Thread Related Operations

- Thread creation
- Thread termination
- Thread synchronization
- Thread scheduling
- Thread data management
- Thread / process interaction

Thread Creation

- pthread_create
- extern int pthread_create (pthread_t *tid, __const pthread_attr_t *attr, void *(*__start_routine) (void *), void *arg)
  - Creates a new thread of control that executes concurrently with the calling thread.
  - The new thread applies the function start_routine passing it arg as first argument.
  - The new thread terminates either explicitly, by calling pthread_exit(3), or implicitly, by returning from the start_routine function.
  - The attr argument specifies thread attributes to be applied to the new thread.
  - The attr argument can also be NULL, in which case default attributes are used

Thread Creation

- Return value
  - On success, the identifier of the newly created thread is stored in the location pointed by the thread argument,
  - and a 0 is returned.
  - On error, a non-zero error code is returned.
- Errors
  - EAGAIN not enough system resources to create a process for the new thread.
  - EAGAIN more than PTHREAD_THREADS_MAX threads are

- Setting attributes for threads is achieved by filling a thread attribute object attr of type pthread_attr_t, then passing it as second argument to pthread_create.
- Passing NULL is equivalent to passing a thread attribute object with all attributes set to their default values.
- Thread attribute structure is in /usr/include/bits/pthreadtypes.h

```
#define __SIZEOF_PTHREAD_ATTR_T 56
typedef union
{
  char __size[__SIZEOF_PTHREAD_ATTR_T];
  long int __align;
} pthread_attr_t;
```

Detachstate, Schedpolicy, Sched_param structure, Inheritsched, Scope will be a part of the attribute

Attribute Initialization & Destroy

- extern int pthread_attr_init (pthread_attr_t *attr)
  - Initializes the thread attribute object attr and fills it with default values for the attributes.
  - Attribute objects are consulted only when creating a new thread.
  - The same attribute object can be used for creating several threads. Modifying an attribute object after a call to pthread_create does not change the attributes of the thread previously created.
- extern int pthread_attr_destroy(pthread_attr_t *attr)
  - Destroys a thread attribute object, which must not be reused until it is reinitialized.
  - pthread_attr_destroy does nothing in the LinuxThreads implementation.

Detach State

- Control whether the thread is created in the joinable state (value PTHREAD_CREATE_JOINABLE) or in the detached state (PTHREAD_CREATE_DETACHED).
- Default value: PTHREAD_CREATE_JOINABLE.
- Joinable state
  - Another thread can synchronize on the thread termination and recover its termination code using pthread_join
  - some of the thread resources are kept allocated after the thread terminates, and reclaimed only when another thread performs pthread_join on that thread.
- Detached state
  - The thread resources are immediately freed when it terminates
  - pthread_join cannot be used to synchronize on the thread termination

Detach State

- A thread created in the joinable state can later be put in the detached thread using pthread_detach.
- extern int pthread_attr_setdetachstate (pthread_attr_t *attr, int detachstate)

- extern int pthread_attr_getdetachstate (___const pthread_attr_t *attr, int *detachstate)

Sched Policy

- Select the scheduling policy for the thread: one of SCHED_OTHER (regular, non-realtime scheduling), SCHED_RR (realtime, round-robin) or SCHED_FIFO (realtime, first-in first-out).
- Default value: SCHED_OTHER.
- The real time scheduling policies SCHED_RR and SCHED_FIFO are available only to processes with super user privileges.
- The scheduling policy of a thread can be changed after creation with pthread_setschedparam

Sched Policy

- extern int pthread_attr_setschedpolicy (pthread_attr_t *attr, int policy)

- extern int pthread_attr_getschedpolicy (___const pthread_attr_t *attr, int *policy)

Sched Param

- Contain the scheduling parameters (essentially, the scheduling priority) for the thread.
- See sched_setparam for more information on scheduling parameters.
- Default value: priority is 0.
- This attribute is not significant if the scheduling policy is SCHED_OTHER; it only matters for the realtime policies SCHED_RR and SCHED_FIFO.
- The scheduling priority of a thread can be changed after creation with pthread_setschedparam

Sched Param

- extern int pthread_attr_setschedparam (pthread_attr_t *attr, __const struct sched_param *param)

- extern int pthread_attr_getschedparam (__const pthread_attr_t *attr, struct sched_param *param)

- See struct sched_param in bits/sched.h

Inheritsched

- Indicate whether the scheduling policy and scheduling parameters for the newly created thread are determined by the values of the schedpolicy and schedparam attributes (value PTHREAD_EXPLICIT_SCHED) or are inherited from the parent thread (value PTHREAD_INHERIT_SCHED).
- Default value: PTHREAD_EXPLICIT_SCHED.
- extern int pthread_attr_setinheritsched (pthread_attr_t *attr, int inherit)
- extern int pthread_attr_getinheritsched (___const pthread_attr_t *attr, int *inherit)

Scope

- Define the scheduling contention scope for the created thread.
- The only value supported in the LinuxThreads implementation is PTHREAD_SCOPE_SYSTEM
  - meaning that the threads contend for CPU time with all processes running on the machine (thread priorities are interpreted relative to the priorities of all other processes on the machine).
- The other value specified by the standard, PTHREAD_SCOPE_PROCESS
  - means that scheduling contention occurs only between the threads of the running process (thread priorities are interpreted relative to the priorities of the other threads of the process, regardless of the priorities of other processes)

Scope

- extern int pthread_attr_setscope (pthread_attr_t *attr, int scope)

- extern int pthread_attr_getscope (___const pthread_attr_t *attr, int *scope)

SetSchedParam

- extern int pthread_setschedparam (pthread_t t_thread, int policy, __const struct sched_param *param)
  - sets the scheduling parameters for the thread t_thread as indicated by policy and param.
  - Policy can be either SCHED_OTHER, SCHED_RR or SCHED_FIFO.
  - param specifies the scheduling priority for the two realtime policies.
- extern int pthread_getschedparam (pthread_t t_thread, int *policy, struct sched_param *param)
  - retrieves the scheduling policy and scheduling parameters for the thread t_thread and store them in the locations pointed to by policy and param, respectively.
- Return value
  - return 0 on success
  - a non-zero error code on error.

Self & equal

- extern pthread_t pthread_self (void)
  - return the thread identifier for the calling thread.
- extern int pthread_equal (pthread_t __thread1, pthread_t __thread2)
  - determines if two thread identifiers refer to the same thread.
  - Returns a non-zero value if thread1 and thread2 refer to the same thread. Otherwise, 0 is returned

Detach

- extern int pthread_detach (pthread_t th)
  - put the thread th in the detached state.
  - applies to threads created in the joinable state, and which needs to be put in the detached state later.
  - After pthread_detach completes, subsequent attempts to perform pthread_join on th will fail.
  - If another thread is already joining the thread th at the time pthread_detach is called, pthread_detach does nothing and leaves th in the joinable state.
- Return value
  - On success, 0 is returned.
  - On error, a non-zero error code is returned.

Exit

- extern void pthread_exit (void *retval)
  - terminates the execution of the calling thread.
  - All cleanup handlers that have been set for the calling thread with pthread_cleanup_push are executed in reverse order.
  - Finalization functions for thread-specific data are then called for all keys that have non- NULL values associated with them in the calling thread (see pthread_key_create).
  - Finally, execution of the calling thread is stopped.
  - The retval argument is the return value of the thread. It can be consulted from another thread using pthread_join.
- Return value
  - The pthread_exit function never returns.

Join

- extern int pthread_join (pthread_t th, void **__thread_return)
  - suspends the execution of the calling thread until the thread identified by th terminates, either by calling pthread_exit or by being cancelled.
  - If thread_return is not NULL, the return value of th is stored in the location pointed to by thread_return.
  - The return value of th is either the argument it gave to pthread_exit, or PTHREAD_CANCELED if th was cancelled.
  - The joined thread th must be in the joinable state
  - When a joinable thread terminates, its memory resources (thread descriptor and stack) are not deallocated until another thread performs pthread_join on it.
  - It is must to call pthread_join once for each joinable thread created to avoid memory leaks.

Join

- At most one thread can wait for the termination of a given thread.
- Calling pthread_join on a thread th on which another thread is already waiting for termination returns an error.

- Cancellation
  - pthread_join is a cancellation point.
  - If a thread is canceled while suspended in pthread_join, the thread execution resumes immediately and the cancellation is executed without waiting for the th thread to terminate.
  - If cancellation occurs during pthread_join, the th thread remains not joined.
- Return value
  - On success, the return value of th is stored in the location pointed to by thread_return, and 0 is returned.
  - On error, a non-zero error code is returned.

Threading Issues

- The fork and exec system calls
  - If one thread in a system calls fork()
    - The new process duplicates all threads
    - The new process duplicates only the calling thread.
- Cancellation
  - Task of terminating the thread before it has completed.
  - Cancellation of target thread (the thread that is to be cancelled) may occur in 2 different scenarios
    - Asynchronous cancellation
      - terminates the target thread immediately
    - Deferred cancellation
      - allows the target thread to periodically check if it should be cancelled

```
#include<pthread.h>                    #include<unistd.h>
   #include<stdio.h>                   #include<sys/types.h>
   #include<asm/unistd.h>
void *runner(void *param);
int main(int argc,char *argv[])
{ pthread_t tid,tid1;
  pthread_attr_t attr;
  pthread_attr_init(&attr);
  pthread_create(&tid,&attr,runner,argv[1]);
  pthread_create(&tid1,&attr,runner,argv[2]);
  printf("1st thread ID=%u & 2nd thread ID=%u\n",tid,tid1);
    if(!fork())
    {   printf("Child PID=%d, PPID=%d\n",getpid(),getppid());
        printf("Child TID=%d, PID=%d\n",syscall(__NR_gettid),getpid());
    }
```

```
else
{
        wait(NULL);
        printf("Parent:PID=%d,PPID=%d\n",getpid(),getppid());
        printf("Parent:TID=%d, PID=%d\n",
                syscall(__NR_gettid),getpid());
}
pthread_join(tid,NULL);
pthread_join(tid1,NULL);
return 0;
}
```

```
// runner function
void *runner ( void *param )
{       int upper=atoi(param);
        int i;
        int sum=0;
        if (upper>0)
        {       for ( i=1; i <= upper; i++ )
                {       sum = sum + i;   }
        }
        printf("From thread:Thread ID=%u,SUM=%d\t PID=%d,
        PPID=%d\n",pthread_self(),sum,getpid(),getppid());
        printf("From thread:TID=%d,PID=%d\n",
        syscall(__NR_gettid),getpid());
        pthread_exit(0);
}
```