The goal of this assignment is to implement a routine that emulates the `softclock` routine of a BSD kernel. `softclock` manages the evaluation of periodic events such as retransmission of dropped packets and watchdog timers. The data structure that records the state of periodic waiting events is called a callout queue:

<table>
<thead>
<tr>
<th>Queue</th>
<th>Delta</th>
<th>Function/Arg</th>
<th>Expiration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 tick</td>
<td>f(x)</td>
<td>10 ms</td>
</tr>
<tr>
<td></td>
<td>3 ticks</td>
<td>g(y)</td>
<td>40 ms</td>
</tr>
<tr>
<td></td>
<td>0 ticks</td>
<td>h(z)</td>
<td>40 ms</td>
</tr>
</tbody>
</table>

The queue is sorted in event time order with the soonest events at the front of the queue and the most distant events at the end of the queue. The above diagram depicts the following:

- There are three events (one event record per event).
- A clock tick is 10 ms (milliseconds).
- The `Delta` field is time (in clock ticks) relative to the preceding event record.
- If a tick is defined to be 10 ms, the first, second, and third event waits should expire respectively in 10 ms (1st tick), 40 ms (4th tick), and 40 ms (4th tick).
- The `Function/Arg` field (actually two fields) indicates the function that should be called when the timer expires for that event.

A process schedules a new event by registering an event:

```c
int rc = registerCallback (void Function(void *), void * Arg, Tick_t Expire);
```

where `Function` is the function to invoke, `Arg` is the argument to pass to `Function`, `Expire` is the event expiration time (in CPU ticks), and `rc` is the return code (0 if successful, -1 otherwise). Whenever an event is registered, an event record is created and inserted in the callout queue with its `Delta` field computed from the `Expire` argument.

Program Guidelines

Write a multi-threaded program where one thread fills the role of a clock interrupt handler and the remaining threads (called clients below) register callbacks. Use the POSIX thread and synchronization functions (e.g., `pthread_mutex_init(3T)`, `pthread_lock(3T)`, `pthread_cond_init(3T)`). Also, see the man pages for `condition(3T)`, `mutex(3T)` and `pthreads(3T)`. This section gives overall guidelines on the program structure and logic. The section Program Notes at the end of this document describes some program details.

The main thread should create and initialize all required resources.
The clock handler thread will run with some period \( P \) which will be an optional command line argument with a default value. It's job is to execute the callback functions that have been registered in the appropriate time intervals.

The client threads should register a callback function, then go to sleep on a condition variable waiting for the specified time to expire. The expiration value \( E \) is either: a) a random value selected by the thread, or b) a value specified on the command line. When the client thread wakes up it must record the wakeup time then reregister the callout. This should be repeated \( C \) times where \( C \) is an optional command line argument with a default value.

There should be just one global callback function `pokeMe` that all client threads will register. This callback function will take an argument that designates which client thread registered it with the callout queue. Note that this function must be reentrant since all client threads use the same callback function! The callback function is a simple function whose sole responsibility is to wake up the thread which registered it.

The callback registration function `registerCallback` adds a new event record to the callout queue. But because there may be more than one thread concurrently executing this function, it should ensure mutual exclusion during execution.

The main program takes three optional command line arguments:

```
```

where \( N \) is the number of client threads; \( C \) is the number of callbacks to be issued by each client; \( P \) is the clock handler period expressed as the number of 10 ms ticks. \( E \) is the client expiration interval (in number of 10 ms ticks). An \( E \)-value of `-` indicates that all client threads will use \( P \) as their callout expiration time (i.e., \( -e P \)). If a `-h` is included on the command line then a help message is printed which summarizes the command syntax and default command line arguments.

**Experiments**

Ideally, the program should achieve periodic scheduling of client threads. For various reasons, this goal can not be achieved, but we would like to know how close we can achieve periodic scheduling. So, each client thread should display the number of measurement samples and the average, standard deviation, maximum and minimum time that it waits on its condition variable. The clock handler thread should display the number of measurement samples and the average, standard deviation, maximum and minimum of the actual (or realized) tick interval. (Note: The Web page will contain code fragments for computing some of these statistics.) Collect a sufficient number of samples to get reasonable results.

**What to Submit**

Hardcopy documentation (described below) should be submitted in class. The electronic submission should be emailed to kenw@arl.wustl.edu using a `shar` format (see the Electronic Submission section below). The electronic submission should include the following: the document (README.txt file), the source code, test scripts, test input data, and short test output data. By short, I mean that the total does not exceed 100 KB. Test output data should NOT be included if it is not referenced in the README.txt file. The README.txt file is a plain text file that contains the information described in follows the organization described in the document Documentation Guideline. Note that the a significant part of this assignment involves performing and presenting the measurement results.
Electronic Submission

Electronically submit your files using the following sequence of commands:

    shar File1 File2 ... > B.shar
    mailx -s B.shar kenw@arl.wustl.edu < B.shar

The end result should be that you mail to me a single shar (shell archive) file containing your files
(source code, test scripts, and test output). Do NOT submit object code or executables.

Late Policy

The hardcopy documentation is due in my CS mail box by noon of the due date, and the
electronic submission is due by midnight of the same day. Late submissions will be accepted up to
one week after the due date but will suffer a 20% penalty.

Extra Credit

See the Web page for a description of the extra credit problem.

Program Notes

Data Structures:

    // type definitions
    typedef unsigned long Tick_t;
    typedef struct {             // callout queue event record
        Event_t    *next;
        Tick_t     delta;
        int        (*fn)();
        void       *arg;
    } Event_t;
    typedef struct {             // callout queue header
        Event_t    *first;
        pthread_mutex_t *lock;
    } EventHead_t;
    typedef struct {
        int         state;   // 1: timer has expired; 0: otherwise
        pthread_mutex_t *lock;  // state mutex
        pthread_cond_t  *expired; // signal when state becomes 1
        Tick_t        t;       // expiration time (in clock ticks)
    } Timer_t;

    // variable declarations
    Tick_t tNow = 0;              // current time (number of ticks)
Control Structures:

- **main Thread**
  - Process command line arguments.
  - Set tick value.
  - Allocate and initialize synchronization structures.
  - Create clock thread and client threads.
  - Wait for all threads to finish.

- **clock Thread**
  - *Atomically* initialize callout queue.
  - Use `select(2)` to set timeout period.
  - *Atomically* remove event record, and call `pokeMe` for each expired event.
  - Repeat the select-remove-call loop until experiment is done.

- **registerCallback** Function
  - Create and initialize new event record.
  - *Atomically* add new event record to callout queue.

- **client Thread**
  - Determine expiration period \( E \) for thread. \( E \) is set once for each thread and is constant thereafter.
  - Do \( C \) times: Register a callback in \( E \) ticks and sleep on a condition variable signal.

- **pokeMe** Callback Function
  - Its argument is a pointer to the timer structure of the thread that registered the callback.
  - Signal waiting thread using a condition variable.