’яті для налагодження та аналізу.

Ця реалізація Best-Fit є оптимальною з точки зору використання пам’яті, оскільки мінімізує «відходи» – різницю між розміром виділеного блоку та запитаним розміром. Однак, це досягається ціною повного перебору списку вільних блоків, що може бути неефективним при великій кількості блоків.

5.2.2.3 Приклад роботи

```
~/D/N/O/l/src ❯❯❯ just run-mem-alloc
After initial allocations (100,200,300):
Block list:
  Block at 0x557b61e222a0 | size: 112 bytes | ALLOCATED
  Block at 0x557b61e22330 | size: 208 bytes | ALLOCATED
  Block at 0x557b61e22420 | size: 304 bytes | ALLOCATED
  Block at 0x557b61e22570 | size: 272 bytes | FREE

After freeing 100- and 200-byte blocks:
Block list:
  Block at 0x557b61e222a0 | size: 352 bytes | FREE
  Block at 0x557b61e22420 | size: 304 bytes | ALLOCATED
  Block at 0x557b61e22570 | size: 272 bytes | FREE

Requesting 150 bytes: (best-fit should pick the 200-byte block)
Block list:
  Block at 0x557b61e222a0 | size: 352 bytes | FREE
  Block at 0x557b61e22420 | size: 304 bytes | ALLOCATED
  Block at 0x557b61e22570 | size: 160 bytes | ALLOCATED
  Block at 0x557b61e22630 | size: 80 bytes | FREE

b2 was at 0x557b61e22350, b4 is at 0x557b61e22590, diff = 576 bytes
```

Рисунок 5.7 – Приклад роботи алокатора

5.2.3 Розробка системи «Calc». Виконання завдання на найвищу оцінку.

Метою даного завдання є створити 3 програми для виконання простих математичних операцій над числами використовуючи певні засоби взаємодії між процесами.

Код складених програм розміщено в додатку Г.

Система складається з 3 програм:

– «NumWriter»: програма, що питає у користувача 2 числа;
– «OpWriter»: програма, що питає у користувача яку операцію необхідно виконати над числами;

– «Calc»: програма, яка виконує над вказаними числами вказану операцію, та виводить результат в консоль.

Для зручного обміну даними створено структуру, що зберігає числа, операцію та має додаткове бітове поле, в якому програми встановлюють певні прапорці після виконання своїх дій. «OpWriter» чекає на сигнал від «NumWriter», а «Calc» чекає на сигнали від «OpWriter» та «NumWriter».

Кожна з програм відкриває один і той самий файл, та створює його проекцію в оперативну пам’ять, після чого в цю область пам’яті записується структура даних, створена для обміну даними між процесами. По завершенню своєї частини роботи кожна програма встановлює прапорець готовності в структурі, після чого наступна програма приступає до виконання своєї частини.

5.3 Висновки

Під час даної лабораторної роботи ми дослідили особливості використання та функції для роботи з віртуальною та фізичною пам’яттю.

ДОДАТОК А

Вміст файлу «Justfile» із інструкціями збірки розроблених програм

```
cc := "clang"
cc_flags := "-Wall -Wextra -std=c11"

@clean:
  rm -rf build

@mkdir-build: clean
  mkdir -p build

@build-info: mkdir-build
    {{cc}} -ggdb -o build/info1 info/main.c
    {{cc}} -ggdb -o build/info2 info/main.c

@debug-info: build-info
    gf2 build/info1 & disown
    gf2 build/info2 & disown

@build-mem-alloc: mkdir-build
    {{cc}} -o build/mem-alloc mem-alloc/main.c mem-alloc/
best_fit_allocator.c

@run-mem-alloc: build-mem-alloc
    build/mem-alloc

@build-calc: mkdir-build
    {{cc}} -o build/calc calc/calc.c
    {{cc}} -o build/num-writer calc/num-writer.c
    {{cc}} -o build/op-writer calc/op-writer.c
```

ДОДАТОК Б

Код програми «Info»

Б.1 Вміст файлу «main.c»

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/sysinfo.h>
#include <unistd.h>

#define bool int
#define true 1
#define false 0

#define MAX_LINE_LENGTH 256
#define GB (1024 * 1024 * 1024)
#define MB (1024 * 1024)

typedef enum {
  RG_ALL,
  RG_PROGRAM_ONLY,
  RG_ANONYMOUS_ONLY,
  RG_ANONYMOUS_PROGRAM
} RG_MODE;

typedef struct {
  void *start_addr;
  void *end_addr;
  size_t size;
  char permissions[5];
  unsigned long offset;
  char device[8];
  unsigned long inode;
  char pathname[256];
} mem_rg;

void get_sys_mem_info() {
  printf("\n=== System Memory Information ===\n");

  struct sysinfo inf;
  sysinfo(&inf);

  printf("Total RAM:\t%5.2f GB\n", (float)(inf.totalram *
inf.mem_unit) / GB);
  printf("Free RAM:\t%5.2f GB\n", (float)(inf.freeram *
inf.mem_unit) / GB);
  printf("Shared RAM:\t%5.2f GB\n", (float)(inf.sharedram *
inf.mem_unit) / GB);
  printf("Buffer RAM:\t%5.2f GB\n", (float)(inf.bufferram *
inf.mem_unit) / GB);
  printf("Total Swap:\t%5.2f GB\n", (float)(inf.totalswap *
inf.mem_unit) / GB);
  printf("Free Swap:\t%5.2f GB\n", (float)(inf.freeswap *
inf.mem_unit) / GB);
  printf("Number of processes: %d\n", inf.procs);
```

```
  printf("\n=== Additional Memory Info (/proc/meminfo) ===\n");

  FILE *proc_meminfo = fopen("/proc/meminfo", "r");

  char line[MAX_LINE_LENGTH];
  while (fgets(line, MAX_LINE_LENGTH, proc_meminfo) != NULL) {
    if (strstr(line, "MemTotal:") || strstr(line, "MemFree:") ||
        strstr(line, "MemAvailable:") || strstr(line, "Buffers:") ||
        strstr(line, "Cached:") || strstr(line, "SwapTotal:") ||
        strstr(line, "SwapFree:") || strstr(line, "Dirty:") ||
        strstr(line, "Writeback:") || strstr(line, "Shmem:")) {
      printf("%s", line);
    }
  }

  fclose(proc_meminfo);

  printf("\n=== Memory Pages Information ===\n");

  long p_size = sysconf(_SC_PAGESIZE);
  long p_phys = sysconf(_SC_PHYS_PAGES);
  long p_avai = sysconf(_SC_AVPHYS_PAGES);

  printf("Page Size:              %8ld bytes\n", p_size);
  printf("Total Phys Pages:       %8ld\n", p_phys);
  printf("Available Phys Pages:   %8ld\n", p_avai);
  printf("Total Phys Memory:      %8.2f GB\n", (float)(p_phys *
p_size) / GB);
  printf("Available Phys Memory: %8.2f GB\n", (float)(p_avai *
p_size) / GB);
}

mem_rg parse_rg(char *line) {
  mem_rg region;
  memset(&region, 0, sizeof(region));

  sscanf(line, "%p-%p %4s %lx %7s %lu %255[^\n]", &region.start_addr,
         &region.end_addr, region.permissions, &region.offset,
region.device,
         &region.inode, region.pathname);

  region.size = (size_t)region.end_addr - (size_t)region.start_addr;

  return region;
}

int is_valid_rg(mem_rg *region, RG_MODE mode) {
  switch (mode) {
  case RG_PROGRAM_ONLY:
    return strstr(region->pathname, "info") != NULL ||
           region->pathname[0] == '\0';

  case RG_ANONYMOUS_ONLY:
    return strstr(region->pathname, "[heap]") != NULL ||
           strstr(region->pathname, "[stack]") != NULL;

  case RG_ANONYMOUS_PROGRAM:
```

```
      return strstr(region->pathname, "[heap]") != NULL ||
             strstr(region->pathname, "[stack]") != NULL ||
             strstr(region->pathname, "info") != NULL;

    case RG_ALL:
    default:
      return 1;
    }
}

void get_prog_addr(FILE *fp) {
  printf("\n=== Program Memory Addresses ===\n");

  rewind(fp);

  char line[MAX_LINE_LENGTH];
  fgets(line, MAX_LINE_LENGTH, fp);

  mem_rg start_region = parse_rg(line);
  mem_rg end_region = {0};

  while (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
    mem_rg curr_region = parse_rg(line);

    if (is_valid_rg(&curr_region, RG_PROGRAM_ONLY) &&
        curr_region.end_addr > end_region.end_addr)
      end_region = curr_region;
  }

  printf("Program Start Address: %p\n", start_region.start_addr);
  printf("Program End Address:   %p\n", end_region.end_addr);

  printf("\n================================\n");
}

void list_rg(FILE *fp) {
  rewind(fp);

  RG_MODE mode = RG_PROGRAM_ONLY;

  char line[MAX_LINE_LENGTH];
  int included_count = 0;
  int total_count = 0;

  while (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
    total_count++;

    mem_rg region = parse_rg(line);

    if (!is_valid_rg(&region, mode))
      continue;

    included_count++;

    printf("--- Memory Region №%2d ---\n", included_count);
    printf("Start Address: %p\n", region.start_addr);
    printf("End Address:   %p\n", region.end_addr);
    printf("Size:          %zu bytes (%.2f MB)\n", region.size,
```

```
            (float)region.size / MB);
    printf("Permissions:    %s\n", region.permissions);
    printf("Offset:         0x%lx\n", region.offset);
    printf("Device:         %s\n", region.device);
    printf("Inode:          %lu\n", region.inode);
    printf("Pathname:       %s\n",
           region.pathname[0] ? region.pathname : "[anonymous]");
    printf("-------------------------\n");
  }

  printf("Blocks Total:  %d\n", total_count);
  printf("Blocks Shown:  %d\n", included_count);
}

void alloc_cmp(FILE *fp) {
  printf("\n=== Allocating Memory ===\n");

  printf("Before allocation:\n");
  list_rg(fp);

  size_t allocation_size = 10 * MB;
  printf("\nAllocating %zu bytes (%.2f MB) of memory...\n",
allocation_size,
         (float)allocation_size / MB);

  void *memory = mmap(NULL, allocation_size, PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

  printf("Memory is allocated at %p\n", memory);

  printf("\nAfter allocation:\n");
  list_rg(fp);

  if (munmap(memory, allocation_size) == 0) {
    printf("\nMemory freed.\n");
  } else {
    perror("munmap");
  }
}

int main(int argc, char *argv[]) {
  get_sys_mem_info();

  FILE *fp = fopen("/proc/self/maps", "r");
  get_prog_addr(fp);

  alloc_cmp(fp);

  fclose(fp);

  return EXIT_SUCCESS;
}
```

ДОДАТОК В

Код програми «MemAlloc»

В.1 Вміст файлу «main.c»

```c
#include "best_fit_allocator.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

int main() {
  /* small pool for clarity */
  best_fit_allocator_t *pool = bf_allocator_create(1024);
  if (!pool) {
    fprintf(stderr, "Failed to create allocator\n");
    return 1;
  }

  /* Step 1: allocate three blocks */
  void *b1 = bf_allocator_alloc(pool, 100);
  void *b2 = bf_allocator_alloc(pool, 200);
  void *b3 = bf_allocator_alloc(pool, 300);
  printf("After initial allocations (100,200,300):\n");
  bf_allocator_print(pool);

  /* Step 2: free the first two */
  bf_allocator_free(pool, b1);
  bf_allocator_free(pool, b2);
  printf("\nAfter freeing 100- and 200-byte blocks:\n");
  bf_allocator_print(pool);

  /* Step 3: allocate 150 bytes—should go into the 200-byte hole */
  void *b4 = bf_allocator_alloc(pool, 150);
  printf("\nRequesting 150 bytes: (best-fit should pick the 200-byte
block)\n");
  bf_allocator_print(pool);

  /* Verify that b4 lies inside the old b2 region */
  size_t diff = (uintptr_t)b4 - (uintptr_t)b2;
  printf("\nb2 was at %p, b4 is at %p, diff = %zu bytes\n", b2, b4,
diff);

  bf_allocator_destroy(pool);
}
```

В.2 Вміст файлу «best_fit_allocator.h»

```c
#ifndef BEST_FIT_ALLOCATOR_H
#define BEST_FIT_ALLOCATOR_H

#include <stddef.h>

#ifdef __cplusplus
extern "C" {
#endif
```

```
typedef struct best_fit_allocator best_fit_allocator_t;

best_fit_allocator_t *bf_allocator_create(size_t pool_size);
void bf_allocator_destroy(best_fit_allocator_t *allocator);
void *bf_allocator_alloc(best_fit_allocator_t *allocator, size_t
size);
void bf_allocator_free(best_fit_allocator_t *allocator, void *ptr);
void bf_allocator_print(const best_fit_allocator_t *allocator);

#ifdef __cplusplus
}
#endif
#endif /* BEST_FIT_ALLOCATOR_H */
```

В.3 Вміст файлу «best_fit_allocator.c»

```
#include "best_fit_allocator.h"
#include <assert.h>
#include <stdalign.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

static size_t align_up(size_t n, size_t align) {
  return (n + align - 1) & ~(align - 1);
}

struct block {
  size_t size;
  int free;
  struct block *next;
  struct block *prev;
};

struct best_fit_allocator {
  void *buffer;
  size_t pool_size;
  struct block *head;
};

best_fit_allocator_t *bf_allocator_create(size_t pool_size) {
  if (pool_size <= sizeof(struct block))
    return NULL;
  size_t alignment = alignof(max_align_t);

  if (pool_size % alignment != 0) {
    pool_size = ((pool_size + alignment - 1) / alignment) *
alignment;
  }

  void *buf = aligned_alloc(alignment, pool_size);
  if (!buf)
    return NULL;
  best_fit_allocator_t *a = malloc(sizeof(*a));

  if (!a) {
```

```
    free(buf);
    return NULL;
  }

  a->buffer = buf;
  a->pool_size = pool_size;
  a->head = (struct block *)buf;
  a->head->size = pool_size - sizeof(struct block);
  a->head->free = 1;
  a->head->next = NULL;
  a->head->prev = NULL;

  return a;
}

void bf_allocator_destroy(best_fit_allocator_t *allocator) {
  if (!allocator)
    return;
  free(allocator->buffer);
  free(allocator);
}

void *bf_allocator_alloc(best_fit_allocator_t *allocator, size_t
size) {
  if (!allocator || size == 0)
    return NULL;

  size = align_up(size, alignof(max_align_t));
  struct block *best = NULL;

  for (struct block *it = allocator->head; it; it = it->next) {
    if (it->free && it->size >= size) {
      if (!best || it->size < best->size) {
        best = it;
        if (best->size == size)
          break;
      }
    }
  }

  if (!best)
    return NULL;

  if (best->size >= size + sizeof(struct block) +
alignof(max_align_t)) {
    uint8_t *raw = (uint8_t *)best;
    struct block *next_blk =
        (struct block *)(raw + sizeof(struct block) + size);

    next_blk->size = best->size - size - sizeof(struct block);
    next_blk->free = 1;
    next_blk->next = best->next;
    next_blk->prev = best;

    if (best->next)
      best->next->prev = next_blk;

    best->next = next_blk;
```

```
    best->size = size;
  }

  best->free = 0;
  return (uint8_t *)best + sizeof(struct block);
}

void bf_allocator_free(best_fit_allocator_t *allocator, void *ptr) {
  if (!allocator || !ptr)
    return;

  uint8_t *start = (uint8_t *)allocator->buffer + sizeof(struct
block);
  uint8_t *end = (uint8_t *)allocator->buffer + allocator->pool_size;

  assert(ptr >= (void *)start && ptr < (void *)end);
  struct block *blk = (struct block *)((uint8_t *)ptr - sizeof(struct
block));
  assert(!blk->free);
  blk->free = 1;

  if (blk->next && blk->next->free) {
    blk->size += sizeof(struct block) + blk->next->size;
    blk->next = blk->next->next;
    if (blk->next)
      blk->next->prev = blk;
  }

  if (blk->prev && blk->prev->free) {
    blk->prev->size += sizeof(struct block) + blk->size;
    blk->prev->next = blk->next;
    if (blk->next)
      blk->next->prev = blk->prev;
  }
}

void bf_allocator_print(const best_fit_allocator_t *allocator) {
  if (!allocator)
    return;

  printf("Block list:\n");
  for (const struct block *it = allocator->head; it; it = it->next) {
    printf("  Block at %p | size: %zu bytes | %s\n", (void *)it, it-
>size,
           it->free ? "FREE" : "ALLOCATED");
  }
}
```

ДОДАТОК Г

Код програм «Calc», «NumWriter» та «OpWriter»

Г.1 Вміст файлу «shared.h»

```
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define SF_NAME "/tmp/calc_data"
#define SF_SIZE 256

typedef struct {
  float num1;
  float num2;
  char operation;
  int ready_flags; // 1 - nums-ready, 2 - op-ready
} calc_data;

#define err(exit_code, msg, line)
\
  do
{                                                                    \
    EXIT_CODE = exit_code;
\
    char buf[256];
\
    snprintf(buf, sizeof(buf), "%s:%d: error: %s", __FILE__, line,
msg);        \
    perror(buf);
\
  } while (0)

#define SPINNER "|/-\\"
#define spin(msg, delay_ms)
\
  do
{                                                                    \
    static int i = 0;
\
    printf("\r%s %c ", msg, SPINNER[i % (sizeof(SPINNER) - 1)]);
\
    fflush(stdout);
\
    usleep(delay_ms * 1000);
\
    i++;
\
  } while (0)
```

Г.2 Вміст файлу «calc.c»

```
#include <fcntl.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>

#include "shared.h"

int EXIT_CODE = 0;

int main() {
  int fd;
  calc_data *data;

  fd = open(SF_NAME, O_CREAT | O_RDWR, 0666);
  if (fd == -1) {
    err(EXIT_FAILURE, "open", __LINE__ - 1);
    goto exit;
  }

  if (ftruncate(fd, SF_SIZE) == -1) {
    err(EXIT_FAILURE, "ftruncate", __LINE__ - 1);
    goto fd_close;
  }

  data = (calc_data *)mmap(NULL, SF_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED,
                           fd, 0);
  if (data == MAP_FAILED) {
    err(EXIT_FAILURE, "mmap", __LINE__ - 1);
    goto fd_close;
  }

  while (!(data->ready_flags & 1))
    spin("Waiting for numbers", 250);
  printf("\rNumbers are '%.2f' and '%.2f'.\n", data->num1, data-
>num2);

  while (!(data->ready_flags & 2))
    spin("Waiting for operation", 250);
  printf("\rOperation is '%c'.       \n", data->operation);

  double result = 0;

  switch (data->operation) {
  case '+':
    result = data->num1 + data->num2;
    break;
  case '-':
    result = data->num1 - data->num2;
    break;
  case '*':
    result = data->num1 * data->num2;
    break;
  case '/':
    if (data->num2 != 0) {
      result = data->num1 / data->num2;
    } else {
      fprintf(stderr, "Division by zero is not allowed!\n");
      goto munmap;
```

```
    }
    break;
  default:
    fprintf(stderr, "Unknown operation: %c\n", data->operation);
  }

  printf("Result: %.2f %c %.2f = %.2f\n", data->num1, data-
>operation,
         data->num2, result);

munmap:
  munmap(data, SF_SIZE);

fd_close:
  close(fd);

  unlink(SF_NAME);
exit:
  return EXIT_CODE;
}
```

Г.3 Вміст файлу «num-writer.c»

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>

#include "shared.h"

int EXIT_CODE = 0;

int main() {
  int fd;
  calc_data *data;

  fd = open(SF_NAME, O_CREAT | O_RDWR, 0666);
  if (fd == -1) {
    err(EXIT_FAILURE, "open", __LINE__ - 1);
    goto exit;
  }

  if (ftruncate(fd, SF_SIZE) == -1) {
    err(EXIT_FAILURE, "ftruncate", __LINE__ - 1);
    goto fd_close;
  }

  data = (calc_data *)mmap(NULL, SF_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED,
                           fd, 0);
  if (data == MAP_FAILED) {
    err(EXIT_FAILURE, "mmap", __LINE__ - 1);
    goto fd_close;
  }

  printf("Enter number 1: ");
```

```
  if (scanf("%f", &data->num1) != 1) {
    err(EXIT_FAILURE, "Getting number 1", __LINE__ - 1);
    goto munmap;
  }

  printf("Enter number 2: ");
  if (scanf("%f", &data->num2) != 1) {
    err(EXIT_FAILURE, "Getting number 2", __LINE__ - 1);
    goto munmap;
  }

  __sync_or_and_fetch(&data->ready_flags, 1);

  printf("Numbers '%.2f' and '%.2f' are written to the shared memory.
\n",
         data->num1, data->num2);

munmap:
  munmap(data, SF_SIZE);

fd_close:
  close(fd);

exit:
  return EXIT_CODE;
}
```

Г.4 Вміст файлу «op-writer.c»

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <unistd.h>

#include "shared.h"

int EXIT_CODE = 0;

int main() {
  int fd;
  calc_data *data;

  fd = open(SF_NAME, O_CREAT | O_RDWR, 0666);
  if (fd == -1) {
    err(EXIT_FAILURE, "open", __LINE__ - 1);
    goto exit;
  }

  if (ftruncate(fd, SF_SIZE) == -1) {
    err(EXIT_FAILURE, "ftruncate", __LINE__ - 1);
    goto fd_close;
  }

  data = (calc_data *)mmap(NULL, SF_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED,
                           fd, 0);
```

```
  if (data == MAP_FAILED) {
    err(EXIT_FAILURE, "mmap", __LINE__ - 1);
    goto fd_close;
  }

  while (!(data->ready_flags & 1))
    spin("Waiting for numbers", 250);

  printf("\rEnter operation (+, -, *, /):   ");
  if (scanf(" %c", &data->operation) != 1) {
    err(EXIT_FAILURE, "Getting operation", __LINE__ - 1);
    goto munmap;
  }

  __sync_or_and_fetch(&data->ready_flags, 2);

  printf("Operation '%c' is written to the shared memory.\n", data-
>operation);

munmap:
  munmap(data, SF_SIZE);

fd_close:
  close(fd);

exit:
  return EXIT_CODE;
}
```