```c
/*
* Name: Marcos Ondruska
* Panther ID: 2685885
*
* Usage: Application runs two competing threads to update a shared glabal counter variable for
* a max of 2000000 updates to each thread. Thread 1 has a bonus of 100 added every time the counter
* variable is divisible by 100. The current value of the counter is also printed at the end of the
* remainder for each individual thread.
*/

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define MAX_UPDATES 2000000

/* Struct a shared variable to store result */
struct shared_data
{
int value;
};

/* Global shared variable */
struct shared_data *counter;

/* Mutex lock */
pthread_mutex_t mutex;

/* Thread1 function */
void *thread1()
{
int i = 0;
int bonus = 0;
int currentValue = 0;
```

```
while (i < MAX_UPDATES)
{
/* Entry section */
if (pthread_mutex_trylock(&mutex) == 0)
{
/* Critical section */
if ((counter->value) < 4000000)
{
if ((counter->value % 100) == 0)
{
bonus++;
counter->value += 100;
}

counter->value++;
currentValue = counter->value;
}
i++;
/* Exit section */
pthread_mutex_unlock(&mutex);
}
}
/* Remainder section */
printf("I'm thread1, I did %d updates and I got the bonus %d times, counter =
%d\n",
i,
bonus,
currentValue);
return NULL;
}

/* Thread2 function */
void *thread2()
{
int i = 0;
int currentValue = 0;

while (i < MAX_UPDATES)
```

```
{
/* Entry section */
if (pthread_mutex_trylock(&mutex) == 0)
{
/* Critical section */
if ((counter->value) < 4000000)
{
counter->value++;
currentValue = counter->value;
}
i++;
/* Exit section */
pthread_mutex_unlock(&mutex);
}
}
/* Remainder section */
printf("I'm thread2, I did %d updates, counter = %d\n",
i,
currentValue);
return NULL;
}

int main()
{
pthread_t tid[2];
int rc;

/* Allocate memory for shared data */
counter = (struct shared_data *)malloc(sizeof(struct shared_data));
counter->value = 0;

/* Initialize mutex lock */
if ((pthread_mutex_init(&mutex, NULL)))
{
printf("Error occured when initialize mutex lock.");
exit(0);
}
```

```
/* Required to schedule thread independently */
pthread_attr_t attr;
if ((pthread_attr_init(&attr)))
{
printf("Error occured when initialize pthread_attr_t.");
exit(0);
}
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

/* Create thread1 */
if ((rc = pthread_create(&tid[0], &attr, thread1, NULL)))
{
fprintf(stderr, "ERROR: pthread_create, rc: %d\n", rc);
exit(0);
}

/* Create thread2 */
if ((rc = pthread_create(&tid[1], &attr, thread2, NULL)))
{
fprintf(stderr, "ERROR: pthread_create, rc: %d\n", rc);
exit(0);
}

/* Wait for threads to finish */
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);

printf("From parent counter = %d\n", counter->value);

/* Clean up */
pthread_mutex_destroy(&mutex);
free(counter);
pthread_exit(NULL);

return 0;
}
```