Computer Science and Engineering & Information Technology (2^{nd} Year B.Tech.)
IIIT Kalyani, West Bengal

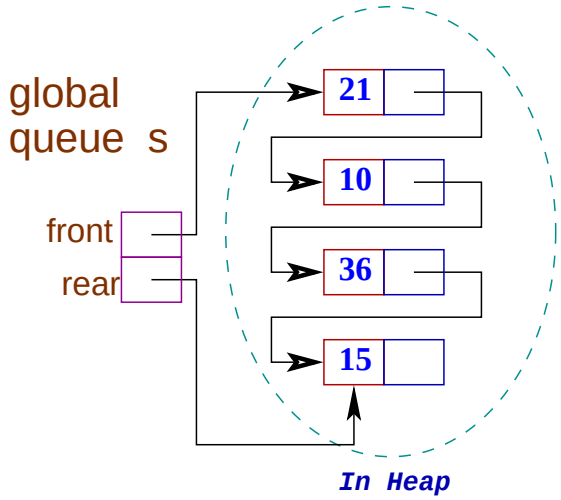
Operating System Lab (CS 411): (Spring: 2019-2020)

Assignment - 9 *Marks: 10*
Assignment Out: *20^{th} March, 2020*

In this experiment we shall see how a thread adds its thread ID in a global queue and suspends itself. Another thread gets a thread ID of a suspended thread from the queue and restarts it. Both add and delete queue operations are performed atomically as more than one threads update the queue.

Organize the program in the following way.

1. Implement a queue using a linked list.




```
// queue.h Queue of thread ID
#ifndef QUEUE_H
#define QUEUE_H

#define ERROR (-1)

typedef struct node {
        int data;
        struct node *next;
} node_t;

class queue{
      node_t *front, *rear;
      public:
      queue();
      bool isEmptyQ();
      void addQ(int n);
      int deleteQ();   // top + delete
                       // returns ERROR when the queue is empty
};
#endif
```

2. Implement
`void tasLockInit(int &lock)`,
`void tasLock(int *lockp)`, and
`void tasUnlock(int &lock)`
using inline assembly code. Use the *bit-test-and-set* (`btsl`) for `tasLock()`. The formal parameter `lockp` is a pointer to a global variable used as a lock. Similarly, `lock` is a reference parameter. A global lock-variable is passed as an actual parameter.

3. The `main()` thread does the following:
   (a) Initializes the lock(s).
   (b) Reads a small positive integer n.
   (c) Creates space for n stacks used by n child threads and keep their base addresses in an array `char *sp[n]`.
   (d) It creates n number of child threads using `clone()`.
   (e) Finally the `main()` thread deletes (`atomically` using `tasLock()` and `tasUnlock()`) nodes from the queue of suspended thread IDs. For each thread ID it sends the signal `SIGCONT` to restart the thread using the system call `kill()`.
   (f) The `main()` thread keeps track of the number of deleted threads. It terminates when the number is n.
4. Each child thread does the following:
   (a) Extracts the thread ID (`syscall(SYS_gettid)`).
   (b) Prints a message.
   (c) Adds the thread ID, a positive integer, in the queue atomically using `tasLock()` and `tasUnlock()`. The lock is already initialized in the `main()` thread.
   (d) It suspends itself using the system call `kill()` with the signal `SIGSTOP`.
   (e) When it comes out of suspended state (after receiving the signal from the `main()` thread) , it prints a second message and terminates.
5. *Note:*
   (a) You may introduce a delay between thread creation, suspension and restarting.
   (b) It may be necessary to use a another lock for printing messages.

Input/Output:

```
$ ./a.out
Enter a +ve integer: 1
Thread: 4170 going to sleep
Thread: 4170 going to terminate

$ ./a.out
Enter a +ve integer: 2
Thread: 4174 going to sleep
Thread: 4175 going to sleep
Thread: 4175 going to terminate
Thread: 4174 going to terminate

$ ./a.out
Enter a +ve integer: 20
Thread: 4177 going to sleep
Thread: 4178 going to sleep
Thread: 4179 going to sleep
Thread: 4185 going to sleep
Thread: 4183 going to sleep
Thread: 4184 going to sleep
Thread: 4188 going to sleep
Thread: 4193 going to sleep
Thread: 4180 going to sleep
Thread: 4186 going to sleep
Thread: 4187 going to sleep
Thread: 4195 going to sleep
Thread: 4196 going to sleep
Thread: 4182 going to sleep
Thread: 4194 going to sleep
```

```
Thread: 4190 going to sleep
Thread: 4189 going to sleep
Thread: 4192 going to sleep
Thread: 4181 going to sleep
Thread: 4191 going to sleep
Thread: 4177 going to terminate
Thread: 4179 going to terminate
Thread: 4178 going to terminate
Thread: 4185 going to terminate
Thread: 4183 going to terminate
Thread: 4195 going to terminate
Thread: 4193 going to terminate
Thread: 4188 going to terminate
Thread: 4190 going to terminate
Thread: 4186 going to terminate
Thread: 4181 going to terminate
Thread: 4191 going to terminate
Thread: 4184 going to terminate
Thread: 4189 going to terminate
Thread: 4196 going to terminate
Thread: 4187 going to terminate
Thread: 4180 going to terminate
Thread: 4194 going to terminate
Thread: 4182 going to terminate
Thread: 4192 going to terminate
$
```