Threads

1. Creating Threads

We will be using POSIX Pthread library for creating threads in C. (`pthread.h`)

To create a new thread we use the function `pthread_create()` to add a thread to the current process.

Syntax:
```
int pthread_create(pthread_t *tid, const pthread_attr_t
    *tattr, void*(*start_routine)(void *), void *arg);
```

a. `tid` refers to the thread ID which is of type `pthread_t` i.e. unsigned int.
b. `tattr` refers to the attributes of the thread. A default thread has the following attributes:
    - It is joinable.
    - It has default stack and stack size.
    - It inherits the parent's priority.
c. `start_routine` refers to the function the thread will execute. The return type of the function is `void *`.
d. `void *arg` is the argument which is passed to the function `start_routine`.

2. Initializing Attributes

We need to initialize the attributes of the thread i.e. `tattr`. A thread can have various attributes other than 3 mentioned above such as scheduling policy or scope of the thread.

Syntax: `int pthread_attr_init(const pthread_attr_t *tattr)`

3. Waiting for a Thread to join back

Since by default threads are joinable, we need to wait for the thread to join. We can also create detached threads by setting appropriate attributes or detaching the thread within the function itself.

Syntax:
```
int pthread_join(pthread_t tid, void **status);
```

Questions

1. Create a thread that computes the factorial of a number. The number is an input by the user.
    a. Use global variable
    b. Passing arguments to thread function

    The thread joins back with the main process and the main process prints the factorial.

    STEPS:
    1. Read the input from the user using `scanf`.
    2. Store the input into a global variable called `inputNumber`.
    3. Create a thread with Factorial function which calculates factorial of `inputNumber`.
    4. Store the value of the factorial in a global variable `fact`.
    5. After the thread joins back the main process, the main process prints the value of `fact` to the user.

    Similarly for b. pass the `inputNumber` as an argument of the function. Assume `fact` to be a global variable in both the cases.

2. Create multiple threads and print their tid's using the function `pthread_self()`.

Note: You can use the references below for doubts or further understanding threads.

References

http://www.cs.cf.ac.uk/Dave/C/node29.html

http://www.cs.cf.ac.uk/Dave/C/node30.html

http://pubs.opengroup.org/onlinepubs/007904975/functions/xsh_chap02_09.html