Bounded Queue

The Producer–Consumer Problem requires the synchronization of two concurrent threads, one that produces new items, and one that consumes them. The producer and the consumer share a fixed-size buffer (bounded queue) for the items.

Implement a bounded queue datatype and operations, as specified in this header:

queue.h

```
struct queue;
struct queue *queue_init(unsigned int size);
void queue_put(struct queue *q, int value);
int queue_get(struct queue *q);
void queue_destroy(struct queue *q);
```

Define the contents of the `struct queue` datatype, as well as the bodies of the four required functions, into `queue.c`. A `main.c`, which implements the producer and consumer threads that exercise the bounded queue, will be provided; build the whole thing into a program called `queue` with this Makefile:

Makefile

```
CFLAGS=-Wall -g -pthread
LDFLAGS=-pthread

OBJECTS=\
  queue.o \
  main.o

queue: $(OBJECTS)
queue.o: queue.c queue.h
main.o: main.c queue.h

.PHONY: clean
clean:
        rm -f queue $(OBJECTS)
```

The `queue` program spawns two threads, a producer and a consumer, which wake up every second to do their jobs. A few options can manipulate their behavior, as shown in the usage information:

```
$ ./queue -?
Usage: solution/queue [OPTIONS]

Options:
  -n SIZE     Let the queue hold SIZE items at once
  -p NUM      Produce NUM items each second
  -c NUM      Consume NUM items each second
  -d SECONDS  Stop after SECONDS duration
```

Regarding the actual bounded queue implementation, the manner in which the `int` values are actually stored is not important for this assignment; a circular buffer might be easiest, since that data structure lends itself to a fixed-size queue. The synchronization itself can be gleaned from the description of the Producer–Consumer Problem above, with the semaphores implemented using the POSIX semaphore interface (where `sem_wait` corresponds to the `down` operation in the pseudocode description, and `sem_post` corresponds to `up`).

For good karma, you may—optionally—attack the multiple-producers, multiple-consumers problem, in which there may be more than one of each kind of thread. A pseudocode solution for that may be found in the same place.

When finished, submit `queue.c` on CMS.