Лабораторная работа № 3 по курсу : Операционные системы

Выполнил студент группы М8О-206Б-17 МАИ *Новиков Павел Сергеевич.*

Условие

- Постановка задачи:

  Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

- Вариант задания

  Поиск длин кратчайших путей из одной вершины графа в другую.

Метод решения

Для поиск длин кратчайших путей был использован алгоритм Флойда-Уоршела для матриц смежности. Алгоритм имеет простую параллельную схему: на каждой итерации для каждой вершины параллельно запускается перебор кратчайших путей.

Для реализации многопоточных вычислений были использованы следующие функции:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
```

(для создания нового потока)
Аргумент pthread_t *thread – по этому адресу, в случае успешного создания потока записывается id потока
Аргумент const pthread_attr_t *attr – атрибуты потока. В случае если используются атрибуты по умолчанию, то можно передавать NULL
Аргумент start_routin – это функция, которая будет выполняться в новом потоке.
Аргумент *arg – собственно входные данные, передающиеся функции как единственный аргумент (для передачи нескольких аргументов придётся использовать структуру)

```
int pthread_join(pthread_t thread, void **retval);
```

(ожидает завершения потока обозначенного *thread*.)
При удачном завершении возвращает 0, ненулевое значение сигнализирует об ошибке.

Описание программы

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <pthread.h>
void print_matrix(int**, int);
void floyd(int**, int, int);
void *floyd_parallel(void*);
int min(int x, int y) {
    if (x < y) {
        return x;
    }
    return y;
}
typedef struct _args {
    int i, k;
    int** matrix;
    int n;
} Args;
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Use ./prog \"thread count\"\n");
        return 0;
    }
    int thread_count = atoi(argv[1]);
    if (thread_count < 1) {
        printf("Wrong thread count: %d\n", thread_count);
        return 0;
    }
    printf("Enter number of vertexes:\n");
    int n;
    scanf("%d", &n);
    if (n <= 0) {
        printf("Vertex count must be > 0\n");
        return 0;
    }
    printf("Enter adjency matrix:\n");
    int **adjency_matrix = malloc(n * sizeof(*adjency_matrix));
    for (int i = 0; i < n; ++i) {
        adjency_matrix[i] = malloc(n * sizeof(int));
        for (int j = 0; j < n; ++j) {
            scanf("%d", &adjency_matrix[i][j]);
            if (adjency_matrix[i][j] < 0)
                adjency_matrix[i][j] = INT_MAX / 2;
            if (i == j)
                adjency_matrix[i][j] = 0;
        }
    }
    floyd(adjency_matrix, n, thread_count);
    int u = 0, v = 0;
    do {
        --u;
        --v;
        if ((u >= 0) && (v >= 0) && (v < n) && (u < n)) {
            int path = adjency_matrix[u][v];
```

```
            if (path == INT_MAX) {
                printf("Path from v%d to v%d doesnt exist\n", u + 1, v + 1);
            } else {
                printf("Shoret path length from v%d to v%d equals: %d\n", u + 1, v + 1, path);
            }
        }
        printf("Enter start & end vertex to show shortest path length:\n");
    } while (scanf("%d%d", &u, &v) != EOF);
    for (int i = 0; i < n; ++i) {
        free(adjency_matrix[i]);
    }
    free(adjency_matrix);
}
void print_matrix(int** matrix, int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
    return;
}
void floyd(int** matrix, int n, int thread_count) {
    thread_count = min(thread_count - 1, n);
    pthread_t *t_id = malloc(thread_count * sizeof(pthread_t));
    Args *args = malloc(thread_count * sizeof(Args));
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < thread_count; ++i) {
            args[i].i = i;
            args[i].k = k;
            args[i].matrix = matrix;
            args[i].n = n;
            pthread_create(&t_id[i], NULL, floyd_parallel,(void*) &args[i]);
        }
        for (int i = thread_count; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j]);
            }
        }
        for (int i = 0; i < thread_count; ++i) {
            pthread_join(t_id[i], NULL);
        }
    }
    free(t_id);
    free(args);
    return;
}
void *floyd_parallel(void* args) {
    Args *ptr = (Args*)args;
    for (int j = 0; j < (ptr -> n); ++j) {
        (ptr -> matrix)[ptr -> i][j] = min((ptr -> matrix)[ptr -> i][j], (ptr -> matrix)[ptr -> i][ptr -> k
    }
    pthread_exit(0);
}
```

Тесты

```
tilt@tilt:~/os/lab03$ ./prog 10
Enter number of vertexes:
3
```

```
Enter adjency matrix:
0 2 3
2 0 6
3 6 0
Enter start & end vertex to show shortest path length:
1 2
Shoret path length from v1 to v2 equals: 2
Enter start & end vertex to show shortest path length:
2 3
Shoret path length from v2 to v3 equals: 5
Enter start & end vertex to show shortest path length:
1 3
Shoret path length from v1 to v3 equals: 3
Enter start & end vertex to show shortest path length:
```

Вывод strace

```
execve("./prog", ["./prog", "10"], 0x7fffe41883e8 /* 49 vars */) = 0
arch_prctl(0x3001 /* ARCH_??? */,
0x7fff960bb7b0) = -1 EINVAL (Invalid argument)
arch_prctl(ARCH_SET_FS, 0x7fa0c7ff3740) = 0
Enter number of vertexes:
3
Enter adjency matrix:
1 1 1
1 1 1
1 1 1
clone(child_stack=0x7fa0c7ff1fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c7ff29d0, tls=0

child_tidptr=0x7fa0c7ff29d0) = 18222
clone(child_stack=0x7fa0c77f0fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c77f19d0, tls=0
child_tidptr=0x7fa0c77f19d0) = 18223
clone(child_stack=0x7fa0c6fd5fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c6fd69d0, tls=0
child_tidptr=0x7fa0c6fd69d0) = 18224
clone(child_stack=0x7fa0c6fd5fb0,
```

```
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c6fd69d0, tls=0
child_tidptr=0x7fa0c6fd69d0) = 18225
clone(child_stack=0x7fa0c77f0fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c77f19d0, tls=0
child_tidptr=0x7fa0c77f19d0) = 18226
clone(child_stack=0x7fa0c7ff1fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c7ff29d0, tls=0
child_tidptr=0x7fa0c7ff29d0) = 18227
clone(child_stack=0x7fa0c7ff1fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c7ff29d0, tls=0
child_tidptr=0x7fa0c7ff29d0) = 18228
clone(child_stack=0x7fa0c77f0fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c77f19d0, tls=0
child_tidptr=0x7fa0c77f19d0) = 18229
clone(child_stack=0x7fa0c6fd5fb0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fa0c6fd69d0, tls=0
child_tidptr=0x7fa0c6fd69d0) = 18230
Enter start & end vertex to show shortest path length:
1 5
Enter start & end vertex to show shortest path length:
exit_group(0)                           = ?
+++ exited with 0 +++
```

Выводы

Многопоточность полезна при выполнении независимых расчетов, но зачастую параллельная схема алгоритма бывает сложна и неочевидна. Многопоточность имеет большое значения для клиент-серверных приложений: каждого пользователя можно обработать в отдельном потоке, в итоге пользователи не будут становится в длинную очередь последовательной обработки запросов.