Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 3
по курсу «Операционные системы»

Студент:	Обыденкова Ю. Ю.
Группа:	М8О-208Б-18
Вариант:	14
Преподаватель:	Соколов А.А.
Оценка:	
Дата:	

Москва, 2019

1. Постановка задачи

Вариант 14:

Рассчитать детерминант матрицы.

Операционная система: Unix.

Цель работы

Приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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.

2. Решения задачи

Поток выполнения — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени). В качестве аналогии потоки выполнения процесса можно уподобить нескольким вместе работающим поварам. Все они готовят одно блюдо, читают одну и ту же кулинарную книгу с одним и тем же рецептом и следуют его указаниям, причём не обязательно все они читают на одной и той же странице.

Программа получает на вход размер матрицы и генерирует ее, заполняя случайными числами от 0 до 9. Затем рекурсивно вычисляет определитель матрицы по формуле разложения по строке. Каждый этап рекурсии выполняется в отдельном потоке, таким образом производя параллельный подсчет.

Использованные системные вызовы:

`int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)` --- функция получает в качестве аргументов указатель на поток, переменную типа `pthread_t`, в которую, в случае удачного завершения сохраняет `id` потока. `pthread_attr_t` – атрибуты потока. В случае если

используются атрибуты по умолчанию, то можно передавать NULL. start_routin – это непосредственно та функция, которая будет выполняться в новом потоке. arg – это аргументы, которые будут переданы функции.

int pthread_join(pthread_t thread, void **value_ptr) --- откладывает выполнение вызывающего (эту функцию) потока, до тех пор, пока не будет выполнен поток thread. Когда pthread_join выполнилась успешно, то она возвращает 0. Если поток явно вернул значение (это то самое значение SUCCESS, из нашей функции), то оно будет помещено в переменную value_ptr.

Тесты программы:

Input	Output	Цель теста
./a.out 0	Insert the demention of matrix: 0 Insert matrix: Det is: 0.000000	Программа завершила работу, посчитав определитель матрицы 0
./a.out 2	Insert the demention of matrix: 2 Insert matrix: 3 2 2 1 Det is: -1.000000	Программа завершила работу, посчитав определитель матрицы 2x2
./a.out 3	Insert the demention of matrix: 3 Insert matrix: 2 4 3 3 2 2 0 1 4 Det is: -27.000000	Программа завершила работу, посчитав определитель матрицы 3x3
./a.out 4	Insert the demention of matrix: 4 Insert matrix: 2 4 4 4 2 2 0 2 1 4 2 1 4 3 1 2 Det is: 52.000000	Программа завершила работу, посчитав определитель матрицы 4x4

3. Руководство по использованию программы

Компиляция и запуск программного кода в *Ubuntu* :

gcc -pthread os3.c -lm

4. Листинг программы

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>
```

```
#include <time.h>
#include <malloc.h>

pthread_mutex_t mutex;

typedef struct {
      int **matrix; // matrix
      int size; // size
} compare;

void* determinant(void *together) {
      compare* tmp = (compare*) together;
      int i,j;
      double det = 0;
      pthread_t *array = malloc(sizeof(pthread_t) * tmp->size);
      compare   *mtarr = malloc(sizeof(compare)   * tmp->size);

      if (tmp->size == 1) {
            det = tmp->matrix[0][0];
      } else if (tmp->size == 2) {
            det = tmp->matrix[0][0] * tmp->matrix[1][1] - tmp->matrix[0][1] *
tmp->matrix[1][0];
      } else {

            for (i = 0; i < tmp->size; ++i) {
                  mtarr[i].matrix = (int **)malloc(sizeof(int *) * tmp->size);
                  mtarr[i].size = tmp->size - 1;

                  for (j = 0; j < tmp->size - 1; ++j) {
                        if (j < i)
                              mtarr[i].matrix[j] = tmp->matrix[j];
                        else
                              mtarr[i].matrix[j] = tmp->matrix[j + 1];
                  }

                  pthread_create(&array[i], NULL, determinant, mtarr + i);

            }

            for (i = 0; i < tmp->size; ++i) {
                  void *res;
                  for (j = 0; j < tmp->size - 1; ++j) {
                  }
                  pthread_join(array[i], &res);
                  double x = *(double *)&res;
                  det += (-1 + 2 * !(i % 2)) * x * tmp->matrix[i][tmp->size -
1];


                  double answer = *(double*)&det;
                  free(mtarr[i].matrix);
            }


      }
      free(mtarr);
      free(array);

      void* ans = *(void **)&det;
      return ans;
}


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

```
	srand(time(NULL));
	int **matrix;
	int n = 0;
	int a;

	pthread_t tid;
	pthread_attr_t attr;
	pthread_attr_init(&attr);

	printf("Insert the demention of matrix:\n");
	scanf("%d", &n);

	matrix = (int**)malloc(n * sizeof(int*));

	for (int i=0; i<n; ++i)
		matrix[i] = (int*)malloc(n * sizeof(int));


	printf("Insert matrix:\n");


	for (int i = 0; i < n; ++i) {
		for (int j = 0; j < n; ++j) {
			matrix[i][j]=0+rand()%5-1+1;
		}
	}

	for (int i = 0; i < n; ++i) {
		for (int j = 0; j < n; ++j) {
			printf("%d ", matrix[i][j]);
		}
		printf("\n");
	}


	compare* together = (compare*)malloc(sizeof(compare));
	together->matrix = matrix;
	together->size = n;

	void *det;
	pthread_mutex_init(&mutex, NULL);
	pthread_create(&tid, NULL, determinant, together);
	pthread_join(tid, &det);

	double answer = *(double*)&det;
	printf("Det is: %f\n", answer);

	for (int i = 0; i < n; ++i)
		free(matrix[i]);
	free(matrix);
	free(together);

	return 0;
}
```

5. Вывод

В данной лабораторной работе мы реализовали программу, работающую с потоками.

Многопоточность - полезная вещь в программировании. Она обладает следующими преимуществами:

• Упрощение программы в некоторых случаях за счёт использования общего адресного пространства.

• Меньшие относительно процесса временные затраты на создание потока.

• Повышение производительности процесса за счёт распараллеливания процессорных вычислений и операций ввода-вывода.