Object Oriented Methodology

Week – *7*, Lecture – *3*
Introduction to POSIX Threads

SAURABH SRIVASTAVA

VISITING FACULTY

IIIT LUCKNOW

About Function Pointers

A function pointer is something that we did not discuss in detail before

- So let us have a look at it in brief first

A function pointer in C/C++ is a pointer that can be used to call the function

Such a pointer is helpful when you wish to treat a function similar to a value of a variable …

- … i.e. you want to have a “variable”, whose values are different functions
- The value of this variable can be changed as often as required
- This is useful, if you the function that must be called for a particular case is not known at compile time
- A function pointer can be used at such places, and the value of the pointer decides the execution trail

Similar to a regular pointer, a function pointer can point to only *functions that match its type*

- Its type is decided by its return type as well as its arguments list
- The value of a “compatible” function can be assigned to the pointer as `<ptr> = <function name>`
- There is a link in the **Additional Reading** section where you can read more about them

About `void` Pointers

In C/C++, there is a pointer to a general type, which can point to data of “any data type”

These pointers are called `void` pointers

A void pointer is declared like this:

- `void* ptr;`

You can assign it the address of any variable – primitive or objects

- For example, this is perfectly fine:

```
ptr = &a_float_variable; // assigning a float pointer to a void pointer
ptr = &a_int_variable; // assigning an int pointer to a void pointer
```

A the time of using them, you are required to perform the correct type cast

- For example, the values of the variables above can be accessed like:
  `cout<<*((float*) ptr);` or `cout<<*((int*) ptr);`

Thus, a void pointer can be used to point to *any* type of data in C/C++

Creating a new *Thread of Execution*

To create a new *Thread of Execution*, you can use the `pthread_create()` method

This method expects four parameters

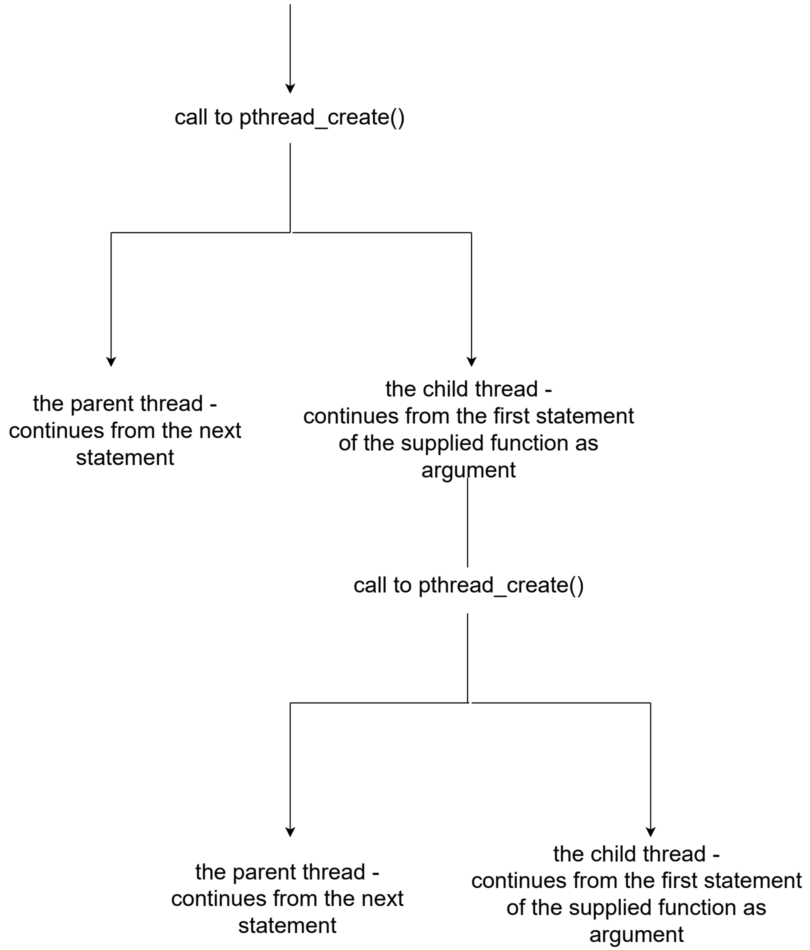
- The first parameter is a pointer to the type `pthread_t`, used to return an id for the new thread
- The second parameter is a pointer to the type `pthread_attr_t`, used to provide any preferences
- The third parameter is a *function pointer*, the name of an “appropriate” function containing thread’s logic
- The fourth parameter is a pointer to a single argument of a `void` type, passed to the above function

It returns an integer which

- if `0`, means that the thread was created successfully
- otherwise, it means that an error during the thread creation process

To check out details about the same, you can use the command `man pthread_create`

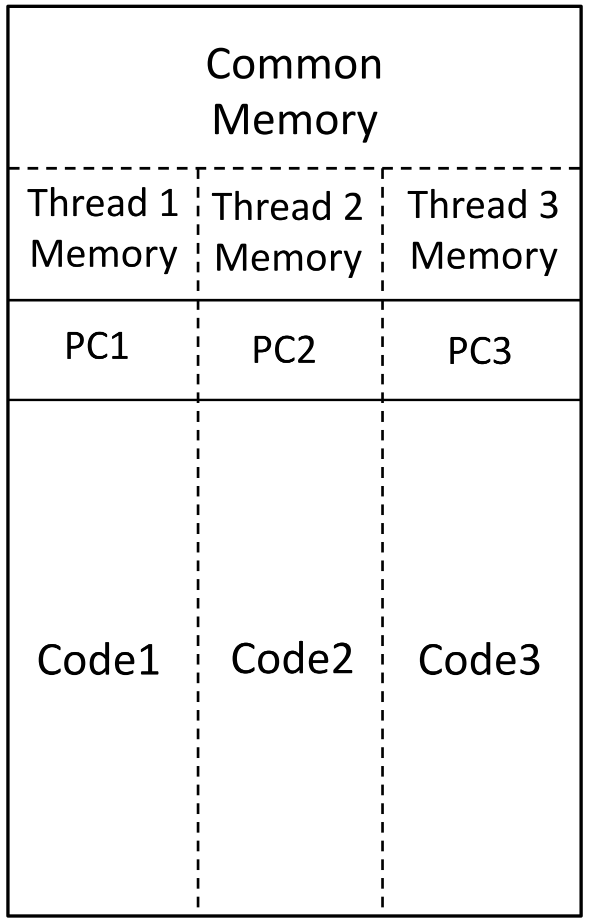
How “threads” work simultaneously


call to pthread_create()
the parent thread - continues from the next statement
the child thread - continues from the first statement of the supplied function as argument
call to pthread_create()
the parent thread - continues from the next statement
the child thread - continues from the first statement of the supplied function as argument

Communication between threads

Remember the simplified pictorial view of threads that we saw in the previous lecture?

“A simplified view of” Threads



This one !!

Communication between threads

The common memory here can be used for communication between threads

For example, global variables are accessible to all the threads

- They can be set by one thread, and read by another
- There is an inherent issue with this – called thread safety – but that is beyond our coverage for now

A typical scenario that may be useful for you looks like this:

```
.                                      .
.                                      .
// Thread i                            // Thread j
.                                      .
.                                      .
while(global_var != desired_val)       global_var = desired_val
    // keep waiting                    .
// do required stuff                   .
.                                      .
.                                      .
```

Let us see some code now !!

Homework !!

Think about your projects, and find out if there are any places where threading could be useful

- In particular, you may wish to do two tasks in parallel, and communicate between the two periodically

Additional Reading

Function Pointers are not required often – specially when you write your own code …

- … but they are often used for system tasks such as threading or signal handling

Read the following articles, if you find the concept interesting

- https://www.geeksforgeeks.org/function-pointer-in-c/
- https://www.cprogramming.com/tutorial/function-pointers.html