Министерство высшего образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)
Кафедра безопасности информационных систем (БИС)

ПОТОКИ

Отчет по лабораторной работе №5
по дисциплине «Системное программирование»

Студентка гр.737-1
_____ Агеева В.С.
__.__.2021г

Принял
Руководитель
доцент кафедры БИС
_____ Романов А.С.
_.__.2021г

Томск 2021

1 Введение

Цель работы: изучить работу с потоками. Научиться разбивать задачу на части, для последующего их выполнения различными потоками. Познакомиться с основными функциями WinAPI для работы с потоками в Windows и библиотекой Pthread для работы с потоками в Linux.

Задание:

1. Изучить краткие теоретические сведения и лекционный материал по теме практического задания.

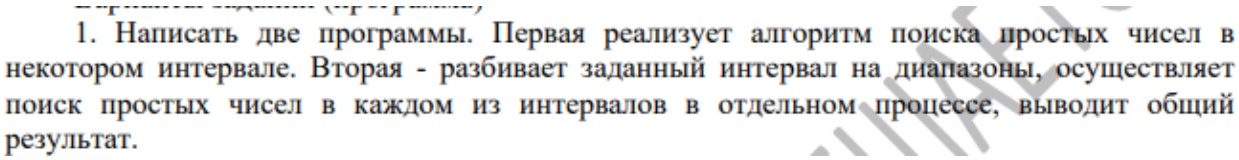
2. Реализовать приведенные примеры программы и продемонстрировать их работу.

3. Выбрать модель многопоточного приложения, наиболее точно отвечающую специфике задачи. Разработать алгоритм решения задания, с учетом разделения вычислений между несколькими потоками. Желательно избегать ситуаций изменения одних и тех же общих данных несколькими потоками. Если же избежать этого невозможно, необходимо использовать алгоритмы с активным ожиданием или неделимые операции.

4. Реализовать алгоритм с применением функций библиотеки Pthread.

5. Реализовать алгоритм с применением функций WinAPI.

6. Сравнить возможности обоих подходов, сделать выводы.



1. Написать две программы. Первая реализует алгоритм поиска простых чисел в некотором интервале. Вторая - разбивает заданный интервал на диапазоны, осуществляет поиск простых чисел в каждом из интервалов в отдельном процессе, выводит общий результат.

Рисунок 1.1 – Задание на лабораторную работу

2 Ход работы

Для выполнения лабораторной работы напишем код на C++, в котором объединены две программы из прошлой лабораторной работы (рисунок 2.1).

```
void* start_func (void *arg)
{
   int start = ((int*)arg)[0];
   int end = ((int*)arg)[1];
   int number_pthread = ((int*)arg)[2];

   int count = 0, flag;
   int tmp_arr[(end - start) / 2];

   for(int i = start ; i <= end ; i++)
   {
      flag = 1;
      for (int t = 2; t <= sqrt(i) ;t++)
      {
         if (i % t == 0) { flag = 0; break; }
      }
      if (flag)
      {
         tmp_arr[count] = i;
         count++;
      }
   }

   sem_wait(&empty);

   ptr_arr[0] = number_pthread;
   ptr_arr[1] = count;
   ptr_arr[2] = start;
   ptr_arr[3] = end;

  for (int i = 0; i < count; ++i)
  {
      ptr_arr[i + 4] = tmp_arr[i];
  }

   sem_post(&full);
   pthread_exit(NULL);
}
```

```
int main(int argc, char* argv[])
{
   int length, int_round, start_interval , end_interval, last;
   int rv , tmp;
   int max_number_pthread = 50;
   int base_length = 20;

   if (argc != 3) {
      printf("Not enought params\n");
      return 0;
   }

   sscanf(argv[1], "%d", &start_interval);
   sscanf(argv[2], "%d", &end_interval);

   if(start_interval > end_interval){
      tmp = end_interval;
      end_interval = start_interval;
      start_interval = tmp;
   }

   int_round = (end_interval - start_interval) / base_length;
   if(int_round > max_number_pthread) {
      int_round = max_number_pthread;
      length = (end_interval - start_interval) / int_round ;
   }
   else {
      length = base_length;
   }

   last = (end_interval - start_interval) % length;
   if (last != 0){
      int_round++;
   }
   int buf_size = (length / 2) * 4;
   int arr[buf_size];
   ptr_arr = arr;

   pthread_t tid[int_round];
   int in_arr[int_round][3];

   for (int i = 0; i < int_round; ++i)
   {
      int first = i * length + start_interval;
      int second = i * length + start_interval + length;
      if (second > end_interval){
         second = end_interval;
      }
      in_arr[i][0] = first;
      in_arr[i][1] = second;
      in_arr[i][2] = i;
   }
```

```
    if (sem_init(&empty, 0, 1) ||
        sem_init(&full, 0, 0)) {
        fprintf(stderr, "Failed to init semaphore\n");
        return EXIT_FAILURE;
    }

    for (int i = 0; i < int_round; i++)
    {
        int status = pthread_create( &(tid[i]), NULL, start_func, (void*)(in_arr[i]) );
    }

    printf("Lenghts of intervals %d\n", length );
    for (int i = 0; i < int_round; i++)
    {
        int count = 0;

        sem_wait(&full);

        printf("Thread with number %d found on interval form %d to %d:\n", ptr_arr[0], ptr_arr[2], ptr_arr[3] );
        count = ptr_arr[1];
        for (int i = 0; i < count; ++i)
        {
            printf(" %d ",ptr_arr[i + 4] );
        }
        printf("\n");
        printf("-------------------------------------------------\n");

        sem_post(&empty);
    }

    for (int i = 0; i < int_round; i++)
    {
        pthread_join(tid[i], NULL);
    }
    printf("%s\n", "All threads finished" );


    if (sem_destroy(&full) ||
        sem_destroy(&empty)) {
        fprintf(stderr, "Failed to destroy semaphore\n");
        return EXIT_FAILURE;
    }

    return 0;
}
```

Рисунок 2.1 – Код на языке C++

3 Заключение

В ходе выполнения лабораторной работы была изучена работа с потоками, научились разбивать задачу на части, для последующего их выполнения различными потоками.

Исходные коды программ и Dockerfile приложены в архиве к отчету, а также загружены на GitHub (https://github.com/7371avs/SP).