Multithreading

Thread

- A thread is a flow of execution through the process code, with its own **program counter** that keeps track of which instruction to execute next, system **registers** which hold its current working variables, and a **stack** which contains the execution history.
- Multithreading contains two or more threads that can run concurrently.
- Each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

Thread

- Multiple threads perform multiple tasks within the environment of a single process.
- All threads within a single process have access to the same process components, such as file descriptors and memory.
- Each process is doing only one thing at a time. With multiple threads of control, we can design our programs to do more than one thing at a time within a single process.

Thread

Thread can be implemented by two ways −

- User Level Threads − User managed threads.
- User threads are supported above the kernel and are managed without kernel support
- Kernel Level Threads − Operating System managed threads acting on kernel.
- modern operating systems—including Windows, Linux, and macOS— support kernel threads also.

Thread

- #include <pthread.h>
- int pthread_create(pthread_t *tidp, pthread_attr_t *attr, void *(*start_rtn), void *arg);
- Returns: 0 if OK, error number on failure
- **tidp** is set to the thread ID of the newly created thread.
- **attr** argument is used to customize various thread attributes .
- The newly created thread starts running at the address of the start_rtn function.
- **arg**, is a typeless pointer.

Thread

- When a thread is created, there is no guarantee which will run first: the newly created thread or the calling thread
- Just as every process has a process ID, every thread has a thread ID.
- new thread obtains its thread ID by calling pthread_self

```
pid = getpid();        tid = pthread_self();
```

- threads created within a process, have the same process ID, but different thread IDs.

Thread

Thread Termination

- The thread can call pthread_exit.
- #include <pthread.h>
- void pthread_exit(void *rval_ptr);
- The rval_ptr argument is a typeless pointer, similar to the single argument passed to the start routine.

Thread

- #include <pthread.h>
- int pthread_join(pthread_t thread, void *rval_ptr);
- Returns: 0 if OK, error number on failure
- The calling thread will block until the specified thread calls pthread_exit, returns from its start routine.

Thread

- Fork, pthread_create create a new flow of control
- Exit, pthread_exit exit from an existing flow of control
- Waitpid, pthread_join get exit status from flow of control
- Getpid, pthread_self get ID for flow of control