

Họ và tên: Đồng Quốc Thắng

Mã số sinh viên: 23521421

Lớp: IT007.P11.CTTN

HỆ ĐIỀU HÀNH BÁO CÁO LAB 5

CHECKLIST (Đánh dấu x khi hoàn thành)

Lưu ý mỗi câu phải làm đủ 3 yêu cầu

I. Bài tập thực hành

	BT 1	BT 2	BT 3
Trình bày cách làm	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chụp hình minh chứng	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Giải thích kết quả	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

II. Bài tập ôn tập

	1
Trình bày cách làm	<input type="checkbox"/>
Chụp hình minh chứng	<input type="checkbox"/>
Giải thích kết quả	<input type="checkbox"/>

Tự chấm điểm: 10

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*

<MSSV>_LABx.pdf

I. Bài tập thực hành

1.

Cách làm: sử dụng mutex để bảo vệ các biến sells và products để không bị cả hai thread truy cập cùng lúc. Sử dụng pthread_mutex_lock để lock lại và chạy phần critical section và pthread_mutex_unlock để unlock cho thread khác làm việc.

```
4 #include <semaphore.h>
5 #include <unistd.h>
6 #include <stdbool.h>
7
8 int products = 0;
9 int sells = 0;
10
11 sem_t sem;
12 pthread_mutex_t mutex;
13
14 void* producer(void* arg) {
15     while (true) {
16         sem_wait(&sem);
17         pthread_mutex_lock(&mutex);
18         if (sells < products) {
19             sells++;
20         }
21         pthread_mutex_unlock(&mutex);
22     }
23 }
24
25 void* consumer(void* arg) {
26     while (true) {
27         pthread_mutex_lock(&mutex);
28         if (products < sells + 1421) { //4 so cuoi MSSV la 1421
29             products++;
30         }
31         pthread_mutex_unlock(&mutex);
32         sem_post(&sem);
33     }
34 }
```

```
24
25 void* consumer(void* arg) {
26     while (true) {
27         pthread_mutex_lock(&mutex);
28         if (products < sells + 1421) { //4 so cuoi MSSV la 1421
29             products++;
30         }
31         pthread_mutex_unlock(&mutex);
32         sem_post(&sem);
33     }
34 }
35
36 int main() {
37     pthread_t producer_thread, consumer_thread;
38     pthread_mutex_init(&mutex, NULL);
39     sem_init(&sem, 0, 0);
40
41     pthread_create(&producer_thread, NULL, producer, NULL);
42     pthread_create(&consumer_thread, NULL, consumer, NULL);
43
44     for (int i = 0; i < 5; i++) {
45         printf("products is %d and sells is %d\n", products, sells);
46         sleep(0.5);
47     }
48     sem_destroy(&sem);
49     pthread_mutex_destroy(&mutex);
50
51     return 0;
52 }
```

```
(~/IT007/Lab5)
(20:44:40 on main *)-> ./producer_consumer_semaphore
products is 0 and sells is 0
products is 110 and sells is 104
products is 437 and sells is 436
products is 657 and sells is 657
products is 862 and sells is 851

(~/IT007/Lab5)
(20:44:54 on main *)-> ./producer_consumer_semaphore
products is 0 and sells is 0
products is 235 and sells is 165
products is 521 and sells is 520
products is 738 and sells is 737
products is 1069 and sells is 1016

(~/IT007/Lab5)
(20:44:56 on main *)-> ./producer_consumer_semaphore
products is 0 and sells is 0
products is 173 and sells is 171
products is 738 and sells is 533
products is 1481 and sells is 860
products is 2602 and sells is 1579

(~/IT007/Lab5)
(20:44:56 on main *)->
```

Giải thích kết quả: products lúc nào cũng lớn hơn sells và nhỏ hơn sells + 1421(4 số cuối mã số sinh viên). Hàm for trong main có sleep(0.5) nhưng thực sự khi pass argument không phải là số dương cho hàm sleep thì thời gian sleep sẽ là 0 nhưng vẫn sẽ tốn thời gian nên các thread sẽ chạy được tầm vài nghìn vòng lặp. Lý do không để chỉ 1 hàm sleep(1) là vì làm như vậy thì products và sells sẽ lên giá trị rất cao => tràn số.

2.

Cách làm: khởi tạo một biến semaphore có giá trị ban đầu bằng 0 và dùng cho cả producer và consumer. Trước khi vào critical section thì sẽ gọi semaphore wait, lúc đó chỉ có 1 thread được thực thi critical section vì sau khi gọi 1 lần wait thì giá trị của biến

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.

semaphore là 0(nếu gọi wait thì sẽ bị blocked), sau khi vào critical section thì gọi hàm post semaphore và để cho thread tiếp theo thực thi.

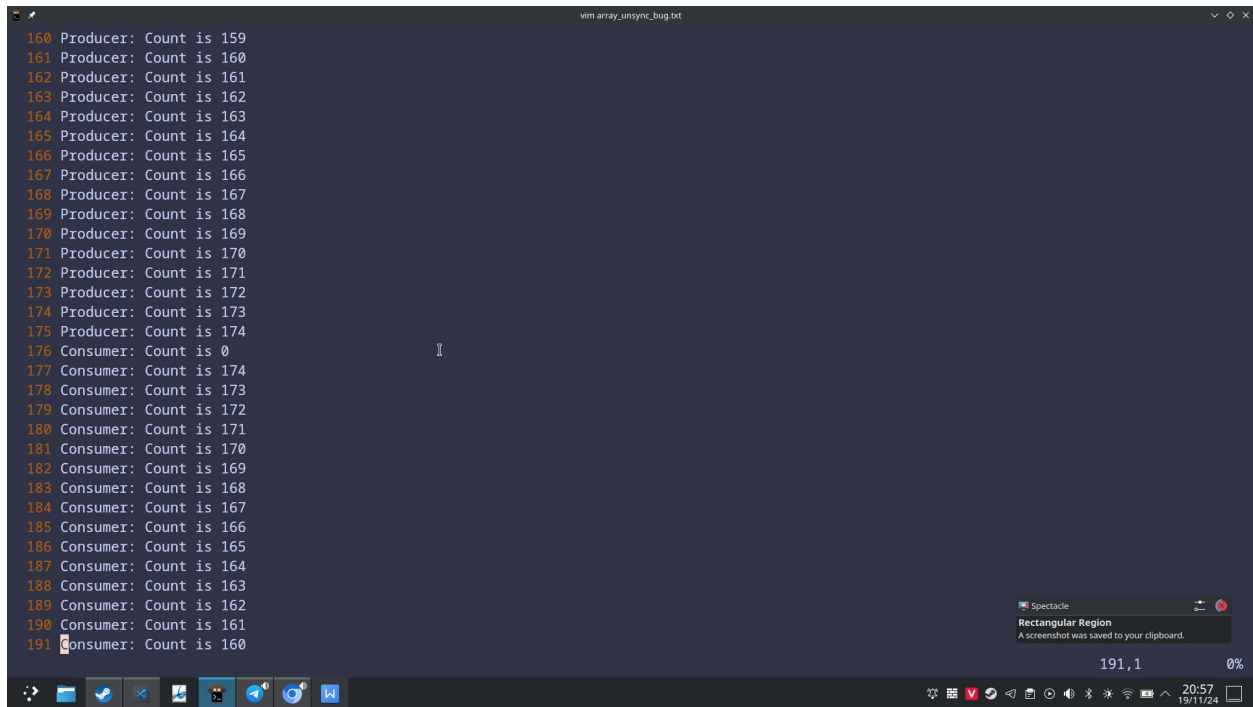
```
array_unsync.c  array_sync.c x
home > turtle > IT007 > Lab5 > array_sync.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6  #include <stdbool.h>
7
8  int arr[1421] = {0};
9  int count = 0;
10 sem_t sem;
11
12 void* producer(void* arg) {
13     srand(23521421);
14     int r;
15     while (true) {
16         sem_wait(&sem);
17         if (count < 1421) {
18             arr[count] = rand();
19             count++;
20         }
21         printf("Producer: Count is %d\n", count);
22         sem_post(&sem);
23     }
24 }
25
26 void* consumer(void* arg) {
27     while (true) {
28         sem_wait(&sem);
29         if (count == 0) {
30             printf("There's nothing in a\n");
31         }
32     }
33 }
```



```
23     }
24 }
25
26 void* consumer(void* arg) {
27     while (true) {
28         sem_wait(&sem);
29         if (count == 0) {
30             printf("There's nothing in a\n");
31         }
32         if (count > 0)
33             count--;
34         printf("Consumer: Count is %d\n", count);
35         sem_post(&sem);
36     }
37 }
38
39 int main() {
40     pthread_t producer_thread, consumer_thread;
41     sem_init(&sem, 0, 1);
42
43     pthread_create(&producer_thread, NULL, producer, NULL);
44     pthread_create(&consumer_thread, NULL, consumer, NULL);
45
46     for (int i = 0; i < 60; i++) {
47         sleep(0.5);
48     }
49     // sem_destroy(&sem);
50     return 0;
51 }
```

Kết quả nếu không có semaphore:

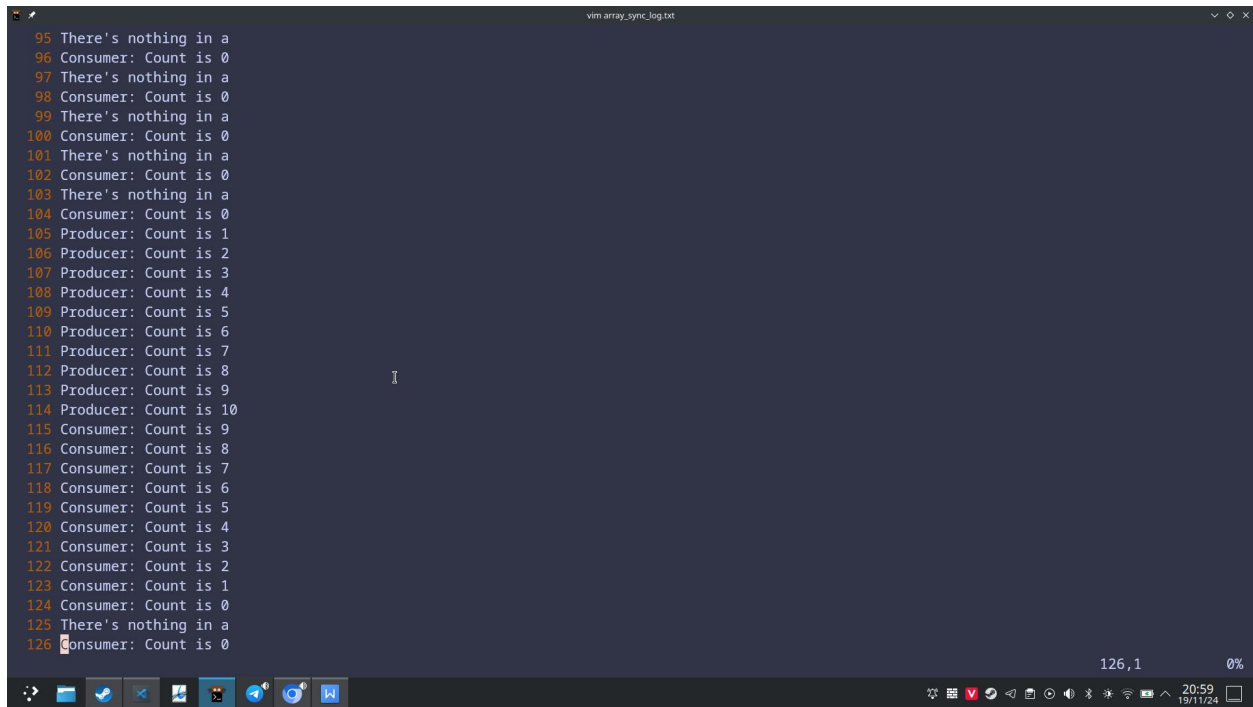
Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.



```
160 Producer: Count is 159
161 Producer: Count is 160
162 Producer: Count is 161
163 Producer: Count is 162
164 Producer: Count is 163
165 Producer: Count is 164
166 Producer: Count is 165
167 Producer: Count is 166
168 Producer: Count is 167
169 Producer: Count is 168
170 Producer: Count is 169
171 Producer: Count is 170
172 Producer: Count is 171
173 Producer: Count is 172
174 Producer: Count is 173
175 Producer: Count is 174
176 Consumer: Count is 0
177 Consumer: Count is 174
178 Consumer: Count is 173
179 Consumer: Count is 172
180 Consumer: Count is 171
181 Consumer: Count is 170
182 Consumer: Count is 169
183 Consumer: Count is 168
184 Consumer: Count is 167
185 Consumer: Count is 166
186 Consumer: Count is 165
187 Consumer: Count is 164
188 Consumer: Count is 163
189 Consumer: Count is 162
190 Consumer: Count is 161
191 Consumer: Count is 160
```

Ở dòng 176, count là 174 ở thread này nhưng lại đọc ra là 0 ở thread sau, sau đó mới đọc được là 174(vẫn sai vì đúng ra phải là 173 vì consume xong rồi mới in ra count). Việc này gây ra do 2 thread cùng tranh nhau đọc và ghi cùng 1 biến count => có race condition.

Kết quả khi có semaphore:

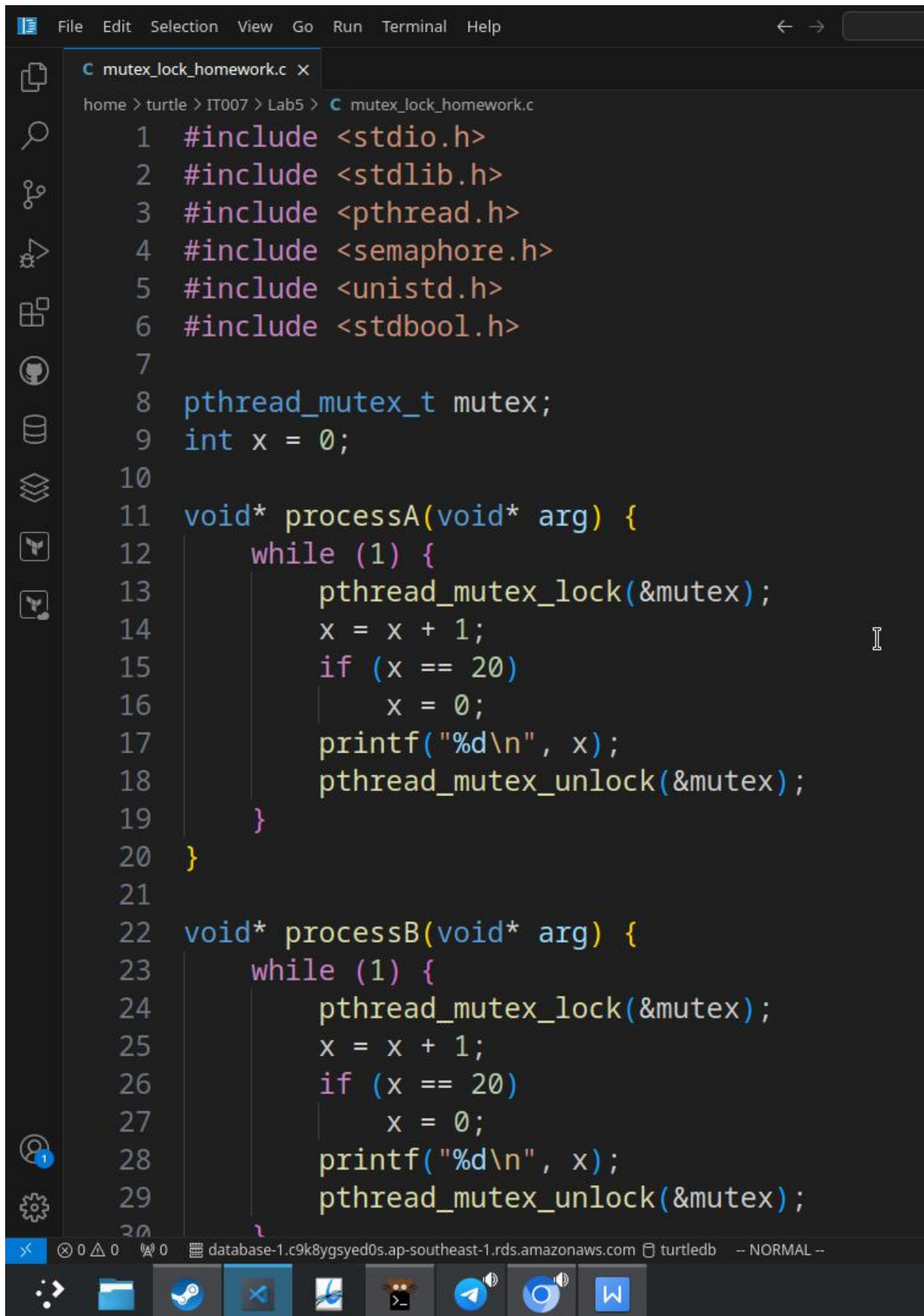


```
95 There's nothing in a
96 Consumer: Count is 0
97 There's nothing in a
98 Consumer: Count is 0
99 There's nothing in a
100 Consumer: Count is 0
101 There's nothing in a
102 Consumer: Count is 0
103 There's nothing in a
104 Consumer: Count is 0
105 Producer: Count is 1
106 Producer: Count is 2
107 Producer: Count is 3
108 Producer: Count is 4
109 Producer: Count is 5
110 Producer: Count is 6
111 Producer: Count is 7
112 Producer: Count is 8
113 Producer: Count is 9
114 Producer: Count is 10
115 Consumer: Count is 9
116 Consumer: Count is 8
117 Consumer: Count is 7
118 Consumer: Count is 6
119 Consumer: Count is 5
120 Consumer: Count is 4
121 Consumer: Count is 3
122 Consumer: Count is 2
123 Consumer: Count is 1
124 Consumer: Count is 0
125 There's nothing in a
126 Consumer: Count is 0
```

Kết quả ra đúng với mong đợi, 2 thread chờ nhau để thực hiện critical section nên biến count tăng và giảm, được đọc ghi đúng theo thứ tự.

3.

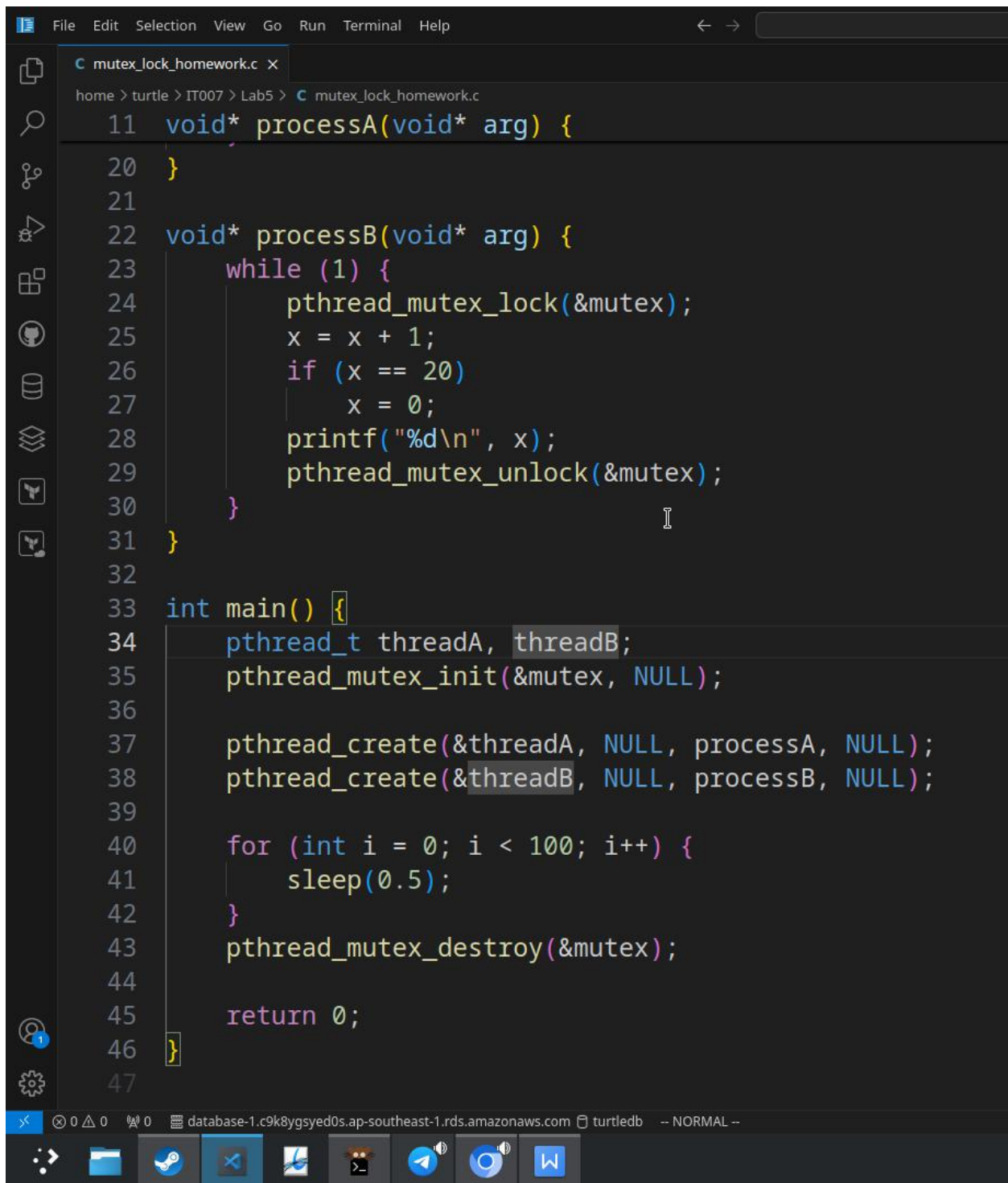
Cách làm: sử dụng mutex để bảo vệ biến x để không bị cả hai thread truy cập cùng lúc. Sử dụng `pthread_mutex_lock` để lock lại và chạy phần critical section và `pthread_mutex_unlock` để unlock.



The image shows a code editor window with a dark theme. The title bar at the top reads "File Edit Selection View Go Run Terminal Help". The editor is open to a file named "mutex_lock_homework.c". The file path in the breadcrumb is "home > turtle > IT007 > Lab5 > mutex_lock_homework.c". The code is as follows:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  #include <unistd.h>
6  #include <stdbool.h>
7
8  pthread_mutex_t mutex;
9  int x = 0;
10
11 void* processA(void* arg) {
12     while (1) {
13         pthread_mutex_lock(&mutex);
14         x = x + 1;
15         if (x == 20)
16             x = 0;
17         printf("%d\n", x);
18         pthread_mutex_unlock(&mutex);
19     }
20 }
21
22 void* processB(void* arg) {
23     while (1) {
24         pthread_mutex_lock(&mutex);
25         x = x + 1;
26         if (x == 20)
27             x = 0;
28         printf("%d\n", x);
29         pthread_mutex_unlock(&mutex);
30     }
```

The editor has a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, Extensions, Docker, Containers, Remote Explorer, and Settings. The status bar at the bottom shows "database-1.c9k8ygsyed0s.ap-southeast-1.rds.amazonaws.com", "turtledb", and "-- NORMAL --". The Windows taskbar is visible at the very bottom with icons for File Explorer, Steam, VS Code, a web browser, a terminal, a chat application, and a music player.

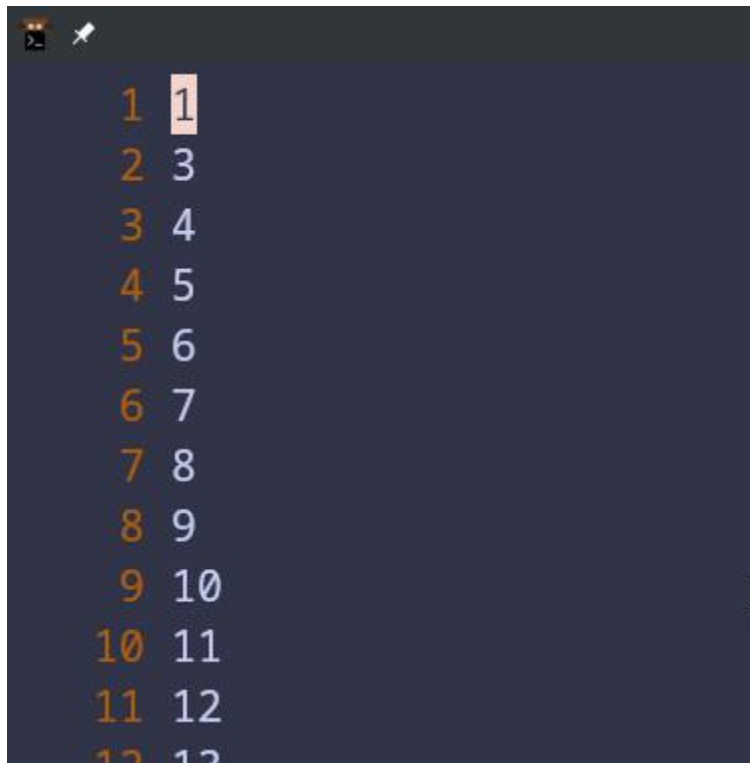


The screenshot shows a code editor with a dark theme. The file name is 'mutex_lock_homework.c'. The code is as follows:

```
11 void* processA(void* arg) {
20 }
21
22 void* processB(void* arg) {
23     while (1) {
24         pthread_mutex_lock(&mutex);
25         x = x + 1;
26         if (x == 20)
27             x = 0;
28         printf("%d\n", x);
29         pthread_mutex_unlock(&mutex);
30     }
31 }
32
33 int main() {
34     pthread_t threadA, threadB;
35     pthread_mutex_init(&mutex, NULL);
36
37     pthread_create(&threadA, NULL, processA, NULL);
38     pthread_create(&threadB, NULL, processB, NULL);
39
40     for (int i = 0; i < 100; i++) {
41         sleep(0.5);
42     }
43     pthread_mutex_destroy(&mutex);
44
45     return 0;
46 }
47
```

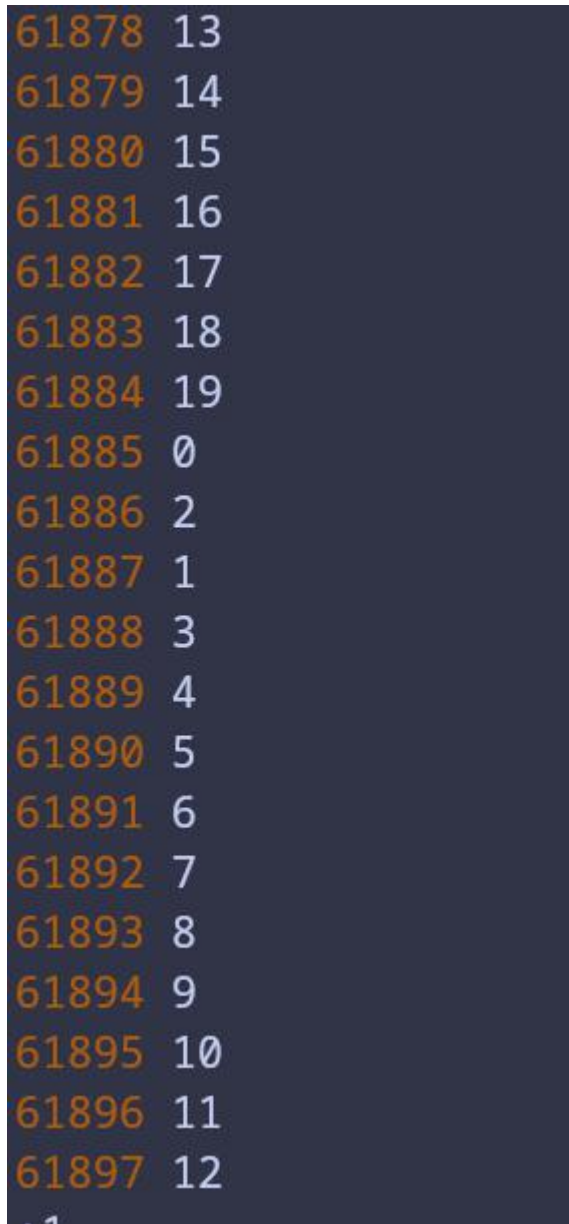
The editor interface includes a menu bar (File, Edit, Selection, View, Go, Run, Terminal, Help), a sidebar with icons for Explorer, Search, Run and Debug, Source Control, and Extensions, and a status bar at the bottom showing connection details and a terminal window.

Kết quả khi không có mutex lock:



```
1 1
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
10 11
11 12
12 13
```

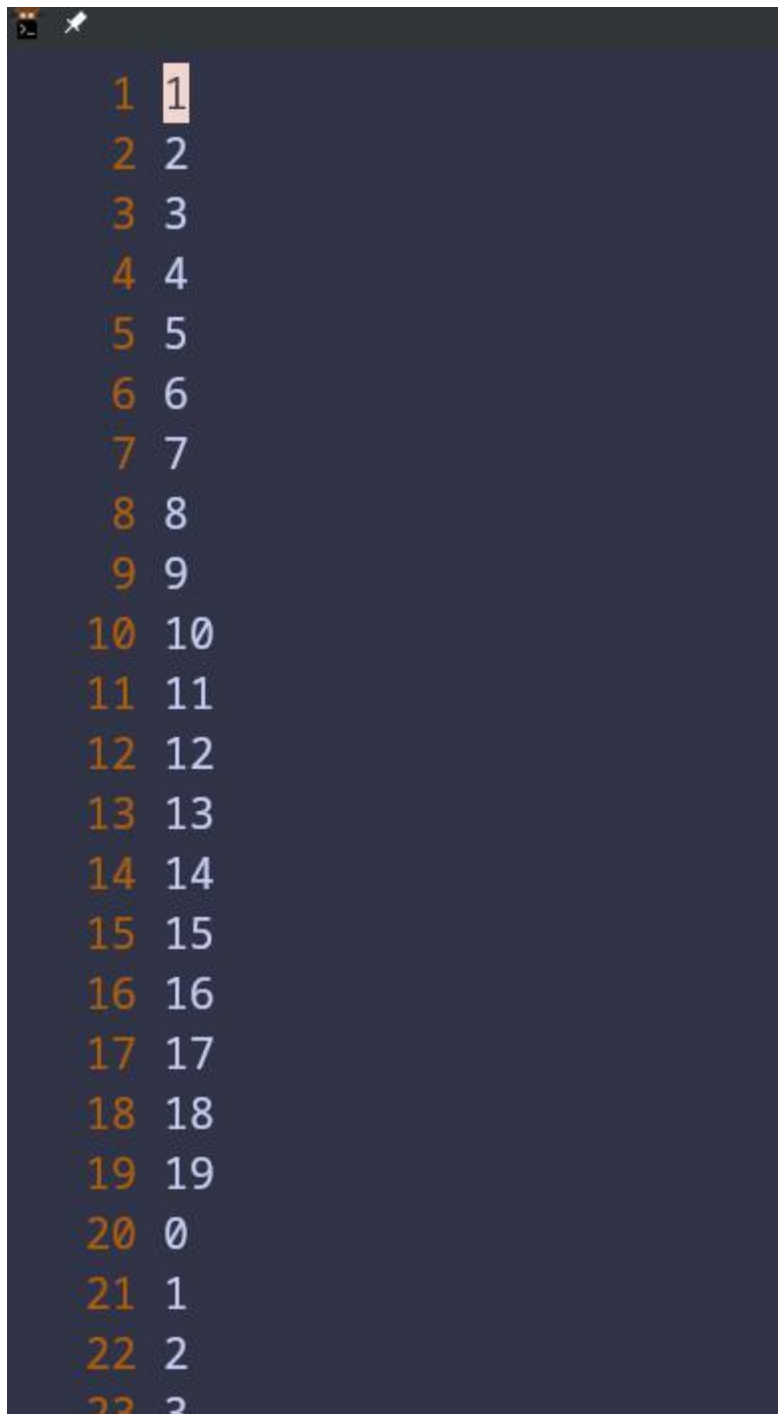
Từ 1 tăng lên thành 3 mà không qua 2.



61878	13
61879	14
61880	15
61881	16
61882	17
61883	18
61884	19
61885	0
61886	2
61887	1
61888	3
61889	4
61890	5
61891	6
61892	7
61893	8
61894	9
61895	10
61896	11
61897	12

hoặc là có trường hợp từ 0 rồi in ra 2 rồi mới in ra 1

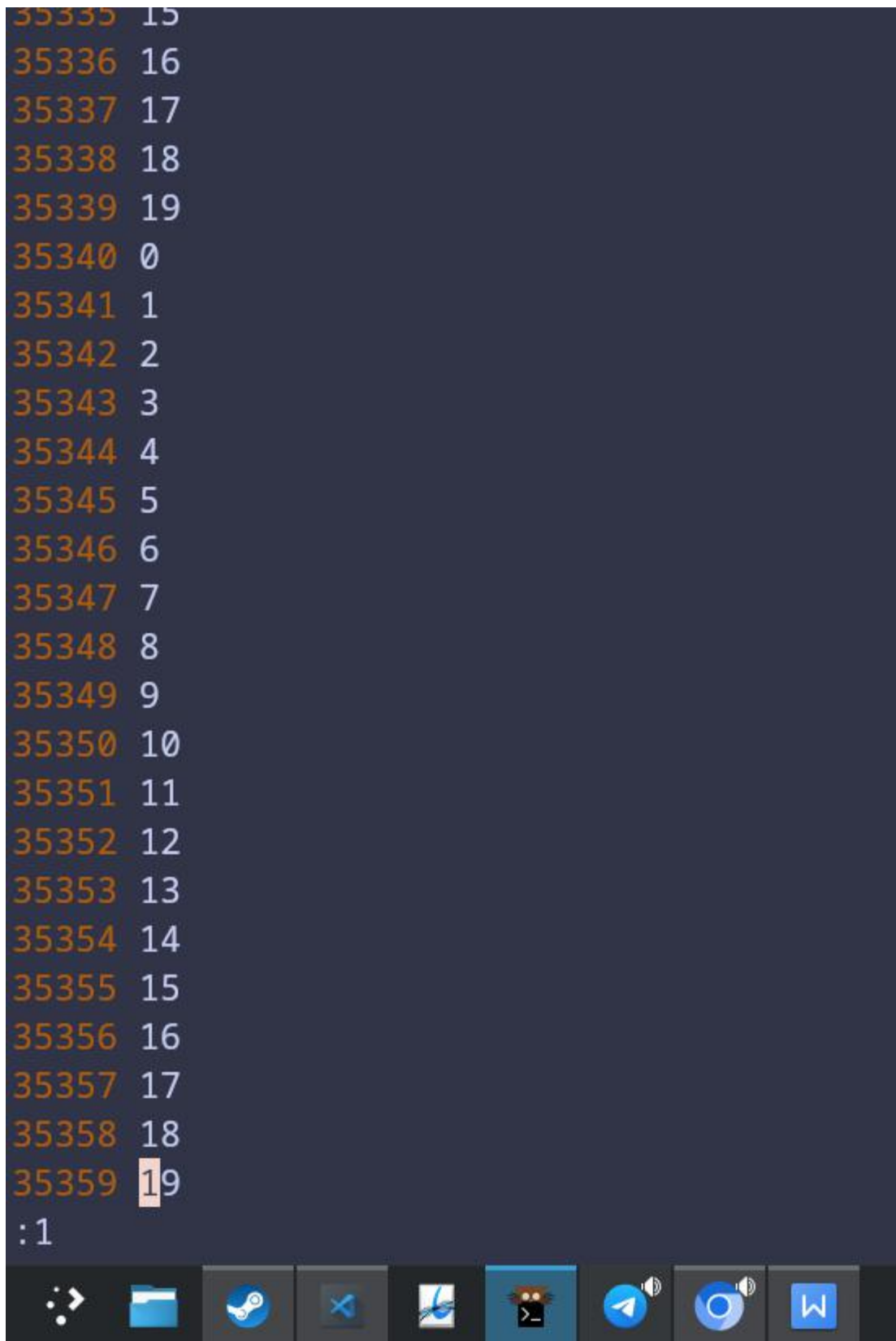
Kết quả khi có mutex lock:



A screenshot of a terminal window with a dark blue background. It displays a list of numbers from 1 to 23. The numbers 1 through 19 are in orange, while 20 through 23 are in white. The first '1' is highlighted with a light pink selection box. The numbers are arranged in two columns: the first column contains numbers 1-19, and the second column contains numbers 1-23. The first row shows '1' in the first column and '1' in the second column.

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	0
21	1
22	2
23	3

```
35335 15
35336 16
35337 17
35338 18
35339 19
35340 0
35341 1
35342 2
35343 3
35344 4
35345 5
35346 6
35347 7
35348 8
35349 9
35350 10
35351 11
35352 12
35353 13
35354 14
35355 15
35356 16
35357 17
35358 18
35359 19
:1
```



Giải thích: Khi không có mutex lock, cả hai thread đều có thể truy cập cùng lúc vào x, có lúc sẽ cùng lúc viết vào x, vì vậy khi đọc biến x lên sẽ có sự nhập nhằng và không được sync. Khi có mutex thì chỉ có 1 thread được truy cập x vào 1 thời điểm => sẽ không có 2 thread vừa đọc viết vào 1 biến cùng 1 lúc => sẽ không có bug.

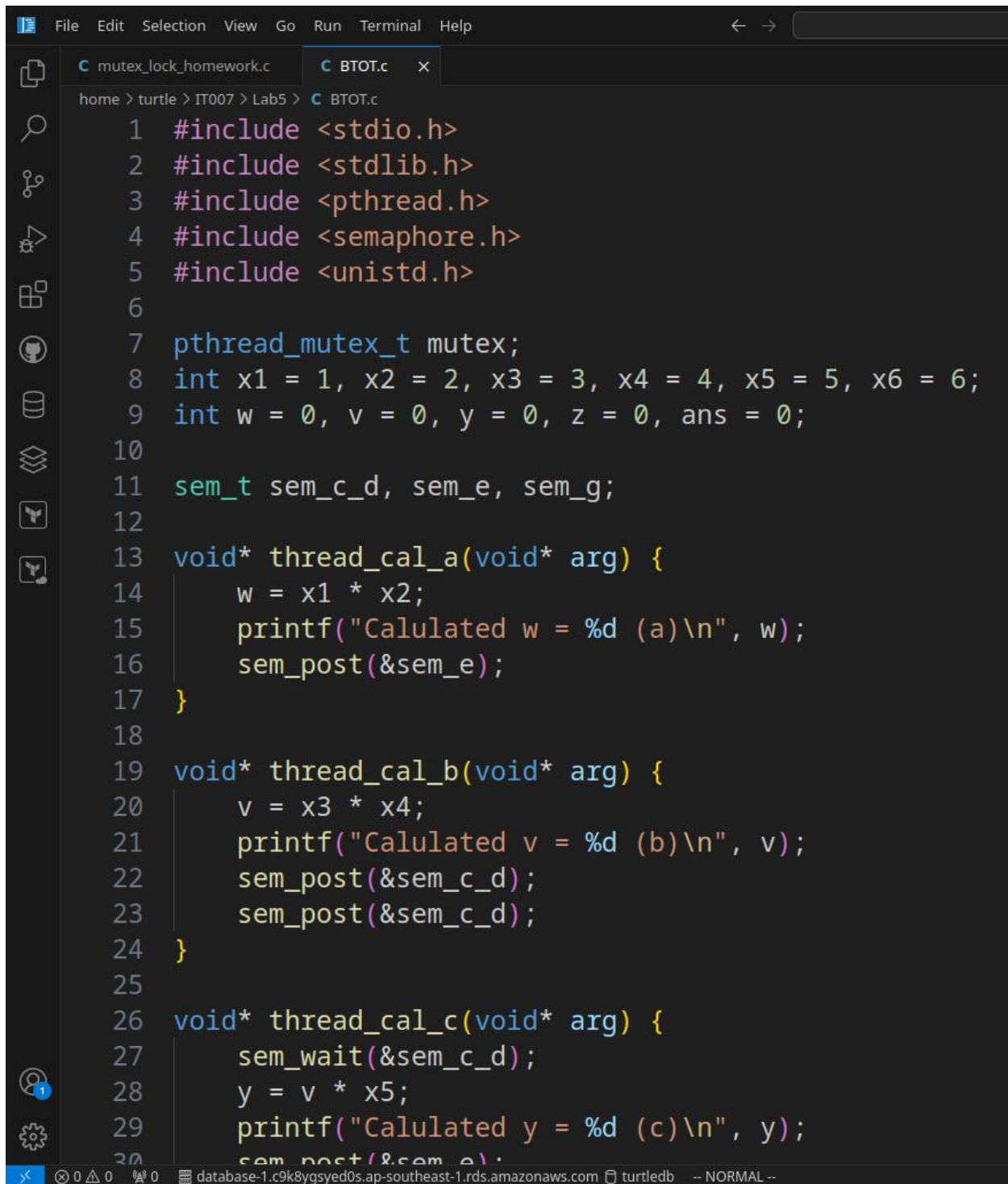
II. Bài tập ôn tập

1.

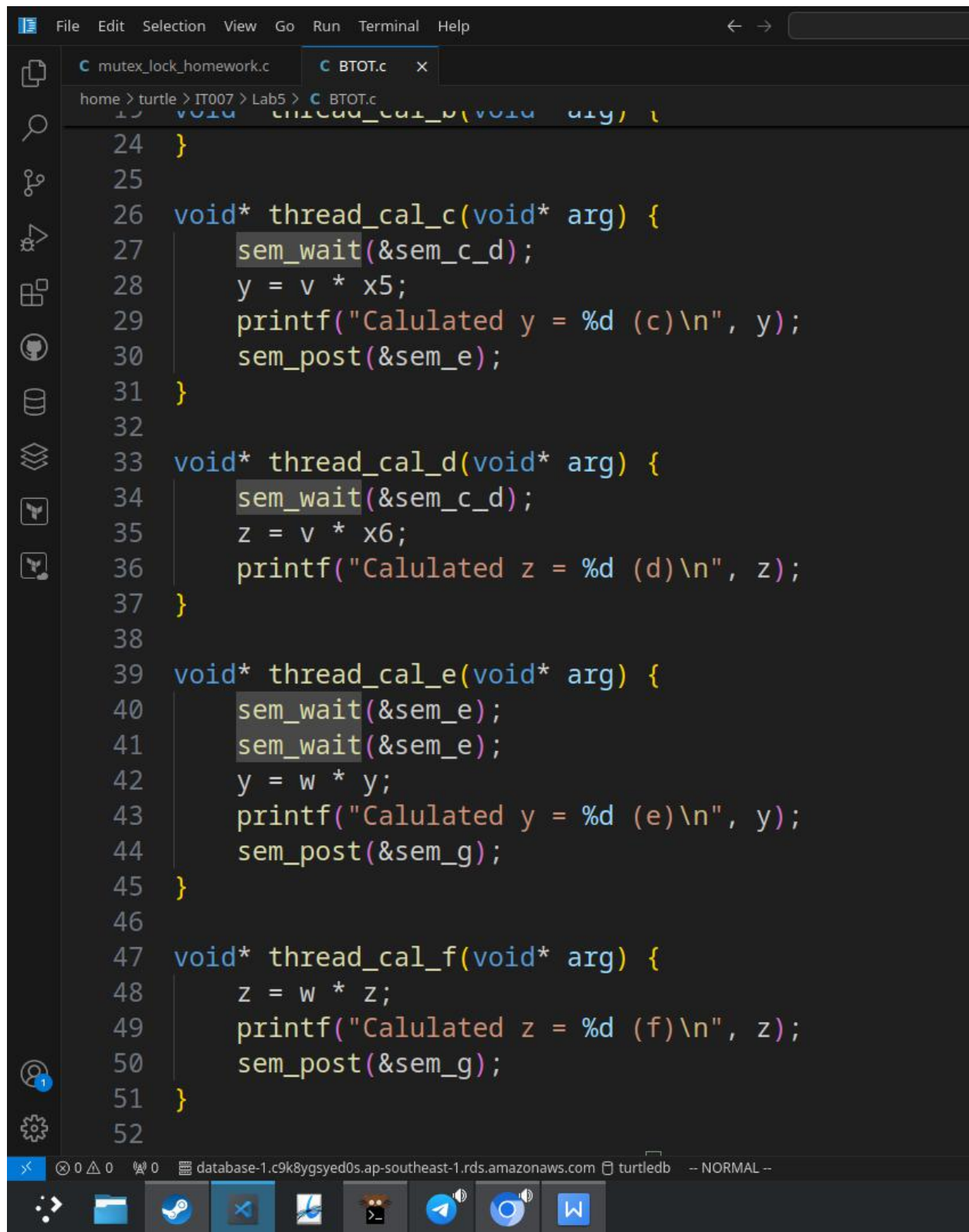
Cách làm: ta sẽ khởi tạo 3 biến semaphore để giúp cho tiến trình thực thi có thứ tự. Ta thấy (c), (d) chỉ thực hiện sau khi có v tức là sau khi (b) đã thực hiện nên ta khởi tạo sem_c_d bằng 0 và gọi wait trong thread của (c) và (d) trước khi chạy phần tính toán. Ta sẽ đặt 2 lần hàm sem_post ở cuối (b) => sau khi (b) được thực hiện xong thì (c) với (d) mới được thực hiện. Vì (c) và (d) chỉ đọc v sau khi giá trị của v đã ổn định nên (c) và (d) có thể thực thi đồng thời.

Ta thấy (e) chỉ thực thi khi w và y được tính có nghĩa là (e) chỉ nên được thực thi sau (a) và (c) đã được thực thi. Nên ta khởi tạo biến semaphore này là 0 và gọi 2 lần hàm wait ở (e), để hàm sem_post ở cuối (a) và (c) để e chỉ được thực thi sau khi (a) và (c) được thực

thi. Cách xử lý (g) giống như cách xử lý (e).



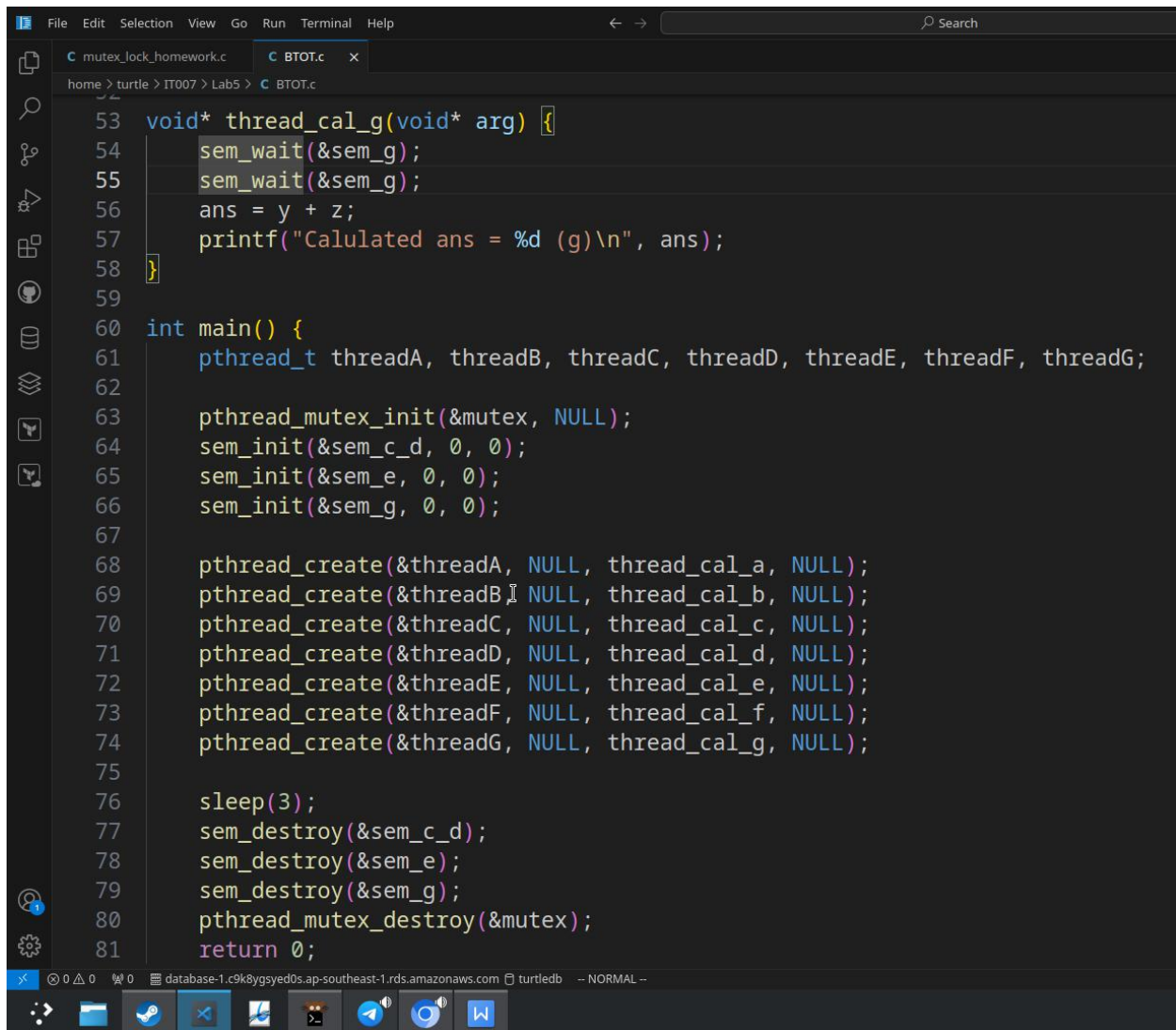
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5 #include <unistd.h>
6
7 pthread_mutex_t mutex;
8 int x1 = 1, x2 = 2, x3 = 3, x4 = 4, x5 = 5, x6 = 6;
9 int w = 0, v = 0, y = 0, z = 0, ans = 0;
10
11 sem_t sem_c_d, sem_e, sem_g;
12
13 void* thread_cal_a(void* arg) {
14     w = x1 * x2;
15     printf("Calulated w = %d (a)\n", w);
16     sem_post(&sem_e);
17 }
18
19 void* thread_cal_b(void* arg) {
20     v = x3 * x4;
21     printf("Calulated v = %d (b)\n", v);
22     sem_post(&sem_c_d);
23     sem_post(&sem_c_d);
24 }
25
26 void* thread_cal_c(void* arg) {
27     sem_wait(&sem_c_d);
28     y = v * x5;
29     printf("Calulated y = %d (c)\n", y);
30     sem_post(&sem_e);
31 }
```



The image shows a screenshot of a code editor with a dark theme. The editor has a menu bar at the top with 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. Below the menu bar, there are two tabs: 'mutex_lock_homework.c' and 'BTOT.c'. The active tab is 'BTOT.c', which shows the following C code:

```
24 }
25
26 void* thread_cal_c(void* arg) {
27     sem_wait(&sem_c_d);
28     y = v * x5;
29     printf("Calulated y = %d (c)\n", y);
30     sem_post(&sem_e);
31 }
32
33 void* thread_cal_d(void* arg) {
34     sem_wait(&sem_c_d);
35     z = v * x6;
36     printf("Calulated z = %d (d)\n", z);
37 }
38
39 void* thread_cal_e(void* arg) {
40     sem_wait(&sem_e);
41     sem_wait(&sem_e);
42     y = w * y;
43     printf("Calulated y = %d (e)\n", y);
44     sem_post(&sem_g);
45 }
46
47 void* thread_cal_f(void* arg) {
48     z = w * z;
49     printf("Calulated z = %d (f)\n", z);
50     sem_post(&sem_g);
51 }
52
```

At the bottom of the editor, there is a status bar showing 'database-1.c9k8ygsyed0s.ap-southeast-1.rds.amazonaws.com', 'turtledb', and '-- NORMAL --'. The Windows taskbar is visible at the very bottom of the image.



```
53 void* thread_cal_g(void* arg) {
54     sem_wait(&sem_g);
55     sem_wait(&sem_g);
56     ans = y + z;
57     printf("Calulated ans = %d (g)\n", ans);
58 }
59
60 int main() {
61     pthread_t threadA, threadB, threadC, threadD, threadE, threadF, threadG;
62
63     pthread_mutex_init(&mutex, NULL);
64     sem_init(&sem_c_d, 0, 0);
65     sem_init(&sem_e, 0, 0);
66     sem_init(&sem_g, 0, 0);
67
68     pthread_create(&threadA, NULL, thread_cal_a, NULL);
69     pthread_create(&threadB, NULL, thread_cal_b, NULL);
70     pthread_create(&threadC, NULL, thread_cal_c, NULL);
71     pthread_create(&threadD, NULL, thread_cal_d, NULL);
72     pthread_create(&threadE, NULL, thread_cal_e, NULL);
73     pthread_create(&threadF, NULL, thread_cal_f, NULL);
74     pthread_create(&threadG, NULL, thread_cal_g, NULL);
75
76     sleep(3);
77     sem_destroy(&sem_c_d);
78     sem_destroy(&sem_e);
79     sem_destroy(&sem_g);
80     pthread_mutex_destroy(&mutex);
81     return 0;
```

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.

Kết quả:


```
❗ ✎  
[~/IT007/Lab5]—  
[21:03:23 on main *]—> vim mutex_log_bug.txt  
[~/IT007/Lab5]—  
[21:05:24 on main *]—> vim mutex_log.txt  
[~/IT007/Lab5]—  
[21:18:11 on main *]—> ./BTOT  
Calulated w = 2 (a)  
Calulated v = 12 (b)  
Calulated y = 60 (c)  
Calulated z = 72 (d)  
Calulated z = 0 (f)  
Calulated y = 120 (e)  
Calulated ans = 192 (g)  
[~/IT007/Lab5]—  
[21:18:17 on main *]—> ./BTOT  
Calulated v = 12 (b)  
Calulated w = 2 (a)  
Calulated z = 72 (d)  
Calulated z = 144 (f)  
Calulated y = 60 (c)  
Calulated y = 120 (e)  
Calulated ans = 264 (g)  
[~/IT007/Lab5]—  
[21:18:20 on main *]—> ./BTOT  
Calulated w = 2 (a)  
Calulated v = 12 (b)  
Calulated y = 60 (c)  
Calulated z = 72 (d)  
Calulated y = 120 (e)  
Calulated z = 144 (f)  
Calulated ans = 264 (g)  
[~/IT007/Lab5]—  
[21:18:24 on main *]—>
```


Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thế Tùng.

Ta thấy thứ tự thực thi chi tiết có thể khác nhau, nhưng (c), (d) luôn tính sau (b), (e) luôn tính sau (a), (c) và (g) luôn tính sau (e) và (f). Thứ tự thực thi chi tiết khác nhau vì các thread được thực thi đồng thời, ngoài những thread mình quy định thứ tự thực thi thì các thread được thực thi song song nên thread nào được thực thi xong trước sẽ không có quy luật.

Code và log đã được lưu trong các file và up lên github.

Link github: <https://github.com/LowTechTurtle/IT007/tree/main/Lab5>