

운영체제론 개인 과제 3 보고서

제출자 : 이찬영 (2019098068, ICT 융합학부)

1. 본인이 설계한 두 가지 해법에 대한 설명

<bounded_buffer.c>

```
1  /*
2  * Copyright(c) 2021-2023 All rights reserved by Heekuck Oh.
3  * 이 프로그램은 한양대학교 ERICA 컴퓨터학부 학생을 위한 교육용으로 제작되었다.
4  * 한양대학교 ERICA 학생이 아닌 이는 프로그램을 수정하거나 배포할 수 없다.
5  * 프로그램을 수정할 경우 날짜, 학과, 학번, 이름, 수정 내용을 기록한다.
6  * -----한양대학교 ERICA ICT융합학부 2019098068 이찬영-----
7  * 05.09
8  * atomic_compare_exchange_weak(), 즉 CAE 명령어를 이용하여 생산자와 소비자가 동시에 접근하지 않게 스핀락을 구현해 동기화를 하였습니다.
9  * (CODE LINE : 43, 53~56, 72, 81, 95~98, 109, 118, 127)
10 * 공유버퍼 (bounded-buffer) 문제
11 */
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <stdbool.h>
15 #include <unistd.h>
16 #include <pthread.h>
17
18 #include <stdatomic.h>
19
20 #define M 8
21 #define MAX 1024
22 #define BUFSIZE 4
23 #define RED "\e[0;31m"
24 #define RESET "\e[0m"
25
26 /*
27 * 생산자와 소비자가 공유할 버퍼를 만들고 필요한 변수를 초기화한다.
28 */
29 int buffer[BUFSIZE];
30 int in = 0;
31 int out = 0;
32 int counter = 0;
33 int next_item = 0;
34
35 /*
36 * 생산된 아이템과 소비된 아이템의 로그와 개수를 기록하기 위한 변수
37 */
38 int task_log[MAX][2];
39 int produced = 0;
40
41 int consumed = 0;
42
43 /*
44 * alive 값이 false가 될 때까지 스레드 내의 루프가 무한히 반복된다.
45 */
46 bool alive = true;
47 atomic_bool lock = false; // 다른 스레드가 버퍼에 접근하지 못하도록 한다.
48
49 /*
50 * 생산자 스레드로 실행할 함수이다. 아이템을 생성하여 버퍼에 넣는다.
51 */
52 void *producer(void *arg)
53 {
54     int i = *(int *)arg;
55     int item;
56
57     while (alive) {
58         bool expected = false;
59         while(!atomic_compare_exchange_weak(&lock, &expected, true)) // lock 변수에 대한 atomic 연산을 수행함
60             expected = false;
61         if(counter < BUFSIZE) {
62             /*
63             * 새로운 아이템을 생산하여 버퍼에 넣고 관련 변수를 갱신한다.
64             */
65             item = next_item++;
66             buffer[in] = item;
67             in = (in + 1) % BUFSIZE;
68             counter++;
69             /*
70             * 생산자를 기록하고 중복생산이 아닌지 검증한다.
71             */
72             if (task_log[item][0] == -1) {
73                 task_log[item][0] = i;
74                 produced++;
75             }
76             else {
77                 lock = false;
78                 printf("<red>%d....ERROR: 아이템 %d 중복생산\n", i, item, item);
79                 continue;
80             }
81         }
82     }
83 }
```

(code line : 43)

Bool 타입의 원자변수인 lock을 선언합니다. 이 lock 변수는 이후에 스핀락에서 다른 스레드가 버퍼에 접근하지 못하도록 하기 위해 사용됩니다.

(code line : 53~56)

생산자 스레드로 실행할 Producer() 함수 내에서 CAE 명령어를 사용하여 스핀락을 구현하여 동기화를 실행합니다. 동기화란 락을 획득하려는 스레드와 이미 락을 가진 스레드 간의 동시 접근을 막는 것을 의미합니다. 일단 처음에는 bool 타입 expected 변수를 false로 설정해 줍니다. 그 이후에 atomic_compare_exchange_weak 함수 내에서 만약 lock == expected이면, lock = true로 치환되고 함수는 true를 리턴하게 되어 while문을 빠져나오지 못합니다. 반면에 lock != expected이면, expected = lock으로 치환되고 함수는 false를 리턴하게 되어 while문을 빠져나오게 됩니다. 이렇게 하면 여러 스레드가 동시에 lock 변수를 변경하는 상황에서도 안전하게 락을 획득할 수 있습니다. counter은 producer 스레드와 consumer 스레드에서 공유되기 때문에 producer 스레드에서는 counter < BUF_SIZE (메모리 공간이 buffer 크기보다 작을 때, 즉 생산할 공간이 있을 때) 일 때, 버퍼에 넣고 관련 변수를 갱신해야 중복 생산 ERROR를 피할 수 있습니다.

```
75     }
76 }
77 /*
78  * 생산한 아이템을 출력한다.
79  */
80 printf("<P%d,%d>\n", i, item);
81 lock = false;
82 }
83 pthread_exit(NULL);
84 }
85
86 /*
87  * 소비자 스레드로 실행할 함수이다. 버퍼에서 아이템을 읽고 출력한다.
88  */
89 void *consumer(void *arg)
90 {
91     int i = *(int *)arg;
92     int item;
93
94     while (alive) {
95         bool expected = false;
96         while(!atomic_compare_exchange_weak(&lock, &expected, true))
97             expected = false;
98         if(counter > 0) {
99             /*
100              * 버퍼에서 아이템을 꺼내고 관련 변수를 갱신한다.
101              */
102             item = buffer[out];
103             out = (out + 1) % BUFSIZE;
104             counter--;
105             /*
106              * 소비자를 기록하고 미생산 또는 중복소비 아닌지 검증한다.
107              */
108             if (task_log[item][0] == -1) {
109                 lock = false;
110                 printf(RED"<P%d,%d>RESET"....ERROR: 아이템 %d 미생산\n", i, item, item);
111                 continue;
```

(code line 72, 81)

저는 critical section(임계구역)을 code line 56~80으로 보았지만 사실 printf("<P%d,%d>\n", i, item); 까지 임계구역을 볼 필요는 없습니다. 출력은 모든 것이 정상적으로 처리된 다음에 확인 차원에서 하는 것이지 다른 스레드가 들어오지 못하게 차단까지 하면서 할 이유가 전혀 없기 때문입니다. 이후에는 lock = false로 설정해주어 락을 해제합니다. 그렇게 여러 개의 스레드가

while(alive) 안에서 실행되다가 alive = false; 일 때, pthread_exit 함수를 호출하여 종료합니다. 또한 if문 내에 중복 생산이 발생하는 경우에도, lock=false; 를 해줌으로써 다른 스레드가 버퍼에 접근할 수 있도록 해 줍니다.

※ printf("<P%d,%d>Wn", i, item); 위에까지 lock=false; 로 임계구역을 설정해 주면, main 함수 내에서 usleep()을 조정해 주어야 item이 50개 이상 출력됩니다!

(code line 95~98)

소비자 스레드로 실행할 Consumer() 함수 내에서 CAE 명령어를 사용하여 스핀락을 구현합니다. 이후에는 producer() 함수 내에서 설명한 것과 동일합니다. counter은 producer 스레드와 consumer 스레드에서 공유되기 때문에 consumer 스레드에서는 counter > 0 (메모리 공간이 0보다 클 때, 즉 소비할 것이 있을 때) 일 때, 버퍼에서 아이템을 꺼내야 중복 소비 ERROR를 방지할 수 있습니다.

```
112     }
113     else if (task_log[item][1] == -1) {
114         task_log[item][1] = i;
115         consumed++;
116     }
117     else {
118         lock = false;
119         printf(RED"<C%d,%d>\"RESET\"....ERROR: 아이템 %d 중복소비\\n", i, item, item);
120         continue;
121     }
122 }
123 /*
124  * 소비할 아이템을 빨간색으로 출력한다.
125  */
126 printf(RED"<C%d,%d>\"RESET\"\\n", i, item);
127 lock = false;
128 }
129 pthread_exit(NULL);
130 }
131
132 int main(void)
133 {
134     pthread_t tid[N];
135     int i, id[N];
136     /*
137     * 생산자와 소비자를 기록하기 위한 logs 배열을 초기화한다.
138     */
139     for (i = 0; i < MAX; ++i)
140         task_log[i][0] = task_log[i][1] = -1;
141     /*
142     * N/2 개의 소비자 스레드를 생성한다.
143     */
144     for (i = 0; i < N/2; ++i) {
145         id[i] = i;
146         pthread_create(tid+i, NULL, consumer, id+i);
147     }
148 }
```

(code line 109, 118, 126)

임계구역 이후에는 lock = false로 설정해주어 락을 해제합니다. 그렇게 여러 개의 스레드가 while(alive) 안에서 실행되다가 alive = false; 일 때, pthread_exit 함수를 호출하여 종료합니다. 이후에는 bounded_buffer.skeleton.c에서 설계한 로직대로 코드가 구현됩니다. 나머지는 위의 설명과 동일합니다. ((code line 72, 81) 설명 참조)

<bounded_waiting.c>

```
1  /*
2   * Copyright(c) 2021-2023 All rights reserved by Heekuck Oh.
3   * 이 프로그램은 한양대학교 ERICA 컴퓨터학부 학생을 위한 교육용으로 제작되었다.
4   * 한양대학교 ERICA 학생이 아닌 이는 프로그램을 수정하거나 배포할 수 없다.
5   * 프로그램을 수정할 경우 날짜, 학과, 학번, 이름, 수정 내용을 기록한다.
6   * -----한양대학교 ERICA ICT융합학부 2019098068 이찬영-----
7   * 05.09
8   * atomic_compare_exchange_weak, 즉 CAE 명령어를 사용하여 여러 개의 스레드가 정해진 차례 안에서
9   * 락을 얻을 수 있도록 스핀락을 구현해 동기화를 하였습니다.
10  * (CODE LINE : 36, 45~52, 64~72)
11  * 유한대기 (bounded-waiting) 문제
12  */
13  #include <stdio.h>
14  #include <stdbool.h>
15  #include <unistd.h>
16  #include <pthread.h>
17
18  #include <stdatomic.h>
19
20  #define N 8          /* 스레드 개수 */
21  #define RUNTIME 100000 /* 출력량을 제한하기 위한 실행시간 (마이크로초) */
22
23  /*
24   * ANSI 컬러 코드: 출력을 쉽게 구분하기 위해서 사용한다.
25   * 순서대로 BLK, RED, GRN, YEL, BLU, MAG, CYN, WHT, RESET을 의미한다.
26   */
27  char *color[N+1] = {"\e[0;30m", "\e[0;31m", "\e[0;32m", "\e[0;33m", "\e[0;34m", "\e[0;35m", "\e[0;36m", "\e[0;37m", "\e[0m"};
28
29  /*
30   * waiting[i]는 스레드 i가 임계구역에 들어가기 위해 기다리고 있음을 나타낸다.
31   * alive 값이 false가 될 때까지 스레드 내의 루프가 무한히 반복된다.
32   */
33  bool waiting[N];
34  bool alive = true;
35
36  atomic_bool lock = false;
37
```

```
38  /*
39   * N 개의 스레드가 임계구역에 배타적으로 들어가기 위해 스핀락을 사용하여 동기화한다.
40   */
41
42  void *worker(void *arg)
43  {
44      int i = *(int *)arg;
45      int j;
46
47      while (alive) {
48          waiting[i] = true;
49          bool expected = false;
50          while(waiting[i] && !atomic_compare_exchange_weak(&lock, &expected, true))
51              expected = false;
52          waiting[i] = false;
53          /*
54           * 임계구역: 알파벳 문자를 한 줄에 40개씩 10줄 출력한다.
55           */
56          for (int k = 0; k < 400; ++k) {
57              printf("%s%s", color[i], 'A'+i, color[N]);
58              if ((k+1) % 40 == 0)
59                  printf("\n");
60          }
61          /*
62           * 임계구역이 성공적으로 종료되었다.
63           */
64          j = (i + 1) % N;
65          while(j != i && !waiting[j]) {
66              j = (j + 1) % N;
67          }
68          if(j == i) { /* 아무도 기다리는 프로세스가 없을 때 */
69              lock = false;
70          }
71          else /* 기다리는 프로세스가 있을 때 */
72              waiting[j] = false;
73      }
74      pthread_exit(NULL);

```

```
75  }
76
77  int main(void)
78  {
79      pthread_t tid[N];
80      int i, id[N];
81
82      /*
83       * N 개의 자식 스레드를 생성한다.
84       */
85      for (i = 0; i < N; ++i) {
86          id[i] = i;
87          pthread_create(&tid[i], NULL, worker, &id[i]);
88      }
89      /*
90       * 스레드가 출력하는 동안 RUNTIME 마이크로초 쉰다.
91       * 이 시간으로 스레드의 출력량을 조절한다.
92       */
93      usleep(RUNTIME);
94      /*
95       * 스레드가 자연스럽게 무한 루프를 빠져나올 수 있게 한다.
96       */
97      alive = false;
98      /*
99       * 자식 스레드가 종료될 때까지 기다린다.
100     */
101     for (i = 0; i < N; ++i)
102         pthread_join(tid[i], NULL);
103     /*
104      * 메인함수를 종료한다.
105      */
106     return 0;
107 }
```

(code line 33~34)

Bool 타입의 원자변수인 lock과 waiting 배열을 선언합니다. lock은 스핀락을 구현하는 변수이며, waiting 배열은 스레드 i가 임계 구역에 들어가기 위해 기다리고 있는지 여부를 나타내는 데에 사용됩니다.

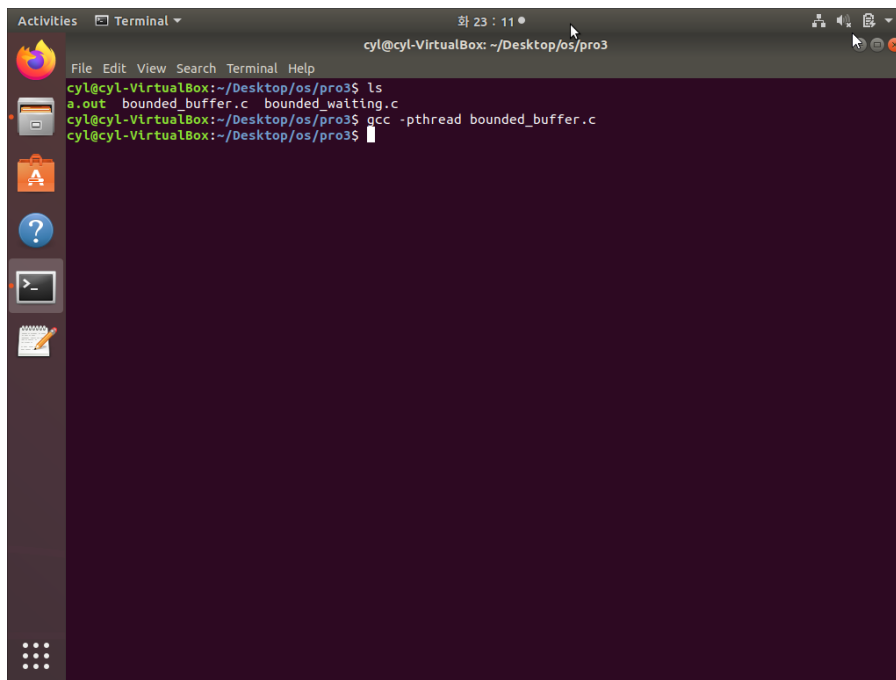
(code line 45~52)

다음 차례의 스레드를 가리키는 변수 j를 선언해 주고, 스레드 i가 임계 구역에 진입하기를 원하는 의미로 `waiting[i] = true;` 를 선언해 줍니다. 이후에는 lock 변수에 대한 `atomic_compare_and_swap` 연산에서 예상값을 나타내는 `expected = false`로 설정해준다. 그 후에는 while 문 내에서 `atomic_compare_exchange_weak` 함수는 lock 변수와 expected 값을 비교하고, 두 값이 같으면 lock 값을 true로 설정하고, 다르면 expected 값에 현재 lock 값을 저장합니다. 이 함수는 약한 비교-교환 연산을 수행하므로 비교시에 expected 값이 변경되었을 경우에도 교환이 일어날 수 있습니다. 다시 말해 이 while문은 스핀락을 통해 임계 구역에 접근하기 위한 동기화를 수행합니다. Code line 52에서는 임계구역에 진입한 후, 스레드 i는 `waiting[i]` 값을 false로 설정하여 다른 스레드가 해당 스레드를 기다리지 않도록 합니다. 이를 통해 다음 스레드가 임계구역에 진입할 수 있도록 합니다.

(code line 64~72)

해당 부분은 스레드가 critical section(임계구역)을 성공적으로 종료한 후 락을 양도하는 과정을 나타냅니다. j변수를 $(i+1) \% N$ 으로 설정하여 다음 스레드의 인덱스를 계산하며 다음 스레드가 기다리고 있는지 아닌지를 `while((j != i) && !waiting[j])`를 통해 검사합니다. 반복문을 통해 얻은 다음 스레드인 j가 현재 스레드(i)와 같은 경우, (즉 맨 마지막) 모든 스레드가 임계구역을 성공적으로 종료한 상태이므로 lock 변수를 0으로 설정하여 락을 해제하게 되며, 그렇지 않은 경우, 다음 스레드가 진입할 수 있도록 `waiting[j]`값을 false로 설정합니다. 이후에는 `bounded_waiting.skeleton.c`에서 설계한 로직대로 코드가 구현됩니다.

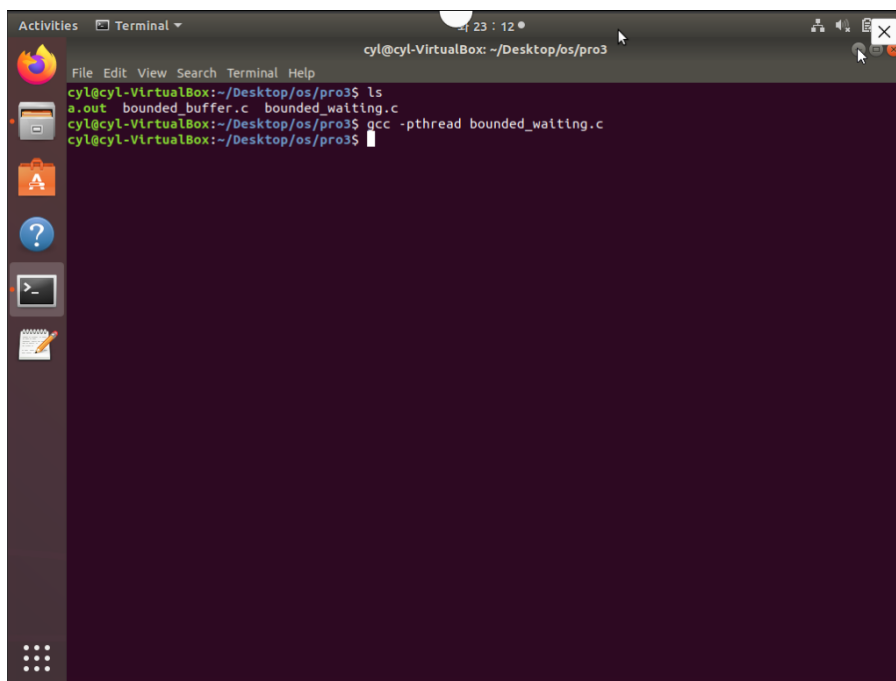
2. 컴파일 과정을 보여주는 화면 캡처



A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (cyl@cyl-VirtualBox: ~/Desktop/os/pro3). The terminal shows the following commands and output:

```
cyl@cyl-VirtualBox:~/Desktop/os/pro3$ ls
a.out bounded_buffer.c bounded_waiting.c
cyl@cyl-VirtualBox:~/Desktop/os/pro3$ gcc -pthread bounded_buffer.c
cyl@cyl-VirtualBox:~/Desktop/os/pro3$
```

아무 오류 없이 bounded_buffer.c 컴파일이 잘 된 것을 확인할 수 있습니다.



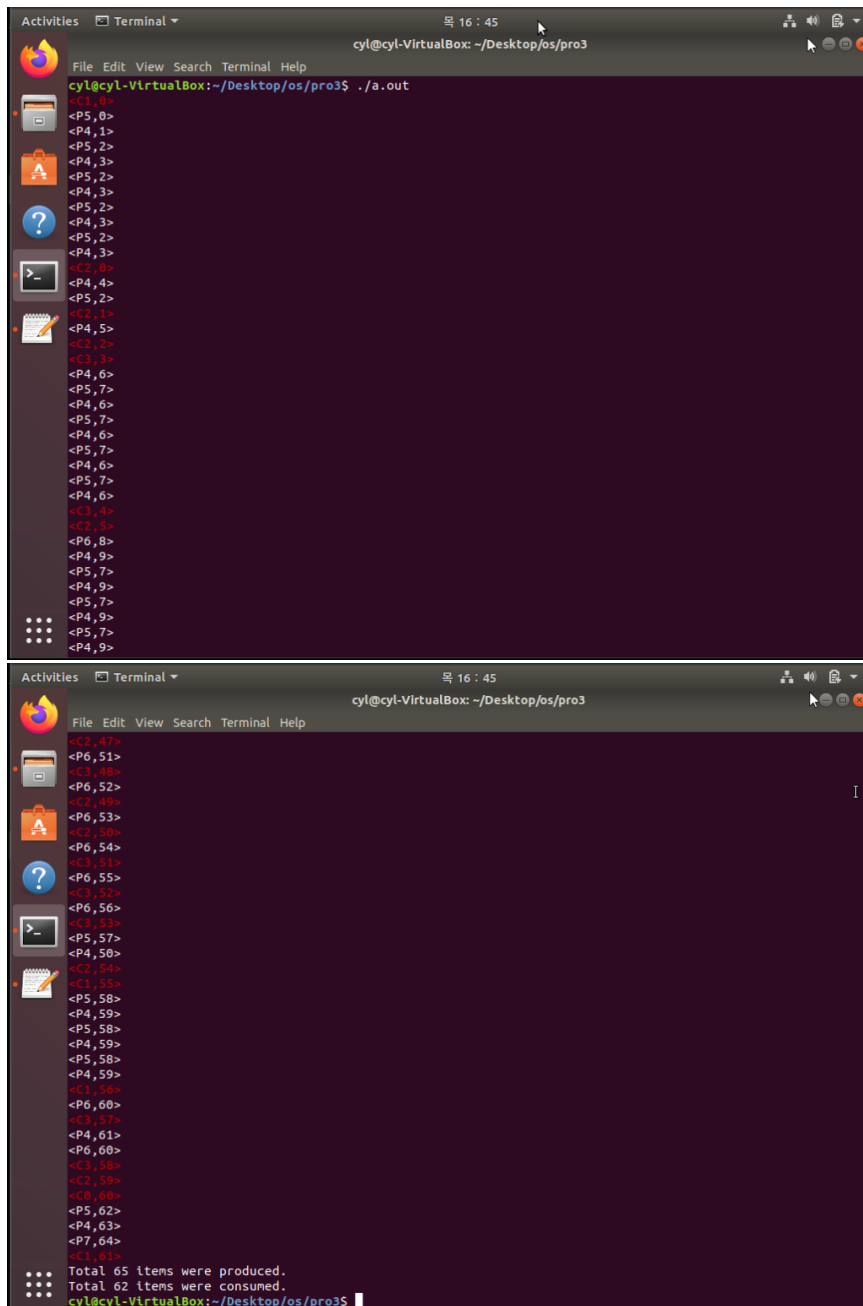
A terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (cyl@cyl-VirtualBox: ~/Desktop/os/pro3). The terminal shows the following commands and output:

```
cyl@cyl-VirtualBox:~/Desktop/os/pro3$ ls
a.out bounded_buffer.c bounded_waiting.c
cyl@cyl-VirtualBox:~/Desktop/os/pro3$ gcc -pthread bounded_waiting.c
cyl@cyl-VirtualBox:~/Desktop/os/pro3$
```

아무 오류 없이 bounded_waiting.c 컴파일이 잘 된 것을 확인할 수 있습니다.

3. 실행 결과물의 주요 장면을 발췌해서 그에 대한 상세한 설명

<bounded_buffer.c>



```
cyll@cyl-VirtualBox: ~/Desktop/os/pro3$ ./a.out
<P5,0>
<P4,1>
<P5,2>
<P4,3>
<P5,2>
<P4,3>
<P5,2>
<P4,3>
<P5,2>
<P4,3>
<C2,0>
<P4,4>
<P5,2>
<C2,1>
<P4,5>
<C2,2>
<C3,3>
<P4,0>
<P5,7>
<P4,6>
<P5,7>
<P4,6>
<P5,7>
<P4,6>
<C3,4>
<C2,5>
<P6,8>
<P4,9>
<P5,7>
<P4,9>
<P5,7>
<P4,9>
<C2,47>
<P6,51>
<C3,48>
<P6,52>
<C3,49>
<P6,53>
<C2,50>
<P6,54>
<C3,51>
<P6,55>
<C3,52>
<P6,56>
<C3,53>
<P5,57>
<P4,50>
<C2,54>
<C3,55>
<P5,58>
<P4,59>
<P5,58>
<P4,59>
<P5,58>
<P4,59>
<C1,56>
<P6,60>
<C3,57>
<P4,61>
<P6,60>
<C3,58>
<C2,59>
<C6,60>
<P5,62>
<P4,63>
<P7,64>
<C1,61>
Total 65 items were produced.
Total 62 items were consumed.
cyll@cyl-VirtualBox:~/Desktop/os/pro3$
```

생산되고 소비되는 개수가 실행할 때마다 랜덤하게 변합니다.

또한 생산되고 소비되는 순서가 매우 유동적으로 변한다는 것도 알 수 있습니다.

중복소비, 중복생산 ERROR는 뜨지 않습니다.

```
Activities Terminal 09:41
cyl@cyl-VirtualBox: ~/Desktop/os/pro3

File Edit View Search Terminal Help

<P6,57>
<C3,57>
<P6,58>
<C3,58>
<P6,59>
<C2,59>
<P6,60>
<C2,60>
<P6,61>
<C2,61>
<P6,62>
<C3,62>
<P6,63>
<C2,63>
<P6,64>
<C2,64>
<P6,65>
<C3,65>
<P6,66>
<C2,66>
<P6,67>
<C2,67>
<P6,68>
<P7,69>
<P6,70>
<C2,70>
<C1,69>
<C3,70>
<P4,71>
<C3,71>
<P6,72>
<P7,73>
<C2,72>
<P4,74>
<P5,75>

... Total 76 items were produced.
... Total 73 items were consumed.
cyl@cyl-VirtualBox:~/Desktop/os/pro35
```

```
Activities Terminal 09:41
cyl@cyl-VirtualBox: ~/Desktop/os/pro3

File Edit View Search Terminal Help

<P6,277>
<P4,278>
<P6,279>
<C3,279>
<P4,280>
<C1,277>
<C8,278>
<C1,279>
<C8,280>
<P7,281>
<C3,281>
<P4,282>
<P6,283>
<P4,284>
<P6,285>
<C3,285>
<P7,286>
<C3,286>
<C8,284>
<C1,285>
<C8,286>
<P6,287>
<P4,288>
<P6,289>
<P4,290>
<C3,287>
<P7,291>
<C3,288>
<C2,289>
<P6,292>
<C1,290>
<C8,291>
<P4,293>
<P5,294>
<P7,295>

... Total 296 items were produced.
... Total 292 items were consumed.
cyl@cyl-VirtualBox:~/Desktop/os/pro35
```

```
Activities Terminal 09:41
cyl@cyl-VirtualBox: ~/Desktop/os/pro3

File Edit View Search Terminal Help

<P4,43>
<C8,43>
<P4,44>
<P4,45>
<C8,45>
<P4,46>
<C8,46>
<P4,47>
<C8,47>
<P4,48>
<C8,48>
<P4,49>
<C8,49>
<P4,50>
<C8,50>
<P4,51>
<C8,51>
<P4,52>
<C8,52>
<P4,53>
<C8,53>
<P4,54>
<C8,54>
<P4,55>
<C8,55>
<P4,56>
<C8,56>
<P5,57>
<P7,58>
<C2,57>
<C8,58>
<P4,59>
<C1,59>
<P6,60>

... Total 61 items were produced.
... Total 60 items were consumed.
cyl@cyl-VirtualBox:~/Desktop/os/pro35
```


<bounded_waiting.c>

[illegible][illegible]

The screenshot shows a Linux desktop with a dark theme. The top panel includes the 'Activities' button, a search bar, and a clock showing '16:47'. The terminal window is titled 'Terminal' and displays the command 'cyl@cyl-VirtualBox: ~/Desktop/os/pro3'. The program's output consists of a large block of 'A's, followed by a block of 'B's, and then a series of 'D's. The desktop background is a dark, textured image. The left sidebar contains icons for the Dash, Home, and Applications menus, along with various application icons like Firefox, LibreOffice, and the Dash icon.

실행할 때마다 스레드는 A~H까지 차례대로 서로 침범하지 않고 각자의 영역을 잘 지켜서 여러 번 구분되게 출력되는 것을 확인할 수 있습니다. (꽤 많은 수의 출력이 나옵니다.)

순서가 여러 번 출력되는 이유는 스핀락이 임계구역에 진입하기 위해 계속해서 루프를 돌면서 상태를 체크하지만, 락을 얻지 못한 여러 스레드는 계속해서 또 루프를 도는 상황이 생기기 때문에 출력이 중복되는 경우가 발생하기 때문입니다. 다시 말해, 락을 얻지 못한 스레드는 여전히 루프를 돌면서 waiting 배열의 값을 체크하다가 락을 획득하는 순간 임계구역에 진입하여 문자를 출력하고 나오게 되며 여러 스레드들이 중복해서 문자를 출력하게 되는 원리입니다.

4. 과제를 수행하면서 경험한 문제점과 느낀 점

이번 과제를 수행하면서 스핀락에 대한 개념을 이해하지 않을 수가 없었습니다. 또한 이전 과제에서 스레드를 활용하는 것은 많이 했었지만 critical section이라는 임계구역을 설정해 놓고 스레드가 중복해서 들어오지 못하게 한다는 문제는 이전에 미처 생각해보지 못했습니다. 왜냐하면 이전에는 스레드가 침범할 여지가 없었기 때문입니다. 다시 말해, 침범하더라도 문제가 발생할 수 있는 여지가 거의 없었습니다.

또한 과제를 수행하던 중 스핀락 구현에는 꼭 한가지 방법만 있는 것은 아니라는 것도 알 수 있었습니다. 다양한 방법으로 스핀락으로의 접근이 가능하다는 것도 같이 알 수 있었습니다.

동기화란 것이 어떻게 보면 프로그램이 잘 돌아갈 수 있도록 원활하게 독립적으로 critical section을 골고루 배분해 주는 것인데 이것이 되지 않으면 에러가 날 수 밖에 없습니다. 아울러, 스핀락은 context switch가 발생하지 않는다는 점에서 CPU 부하를 줄일 수 있다는 장점은 있지만 LOCK이 반환될 때까지 계속 확인하면서 루프를 돌고 있는 busy waiting으로 CPU를 쓸데없이 낭비한다는 단점이 있었습니다. 따라서 스핀락은 멀티 프로세서 시스템에서만 사용할 수 있습니다.

과제 수행 중, bounded_buffer.c에서는 임계 구역을 잘 설정하지 못해 계속 ERROR가 생기는 상황이 반복되었으며, consumer() 스레드 함수와 producer() 스레드 함수가 동시에 있어서 이 부분에서 스핀락을 어떻게 걸어줄지를 놓고 고민을 많이 할 수 밖에 없었습니다. 또한 임계구역을 어디까지로 설정해 줄 것인가의 문제에서 item 출력 개수의 차이가 있었습니다. 상호 배제를 한 상태에서 출력을 하느냐 아니면 상호 배제를 하지 않은 상태에서 출력을 하느냐의 차이인데, 상호 배제를 하지 않은 상태에서 출력하면 빠져나가는 스레드들이 생기기 때문에 시스템 상 당연한 결과

로 적은 수의 item이 출력됩니다. 이 문제는 main 함수 내에 usleep() 지연 시간을 조정해 주면 해결할 수 있습니다.

Bounded_waiting.c에서는 ppt에 있는 내용을 참고하여 별 어려움 없이, compare_and_swap() 함수를 atomic_compare_exchange_weak() 로 바꿔서 스핀락을 수행하게 하였습니다.

아울러 남은 과제 수행과 더불어 운영체제 수업에서 앞으로도 남은 기간 성실히 임하겠습니다. 감사합니다.