Laboratorio: Trabajo de PThreads 1

Jeisson Yato Tintaya

Octubre 2020

Github: https://github.com/JYato/Laboratorio

1. Implementar y comparar las técnicas de sincronización Busy-waiting y Mutex, obtener una tabla similar a la Tabla 4.1 del libro.

   El sistema sobre el que se ejecutó el programa tiene 2 núcleos disponibles.En la figura 1 se puede ver la comparación de ambas formas. Como menciona el libro, la forma busy-waiting empeora a medida que la cantidad de núcleos sea menor a la de threads. Caso diferente con la versión con Mutex.

2. Basado en la sección 4.7, implementar un ejemplo de productor-consumidor. Implementación de pase de mensajes entre threads con semáforos.

   La diferencia crucial entre semáforos y mutexes es es que no hay propiedad(own) asociada con un semáforo. El thread principal puede inicializar todos los semáforos a 0, es decir, "locked", y luego cualquier thread puede ejecutar un sem_post en cualquiera de los semáforos y, de forma similar, cualquier hilo puede ejecutar sem_wait en cualquiera de los semáforos.
   Hay que tener en cuenta que el problema de envío de mensajes no involucró una sección crítica. El problema no era que hubiera un bloque de código que solo podía ser ejecutado por un thread a la vez. Más bien, el thread myrank no pudo continuar hasta que el thread source haya terminado de crear el mensaje. Este tipo de sincronización, cuando un thread no puede continuar hasta que otro thread ha realizado alguna acción, se denomina sincronización productor-consumidor. En la figura 2 se puede ver el código que proporciona el libro. La implementación completa se encuentra en el github.

3. Implementar las diferentes implementaciones de barreras en PThreads mostradas en el libro.

   - Busy waiting y mutex: Figura 3
   - Semáforas: Figura 4
   - Variables condicionales: Figura 5

Threads	Busy-wait	Mutex
1	2.886450	2.865729
2	2.826381	2.824662
4	2.898409	2.845130
8	3.219669	2.806082
16	3.261158	2.837379
32	3.429780	2.880709
64	3.335700	2.900791

Figure 1: Tabla comparativa

```
1  /* messages is allocated and initialized to NULL in main   */
2  /* semaphores is allocated and initialized to 0 (locked) in
        main */
3  void* Send_msg(void* rank) {
4     long my_rank = (long) rank;
5     long dest = (my_rank + 1) % thread_count;
6     char* my_msg = malloc(MSG_MAX*sizeof(char));
7
8     sprintf(my_msg, "Hello to %ld from %ld", dest, my_rank);
9     messages[dest] = my_msg;
10    sem_post(&semaphores[dest])
          /* ''Unlock'' the semaphore of dest */
11
12    /* Wait for our semaphore to be unlocked */
13    sem_wait(&semaphores[my_rank]);
14    printf("Thread %ld > %s\n", my_rank, messages[my_rank]);
15
16    return NULL;
17 }  /* Send_msg */
```

Figure 2: Paso de mensajes entre threads usando semáforas

```
/* Shared and initialized by the main thread */
int counter;  /* Initialize to 0 */
int thread_count;
pthread_mutex_t barrier_mutex;
. . .

void* Thread_work(. . .) {
   . . .
   /* Barrier */
   pthread_mutex_lock(&barrier_mutex);
   counter++;
   pthread_mutex_unlock(&barrier_mutex);
   while (counter < thread_count);
   . . .
}
```

Figure 3: Usando Busy waiting y mutex

```
/* Shared variables */
int counter;        /* Initialize to 0 */

sem_t count_sem;    /* Initialize to 1 */
sem_t barrier_sem;  /* Initialize to 0 */
. . .
void* Thread_work(...) {
   . . .
   /* Barrier */
   sem_wait(&count_sem);
   if (counter == thread_count-1) {
      counter = 0;
      sem_post(&count_sem);
      for (j = 0; j < thread_count-1; j++)
         sem_post(&barrier_sem);
   } else {
      counter++;
      sem_post(&count_sem);
      sem_wait(&barrier_sem);
   }
   . . .
}
```

Figure 4: Usando semáforas

```
/* Shared */
int counter = 0;
pthread_mutex_t mutex;
pthread_cond_t cond_var;
. . .
void* Thread_work(. . .) {
   . . .
   /* Barrier */
   pthread_mutex_lock(&mutex);
   counter++;
   if (counter == thread_count) {
      counter = 0;
      pthread_cond_broadcast(&cond_var);
   } else {
      while (pthread_cond_wait(&cond_var, &mutex) != 0);
   }
   pthread_mutex_unlock(&mutex);
   . . .
}
```

Figure 5: Usando variables condicionales