Studio5 Report

STUDIO5

Step1:

- Bingxin Liu

Step2:

Because processes at least will read the memory area, processes access shared memory need the same layout to access the memory correctly. Only then could these processes read same memory object in the same address. Otherwise, one process may read an int value and another process may read an float value, for example, which obviously doesn't make sense.

Step3:

I prefer memcpy(), and I think it is more efficient than element-wise assignment. This is because the memcpy() function use word pointers instead of byte pointers. So, the memcpy() function copy items block by block rather than byte by byte like loop copy.

Step4:

```
pi@lbxpi:~/studios/studio5 $ gcc leader.c -o leader -lrt
pi@lbxpi:~/studios/studio5 $ gcc follewer.c -o follewer -lrt
pi@lbxpi:~/studios/studio5 $ ./leader
array[0] = 1215069295
array[1] = 1311962008
array[2] = 1086128678
array[3] = 385788725
array[4] = 1753820418
array[5] = 394002377
array[6] = 1255532675
array[7] = 906573271
array[8] = 54404747
array[9] = 679162307
pi@lbxpi:~/studios/studio5 $ ./follewer
array[0] = 1215069295
array[1] = 1311962008
array[2] = 1086128678
array[3] = 385788725
array[4] = 1753820418
array[5] = 394002377
array[6] = 1255532675
array[7] = 906573271
array[8] = 54404747
array[9] = 679162307
pi@lbxpi:~/studios/studio5 $
```

If we didnt remove O_CREAT, then the shm_open function will create a new memory area. As a result, the memory area in leader program will be different from the one in follewer program, which means the memory area is not shared.

Then, if we didnt remove ftruncate, in this case, we will add a new memory area into the allocated shared memory or the memory newly allocated in the follewer program, if we didn'751,879.6993,007,518.796t remove O_CREAT as well.

Step5:

```
pi@lbxpi:~/studios/studio5 $ ./leader
Initialized, waiting for the follower to be created.
```

```
array[0] = 1215069295
array[1] = 1311962008
array[2] = 1086128678
array[3] = 385788725
array[4] = 1753820418
array[5] = 394002377
array[6] = 1255532675
array[7] = 906573271
array[8] = 54404747
array[9] = 679162307
writing finished, waiting for the follower to read.
the follower have read, destroy the shared memory.
pi@lbxpi:~/studios/studio5 $


pi@lbxpi:~/studios/studio5 $ ./follewer
follower created, waiting for the leader to finish writing.
array[0] = 1215069295
array[1] = 1311962008
array[2] = 1086128678
array[3] = 385788725
array[4] = 1753820418
array[5] = 394002377
array[6] = 1255532675
array[7] = 906573271
array[8] = 54404747
array[9] = 679162307
reading finished, unlink self.
pi@lbxpi:~/studios/studio5 $
```

As obeserved, there isn't a concurrency situation happens during programs are running. This is because, when the leader is writing or initializing, the follewer is waiting, and when the follewer is reading, the leader is waiting rather than writing. As a result, concurrencies that two programs are waiting the same resource or one program is writing and at the mean time another program is reading didn't happen, which means the concurrency protocol correctly avoids data races, deadlocks, and other hazards.

Step6:

```
// 1,000,000 in pi
real  0m0.133s
user 0m0.104s
sys      0m0.028s
// bandwidth = 4,000,000/0.133 = 30,075,187.96 Bytes/sec ~ 30MB/s

// 2,000,000 in pi
real  0m0.259s
user 0m0.228s
sys      0m0.030s
// bandwidth = 8,000,000/0.259 = 30,888,030.88 Bytes/sec ~ 30MB/s

// 1,000,000 in cluster
real  0m0.040s
user 0m0.018s
sys      0m0.004s
// bandwidth = 4,000,000/0.040 = 100,000,000 Bytes/sec ~ 100MB/s

// 2,000,000 in cluster
real  0m0.050s
user 0m0.032s
sys      0m0.009s
// bandwidth = 8,000,000/0.050 = 160,000,000 Bytes/sec ~ 160MB/s
```