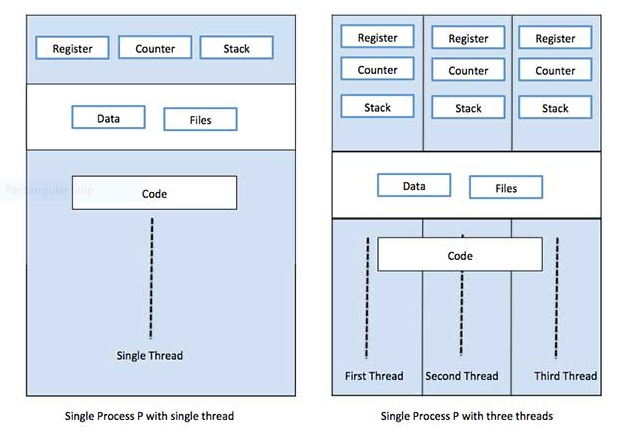
Thread :

1. Threads are the light weight process, helps to improve the Application performance through parallelism.
2. Things that threads share with each other:
   1. Code Memory.
   2. Data Memory.
   3. Open Files.
3. Things those are separate for each thread.
   1. Stack
   2. Program Counter
   3. System Registers
4. Each thread represents a separate flow of control.



The above diagrams shows the single thread and multi thread concept, and specifies the shared and Separate commodities of a thread.

1. There are two types of threads:
   1. User Level (user managed threads)
   2. Kernel Level (operating system managed threads)
2. **Threads API specifications and declerations:**
3. Thread operations include thread creation, termination, synchronization (joins,blocking), scheduling, data management and process interaction.
4. A thread does not maintain a list of created threads, nor does it know the thread that created it.
5. All threads within a process share the same address space.
6. Threads in the same process share:
   1. Process instructions
   2. Most data
   3. open files (descriptors)
   4. signals and signal handlers
   5. current working directory
   6. User and group id
7. Each thread has a unique:
   1. Thread ID
   2. set of registers, stack pointer
   3. stack for local variables, return addresses
   4. signal mask
   5. priority
   6. Return value: errno
8. pthread functions return "0" if OK.

* **Thread Creation and Teremination:**
  + Header file needed : pthread.h
  + Compile time linkage : gcc –pthread **code.c**
  + Pthread function decleration:

int pthread\_create(pthread\_t \* thread,   
 const pthread\_attr\_t \* attr,  
 void \* (\*start\_routine)(void \*),   
 void \*arg);

**Attributes details:**

* + 1. Pthread\_t \*thread :
       - this attribute gives the ThreadID
       - here we have to pass address of variable
    2. const pthread\_attr\_t \*attr:
       - this attribute is for special purpose of pthread\_Create function.(for more detail click here.)
    3. void \* (\*start\_routine)(void \*) :
       - this attribute is for the function call.
       - This functions return type should be a void pointer type
       - And attribute for this function should be of void pointer type
    4. Void \*arg:
       - This attribute provides the argument for the thread function, i.e. function name passed in previous argument.
  + Short details of 2nd argument
    1. There is a structure with the name **pthread\_attr\_t**
    2. If user wants to do some additional things with the threads or user wants to add or remove some functionalities of the thread (e.g. controlling the thread stack size ) then configuring this structure and passing it while creating the thread will do the work.
    3. The structure is as folloes:
       - detached state (joinable?
         * Default: PTHREAD\_CREATE\_JOINABLE.
         * Other option: PTHREAD\_CREATE\_DETACHED)
       - scheduling policy (real-time?
         * PTHREAD\_INHERIT\_SCHED,
         * PTHREAD\_EXPLICIT\_SCHED,
         * SCHED\_OTHER)
       - scheduling parameter
       - inheritsched attribute (
         * Default: PTHREAD\_EXPLICIT\_SCHED
         * Inherit from parent thread: PTHREAD\_INHERIT\_SCHED)
       - scope (
         * Kernel threads: PTHREAD\_SCOPE\_SYSTEM
         * User threads: PTHREAD\_SCOPE\_PROCESS
         * Pick one or the other not both.)
       - guard size
       - stack address (See unistd.h and bits/posix\_opt.h \_POSIX\_THREAD\_ATTR\_STACKADDR)
       - stack size (
         * default minimum PTHREAD\_STACK\_SIZE set in pthread.h),

to access these stack elements there are two API namely:

1. pthread\_attr\_init
2. pthread\_attr\_distroy

(for more details)

link: <http://man7.org/linux/man-pages/man3/pthread_attr_init.3.html>

1. incase we want to change the attributes, we need to call the pthread\_attr\_init

function to set the **pthread\_attr\_t** structure values , and in case of resetting we have to use pthread\_attr\_ destroy function.

1. If once the attribute is set using pthread\_attr\_init function the same value cannot be set by calling the pthread\_attr\_init function with the same argument.
2. Same confition is for pthread\_attr\_ destroy.
3. Calling **pthread\_attr\_init**() on a thread attributes object that has already been initialized results in undefined behavior.
4. When a thread attributes object is no longer required, it should be destroyed using the **pthread\_attr\_destroy**() function. Destroying a thread attributes object has no effect on threads that were created using that object.
5. Once a thread attributes object has been destroyed, it can be reinitialized using **pthread\_attr\_init**(). Any other use of adestroyed thread attributes object has undefined results.
6. These two functions returns 0 on success, and returns non zero values as error number on error.
7. Mostly on linux platform these functions return 0 but on portable platform these return should be handled.

REFERENCES:

1. <https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html> (thread basics)
2. <http://man7.org/linux/man-pages/man3/pthread_attr_init.3.html> (pthread attributs)
3. <https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html> (pthread related api)
4. <http://man7.org/linux/man-pages/man3/pthread_attr_setstack.3.html> (setstack related)
5. <https://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/Errors/unix_system_errors.html> (ERROR RELATED)