Programming Project Checkpoint 4

IPNS 106010018 Yen-Fong Li

```
void Producer1(void) {
        /*
         * @@@ [2 pt]
         * initialize producer data structure, and then enter
         * an infinite loop (does not return)
         */


        while (1) {
        /* @@@ [6 pt]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
                //SemaphoreWait(p2);
                SemaphoreWait(empty);
                SemaphoreWait(mutex);


                __critical{
                        //buffer = letter;

                        buffer[full] = letter;
                        /*
                        if (empty == 1)
                                buffer[2] = letter;
                        if (empty == 2)
                                buffer[1] = letter;
                        if (empty == 3)
                                buffer[0] = letter;
                        */
                }

                letter = (letter == 'Z') ? 'A' : letter + 1;


                SemaphoreSignal(mutex);
                SemaphoreSignal(full);
                //SemaphoreSignal(p1);
        }
}
```

```
void Producer2(void) {
        /*
         * @@@ [2 pt]
         * initialize producer data structure, and then enter
         * an infinite loop (does not return)
         */


        while (1) {
        /* @@@ [6 pt]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
                //SemaphoreWait(p1);
                SemaphoreWait(empty);
                SemaphoreWait(mutex);


                __critical{
                        //buffer = letter;

                        buffer[full] = num;
                        /*
                        if (empty == 1)
                                buffer[2] = letter;
                        if (empty == 2)
                                buffer[1] = letter;
                        if (empty == 3)
                                buffer[0] = letter;
                        */
                }

                num = (num == '9') ? '0' : num + 1;


                SemaphoreSignal(mutex);
                SemaphoreSignal(full);
                //SemaphoreSignal(p2);
        }
}
```

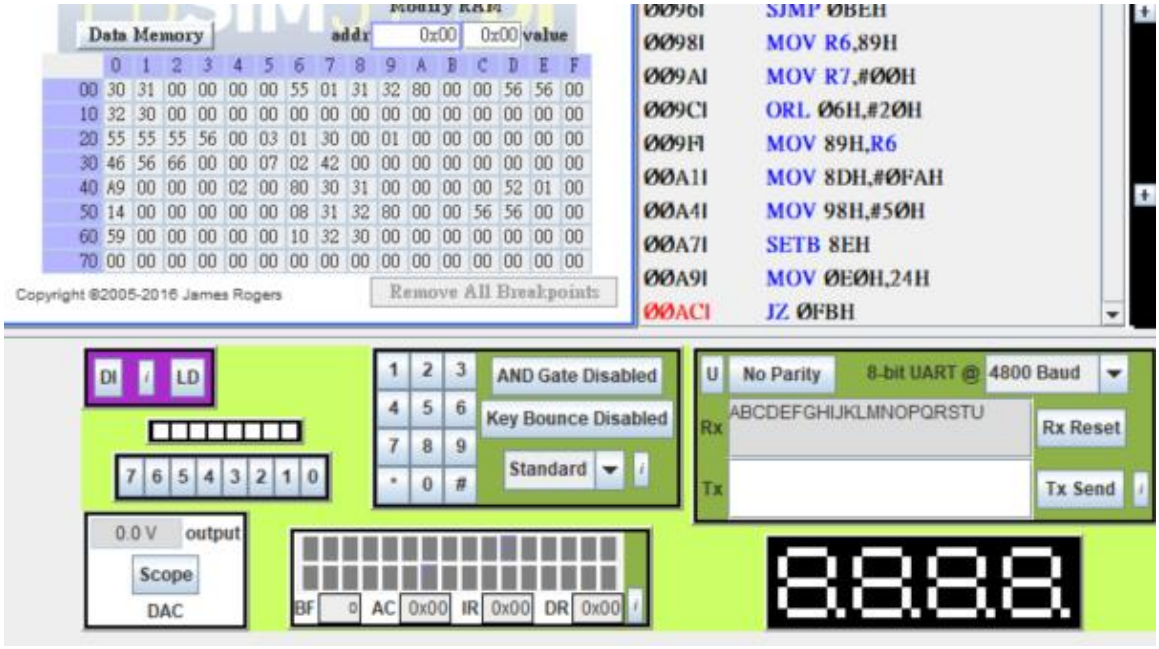
Duplicate the function Producer() before then change “Z” to “9” and “A” to “0”. Rename their function name as Producer1() and Producer2().

```
ThreadCreate(Producer1);
ThreadCreate(Producer2);
Consumer();
```

In main(), we now have to create another thread since we have a new Producer2.

Unfair Version

If we run the code like this, we will observe the output below.



Execution order : Producer1 -> Producer2 -> Consumer

1. When running Producer1, it will feed more than three letters to the buffer so the buffer become full.
2. When it is the turn for the Producer2 to run, it can't feed any number to the buffer since it is full.
3. Consumer will get three letters, which are from Producer1(), out from the buffer and then go back to Producer1. It makes Producer2 starvation.

If we change the create order (Create Producer2 first), we will get "01234567890123…." for the output.

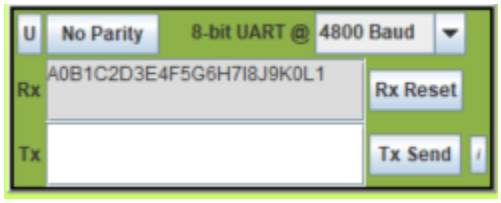
Fair Version

Add two new semaphore – p1 and p2. The idea is that after Producer1 feed a letter to the buffer, it has to wait Producer2 to feed a number. Same as Producer2 in the opposite way.

```
void Producer1(void) {
        /*
         * @@@ [2 pt]
         * initialize producer data structure, and then enter
         * an infinite loop (does not return)
         */


        while (1) {
        /* @@@ [6 pt]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
                SemaphoreWait(p2);
                SemaphoreWait(empty);
                SemaphoreWait(mutex);


                __critical{
                        //buffer = letter;

                        buffer[full] = letter;
                        /*
                        if (empty == 1)
                                buffer[2] = letter;
                        if (empty == 2)
                                buffer[1] = letter;
                        if (empty == 3)
                                buffer[0] = letter;
                        */
                }

                letter = (letter == 'Z') ? 'A' : letter + 1;


                SemaphoreSignal(mutex);
                SemaphoreSignal(full);
                SemaphoreSignal(p1);
        }
}
```

```
void Producer2(void) {
        /*
         * @@@ [2 pt]
         * initialize producer data structure, and then enter
         * an infinite loop (does not return)
         */


        while (1) {
        /* @@@ [6 pt]
         * wait for the buffer to be available,
         * and then write the new data into the buffer */
                SemaphoreWait(p1);
                SemaphoreWait(empty);
                SemaphoreWait(mutex);


                __critical{
                        //buffer = letter;

                        buffer[full] = num;
                        /*
                        if (empty == 1)
                                buffer[2] = letter;
                        if (empty == 2)
                                buffer[1] = letter;
                        if (empty == 3)
                                buffer[0] = letter;
                        */
                }

                num = (num == '9') ? '0' : num + 1;


                SemaphoreSignal(mutex);
                SemaphoreSignal(full);
                SemaphoreSignal(p2);
        }
}
```

In this way we can get the output below.



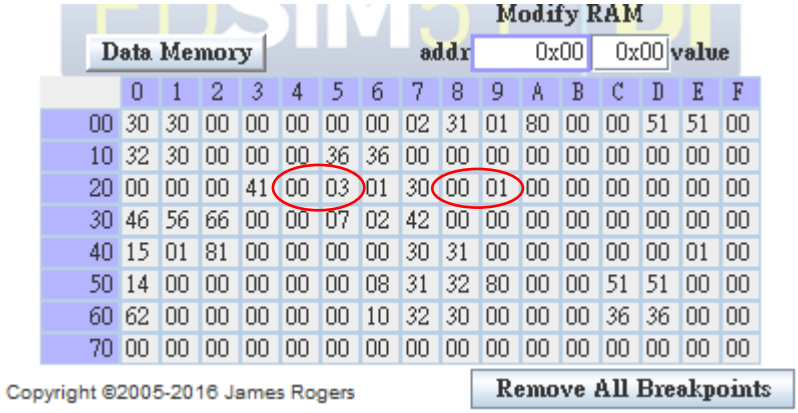
It seems pretty fair for Producer1 and Producer2.

Semaphore Change

Empty and full just work like the last checkpoint

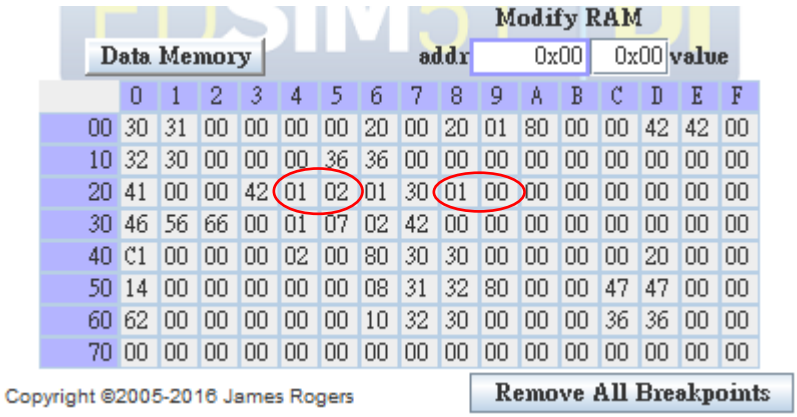
Below is the explanation of the new semaphores p1 and p2 (0x28 and 0x29)

1. Initialize p1 as 0, p2 as 1.



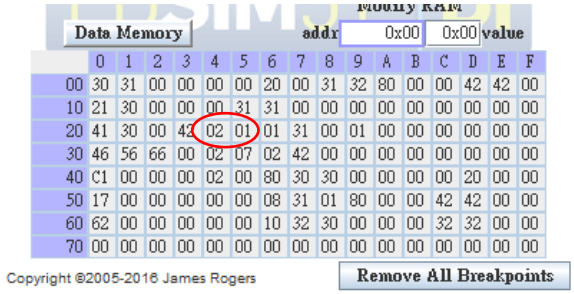
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	30	00	00	00	00	00	02	31	01	80	00	00	51	51	00
10	32	30	00	00	00	36	36	00	00	00	00	00	00	00	00	00
20	00	00	00	41	00	03	01	30	00	01	00	00	00	00	00	00
30	46	56	66	00	00	07	02	42	00	00	00	00	00	00	00	00
40	15	01	81	00	00	00	00	30	31	00	00	00	00	00	01	00
50	14	00	00	00	00	00	08	31	32	80	00	00	51	51	00	00
60	62	00	00	00	00	00	10	32	30	00	00	00	36	36	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

2. When Producer1 feed a letter to the buffer, the semaphore empty and full will change(0x25 and 0x26).
   Semaphore p1 and p2 change to 01 and 00. Same as Producer2 in the opposite way.

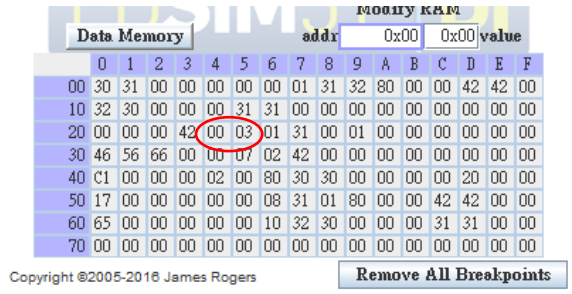


	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	00	00	20	00	20	01	80	00	00	42	42	00
10	32	30	00	00	00	36	36	00	00	00	00	00	00	00	00	00
20	41	00	00	42	01	02	01	30	01	00	00	00	00	00	00	00
30	46	56	66	00	01	07	02	42	00	00	00	00	00	00	00	00
40	C1	00	00	00	02	00	80	30	30	00	00	00	00	20	00	00
50	14	00	00	00	00	00	08	31	32	80	00	00	47	47	00	00
60	62	00	00	00	00	00	10	32	30	00	00	00	36	36	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

3. Execution order : Producer1 -> Producer2 -> Consumer
   Since there are semaphore p1 and p2, Producer1 and Producer2 will feed a letter and a number respectively. Then Consumer will output two things. The semaphore empty and full will change. (0x25 and 0x26)



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	00	00	20	00	31	32	80	00	00	42	42	00
10	21	30	00	00	00	31	31	00	00	00	00	00	00	00	00	00
20	41	30	00	42	02	01	01	31	00	01	00	00	00	00	00	00
30	46	56	66	00	02	07	02	42	00	00	00	00	00	00	00	00
40	C1	00	00	00	02	00	80	30	30	00	00	00	20	00	00	00
50	17	00	00	00	00	00	08	31	01	80	00	00	42	42	00	00
60	62	00	00	00	00	00	10	32	30	00	00	00	32	32	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	30	31	00	00	00	00	00	01	31	32	80	00	00	42	42	00
10	32	30	00	00	00	31	31	00	00	00	00	00	00	00	00	00
20	00	00	00	42	00	03	01	31	00	01	00	00	00	00	00	00
30	46	56	66	00	00	07	02	42	00	00	00	00	00	00	00	00
40	C1	00	00	00	02	00	80	30	30	00	00	00	00	20	00	00
50	17	00	00	00	00	00	08	31	01	80	00	00	42	42	00	00
60	65	00	00	00	00	00	10	32	30	00	00	00	31	31	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00