Дослідження, моделювання та нестандартні підходи до аналізу процесів, файлових систем, безпеки та ресурсів в Linux

1. Теорія

Системне програмування в Linux охоплює розробку програм, які взаємодіють безпосередньо з ядром операційної системи, апаратними ресурсами та низькорівневими сервісами. Цей підхід виходить за межі використання високорівневих інструментів (таких як ps, top, cron, strace) і вимагає глибокого розуміння механізмів ядра, системних викликів (syscalls), файлових дескрипторів, прав доступу, планування процесів, керування пам'яттю та міжпроцесної взаємодії.

1.1. Моделювання черги задач (Job Queue) без потоків і сигналів

Черга задач (job queue) - це структура, яка дозволяє відкладене або послідовне виконання задач у порядку їх надходження. Зазвичай у таких системах реалізується за допомогою функцій:

- enqueue (додавання задачі),
- dequeue (виконання задачі),
- pause (тимчасове зупинення виконання),
- resume (відновлення),
- cancel (скасування задачі).

У класичних реалізаціях використовуються багатопоточність або сигнали (наприклад, SIGSTOP, SIGCONT), але в цьому завданні заборонено їх використання. Це вимагає симуляції через структуровану логіку виконання в одному потоці — наприклад, циклічну перевірку станів задач.

Приклад

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


 typedef enum { PENDING, RUNNING, PAUSED, CANCELLED, COMPLETED } JobState;
 typedef struct {
```

```
    int id;
    JobState state;
    void (*function)(void);
} Job;
#define MAX_JOBS 10
Job queue[MAX_JOBS];
int job_count = 0;

void dummy_task() {
    for (int i = 0; i < 5; ++i) {
        printf("Running task iteration %d...\n", i + 1);
        sleep(1);
    }
}

void enqueue(void (*func)(void)) {
    if (job_count < MAX_JOBS) {
        queue[job_count++] = (Job){ .id = job_count, .state = PENDING, .function = func };
    }
}

void run_queue() {
    for (int i = 0; i < job_count; ++i) {
        Job* job = &queue[i];
        if (job->state == PENDING) {
            job->state = RUNNING;
            job->function();
            job->state = COMPLETED;
        }
    }
}

int main() {
    enqueue(dummy_task);
    run_queue();
    return 0;
}
```

2. Практика

2.1 Загальні завдання

Задача 1

Використайте popen(), щоб передати вивід команди rwho (команда UNIX) до more (команда UNIX) у програмі на C.

Задача 2

Напишіть програму мовою C, яка імітує команду ls -l в UNIX — виводить список усіх файлів у поточному каталозі та перелічує права доступу тощо.
(Варіант вирішення, що просто виконує ls -l із вашої програми, — не підходить.)

Задача 3

Напишіть програму, яка друкує рядки з файлу, що містять слово, передане як аргумент програми (проста версія утиліти grep в UNIX).

Задача 4

Напишіть програму, яка виводить список файлів, заданих у вигляді аргументів, з зупинкою кожні 20 рядків, доки не буде натиснута клавіша (спрощена версія утиліти more в UNIX).

Задача 5

Напишіть програму, яка перелічує всі файли в поточному каталозі та всі файли в підкаталогах.

Задача 6

Напишіть програму, яка перелічує лише підкаталоги у алфавітному порядку.

Задача 7

Напишіть програму, яка показує користувачу всі його/її вихідні програми на C, а потім в інтерактивному режимі запитує, чи потрібно надати іншим дозвіл на читання (read permission); у разі ствердної відповіді — такий дозвіл повинен бути наданий.

Задача 8

Напишіть програму, яка надає користувачу можливість видалити будь-який або всі файли у поточному робочому каталозі. Має з’являтися ім’я файлу з запитом, чи слід його видалити.

Задача 9

Напишіть програму на C, яка вимірює час виконання фрагмента коду в мілісекундах.

Задача 10

Напишіть програму мовою C для створення послідовності випадкових чисел з плаваючою комою у діапазонах:

(a) від 0.0 до 1.0

(b) від 0.0 до n, де n — будь-яке дійсне число з плаваючою точкою.

Початкове значення генератора випадкових чисел має бути встановлене так, щоб гарантувати унікальну послідовність.

Примітка: використання прапорця -Wall під час компіляції є обов'язковим.

Завдання по варіантах:

1. Напишіть програму, яка визначає структуру ієрархії директорій для поточного користувача, починаючи з кореня, виключаючи циклічні посилання, але зберігаючи символьні.

2. Реалізуйте утиліту командного рядка, яка виводить процеси, запущені лише з нестандартних шеллів, не використовуючи ps, top, htop.

3. Розробіть засіб, що аналізує, які файли у вказаній директорії змінювалися найчастіше за останні 7 днів, використовуючи лише доступні через стандартну бібліотеку функції C.

4. Напишіть програму, яка імітує роботу черги задач (job queue), з підтримкою паузи, скасування та відновлення, але без використання потоків або сигналів.

5. Створіть команду, яка виводить дерева викликів системних викликів (syscalls) під час виконання довільної програми без використання strace.

6. Напишіть інструмент, який змінює права доступу до файлів залежно від кількості їх використання іншими користувачами.

7. Змоделюйте файлову систему на основі одного великого двійкового файлу з підтримкою створення, видалення та читання файлів.

8. Реалізуйте функціонал cron, який не використовує жодного фону або демонів.

9. Розробіть сканер портів у локальній мережі, який не використовує сокети або будь-які мережеві бібліотеки напряму.

10. Створіть утиліту, яка виводить таблицю відкритих файлів усіх процесів у системі без доступу до /proc.

11. Розробіть програму, яка визначає найменш використовувані команди в історії користувача (.bash_history), не аналізуючи сам файл напряму.

12. Зробіть систему логування запусків програм, яка не використовує жодного лог-файлу.

13. Напишіть утиліту, яка визначає "аномальні" виконувані файли в системі.

14. Реалізуйте засіб, який дозволяє відновити вилучений файл, якщо доступ до диску не було втрачено повністю.

15. Створіть програму, яка виводить ієрархію груп користувача, включаючи перетин між ними.

16. Реалізуйте команду, яка визначає рівень фрагментації окремого файлу.

17. Напишіть програму, яка визначає найбільш "параноїдальну" конфігурацію безпеки у вашій системі.

18. Створіть команду, яка дає змогу "перемотати назад" виконання shell-команд (але не через історію).

19. Зробіть скрипт, який виявляє неочевидні залежності між файлами і процесами, що їх використовують.

20. Реалізуйте обмежувач використання CPU для вибраного процесу без використання nice, cgroups або taskset.

21. Напишіть утиліту, яка аналізує "заплутаність" структури директорій за спеціальним критерієм вашого вибору.

22. Розробіть "детектор сплячих процесів", які технічно активні, але нічого не роблять.

23. Створіть програму, яка моделює втрату зв’язку з файловою системою і відновлення після неї (у межах звичайного користувача).

24. Реалізуйте просту модель кешу з підтримкою TTL і LRU-алгоритмом, не використовуючи жодних сторонніх структур.

25. Напишіть програму, яка знаходить "найстаріший живий процес", не використовуючи ps або /proc.

26. Реалізуйте власну версію kill, яка працює без використання kill() або сигналів.

27. Створіть команду, яка знаходить конфігураційні файли з потенційно небезпечними параметрами, не спираючись на відомі шаблони.