ЛАБОРАТОРНАЯ РАБОТА №1

Тема: Сравнение парадигм конкурентности и параллелизма при разработке многопоточных программ в ОС Linux.

Цель: исследовать чувствительность вычислительной схемы из индивидуального задания к:

а) ситуациям конкурентности, когда несколько потоков разделяют одно процессорное ядро.
б) ситуациям параллелизма, когда каждый поток выполняется на отдельном ядре процессора (нет конкуренции за вычислительные ресурсы).

Цель работы обуславливает постановку и решение следующих **задач**:

1) Изучить основные принципы многопоточного программирования в ОС Linux, включая различия между конкурентностью и параллелизмом.
2) Получить навыки работы с POSIX Threads (pthread) и инструментами управления потоками, такими как sched_setaffinity и taskset.
3) Ознакомиться с механизмами планирования потоков, управления процессорным временем и анализа производительности многопоточных программ.
4) Выполнить индивидуальное задание, связанное с использованием POSIX Threads для реализации вычислительной задачи с контролируемым распределением потоков по процессорным ядрам. **Следует декомпозировать вычислительную задачу, вычленив сущности для потоковой обработки.**
5) Построить графики зависимости вычислительной эффективности программы от числа потоков для ситуаций (а) и (б), проанализировать накладные расходы, связанные с переключением контекста, оценить влияние гиперпоточности.

Ход лабораторной работы

1. Рассмотреть пример кода для сравнения ситуаций конкурентности и параллелизма.

```c
#define _GNU_SOURCE
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
#include <stdint.h>

#define NUM_THREADS 4
#define NUM_ITERATIONS 500000000

void *compute(void *arg) {
    volatile uint64_t sum = 0;
    for (uint64_t iter = 0; iter < NUM_ITERATIONS; iter++)
        sum +=  iter;
}

// Принудительное закрепление потока за конкретным ядром
void pin_thread_to_core(int core_id) {
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(core_id, &cpuset);
    pthread_t current_thread = pthread_self();
    pthread_setaffinity_np(current_thread, sizeof(cpu_set_t), &cpuset);
}
```

```c
void *compute_pinned(void *arg) {
    int core_id = (int)(long)arg;
    pin_thread_to_core(core_id);
    return compute(arg);
}

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

    pthread_t threads[NUM_THREADS];

    if (argc < 2) {
        fprintf(stderr, "Запуск: %s <mode>\n"
                        "Опции:\n 1 - конкурентность, одно ядро\n"
                                "2 - параллелизм, разные ядра\n", argv[0]);
        return EXIT_FAILURE;
    }

    int mode = atoi(argv[1]);

    printf("Старт: %s, %d поток(а)(ов)...\n",
           mode == 1 ? "конкурентность, одно ядро" :
                       "параллелизм, разные ядра", NUM_THREADS);

    for (size_t iter = 0; iter < NUM_THREADS; iter++)
        if (mode == 1)
            pthread_create(&threads[iter], NULL, compute, (void *)iter);
         else
            pthread_create(&threads[iter], NULL, compute_pinned, (void *)iter);

    for (size_t iter= 0; iter < NUM_THREADS; iter++)
        pthread_join(threads[iter], NULL);

    return EXIT_SUCCESS;
}
```

Компиляция:

```
gcc -pthread -o lab1.o lab1.c
```

Запуск в режиме конкурентности (все потоки на одном ядре, на нулевом):

```
time taskset -c 0 ./lab1.o 1
```

Запуск в режиме параллелизма (разные потоки на разных ядрах):

```
time ./lab1.o 2
```

Индивидуальные задания

Вычислить сумму, N подобрать эмпирически для обеспечения эффективной нагрузки в рамках вычислительного эксперимента. При необходимости предусмотреть проверку подвыражений на принадлежность областям допустимых значений.

1.	$S = \sum_{i=1}^{N} \frac{i^2 + \cos(i) \cdot e^i}{i + \sin(i) + 1}$	2.	$S = \sum_{i=1}^{N} \frac{\sin(i) + \cos(2i) + i^3}{\sqrt{i+1} + \ln(i+1)}$

3.	$S = \sum_{i=1}^{N} \frac{(i+1)! + \sin^2(i) \cdot e^{-i}}{(2i+1)^2 + \cos(i)}$	4.	$S = \sum_{i=1}^{N} \frac{\tan(i) + i^2 e^{-\cos(i)}}{(i+1)! + \ln(i+2)}$
5.	$S = \sum_{i=1}^{N} \frac{\sin(i) + \cos(i^2) + i^4}{\sqrt{(i+1)! + e^i} + \tan(i)}$	6.	$S = \sum_{i=1}^{N} \frac{\cos(i^3) + i^4 e^{-i} + \ln(i+1)}{\sqrt{i^2 + \tan(i) + 1} + i!}$
7.	$S = \sum_{i=1}^{N} \frac{e^{\sin(i)} + i^3}{(i+1)! + \ln(i+2) + \cos^2(i)}$	8.	$S = \sum_{i=1}^{N} \frac{\tan(i) + i^{\frac{3}{2}} e^{-\sin(i^2)}}{(i+2)! + \sqrt{i+3} + \cos(i)}$
9.	$S = \sum_{i=1}^{N} \frac{\cos(i^2) + \tan(\ln(i+1)) + e^{-i}}{\sqrt{(i+1)! + i^5 + \sin^2(i)}}$	10.	$S = \sum_{i=1}^{N} \frac{\sin(i) + i^{2.5} e^{-\tan(i)} + \cos(i^3)}{\sqrt{(i+2)! + e^i} + \tan^2(i)}$
11.	$S = \sum_{i=1}^{N} \frac{\sin(i^2) + i^{\frac{3}{2}} e^{-\cos(i)}}{(i+3)! + \sqrt{i^3 + \tan(i)}}$	12.	$S = \sum_{i=1}^{N} \frac{\tan(i) + e^{\sin(i^3)} + i^2}{(i+2)! + \ln(i+1) + \cos(i^4)}$
13.	$S = \sum_{i=1}^{N} \frac{\cos(e^{\sin(i)}) + i^3}{\sqrt{\tan(i^2) + \ln(i+2) + e^{-i}}}$	14.	$S = \sum_{i=1}^{N} \frac{\sin(\ln(i^2+1)) + e^{-\tan(i/2)}}{\cos^2(i^3 + e^{-\sqrt{i}}) + \ln(i+3)}$
15.	$S = \sum_{i=1}^{N} \frac{\sin(e^{-\tan(i)}) + \cos(\ln(i^3+1)) + i^{1.5}}{\tan(\sin^2(i+1) + e^{-i^2}) + \sqrt{\ln(i^2+2) + \cos^2(i)}}$	16.	$S = \sum_{i=1}^{N} \frac{\ln(e^{-\cos(i^2)} + \tan(\sin(i+1)))}{\sqrt{\tan^2(e^{-\ln(i+3)}) + \sin^2(i^3 + \cos(i))}}$
17.	$S = \sum_{i=1}^{N} \frac{\tan(e^{-\sin(i)}) + \cos(\ln(i^3+2))}{\sqrt{\sin^2(i^2 + \tan(i+1)) + e^{-i^3} + \ln(i+5)}}$	18.	$S = \sum_{i=1}^{N} \frac{\sin(\cos(e^{-i^2})) + i^{3/2} e^{-\tan(i^3)}}{\tan^2(i+1) + \sqrt{\ln(i^4+1) + e^{-i}}}$
19.	$S = \sum_{i=1}^{N} \frac{\cos^2(i^2 + \tan(e^{-i})) + \ln(\sin(i+1))}{\sqrt{e^{-\cos(i^3)} + \tan^2(i+2) + \ln(i+3)}}$	20.	$S = \sum_{i=1}^{N} \frac{\tan(e^{-\cos(i^2)}) + \ln(\sin(i^3+1))}{\sqrt{\cos^2(e^{-\tan(i+2)}) + \sin^2(i^4 + e^{-i})}}$
21.	$S = \sum_{i=1}^{N} \frac{\sin(e^{-\tan(i)}) + \cos(\ln(i^3 + e^{-i}))}{\sqrt{\tan(\sin^2(i+1)) + e^{-\cos(i^2+i)}} + \ln(i+4)}$	22.	$S = \sum_{i=1}^{N} \frac{\tan(\cos(\ln(i+1))) + e^{-\sin(i^3+i^2)}}{\sqrt{\cos^2(e^{-i^2}) + \ln(\tan(i+2) + e^{-i})}}$

Контрольные вопросы

1. Что такое конкурентность в многопоточных программах?
2. Чем конкурентность отличается от параллелизма?
3. Какие преимущества и недостатки у конкурентного выполнения потоков?
4. Какие преимущества и недостатки у параллельного выполнения потоков?
5. Как операционная система управляет потоками?
6. Какое влияние на производительность оказывает переключение контекста потоков?
7. Какие механизмы синхронизации используются для управления конкурентным доступом к ресурсам?
8. Как работает библиотека POSIX Threads (pthread)?
9. Как создать поток с помощью pthread в языке C?
10. Как происходит закрепление потока за конкретным процессорным ядром?
11. Какие системные вызовы используются для управления процессорной привязкой потоков?
12. Как использовать sched_setaffinity для управления распределением потоков по ядрам?
13. Как работает утилита taskset и зачем она нужна?
14. Почему в эксперименте используются команды taskset и sched_setaffinity?
15. Как можно измерить время выполнения многопоточной программы?
16. Какие накладные расходы возникают при конкурентном выполнении потоков?
17. Что такое гиперпоточность и как она влияет на производительность?
18. Как измерить влияние количества потоков на скорость выполнения программы?
19. Почему важно эмпирически подбирать N для эффективной загрузки процессора?
20. Как интерпретировать результаты сравнения конкурентности и параллелизма?
21. Почему не всегда линейное увеличение количества потоков ведет к линейному ускорению?
22. Как влияет планировщик ОС на многопоточные программы?
23. Какие существуют стратегии планирования потоков в Linux?
24. Как можно уменьшить накладные расходы, связанные с переключением контекста?
25. В каких ситуациях конкурентность выгоднее параллелизма?
26. В каких ситуациях параллелизм выгоднее конкурентности?
27. Как избежать ложного параллелизма при использовании потоков?
28. Какие факторы могут ограничивать эффективность параллельных вычислений?
29. Как проверить корректность работы многопоточной программы?
30. Как диагностировать проблемы производительности многопоточной программы?
31. Какова роль кэширования данных в многопоточных вычислениях?
32. Какие инструменты можно использовать для профилирования многопоточных программ?
33. Какие проблемы могут возникнуть при использовании pthread_setaffinity_np?
34. Как использовать htop и top для мониторинга многопоточных программ?
35. Как анализировать влияние гиперпоточности на производительность?
36. Как можно оптимизировать вычислительную схему для многопоточного выполнения?
37. Какие альтернативные библиотеки для работы с потоками существуют помимо pthread?
38. Почему многопоточное программирование не всегда ускоряет выполнение программы?