CG2271 Real Time Operating Systems

Lab 6 – Synchronization Mechanisms

Answer Book

Name: Song Yangyu Matric No: A0077863N

Name: Li Ruize Matric No: A0078040M

Lab day: Friday Lab time: 4pm ~ 6pm

Question 1 (2 marks)

Output is pasted below:

```
ADD: New value of ctr is 1
PRINT: Current value of ctr is 1
PRINT: ctr is too small. Zzzz.. g'night!
ADD: New value of ctr is 2
ADD: New value of ctr is 3
ADD: New value of ctr is 4
ADD: New value of ctr is 5
ADD: New value of ctr is 6
ADD: New value of ctr is 7
ADD: New value of ctr is 8
ADD: New value of ctr is 9
ADD: New value of ctr is 10
ADD: Reached limit of 10! Waking up print thread
PRINT: ctr is now 10! Exiting thread.
```

This program would first create two threads: add and print.

When print is executed, it would first lock ctr_mutex, then check if ctr reaches the MAX_COUNT; if not, it would be blocked, unlock the ctr_mutex, and wait for the ctr_cond signal.

In the add function, it would add ctr each time in the while loop; if ctr reaches hte MAX_COUNT, it would signal “ctr_cond” so that the blocked print can be executed.

Question 2 (3 marks)

This program does not deadlock because the “pthread_cond_wait” would automatically unlock the “ctr_mutex”.

Question 3 (2 marks)

At least one child thread does not block because the initial value of “sema1” is 1; therefore when the sem_wait is first called, it's positive therefore no need to be blocked.

Question 4 (2 marks)

Each time the main thread calls sem_post, 1 thread is woken.

Question 5 (2 marks)

POSIX semaphores are different from mutexes because semaphore is implemented as a counter – it would count up when there's a “post” signal, and count-down when there's a “wait” signal – by doing so, it allows mulitple threads to be in “run-state” even if the “wait” is being called in each thread; while for the mutex, it's a global boolean variable – it would have no use when one tries to unlock a unlocked process. (but for semophere by doing a “post” would count up, therefore in the future this count-up can cancel a “wait” signal.)

Question 6 (3 marks)

The changes we made were to use a semophare to control the order of execution.

1. add in the inlcude file for semophare:

   #include <semaphore.h>

2. add in a semaphore variable:

   sem_t sema1;

3. in the “add” function, wait for sema1 at the start:

   sem_wait(&sema1);

4. in the “print” function, after the printf statement, add in the “sem_post(&sema1); ” to wake up add function

Question 7 (2 marks)

The final sum is 0 + 1 + 2 + ... + 49 = 1225

Question 8 (3 marks)

The sum is incorrect. This is because the function reader executes before writter has written anything into the queue – meaning, even if there's nothing to read from the queue

Question 9 (6 marks)

The changes we made were:

1. add in “semaphore.h” library to pcomm.h
2. in the struct t (struct for the queue), add in two semaphore:
   1. sem_t sema_full; // wait on full push
   2. sem_t sema_empty; // wait on empty get

3. in pq_create, initialize the two semaphores:
    1. sem_init(&(tmp->sema_full),1,0);
    2. sem_init(&(tmp->sema_empty),1,0);
4. in pq_destory, destory the two semaphores:
    1. sem_destroy(&(q->sema_full));
    2. sem_destroy(&(q->sema_empty));
5. In pq_put, add in wait for semaphore when the queue is full:

```
// Wait if full.
 while(q->count == q->size){
    sem_wait(&(q->sema_full));
 }
```

6. also in this function, when the number of elements changed from 0 to one, give a post to sema_empty:

```
 if(q->count == 1)   sem_post(&q->sema_empty);
```

7. Similarly in pq_get function, add in this
    1.
```
// Wait on empty -- a while loop is needed to use up all the positive posts
   while(q->count == 0)    sem_wait(&(q->sema_empty));
```

    2.
```
// check to wake up the put, if it's no longer full
   if(q->count == q->size -1)  sem_post(&q->sema_full);
```

Question 10 (5 marks)

The changes we made, in the file “lab6_3.c” were:

1. include the library “semaphore.h”
2. create a global variable (semaphore): “sem_t all_completed”
3. in the main function:
    1. initialize this semaphore: sem_init(&all_completed,1,0);
    2. before destorying the queue, wait 6 times for all programs to complete :

for(i=0; i<6; i++) sem_wait(&all_completed);

4. at the end of the writter function and the end of the reader function, post to semaphore all_complete:

   sem_post(&all_completed); // finish execution, post to allow main to go

ANSWER BOOK TOTAL: _____/30

DEMO: ____/5

TOTAL: ____/35