CSE3221
Assignment #2

The Bounded-Buffer Problem


```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0
```

```
while (true) {
    . . .
  /* produce an item in next_produced */
    . . .
  wait(empty);
  wait(mutex);
    . . .
  /* add next_produced to the buffer */
    . . .
  signal(mutex);
  signal(full);
}
```

Figure 7.1 The structure of the producer process.

The Bounded-Buffer Problem

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0
```



```
while (true) {
  wait(full);
  wait(mutex);
    . . .
  /* remove an item from buffer to next_consumed */
    . . .
  signal(mutex);
  signal(empty);
    . . .
  /* consume the item in next_consumed */
    . . .
}
```

Figure 7.2 The structure of the consumer process.

The Thread Code

```
/*** thread code to execute ***/

void *producer(void *param) {
        while(TRUE)     {
                nsleep();
                read_byte(...);
                nsleep();
                produce(...);
        }
}

void *consumer(void *param) {
        while(TRUE)     {
                nsleep();
                consume(...);
                nsleep();
                write_byte(...);
        }
}
```

Nanosleep

```
// nanosleep for <10ms


int nsleep() {
  struct timespec tim = {0, 0};
  tim.tv_sec = 0;
  tim.tv_nsec = rand() % 10000000L;
  if(nanosleep(&tim , NULL) < 0 ) {
    fprintf(stderr, "Nano sleep system call failed \n");
    exit(1); //ERROR
  }
 }
```

read_byte

```
void read_byte(int thread, BufferItem *item) {

  /* Acquire mutex lock to protect read and toLog*/
  pthread_mutex_lock(&mutex);
  //get current offset
  if ((item->offset = lseek(fin, 0, SEEK_CUR)) < 0) {
      pthread_mutex_unlock(&mutex); /* Release read mutex lock */
      fprintf(stderr, "Cannot seek output file.\n");
      exit(1); //ERROR
  }
 //read the byte
  if( read(fin, &(item->data), 1) < 1) {
      printf("read_byte PT%d EOF pthread_exit(0)\n", thread);
      pthread_mutex_unlock(&mutex); /* Release read mutex lock */
      pthread_exit(0); //EOF
  }

  toLog("read_byte", thread, *item, -1); //log data

  pthread_mutex_unlock(&mutex); /* Release read mutex lock */
}
```

write_byte

```
void write_byte(int thread, BufferItem item) {

   /* Acquire mutex lock to protect write and toLog*/
   pthread_mutex_lock(&mutex);
   //set current offset
   if (lseek(fout, item.offset, SEEK_SET) < 0) {
       pthread_mutex_unlock(&mutex); /* Release write mutex lock */
       fprintf(stderr, "Cannot seek output file.\n");
       exit(1); //ERROR
   }
   //write the byte
   if( write(fout, &item.data, 1) < 1) {
       pthread_mutex_unlock(&mutex); /* Release write mutex lock */
       fprintf(stderr, "Cannot write to output file.\n");
       exit(1); //ERROR
   }

   toLog("write_byte", thread, item, -1); //log data

   pthread_mutex_unlock(&mutex); /* Release write mutex lock */
}
```

produce

```
// produce to buffer

void produce(int thread, BufferItem item) {

   sem_wait(&empty); /* Acquire empty semaphore */

    /* Acquire mutex lock to protect buffer (insertPointer ) and toLog*/
   pthread_mutex_lock(&mutex);

   /* insert_item */
   buffer[insertPointer] = item;
   toLog("produce", thread, item, insertPointer); //log data
   insertPointer = (insertPointer + 1) % bufSize;

   /* Release mutex lock and signal full semaphore */
   pthread_mutex_unlock(&mutex);
    sem_post(&full);
}
```

consume

```
// consume from buffer
void consume(int thread, BufferItem *item) {

   sem_wait(&full); /* Acquire full semaphore */
/*To count the consumer threads waiting on the semaphore, you can
create your own counting function to call here (note that on Linux
sem_getvalue() may not provide you with the values you need):
   sem_wait_full_count(); //Acquire full semaphore with counting*/

/* Acquire mutex lock to protect buffer (removePointer) and toLog*/
   pthread_mutex_lock(&mutex);

   /* remove item */
   *item = buffer[removePointer];
   toLog("consume", thread, *item, removePointer); //log data
   removePointer = (removePointer + 1) % bufSize;

   /* Release mutex lock and signal empty semaphore */
   pthread_mutex_unlock(&mutex);
   sem_post(&empty);
}
```

End Condition?

Producer threads finish on EOF.
The main program can thus joint the producer threads first.

Consumer threads will never finish (will wait on semaphore full!)
The main program can NOT joint the consumer threads.

Possible solutions:
1) Finish after sleep- no guarantee that the copy will be complete
2) Finish when ALL consumer threads are waiting on semaphore full. You will need to count the consumer threads waiting on semaphore full, etc. This is more difficult than 1) but may earn you some bonus.

The Log File

The Log file allows us to track exactly what happened (action, etc), when it did happen (before/after), and how it did happen (involved data).

If discrepancies between the original file and the created copy are found the processing of the mismatching bytes can be investigated in detail by looking into the log.

Looking for specific information, patterns, etc.?

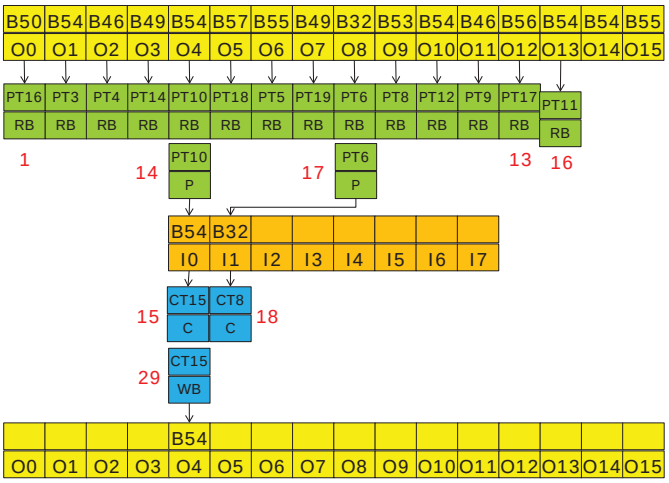
USE grep!

The Log File

P(roducer)**T**(hread); **O**(ffset in the file) **I**(ndex in the buffer) **C**(onsumer)**T**(hread)

1. read_byte PT16 O0 B50 I-1
2. read_byte PT3 O1 B54 I-1
3. read_byte PT4 O2 B46 I-1
4. read_byte PT14 O3 B49 I-1
5. read_byte PT10 O4 B54 I-1
6. read_byte PT18 O5 B57 I-1
7. read_byte PT5 O6 B55 I-1
8. read_byte PT19 O7 B49 I-1
9. read_byte PT6 O8 B32 I-1
10. read_byte PT8 O9 B53 I-1
11. read_byte PT12 O10 B54 I-1
12. read_byte PT9 O11 B46 I-1
13. read_byte PT17 O12 B56 I-1
14. produce PT10 O4 B54 I0
15. consume CT15 O4 B54 I0
16. read_byte PT11 O13 B55 I-1
17. produce PT6 O8 B32 I1
18. consume CT8 O8 B32 I1
19. produce PT14 O3 B49 I2
20. consume CT16 O3 B49 I2
21. produce PT5 O6 B55 I3
22. produce PT19 O7 B49 I4
23. consume CT17 O6 B55 I3
24. consume CT28 O7 B49 I4
25. read_byte PT7 O14 B57 I-1
26. read_byte PT0 O15 B54 I-1
27. produce PT3 O1 B54 I5
28. consume CT29 O1 B54 I5
29. write_byte CT15 O4 B54 I-1
30. produce PT4 O2 B46 I6
31. consume CT21 O2 B46 I6
32. read_byte PT15 O16 B56 I-1
33. read_byte PT13 O17 B32 I-1
34. produce PT8 O9 B53 I7
35. consume CT10 O9 B53 I7
36. read_byte PT1 O18 B52 I-1
37. read_byte PT2 O19 B54 I-1
38. produce PT11 O13 B55 I8
39. produce PT15 O16 B56 I9
40. consume CT23 O13 B55 I8

The Log File Graphically



grep

P(roducer)**T**(hread); **O**(ffset in the file) **I**(ndex in the buffer) **C**(onsumer)**T**(hread)

grep -n PT10 dataset4log.txt

5:read_byte PT10 O4 B54 I-1
14:produce PT10 O4 B54 I0
62:read_byte PT10 O24 B54 I-1
93:produce PT10 O24 B54 I27
140:read_byte PT10 O58 B53 I-1
164:produce PT10 O58 B53 I57
189:read_byte PT10 O80 B55 I-1
210:produce PT10 O80 B55 I77
217:read_byte PT10 O93 B50 I-1
251:produce PT10 O93 B50 I99
294:read_byte PT10 O129 B49 I-1
306:produce PT10 O129 B49 I24
324:read_byte PT10 O142 B53 I-1
369:produce PT10 O142 B53 I44
375:read_byte PT10 O163 B56 I-1
443:produce PT10 O163 B56 I63
448:read_byte PT10 O183 B49 I-1
522:produce PT10 O183 B49 I82

PT10 properly interleaved

grep -n O0 dataset4log.txt

1:read_byte PT16 O0 B50 I-1
46:produce PT16 O0 B50 I11
47:consume CT0 O0 B50 I11
185:write_byte CT0 O0 B50 I-1

The 4 actions in proper order

grep -n I0 dataset4log.txt

14:produce PT10 O4 B54 I0
15:consume CT15 O4 B54 I0
254:produce PT8 O104 B49 I0
591:consume CT28 O104 B49 I0
592:produce PT18 O200 B48 I0
990:consume CT14 O200 B48 I0
991:produce PT2 O300 B32 I0
1390:consume CT11 O300 B32 I0
1391:produce PT10 O399 B32 I0
1791:consume CT2 O399 B32 I0
1792:produce PT9 O501 B53 I0

Use of I0; Buffer Full?

grep

P(roducer)**T**(hread); **O**(ffset in the file) **I**(ndex in the buffer) **C**(onsumer)**T**(hread)

grep -n prod dataset4log.txt

14:produce PT10 O4 B54 I0
17:produce PT6 O8 B32 I1
19:produce PT14 O3 B49 I2
21:produce PT5 O6 B55 I3
22:produce PT19 O7 B49 I4
27:produce PT3 O1 B54 I5
30:produce PT4 O2 B46 I6
34:produce PT8 O9 B53 I7
38:produce PT11 O13 B55 I8
39:produce PT15 O16 B56 I9
44:produce PT0 O15 B54 I10
46:produce PT16 O0 B50 I11
48:produce PT18 O5 B57 I12
51:produce PT15 O21 B55 I13
53:produce PT9 O11 B46 I14
54:produce PT17 O12 B56 I15
57:produce PT3 O20 B46 I16
60:produce PT13 O17 B32 I17

Ixx in proper increasing order

grep -n cons dataset4log.txt

15:consume CT15 O4 B54 I0
18:consume CT8 O8 B32 I1
20:consume CT16 O3 B49 I2
23:consume CT17 O6 B55 I3
24:consume CT28 O7 B49 I4
28:consume CT29 O1 B54 I5
31:consume CT21 O2 B46 I6
35:consume CT10 O9 B53 I7
40:consume CT23 O13 B55 I8
41:consume CT24 O16 B56 I9
45:consume CT26 O15 B54 I10
47:consume CT0 O0 B50 I11
49:consume CT14 O5 B57 I12
52:consume CT2 O21 B55 I13
55:consume CT13 O11 B46 I14
56:consume CT22 O12 B56 I15
58:consume CT7 O20 B46 I16
61:consume CT3 O17 B32 I17

Ixx in proper increasing order

End of Assignment #2