Лабораторная работа №7

Выполнил: Крахмальный К.В.

Группа: ИКС-433

Отчет по данной работе:

1) Написать программу, которая создает поток с помощью pthread_create(). Использовать атрибуты по умолчанию. Родительский и дочерний потоки должны вывести на экран по 5 строк текста.

```c
void* thread_func(void* arg) {
    for (int i = 0; i < 5; i++) {
        printf("Child thread: line %d\n", i);
    }
    return NULL;
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, thread_func, NULL);

    for (int i = 0; i < 5; i++) {
        printf("Main thread: line %d\n", i);
    }

    pthread_join(thread, NULL);
    return 0;
}
```

```
Main thread: line 0
Main thread: line 1
Main thread: line 2
Main thread: line 3
Main thread: line 4
Child thread: line 0
Child thread: line 1
Child thread: line 2
Child thread: line 3
Child thread: line 4
```

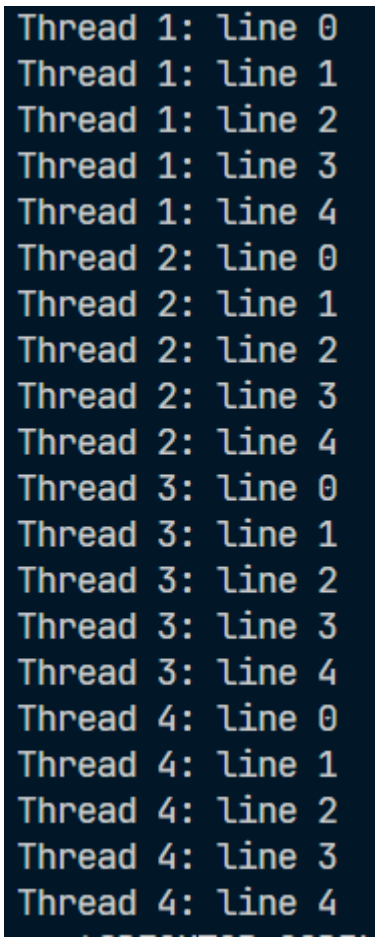
2) Модифицировать упр.1 так, что родительский поток выводит текст после завершения дочернего потока. Подсказка: pthread_join()

```c
void* thread_func(void* arg) {
    for (int i = 0; i < 5; i++) {
        printf("Child thread: line %d\n", i);
    }
    return NULL;
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, thread_func, NULL);

    pthread_join(thread, NULL);

    for (int i = 0; i < 5; i++) {
        printf("Main thread: line %d\n", i);
    }

    return 0;
}
```

```
Child thread: line 0
Child thread: line 1
Child thread: line 2
Child thread: line 3
Child thread: line 4
Main thread: line 0
Main thread: line 1
Main thread: line 2
Main thread: line 3
Main thread: line 4
```

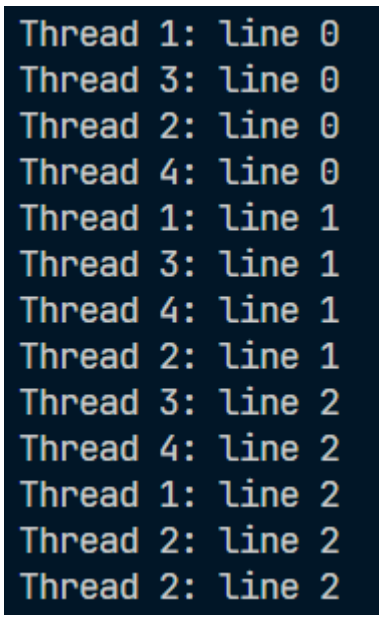
3)Модифицировать упр.2 так, что основной поток создает 4 потока, исполняющих одну и ту же функцию. Эта функция должна распечатать последовательность текстовых строк, переданных как параметр. Каждый из созданных потоков должен распечатать различные последовательности строк.

```c
void* thread_func(void* arg) {
    char* message = (char*)arg;
    for (int i = 0; i < 5; i++) {
        printf("%s: line %d\n", message, i);
    }
    return NULL;
}

int main() {
    pthread_t threads[4];
    char* messages[] = {
        "Thread 1",
        "Thread 2",
        "Thread 3",
        "Thread 4"
    };

    for (int i = 0; i < 4; i++) {
        pthread_create(&threads[i], NULL, thread_func, messages[i]);
    }

    for (int i = 0; i < 4; i++) {
        pthread_join(threads[i], NULL);
    }

    return 0;
    return 0;
}
```



4) Добавить сон с помощью sleep() в функцию потоков между выводами строк. Спустя две секунды после создания дочерних потоков основной поток должен прервать работу всех дочерних потоков с помощью pthread_cancel().

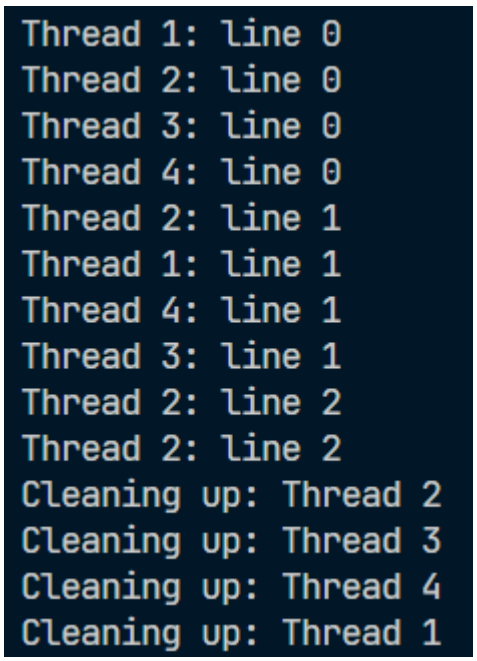
```c
void* thread_func(void* arg) {
    char* message = (char*)arg;
    for (int i = 0; i < 5; i++) {
        printf("%s: line %d\n", message, i);
        sleep(1);
    }
    return NULL;
}

int main() {
    pthread_t threads[4];
    char* messages[] = {
        "Thread 1",
        "Thread 2",
        "Thread 3",
        "Thread 4"
    };

    for (int i = 0; i < 4; i++) {
        pthread_create(&threads[i], NULL, thread_func, messages[i]);
    }

    sleep(2);

    for (int i = 0; i < 4; i++) {
        pthread_cancel(threads[i]);
    }

    return 0;
}
```



5) Модифицировать упр. 4 так, чтобы дочерний поток перед завершение распечатывал сообщение об этом. Использовать pthread_cleanup_push()

```c
void* thread_func(void* arg) {
    char* message = (char*)arg;
    pthread_cleanup_push(cleanup_handler, message);

    for (int i = 0; i < 5; i++) {
        printf("%s: line %d\n", message, i);
        sleep(1);
    }

    pthread_cleanup_pop(0);
    return NULL;
}

int main() {
    pthread_t threads[4];
    char* messages[] = {
        "Thread 1",
        "Thread 2",
        "Thread 3",
        "Thread 4"
    };

    for (int i = 0; i < 4; i++) {
        pthread_create(&threads[i], NULL, thread_func, messages[i]);
    }
```

```c
    sleep(2);

    for (int i = 0; i < 4; i++) {
        pthread_cancel(threads[i]);
    }

    for (int i = 0; i < 4; i++) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```



6) Реализовать прикольный алгоритм сортировки Sleepsort с асимптотикой O(N) (по времени). Суть алгоритма: на вход подается массив, пусть будет не более 50 элементов и пусть будет состоять из целочисленных значений. Для каждого элемента массива создается отдельный поток, в который в качестве аргумента передается значение элемента. Сам поток должен уйти в сон с помощью sleep() или usleep() с параметром равным аргументу потока (значение элемента массива), а после вывести на экран значение.

```c
void* sleep_sort(void* arg) {
    int value = *(int*)arg;
    sleep(value);
    printf("%d ", value);
    return NULL;
}

int main() {
    int arr[] = {3, 1, 4, 2, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    pthread_t threads[n];

    for (int i = 0; i < n; i++) {
        pthread_create(&threads[i], NULL, sleep_sort, &arr[i]);
    }

    for (int i = 0; i < n; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("\n");
    return 0;
}
```

```
1 2 3 4 5
```