POSIX Threads

Class Notes

Posix Threads (Pthreads)

- Pthreads are user level threading library for UNIX and its variants.
- Posix standard categorizes various threading APIs provided by the library implementation into two groups.

```
NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                          void *(*start_routine) (void *), void *arg);

       Compile and link with -pthread.
```

Thread attributes:

Each thread created using pthread_create has the following attributes.

1. **Detach state**: This attribute defines thread clean up mode on exit.

   Possible values

   1) PTHREAD_CREATE_JOINABLE: This creates a joinable thread which requires any thread in the current thread group to read the exit value for this thread to be destroyed. This is default.
   2) PTHREAD_CREATE_DETACHED: Start this thread in the auto clean up mode.

2. **Scope** : Defines thread visibility to kernel's process scheduler

   Possible values

   1) PTHREAD_SCOPE_SYSTEM: Thread is scheduled by kernel's process scheduler. This is default.
   2) PTHREAD_SCOPE_PROCESS: Thread is scheduled by a process level scheduler.

3. **Stack**: Defines the stack region to be used by current thread.
   Default: 2mb stack per thread. Each thread can be assigned from 16K to 8mb of stack size.

4. **Scheduling**: Defines scheduling policy and priority. Linux kernel supports the following policies by default.
   a) Non real time: SCHED_OTHER
      SCHED_BATCH
      SCHED_IDLE
   b) Real time: SCHED_FIFO
      SCHED_RR

- All the above scheduling policies are pre-empted (highest priority runs first)
- Non real time policies perform scheduling based on dynamic priority process of thread.
- Dynamic priorities are dynamically assigned to a process by the scheduler. This value is not fixed and bound to change each time the process is ready to run.
- Dynamic priorities are decided on the following parameters.
  a) Nice value of the process
  b) Amount of time the process spent in the ready queue.
  c) Type of process (IO bound ,CPU bound)
- Real time policies are based on static priorities assigned. Priority range is 1-99.
- Under real time ,SCHED_FIFO is a co-operative scheduler between same priority process and priority pre-emptive across other process groups
- SCHED _RR is a time scheduler across same process priorities and priority pre-emptive across other process.

- Sample code for thread attributes:

```
pthread_t thr;
pthread_attr_t attr;
int ret;

ret = pthread_attr_init(&attr);

 pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

  pthread_create(&thr, &attr, threadFunc, (void *) 1);
```

- Detach state can be altered to joinable or to detached either during thread creation (using attribute instance) or at thread run time (directly making change in the thread object)

- Sample code for scheduling :

```
int inherit,policy,priority,rc;
pthread_t tcb;
pthread_attr_t attr;
struct sched_param param;

pthread_attr_init(&attr);

pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);

/* Now assign Sched policy and priority */

policy = SCHED_FIFO;
pthread_attr_setschedpolicy(&attr,policy);

param.sched_priority = 20;
pthread_attr_setschedparam(&attr,&param);

pthread_create(&tcb, &attr, t_routine, NULL);


param.sched_priority = 10;

policy = SCHED_RR;
pthread_setschedparam(tcb, policy, &param);
```

- Stack attribute is widely used to assign custom stack size based on thread's local data. Using default stack (2mb) may result in unused bytes and wastage of memory when threads do not contain huge local data structures.

Sample code:

1) Allocating memory (buffer) for stack

```
size_t stacksize =  16900;  /* in Bytes minimum 16 KB */
void *stackaddr;
int align = getpagesize();

posix_memalign(&stackaddr, align, stacksize);
```

2) Fill the attribute structure with stack details

```
pthread_t tcb;
pthread_attr_t attr;

pthread_attr_init(&attr);

pthread_attr_setstack(&attr, stackaddr, stacksize);

pthread_create(&tcb, &attr, t_routine, NULL);
```