

HW8. Concurrency Practice

학번: 20170404

이름: 한종수

제출일: 2021. 5. 25

1. 과제 개요

Counter 와 Bounded Buffer 를 Lock 을 사용하여 올바르게 구현하는 프로그램을 만든다.

2. 소스코드

(수정한 코드를 캡처하고 간단히 설명함)

1) counter.c

```
1 ~ #include <stdio.h>
2 ~ #include <stdlib.h>
3 ~ #include <pthread.h>
4
5 volatile int counter = 0;
6 int loops;
7
8 ~ void *worker(void *arg)
9 {
10     int i;
11 ~     for (i = 0; i < loops; i++)
12     {
13         counter++;
14     }
15
16     return NULL;
17 }
18
19 ~ int main(int argc, char *argv[])
20 {
21 ~     if (argc != 2)
22     {
23         fprintf(stderr, "usage : threads <value>\n");
24         exit(1);
25     }
26
27     loops = atoi(argv[1]);
28     pthread_t p1, p2;
29     printf("Initial value : %d\n", counter);
30
31     pthread_create(&p1, NULL, worker, NULL);
32     pthread_create(&p2, NULL, worker, NULL);
33     pthread_join(p1, NULL);
34     pthread_join(p2, NULL);
35     printf("Final value : %d\n", counter);
36
37     return 0;
38 }
```

main에서 thread 두 개를 생성하여 각 thread가 worker 함수를 실행한다. worker 함수에서는 counter 공유변수를 loops 수 만큼 증가시킨다. counter 변수는 lock이 없어 mutual exclusion이 보장이 되지 않는다.

2) counter_lock.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <pthread.h>
5
6 volatile int counter = 0;
7 int loops;
8
9 void *worker(void *arg)
10 {
11     int i;
12     pthread_mutex_lock((pthread_mutex_t *)arg);
13     for (i = 0; i < loops; i++)
14     {
15         counter++;
16     }
17     pthread_mutex_unlock((pthread_mutex_t *)arg);
18
19     return NULL;
20 }
21
22 int main(int argc, char *argv[])
23 {
24     if (argc != 2)
25     {
26         fprintf(stderr, "usage : threads <value>\n");
27         exit(1);
28     }
29
30     loops = atoi(argv[1]);
31     pthread_t p1, p2;
32     printf("Initial value : %d\n", counter);
33
34     pthread_mutex_t lock;
35     int rc = pthread_mutex_init(&lock, NULL);
36     assert(rc == 0);
37
38     pthread_create(&p1, NULL, worker, &lock);
39     pthread_create(&p2, NULL, worker, &lock);
40     pthread_join(p1, NULL);
41     pthread_join(p2, NULL);
42     printf("Final value : %d\n", counter);
43
44     return 0;
45 }
```

main에서 lock을 선언하고 초기화 시킨다. 그리고 lock을 thread의 인자로 넘겨준다.

worker는 lock을 받아 lock에 대한 권한을 얻은 상태에서 counter를 증가시켜 shared variable에 대한 mutual exclusion을 보장한다.

3) bounded_buffer.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <pthread.h>
5
6 #define MAX 5
7
8 int buffer[MAX];
9 int fill = 0;
10 int use = 0;
11 int count = 0;
12 int psum = 0;
13 int csum = 0;
14
15 void put(int value)
16 {
17     buffer[fill] = value;
18     fill = (fill + 1) % MAX;
19     count++;
20     psum += (value - fill);
21 }
22
23 int get()
24 {
25     int tmp = buffer[use];
26     use = (use + 1) % MAX;
27     count--;
28     csum += (tmp - use);
29
30     return tmp;
31 }
32
33 int loops = 0;
34 pthread_mutex_t _empty = PTHREAD_MUTEX_INITIALIZER;
35 pthread_mutex_t _fill = PTHREAD_MUTEX_INITIALIZER;
36 pthread_cond_t _mutex = PTHREAD_COND_INITIALIZER;
37
38 void *producer(void *arg)
39 {
40     int i;
41     for (i = 0; i < loops; i++)
42     {
43         pthread_mutex_lock(&_mutex);
44         while (count == MAX)
45         {
46             pthread_cond_wait(&_empty, &_mutex);
47         }
48         put(i);
49         pthread_cond_signal(&_fill);
50         pthread_mutex_unlock(&_mutex);
51     }
52 }
53
54 void *consumer(void *arg)
55 {
56     int i;
57     for (i = 0; i < loops; i++)
58     {
59         pthread_mutex_lock(&_mutex);
60         while (count == 0)
61         {
62             pthread_cond_wait(&_fill, &_mutex);
63         }
64         int tmp = get();
65         pthread_cond_signal(&_empty);
66         pthread_mutex_unlock(&_mutex);
67         printf("%d\n", tmp);
68     }
69 }
70
71 int main(int argc, char *argv[])
72 {
73     if (argc != 2)
74     {
75         fprintf(stderr, "usage : threads <value>\n");
76         exit(1);
77     }
78     loops = atoi(argv[1]);
79     pthread_t p1, p2, p3, p4;
80
81     pthread_create(&p1, NULL, producer, NULL);
82     pthread_create(&p2, NULL, producer, NULL);
83     pthread_create(&p3, NULL, consumer, NULL);
84     pthread_create(&p4, NULL, consumer, NULL);
85     pthread_join(p1, NULL);
86     pthread_join(p2, NULL);
87     pthread_join(p3, NULL);
88     pthread_join(p4, NULL);
89
90     if (psum == csum)
91     {
92         printf("Test Ok\n");
93     }
94     else
95     {
96         printf("Wrong\n");
97     }
98
99     return 0;
100 }
```

Max의 크기를 5로 정의하고 buffer의 크기를 그에 맞게 설정한다.

그리고 나머지 공유변수들 fill, use, count, psum, csum 을 초기화한다.

put, get 함수는 buffer에 데이터를 넣고 뺀다. put은 count, fill 를 증가시키고, get은 count를 감소시키고 use를 증가시킨다. psum, csum에는 총 생산, 소비 횟수가 기록된다.

producer는 loop를 돌며 loops수 만큼 데이터를 생산한다. 이때 condition variable에 대한 lock 을 이용하여 mutual exclusion을 보장한다. producer는 empty queue에서 consumer 가 데이터를 소비하는 것을 기다린다. 일을 마친 후 consumer 가 대기하는 fill queue 에서 consumer 하나를 깨운다.

consumer는 loop를 돌며 loops수 만큼 데이터를 소비한다. 이때 condition variable에 대한 lock 을 이용하여 mutual exclusion을 보장한다. consumer는 fill queue에서 producer 가 데이터를 생산하는 것을 기다린다. 일을 마친 후 producer 가 대기하는 empty queue 에서 producer 하나를 깨운다.

main에서 producer, consumer thread를 각각 두개 생성한다. 실행 이후 psum csum이 같으면 mutual exclusion이 잘 보장된 것이다.

3. 결과

(테스트 실행결과를 캡처)

1) counter.c 의 잘못된 결과

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$ ./thread 100000
Initial value : 0
Final value : 136388
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$
```

2) counter_lock.c 의 올바른 결과

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$ ./thread 100000
Initial value : 0
Final value : 200000
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$
```

3) bounded_buffer.c 의 올바른 결과

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab8_Concurrency/src$ ./thread 10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
```