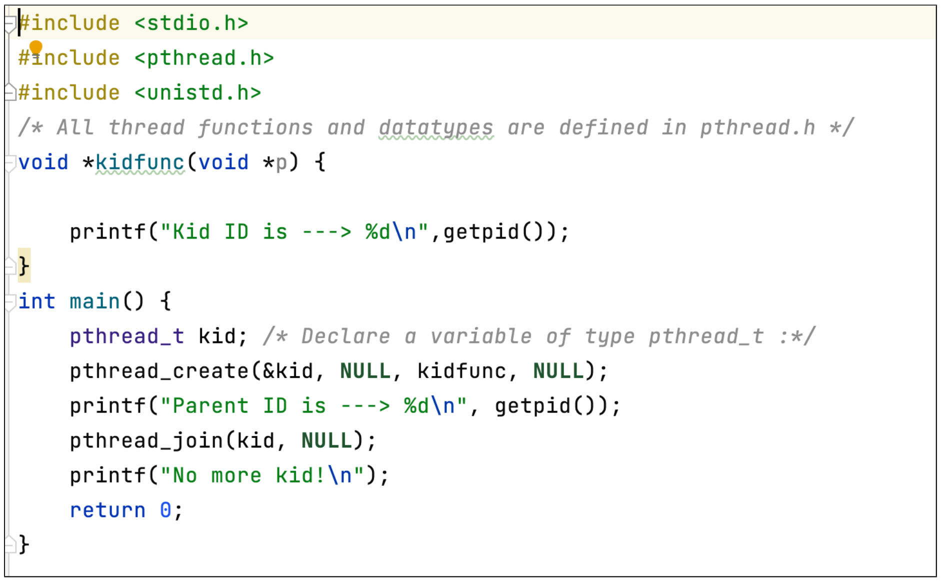


```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
/* All thread functions and datatypes are defined in pthread.h */
void *kidfunc(void *p) {

    printf("Kid ID is ---> %d\n",getpid());
}
int main() {
    pthread_t kid; /* Declare a variable of type pthread_t :*/
    pthread_create(&kid, NULL, kidfunc, NULL);
    printf("Parent ID is ---> %d\n", getpid());
    pthread_join(kid, NULL);
    printf("No more kid!\n");
    return 0;
}
```

Figure 1: Thread and Process

1. For the snippet of code in Figure 1, answer the following questions:

- What is the output of the program?

```
Parent ID is ---> 19291
Kid ID is ---> 19291
No more kid!
```

- Are the process id numbers of the threads same or different? Why? They are the same because they are threads of same process.
- Which thread finishes its execution - parent or child? Why? The child because of pthread_join

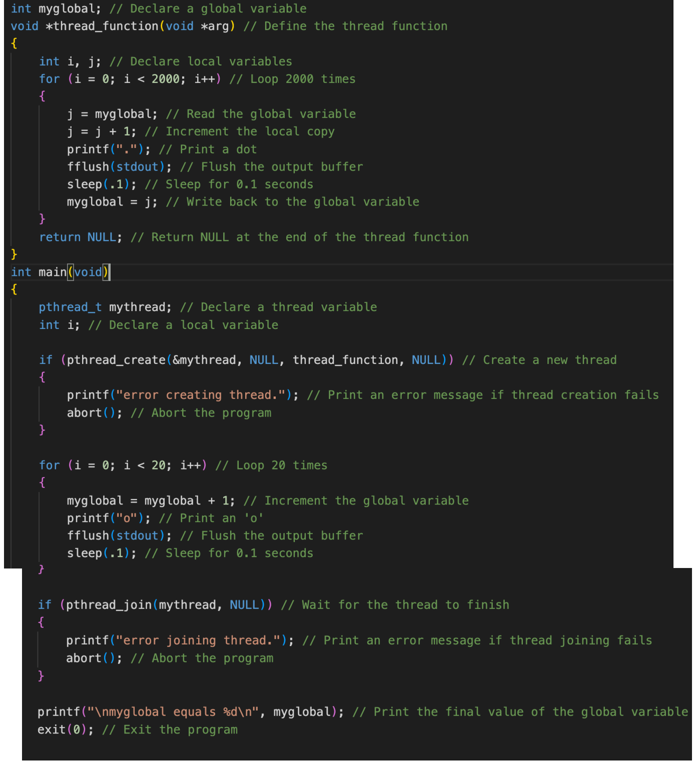
2. A file named "threads.c" is present on Canvas in the same assignment as this Homework, answer the following questions:

- Resubmit the code by adding comments for every line.
- What is the output of this program?

```
o.o.o.o.o..o.o.oo..o.o.oo.o.o.o.o..o.o.o.o.o.o
myglobal equals 21
```

- What would happen if the pthread_join function is removed? if the i in the loop is 20 for both of them than likely nothing but if the i in the loop is larger for the thread than that wont finish since the proccess does not wait for the thread to finish.

3. Write a program in C, that creates two threads. The first thread will be printing 20000 'O's and the second thread prints 30000 'X's.
Note1: Both the threads should be running the same function. The last argument in the pthread_create will come in handy! Go through the programming examples present on Canvas for hints.
Note2: The program should also output the count of 'X's and the count of 'O's printed for everytime a character is printed.



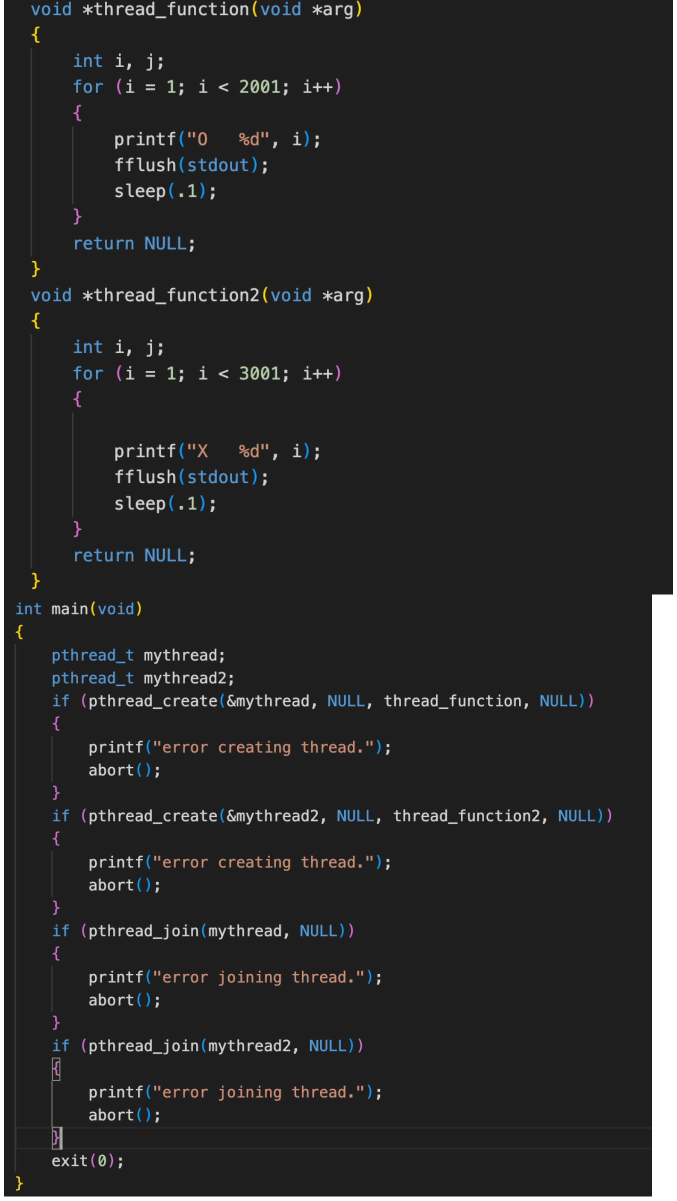
```c
int myglobal; // Declare a global variable
void *thread_function(void *arg) // Define the thread function
{
    int i, j; // Declare local variables
    for (i = 0; i < 2000; i++) // Loop 2000 times
    {
        j = myglobal; // Read the global variable
        j = j + 1; // Increment the local copy
        printf("."); // Print a dot
        fflush(stdout); // Flush the output buffer
        sleep(.1); // Sleep for 0.1 seconds
        myglobal = j; // Write back to the global variable
    }
    return NULL; // Return NULL at the end of the thread function
}
int main(void)
{
    pthread_t mythread; // Declare a thread variable
    int i; // Declare a local variable

    if (pthread_create(&mythread, NULL, thread_function, NULL)) // Create a new thread
    {
        printf("error creating thread."); // Print an error message if thread creation fails
        abort(); // Abort the program
    }

    for (i = 0; i < 20; i++) // Loop 20 times
    {
        myglobal = myglobal + 1; // Increment the global variable
        printf("o"); // Print an 'o'
        fflush(stdout); // Flush the output buffer
        sleep(.1); // Sleep for 0.1 seconds
    }

    if (pthread_join(mythread, NULL)) // Wait for the thread to finish
    {
        printf("error joining thread."); // Print an error message if thread joining fails
        abort(); // Abort the program
    }

    printf("\nmyglobal equals %d\n", myglobal); // Print the final value of the global variable
    exit(0); // Exit the program
```



```c
void *thread_function(void *arg)
{
    int i, j;
    for (i = 1; i < 2001; i++)
    {
        printf("O   %d", i);
        fflush(stdout);
        sleep(.1);
    }
    return NULL;
}
void *thread_function2(void *arg)
{
    int i, j;
    for (i = 1; i < 3001; i++)
    {

        printf("X   %d", i);
        fflush(stdout);
        sleep(.1);
    }
    return NULL;
}
int main(void)
{
    pthread_t mythread;
    pthread_t mythread2;
    if (pthread_create(&mythread, NULL, thread_function, NULL))
    {
        printf("error creating thread.");
        abort();
    }
    if (pthread_create(&mythread2, NULL, thread_function2, NULL))
    {
        printf("error creating thread.");
        abort();
    }
    if (pthread_join(mythread, NULL))
    {
        printf("error joining thread.");
        abort();
    }
    if (pthread_join(mythread2, NULL))
    {
        printf("error joining thread.");
        abort();
    }
    exit(0);
}
```