Lab 6

Operating System

Name: Prakhar Gupta
Roll No: 2103126
Department: CS
Instructor: Dr. Shitala Prasad

Part B:

Q1)
Ans1)
In lab6_1.c trying to solve reader writer problem by synchronizing code using semaphore. The semaphores used are 2 counting semaphore and 1 binary semaphore. One semaphore named buf_used keeps track of buffer size which is filled with data. buf_space tracks buffer size which is free (free of writer to write data into).
Using these semaphore we are setting whether writer / reader should be allowed to access buffer. If buf_used is zero meaning buffer is filled with data and therefore writer should wait before entering/passing new data and same for buf_space and reader. After accessing the buffer (shared memory) now we modify buf_used or buf_space depending on who used the buffer.

Q2)
Ans2)

Instead of using spinlock we are now using semaphore. Using semaphore is better than spinlock as we have to ability to create non busy waiting for process and thus reduce cpu overhead by this. With spinlock we are stuck with busy waiting but using semaphore we can implement non busy waiting.
Lab5 implementation using semaphore.

```
Lab5_6_edited.c:110:4: note: include '<stdlib.h>' or provide
[revai@fedora Lab6]$ ./a.out
The parent process begins.
Parent's report: current index =  0
The child process begins.
Child's report:  current value =  0
Parent's report: current index =  1
Child's report:  current value =  1
Parent's report: current index =  2
Child's report:  current value =  4
Parent's report: current index =  3
Child's report:  current value =  9
Parent's report: current index =  4
Child's report:  current value = 16
Parent's report: current index =  5
Child's report:  current value = 25
Parent's report: current index =  6
Child's report:  current value = 36
Parent's report: current index =  7
Child's report:  current value = 49
Parent's report: current index =  8
Child's report:  current value = 64
Parent's report: current index =  9
Child's report:  current value = 81
The child is done
The parent is done
```

Q3)
Ans3)
Now we have modified our question from Q1 by introducing multiple writer and readers. We now not only need to synchronize each reader and writer but also individual writers also. Here we are using 3 semaphores. Last time, in question 1 we used 2 semaphore, we are going to use them again with procedure and meaning. But we are also going to introduce the third semaphore (binary semaphore) which will help us with multi reader and writer. As there are multiple reader and writer, so if buffer is mostly empty, then multiple writer can access it. But we want to restrict their access to buffer so that this access happens in synchronize way. This is where third semaphore mutex comes. It provides access to only one process at a time whether reader or writer. Thus when a

reader reads, no other reader can read data nor any writer can write data. This is same when a writer writes data into buffer(shared memory).

Q4)
Ans4)
This mutex named semaphore provides exclusive access to shared memory to create synchronization. This is quite important when multiple readers and writers are trying to access data at same time. If not synchronized the value created and update would be vastly affected.
Without mutex Example

T0	Writer1 get access to buffer	buf_space = 5 – 1	in= 0
T1	Writer2 get access to buffer	buf_space = 4-1	in = 0
T2	Writer1 writes 10 in 0 index of buffer		
T3	Writer2 writes 20 in 0 index of buffer		
T4	Writer2 increases in; in++		in = 1
T5	Writer1 increases in; in++		in = 2

We can see the inconsistency caused here as in = 1 does not contain any data in buffer as well as at index = 0 our value got overwrite before reader could read it. Therefore value lost.
This is why we want to give exclusive access to buffer.
Sample Code execution without mutex:

```
The writer process 1 begins.
The writer process 2 begins.
The writer process 3 begins.
The writer process 4 begins.
The writer process 5 begins.
The writer process 6 begins.
The reader process 1 begins.
Reader 1: item  0 == 100
Reader 1: item  1 == 101
Reader 1: item  2 == 102
Reader 1: item  3 == 103
Reader 1: item  4 == 104
The reader process 2 begins.
Reader 2: item  0 == 105
Reader 2: item  1 == 400
Reader 2: item  2 == 200
The reader process 3 begins.
Reader 2: item  3 == 300
Reader 3: item  0 == 500
Reader 3: item  1 == 600
Reader 3: item  2 == 401
All child processes spawned by parent
The reader process 4 begins.
Parent waiting for children to finish
Reader 4: item  0 == 106
Reader 4: item  1 == 201
The reader process 5 begins.
Reader 5: item  0 == 301
Reader 4: item  2 == 601
Reader 3: item  3 == 402
Reader 2: item  4 == 501
Reader 1: item  5 == 107
Reader 5: item  1 == 202
Reader 4: item  3 == 602
Reader 2: item  5 == 502
Reader 1: item  6 == 108
Reader 5: item  2 == 302
Reader 4: item  4 == 602
Reader 3: item  4 == 603
Writer 1 done.
```

Q5)
Ans5)
Our programs will work on principal that reader do not need mutex. Although they still need other semaphore and writer still need all the semaphore (mutex, buf_space,buf_used). When any reader need to read data it does not have any problem with other reader. It just need buf_used to be non negative. Meaning it has new data to read. Thus no mutex. But when writer need to write data, it needs to make sure no other writer is writing on same index of buffer memory. Therefore mutex needed. There will also be no conflict where reader reads while writer is writing on same index as writer will give buf_used only when it has finished writing, therefore buf_used and buf_space will prevent reading before writing conflict.
Output of Code :

```
[revai@fedora Lab6]$ ./a.out
The writer process 4 begins.
The reader process 1 begins.
Reader 1: item  0 == 400
Reader 1: item  1 == 401
Reader 1: item  2 == 402
Reader 1: item  3 == 403
Reader 1: item  4 == 404
Writer 4 done.
The reader process 2 begins.
Reader 2: item  0 == 405
Reader 2: item  1 == 406
Reader 2: item  2 == 407
Reader 2: item  3 == 408
The reader process 4 begins.
Reader 4: item  0 == 409
All child processes spawned by parent
Parent waiting for children to finish
The reader process 5 begins.
The reader process 3 begins.
The writer process 1 begins.
The writer process 2 begins.
The writer process 5 begins.
The writer process 6 begins.
The writer process 3 begins.
Reader 4: item  1 == 100
Reader 5: item  0 == 200
Reader 3: item  0 == 500
Reader 3: item  1 == 101
Reader 3: item  2 == 600
Reader 2: item  4 == 201
Reader 1: item  5 == 300
Reader 2: item  5 == 501
Reader 1: item  6 == 102
Reader 2: item  6 == 601
Reader 1: item  7 == 301
Reader 2: item  7 == 202
Reader 1: item  8 == 502
Reader 2: item  8 == 302
Reader 1: item  9 == 103
Reader 5: item  1 == 602
Reader 4: item  2 == 203
Reader 3: item  3 == 503
Reader 5: item  2 == 303
Reader 4: item  3 == 104
Reader 3: item  4 == 603
Reader 5: item  3 == 204
Reader 4: item  4 == 504
Reader 3: item  5 == 304
Reader 5: item  4 == 105
Reader 4: item  5 == 604
Reader 3: item  6 == 205
Reader 5: item  5 == 505
```