

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школы электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ
о лабораторной работе №2
по дисциплине «Технологии параллельного программирования»

Выполнил:
студент группы КЭ-220
_____/Голенищев А. Б.
_____ 2024 г.

Отчет принял:
_____/Жулев А. Э.
_____ 2024 г.

Челябинск
2024

Задание 4. Общие и частные переменные в OpenMP: программа «Скрытая ошибка»

Представлен вариант кода, где rank является частной переменной, листнинг 1.

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

// Golenishchev Artem, KE-220 Task 4

int main() {
    int k;

    // Ввод количества нитей
    printf("Enter number of threads (k): ");
    scanf("%d", &k);

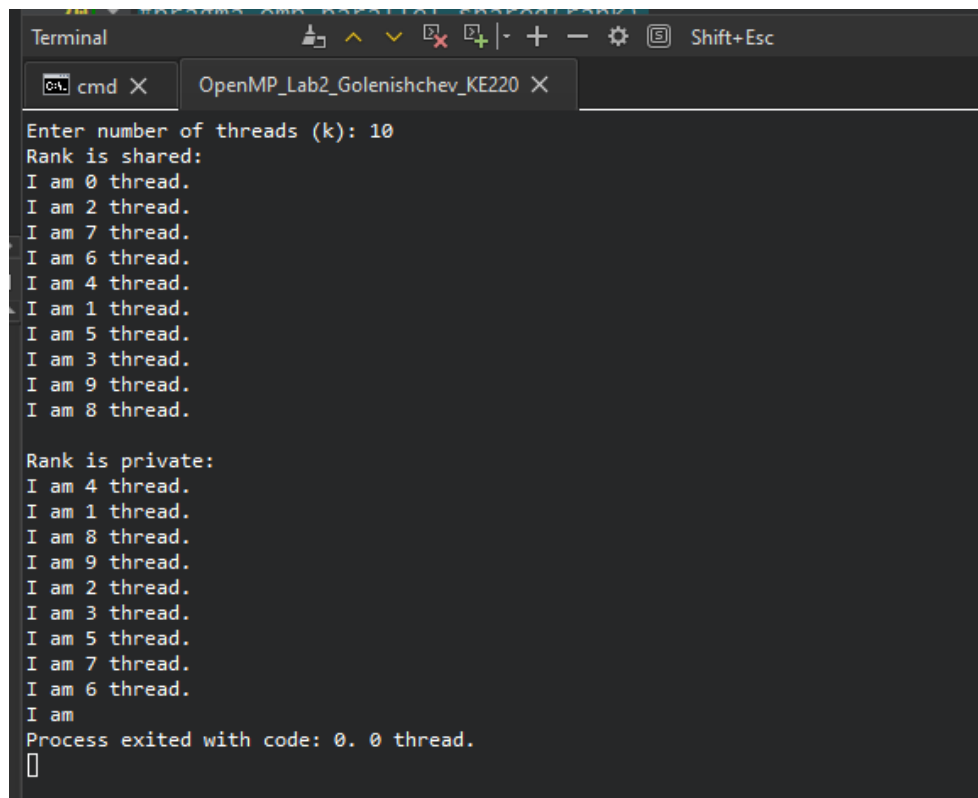
    // Установка количества нитей
    omp_set_num_threads(k);

    int rank;

    // Параллельная область с указанием области видимости
    printf("Rank is shared: \n");
    #pragma omp parallel shared(rank)
    {
        rank = omp_get_thread_num(); // Переменная rank общая для каждой нити
        printf("I am %d thread.\n", rank);
    }
    printf("\nRank is private: \n");
    rank = 0;
    #pragma omp parallel private(rank)
    {
        rank = omp_get_thread_num(); // Переменная rank уникальна для каждой нити
        printf("I am %d thread.\n", rank);
    }
    return 0;
}
```

Листнинг 1. Код, где rank имеет разные области видимости

Результат выполнения программы представлен на рисунке 1.



```
Terminal
cmd X OpenMP_Lab2_Golenishchev_KE220 X
Enter number of threads (k): 10
Rank is shared:
I am 0 thread.
I am 2 thread.
I am 7 thread.
I am 6 thread.
I am 4 thread.
I am 1 thread.
I am 5 thread.
I am 3 thread.
I am 9 thread.
I am 8 thread.

Rank is private:
I am 4 thread.
I am 1 thread.
I am 8 thread.
I am 9 thread.
I am 2 thread.
I am 3 thread.
I am 5 thread.
I am 7 thread.
I am 6 thread.
I am
Process exited with code: 0. 0 thread.
```

Рисунок 1. Результат выполнения программы в Qt Creator

На рисунке 1 видно, что в случае области видимости переменной `private(rank)` потоки с выводом через `printf()` одновременно работают, из-за чего мы наблюдаем некорректный вывод. Скрытая ошибка должна заключаться в повторении несколькими потоками одного и того же значения в выводе `rank` в случае области видимости переменной `shared(rank)`. Для поиска скрытой ошибки повторим запуск еще несколько раз, увеличивая значение `omp_set_num_threads` до количества потоков нашего процессора (24) или больше его. Кроме того, запускать код без изменений бесполезно, для фиксации момента повторений можно использовать `sleep()` или синхронизацию. Модифицируем код, листинги 2-3. Запустим варианты кода для поиска скрытой ошибки, рисунки 2-3. Суть скрытой ошибки заключается в том, что при многопоточности значение переменной `rank` успевает перезаписываться до вывода на экран (гонка данных). Первый способ с использованием `sleep()` позволяет зафиксировать гонку данных (ее наличие), второй способ позволяет дополнительно зафиксировать момент совпадения записанных значений.

```
#include <stdio.h>
#include <omp.h>
#include <windows.h> // Для использования Sleep()
// Golenishchev Artem, KE-220 Task 4
int main() {
    int k;
    // Ввод количества нитей
    printf("Enter number of threads (k): ");
    scanf("%d", &k);
    // Установка количества нитей
    omp_set_num_threads(k);
    int rank;
    // Параллельная область с указанием области видимости
    printf("Rank is shared: \n");
    #pragma omp parallel shared(rank)
    {
        rank = omp_get_thread_num(); // Переменная rank общая для всех потоков
        Sleep(100);                  // Имитация длительных вычислений (100 миллисекунд)
        printf("I am %d thread.\n", rank);
    }
    return 0;
}
```

Листнинг 2. Код, где мы ловим ошибку с помощью sleep()

A screenshot of a terminal window titled "Terminal". The title bar includes standard OS icons: a file icon, up/down arrows, a close button (X), a maximize button (+), and a zoom-in button (magnifying glass). Below the title bar are two tabs: "C:\ cmd X" and "OpenMP_Lab2_Golenishchev_KE220 X". The main area of the terminal displays the following output:

```
Enter number of threads (k): 24  
Rank is shared:  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
Process exited with code: 0..  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.  
I am 23 thread.
```


The cursor is visible at the end of the last line.

Рисунок 2. Результат выполнения программы с sleep()

```

#include <stdio.h>
#include <omp.h>
#include <stdbool.h>
// Golenishchev Artem, KE-220 Task 4
int main() {
    int k;
    // Ввод количества нитей
    printf("Enter number of threads (k): ");
    scanf("%d", &k);
    // Установка количества нитей
    omp_set_num_threads(k);
    int rank;
    bool stop = false;
    // Параллельная область с указанием области видимости
    printf("Rank is shared: \n");
#pragma omp parallel shared(rank, stop)
    {
        for (int i = 0; i < 1000; ++i) { // Ограничение на количество итераций для проверки
            if (stop) continue; // Если ошибка найдена, остальные потоки завершают
            выполнение
            rank = omp_get_thread_num();
#pragma omp critical // Синхронизация проверки
            {
                static int previous_rank = -1;
                static bool first_check = true;

                if (!first_check && previous_rank == rank) {
                    printf("Duplicate rank detected: %d. Stopping...\n", rank);
                    stop = true;
                }
                previous_rank = rank;
                first_check = false;
            }
            if (!stop) {
                printf("I am %d thread.\n", rank);
            }
        }
    }
    if (!stop) {
        printf("\nNo duplicate ranks detected in shared mode.\n");
    }
    return 0;
}

```

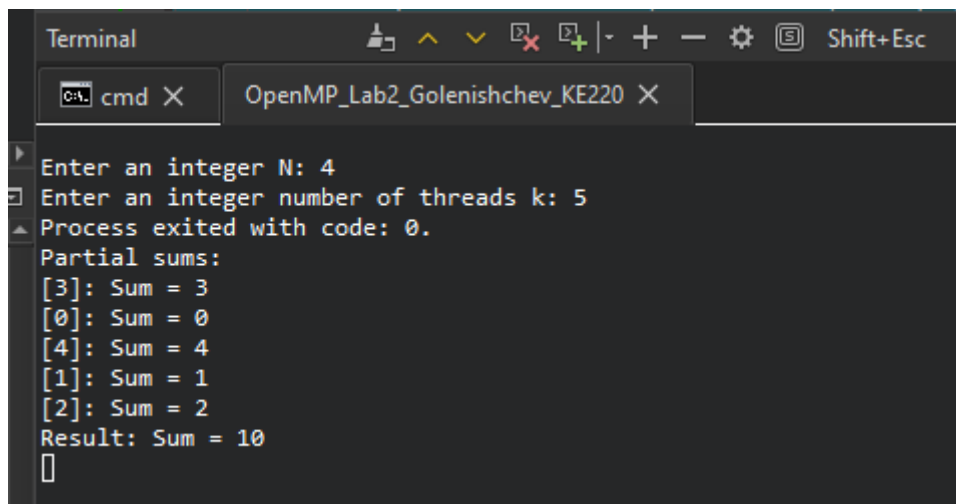
Листнинг 3. Код, где мы ловим ошибку с помощью синхронизации

Задание 5. Общие и частные переменные в OpenMP: параметр *reduction*

Код программы представлен ниже, листнинг 4. Показан результат работы программы, рисунок 4.

```
#include <stdio.h>
#include <omp.h>
// Golenishchev Artem, KE-220 Task 5
int main() {
    int n, k;
    // Ввод числа N и количества нитей k
    printf("Enter an integer N: ");
    scanf("%d", &n);
    printf("Enter an integer number of threads k: ");
    scanf("%d", &k);
    int sum = 0; // Общая сумма
    // Установка количества нитей
    omp_set_num_threads(k);
    printf("Partial sums:\n");
#pragma omp parallel reduction(+:sum)
    {
        int thread_id = omp_get_thread_num(); // Номер текущей нити
        int total_threads = omp_get_num_threads(); // Общее количество нитей
        // Диапазон чисел для текущей нити
        int start = (n * thread_id) / total_threads + 1;
        int end = (n * (thread_id + 1)) / total_threads;
        int local_sum = 0; // Частичная сумма для текущей нити
        for (int i = start; i <= end; ++i) {
            local_sum += i;
        }
        // Добавление локальной суммы в глобальную с использованием reduction
        sum += local_sum;
        // Вывод частичной суммы для текущей нити
        printf("[%d]: Sum = %d\n", thread_id, local_sum);
    }
    // Вывод общей суммы
    printf("Result: Sum = %d\n", sum);
    return 0;
}
```

Листнинг 4. Программа вычисления суммы чисел, работающая на k-нитей



```
Terminal
C:\> cmd X
OpenMP_Lab2_Golenishchev_KE220 X
Enter an integer N: 4
Enter an integer number of threads k: 5
Process exited with code: 0.
Partial sums:
[3]: Sum = 3
[0]: Sum = 0
[4]: Sum = 4
[1]: Sum = 1
[2]: Sum = 2
Result: Sum = 10
█
```

Рисунок 4. Результат выполнения программы расчета суммы чисел

Программа параллельно вычисляет сумму чисел от 1 до NNN, распределяя работу между kkk нитями. Каждый поток обрабатывает свой диапазон чисел, вычисляет частичную сумму и выводит её. Суммы нитей автоматически объединяются в общую сумму с использованием директивы `reduction(+:sum)`, которая безопасно складывает локальные суммы всех потоков в одну общую переменную после завершения параллельной области. Результат работы каждого потока и итоговая сумма выводятся на экран.

Ответы на вопросы к лабораторной работе:

1. Для чего нужны частные переменные? Не противоречит ли их существование реализуемой OpenMP модели программирования в общей памяти? Приведите содержательный пример частной переменной.

Частные переменные необходимы, чтобы каждому потоку выделять свою локальную копию данных, что предотвращает конфликты и обеспечивает независимость вычислений. Их существование не противоречит модели общей памяти OpenMP, так как локальные копии переменных размещаются в памяти отдельных потоков.

```
#pragma omp parallel for private(i)
for (i = 0; i < N; ++i) {
    // Каждая нить использует локальную копию переменной i
}
```

2. Какие новые области видимости появляются в параллельной программе? Как они задаются?

В параллельном программировании `shared()` хранит данные в сегменте стека главной нити, а `private()` внутри сегмента стека нити.



3. Продемонстрируйте конфликт обращений к переменной rank в написанной программе? Всегда ли он возникает? Как его предотвратить?

Конфликт возникает, если несколько потоков одновременно читают и записывают одно и то же значение переменной rank. Это приводит к неопределённому поведению. Конфликт не возникнет, если потоки не пересекаются по времени выполнения или используется синхронизация. Предотвращается `private(rank)`.

```
int rank = 0;
#pragma omp parallel
{
    rank += omp_get_thread_num(); // Одновременная запись несколькими
    потоками
    printf("Thread %d: rank = %d\n", omp_get_thread_num(), rank);
}
```

4. Как оформляются в программе входные данные, промежуточные переменные и результаты выполнения параллельного региона?

Входные данные: Передаются в параллельную область как `shared`, чтобы они были доступны всем потокам.

Промежуточные переменные: Обычно объявляются как `private`, чтобы потоки выполняли независимые вычисления.

Результаты: Объединяются через `reduction` или синхронизируются с помощью директивы `critical`.

Пример:

```
#pragma omp parallel reduction(+:sum)
{
    int local_sum = 0;
    for (int i = start; i < end; i++) {
        local_sum += i;
    }
    sum += local_sum; // Частичная сумма добавляется к общей.
}
```

5. Какие дополнительные действия выполняет директива, если она имеет параметр `reduction`? Чем инициализируются частные переменные, создаваемые параметром `reduction`?

Директива `reduction` автоматически:

1. Создаёт частные копии переменной для каждого потока.
2. Инициализирует эти переменные в зависимости от операции (например, 0 для `+`, 1 для `*`).
3. Выполняет объединение частных значений в конце параллельного региона.

Пример:

```
int sum = 0;
#pragma omp parallel for reduction(+:sum)
for (int i = 1; i <= 10; i++) {
    sum += i; // Каждый поток работает с локальной копией sum.
}
// В конце значения объединяются, и sum содержит итоговую сумму.
```

Выводы:

Изучили основные механизмы управления потоками в OpenMP, такие как использование общих (shared) и частных (private) переменных, распределение задач между потоками, а также предотвращение конфликтов при одновременном доступе к данным. Особое внимание было уделено параметру reduction, который позволяет автоматически создавать локальные копии переменных, выполнять параллельные вычисления и корректно объединять результаты. На практике было продемонстрировано, как эффективно распределять вычисления между потоками, минимизировать ошибки доступа к памяти и организовать синхронный сбор результатов для получения корректного результата.