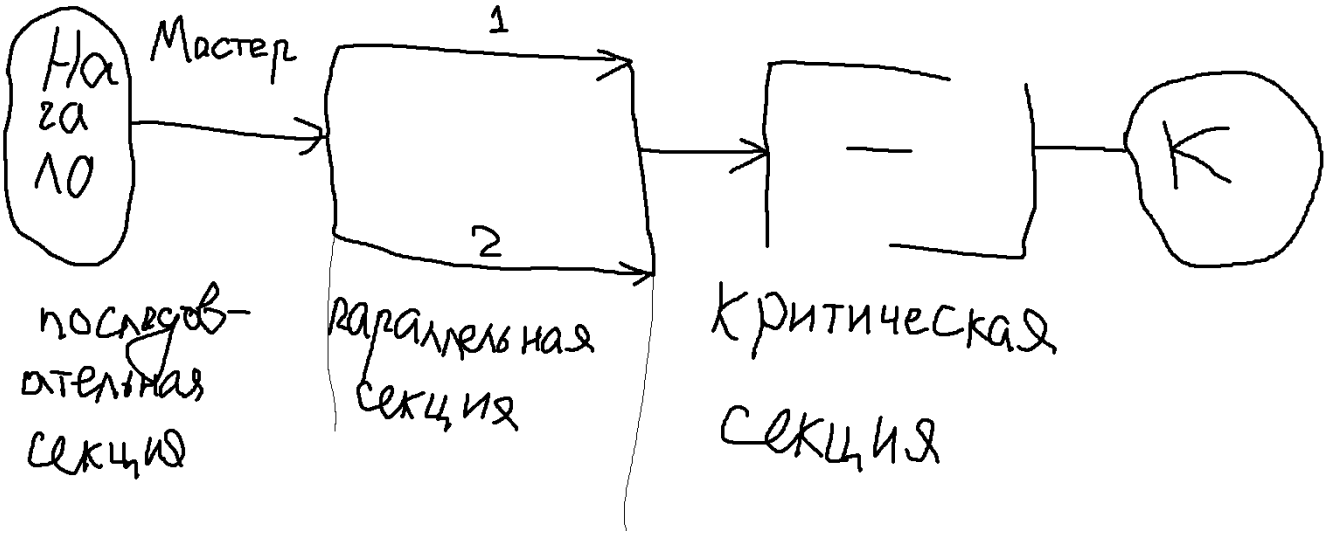
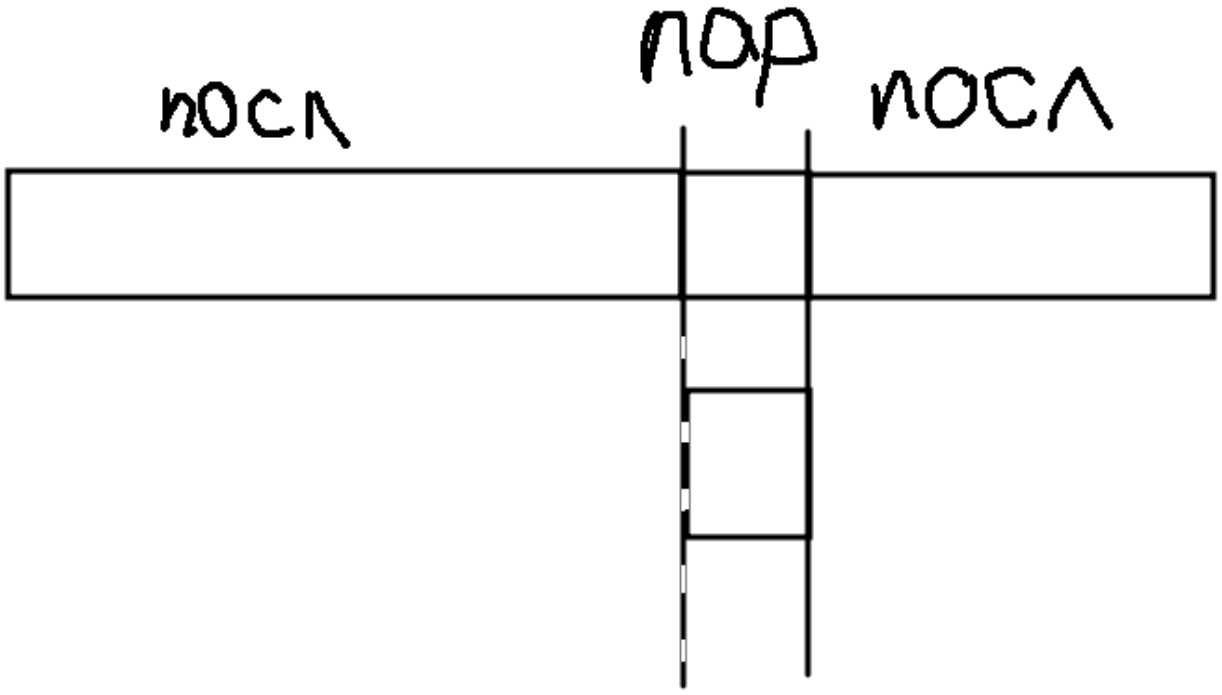
В параллельном сегменте нельзя завершать работу программы
В параллельном сегменте нельзя производить запись в одну и ту же область памяти.
Критическая секция - секция в параллельной секции, в которой все работы выполняются потоками последовательно друг за другом. В критическую секцию одновременно может зайти только один поток (остальные будут ждать). Семафоры, мьютексы (mutual exclusion)
Поток - это, по сути, ещё один экземпляр вашей программы (во время запуска нового потока с точки зрения ОС появляется новый процесс с теми же данными). Можно на него посмотреть через диспетчер задач.





Необходимо добавить компилятору openmp

Мой код в терминале:

```
-g++ -o output_file program.cpp -fopenmp
.\output_file
```

Код на плюсах:

```
#include <iostream>
#include <omp.h>

int main(){
#ifdef _OPENMP
    std::cout << "OpenMP is supported!" << std::endl;
#endif
#define N 1000000
    int pointer[N];
    for (int i = 0; i < N; i++){
        pointer[i]=1;
    }
    int sum = 0;
#pragma omp parallel for
    for (int i = 0; i < N; i++)
    {
        sum = sum + pointer[i];
    }
    std::cout << sum << std::endl;

    return 0;
}
```

Казалось бы, программа должна печатать 1000000, но на самом деле она выводит число значительно меньшее, чем 1000000. Это связано с тем, что когда разные потоки увеличивают значение sum на 1, то может произойти следующая ситуация:
Пусть значение `sum` было равно `100`.
Пусть это значение передалось в потоки 1 и 2. Каждый из поток считает, что значение `sum` до выполнения операций равняется `100`
Поток А обновляет значение `sum`, записав вместо `sum` значение `100+1=101`
Поток Б обновляет значение `sum`, записав вместо `sum` значение `100+1=101` - он не может узнать о том, что другой поток поменял значение `sum`
Если поменять

```
#pragma omp parallel for
```

на

```
#pragma omp parallel for reduction (+: sum)
```

То всё будет хорошо.
Можно также поменять

```
    #pragma omp parallel for
for (int i = 0; i < N; i++){
    sum = sum + pointer[i];
}
```

на

```
#pragma omp parallel for
for (int i = 0; i < N; i++){
#pragma omp critical
    sum += pointer[i];
}
```

2 задача - распараллелить перемножение 2 матриц при использовании обычного метода.

Мой код на C:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define A 32768
#define B 32768
#define C 32768
// D = 2*A*C + A*B + B*C
#define D 4294967296

int main(){

#ifdef _OPENMP
    puts("OpenMP is supported!");
#endif

    char* matrix1[A];
    char* matrix2[B];
    unsigned short* matrix3[A];
    puts("1");
    void* data_pointer = malloc(D);
```

```
    puts("2");
    void* current_pointer = data_pointer;
    for  (int i = 0; i < A; i++){
        matrix1[i] = current_pointer;
        current_pointer = current_pointer + B;
        for  (int j = 0; j < B; j++)
            matrix1[i][j] = 1;
    }
    for  (int i = 0; i < B; i++){
        matrix2[i] = current_pointer;
        current_pointer = current_pointer + C;
        for  (int j = 0; j < C; j++)
            matrix2[i][j] = 1;
    }

    for  (int i = 0; i < A; i++)
    {
        matrix3[i] = current_pointer;
        current_pointer = current_pointer + C*2;
        for  (int j = 0; j < C; j++)
            matrix3[i][j] = 0;
    }

    double tbegin = omp_get_wtime();

    #pragma omp parallel for
    for (int i = 0; i < A; i++){

        for (int j = 0; j < C; j++)
        {
            int result = 0;
            for (int k = 0; k < B; k++)
                result += matrix1[i][k] * matrix2[k][j];
            matrix3[i][j] = result;
        }
        printf("%d\n",i);
    }



    double time = omp_get_wtime() - tbegin;

    printf("%lf\n", time);
    free(data_pointer);
    puts("\7 23");
    return 0;
```

```
}
```

Код выглядит так странно, так как я попытался сделать матрицы максимально большими, использовав половину своей оперативной памяти. Если с памятью не заморачиваться, то аналогичный код на плюсах будет таким:

```
#include <omp.h>
#include <vector>
#include <iostream>


int main()
{
#ifdef _OPENMP
    std::cout << "OpenMP is supported!" << std::endl;
#endif
    const unsigned int A = 1000;
    const unsigned int B = 1000;
    const unsigned int C = 1000;
    using matrix = std::vector<std::vector<int>>;
    // матрицы A на B, B на C, A на C
    matrix ab(A, std::vector<int>(B, 1));
    matrix bc(B, std::vector<int>(C, 1));
    matrix ac(A, std::vector<int>(C, 0));

    double tbegin = omp_get_wtime();

    // ac = ab*bc
    #pragma omp parallel for
    for (int i = 0; i < A; i++)
        for (int j = 0; j < C; j++)
            for (int k = 0; k < B; k++)
                ac[i][j] += ab[i][k] * bc[k][j];

    double time = omp_get_wtime() - tbegin;
    std::cout << time << std::endl;
    return 0;
}
```

Можно итерировать по этим индексам по-разному: выше расположен порядок индексов ijk.
Все варианты итераций:
ijk

ikj
jik
jki
kij
kji

Например, в случае `jik` цикл будет выглядеть как

```
#pragma omp parallel for
 for (int j = 0; j < C; j++)
     for (int i = 0; i < A; i++)
         for (int k = 0; k < B; k++)
             ac[i][j] += ab[i][k] * bc[k][j];
```

Для каждого варианта надо написать свою программу и протестировать, сколько времени занимает подсчёт результата.