/\*

\* 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;

}