Условие:

Вам предоставлен исходный текст программы (main.c).

1.Определите и перечислите факторы, влияющие на производительность.

2.Минимизируйте количество системных вызовов, попытайтесь ускорить работу программы. Перечислить какие вызовы были удалены и почему? Предоставьте исправленный вариант программы. Предоставьте методологию измерений и численные показатели ускорения.

3.Обеспечьте одновременную работу нескольких экземпляров данного приложения с одной SQLITE-базой не полагаясь на системный вызов fcntl (например, если в ядре системы некорректно реализованы блокировки файлов). Внесите исправления в оригинальный код программы. Задокументируйте использованные методы.

Время на реализацию — 6 часов.

main.c

```
/*
  * gcc main.c -o test -lpthread -luuid -lsqlite3
  */

#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <uuid/uuid.h>

#include <sqlite3.h>

sqlite3* db;

static int callback(void* NotUsed, int argc, char** argv, char** azColName)
{
	int i;
	for (i = 0; i < argc; i++)
	{
		printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
	}
	printf("\n");
	return 0;
}

static void* thread_routine(void* arg)
{
	uuid_t uuid;
	char str[37];
	char* zErrMsg = 0;
	int rc;
	int i;
	char sql[100];

	for (i = 0; i < 1000000; i++)
	{

		uuid_generate(uuid);
```

```
		uuid_unparse(uuid, str);

		printf("[tid = %lu] thread_routine1 i = %i UUID:[%s]\n", pthread_self(), (int)arg, str);

		snprintf(sql, 100, "INSERT INTO UUIDS (ID,UUID) VALUES ('%s', %i ); ", str, (int)arg);

		rc = SQLITE_BUSY;

		while (rc == SQLITE_BUSY)
		{
			rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
			usleep(100);
		}

		if (rc != SQLITE_OK)
		{
			fprintf(stderr, "SQL error: %s\n", zErrMsg);
			sqlite3_free(zErrMsg);
		}
		else
		{
			printf("Records created successfully\n");
		}
	}

	pthread_exit((void*)0);
	return 0;
}

int main(int argc, const char* argv[])
{
	int res = 0;
	int rc = 0;
	char* sql = 0;
	char* zErrMsg = 0;

	pthread_t thread1;
	pthread_t thread2;
	pthread_attr_t attr;

	res = pthread_attr_init(&attr);
	/* res = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED); */

	/* unlink("test.db"); */
	rc = sqlite3_open("test.db", &db);
	if (rc)
	{
		fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
		return EXIT_FAILURE;
	}
	else
	{
		printf("Opened database successfully\n");
	}

	sql = "CREATE TABLE UUIDS("
		"ID CHAR(50) PRIMARY KEY NOT NULL,"
		"UUID	INT);";

	rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
```

```
            fprintf(stderr, "SQL error: %s\n", zErrMsg);
            sqlite3_free(zErrMsg);
        }
        else
        {
            printf("Table created successfully\n");
        }

        if ((res = pthread_create(&thread1, &attr, thread_routine, 1) != 0))
        {
            printf("pthread_create failure failure: %d errno: %d:%s\n", res, errno, strerror(errno));
        }

        if ((res = pthread_create(&thread2, &attr, thread_routine, 2) != 0))
        {
            printf("pthread_create failure failure: %d errno: %d:%s\n", res, errno, strerror(errno));
        }

        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);

        sqlite3_close(db);

        return EXIT_SUCCESS;
}
```