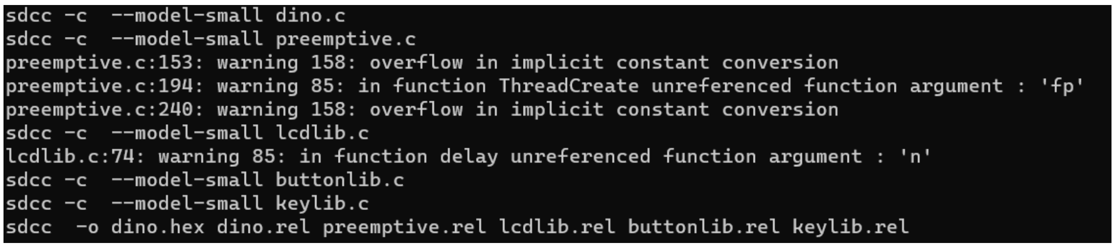
110062118
CS25 Zi Yun, Lai

Operating System

Programming Project Checkpoint 5

DINOGAME

1. Typescript



```
sdcc -c  --model-small dino.c
sdcc -c  --model-small preemptive.c
preemptive.c:153: warning 158: overflow in implicit constant conversion
preemptive.c:194: warning 85: in function ThreadCreate unreferenced function argument : 'fp'
preemptive.c:240: warning 158: overflow in implicit constant conversion
sdcc -c  --model-small lcdlib.c
lcdlib.c:74: warning 85: in function delay unreferenced function argument : 'n'
sdcc -c  --model-small buttonlib.c
sdcc -c  --model-small keylib.c
sdcc  -o dino.hex dino.rel preemptive.rel lcdlib.rel buttonlib.rel keylib.rel
```

2. Explanation

a. Initialization

Location	Type	Name	Purpose
0x20	unsigned char	lcd_ready	
0x21	unsigned char	state	To identify game state. There are 3 states INIT,GAME,OVER
0x22	unsigned char	key_full	Whether there is any unhandled key operation.
0x23	unsigned char	dinosaur_row	To store dinosaur's position.
0x24	unsigned char	local_dinosaur_row	For render to quickly grab the game information to avoid occupying critical section for too long.
0x25	Unsigned int	map[2]	To store map info. It uses bit to record the cactus' position. 1 means cactus and 0 means empty space.
0x26			
0x27			
0x28			
0x29	Unsigned int	localmap[2]	For render to quickly grab the game information to avoid occupying critical section for too long.
0x2A			
0x2B			
0x2C			

0x2D	char	game_empty	Whether the gamectrl can update new game info.
0x2E	char	counter	For adjusting the shift speed.
0x2F	signed char	i	For for loop
0x30	char	savedSP	For switching to another thread.
0x31			
0x32			
0x33			
0x34	char	thread_Bitmap	For switching to another thread.
0x35	char	cur_Thread	For switching to another thread.
0x36	char	new_Thread	For switching to another thread.
0x37	char	tmp_SP	For switching to another thread.
0x38	char	score	For recording the score.
0x39	char	level	For recording the level.
0x3A	char	buffer	For key information exchange between threads.
0x3B	char	Number	Input keypad value.
0x3C	char	key_empty	Whether the keypad ctrl can update new key operation.
0x3D	char	game_full	Whether there is unhandled key operation.
0x3E	char	mutex	For protecting critical section.
0x3F	char	lastkey	To detect whether the keypad is clean.

b. Structure

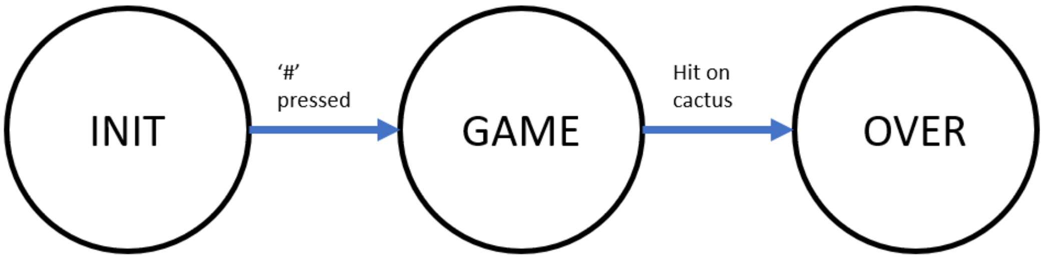
There are total 3 threads keypad_ctrl, game_ctrl, render_task. Main will spawn keypad_ctrl, game_ctrl first and run render_task.

name	job	comsume	Produce
keypad_ctrl	Monitor key status and store it to the buffer.		Key
game_ctrl	Take in the key value in buffer. Update the total game state.	key	State, dinosaur_row, map,
render_task	Print image on LCD based on the information produced	State, dinosaur_row, map,	

	by game		

*Highlighted variable may have racing condition since its shared by two threads.

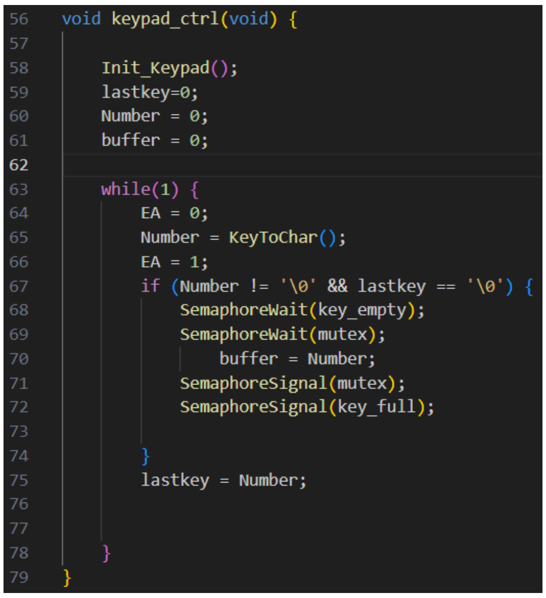
c. State transition



d. Implementation

In the section, I will introduce the implementation of three threads respectively.

● keypad_ctrl

Code	Explanation
	**Line 58-61:** It's for initialization. **Line 64-66:** Read in key. Based on the experience, I add EA operation to make it functional. **Line 67:** I use lastkey to store the key on previous timestep, and use it to check whether the key pressed after all key had been released. It will definitely capture every key event since human cannot double click in such a short time between two keypad_ctrl turns. **Line 68-72:** Key_empty and key_full is adopted to prevent any key from replaced before it had got handled. Mutex is used to protect critical section, since key value is shared by two threads.

```
56  void keypad_ctrl(void) {
57  
58      Init_Keypad();
59      lastkey=0;
60      Number = 0;
61      buffer = 0;
62  
63      while(1) {
64          EA = 0;
65          Number = KeyToChar();
66          EA = 1;
67          if (Number != '\0' && lastkey == '\0') {
68              SemaphoreWait(key_empty);
69              SemaphoreWait(mutex);
70                  buffer = Number;
71              SemaphoreSignal(mutex);
72              SemaphoreSignal(key_full);
73  
74          }
75          lastkey = Number;
76  
77  
78      }
79  }
```

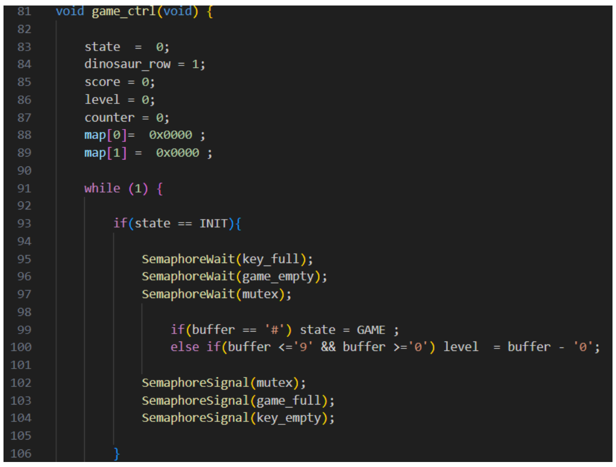
● game_ctrl

game_ctrl operates differently in different state. In the section, I will introduce its behavior under different state respectively.

110062118
CS25 Zi Yun, Lai

INIT

Code



```
81  void game_ctrl(void) {
82  
83      state  =  0;
84      dinosaur_row = 1;
85      score = 0;
86      level = 0;
87      counter = 0;
88      map[0]=  0x0000 ;
89      map[1] =  0x0000 ;
90  
91      while (1) {
92  
93          if(state == INIT){
94  
95              SemaphoreWait(key_full);
96              SemaphoreWait(game_empty);
97              SemaphoreWait(mutex);
98  
99                  if(buffer == '#') state = GAME ;
100                 else if(buffer <='9' && buffer >='0') level  = buffer - '0';
101 
102             SemaphoreSignal(mutex);
103             SemaphoreSignal(game_full);
104             SemaphoreSignal(key_empty);
105 
106         }
```

Explanation

Line 83-89:

It's for initialization.

Line 95-104:

First, It waits for new input key information.

If it existed, then game_ctrl waits for the latest game info to be handled by render.

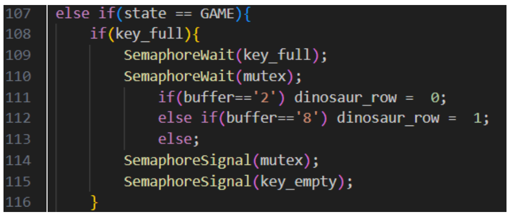
Finally, it waits for mutex to be empty.

If '#' is pressed, it will switch game state.

If '0'-'9' is pressed, it will update level.

GAME

Code



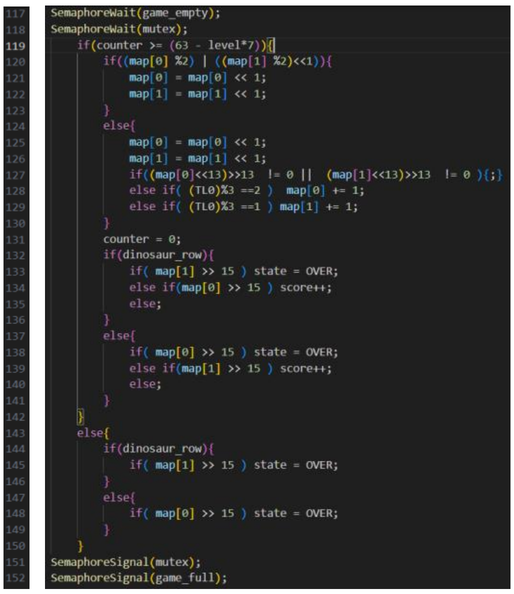
```
107 else if(state == GAME){
108     if(key_full){
109         SemaphoreWait(key_full);
110         SemaphoreWait(mutex);
111             if(buffer=='2') dinosaur_row =  0;
112             else if(buffer=='8') dinosaur_row =  1;
113             else;
114         SemaphoreSignal(mutex);
115         SemaphoreSignal(key_empty);
116     }
```

Explanation

Line 108-115:

If there is unhandled key operation, adjust the dinosaur's position based on the input. If no, just keep going, since the game will still be updated even without key input(cactus shift)

Code



```
117 SemaphoreWait(game_empty);
118 SemaphoreWait(mutex);
119     if(counter >= (63 - level*7)){
120         if((map[0] %2) | ((map[1] %2)<<1)){
121             map[0] = map[0] << 1;
122             map[1] = map[1] << 1;
123         }
124         else{
125             map[0] = map[0] << 1;
126             map[1] = map[1] << 1;
127             if((map[0]<<13)>>13  != 0 ||  (map[1]<<13)>>13  != 0 ){;}
128             else if( (TL0)%3 ==2 )  map[0] += 1;
129             else if( (TL0)%3 ==1 ) map[1] += 1;
130         }
131         counter = 0;
132         if(dinosaur_row){
133             if( map[1] >> 15 ) state = OVER;
134             else if(map[0] >> 15 ) score++;
135             else;
136         }
137         else{
138             if( map[0] >> 15 ) state = OVER;
139             else if(map[1] >> 15 ) score++;
140             else;
141         }
142     }
143     else{
144         if(dinosaur_row){
145             if( map[1] >> 15 ) state = OVER;
146         }
147         else{
148             if( map[0] >> 15 ) state = OVER;
149         }
150     }
151 SemaphoreSignal(mutex);
152 SemaphoreSignal(game_full);
```

Explanation

Line 117-118:

Game empty is used to make sure the latest game state had already been taken.

Line 119:

To adjust the shifting speed according to level. The cactus will shift only if the condition is satisfied. The counter will be incremented each time of timer interrupt.

Map generated policy:

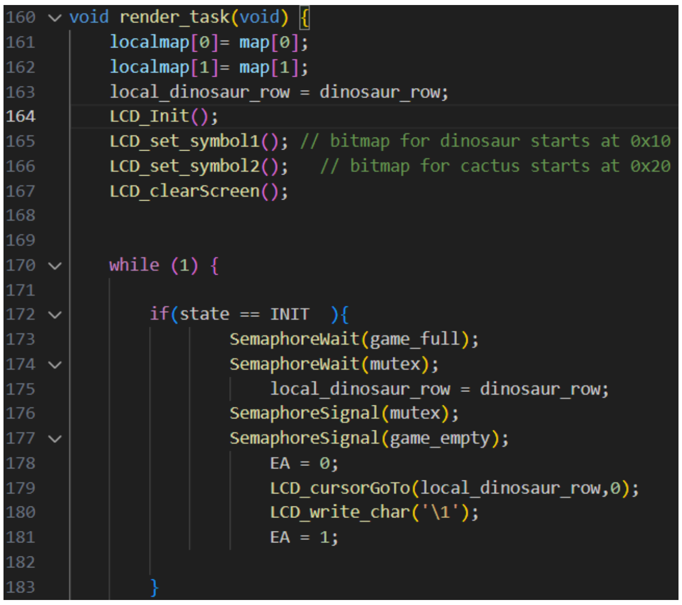
If there is cactus on the leftmost column, the next column will be empty. If the leftmost column is empty, it will keep generating empty column until there are enough successive empty columns. And then, I use TL0%3 to randomly generate column with cactus on top or at the bottom.

Score updating:

	Whether the score will be incremented will only be evaluated when the map is updated. It will be operated after the map and dinosaur's position are updated. **Check death:** If dinosaur's position and cactus is overlapping after an update, go to OVER state.
Over state: Nothing need to be done in this state.	

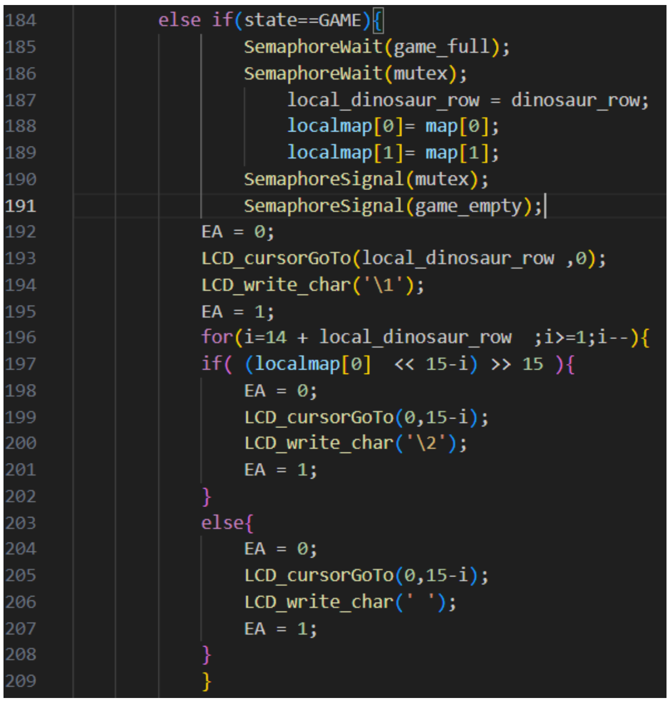
- **Render_task**

Render_task operates differently in different state. In the section, I will introduce its behavior under different state respectively. For each LCD related operation, I use EA0/1 to envelope them, since it would cause error if I took off that. In addition, I modified the LCDInit so the LCD will not shift.

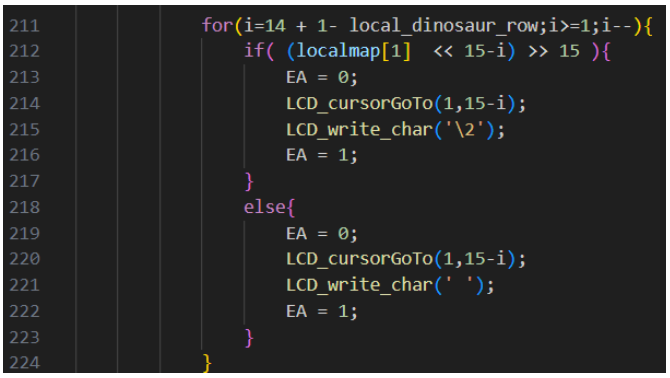
INIT	
Code	Explanation
	**Line 161-167:** It's for initialization. **Set_symbol:** Since the offered code has some bug, I wrote my own version to set the customized note. **TASK:** Print the dinosaur at the default location. The mutex part is not necessary.
GAME	
Code	Explanation

```
160  void render_task(void) {
161      localmap[0]= map[0];
162      localmap[1]= map[1];
163      local_dinosaur_row = dinosaur_row;
164      LCD_Init();
165      LCD_set_symbol1(); // bitmap for dinosaur starts at 0x10
166      LCD_set_symbol2();   // bitmap for cactus starts at 0x20
167      LCD_clearScreen();
168
169
170      while (1) {
171
172          if(state == INIT  ){
173                  SemaphoreWait(game_full);
174                  SemaphoreWait(mutex);
175                      local_dinosaur_row = dinosaur_row;
176                  SemaphoreSignal(mutex);
177                  SemaphoreSignal(game_empty);
178                      EA = 0;
179                      LCD_cursorGoTo(local_dinosaur_row,0);
180                      LCD_write_char('\1');
181                      EA = 1;
182
183          }
```

110062118
CS25 Zi Yun, Lai



```
184         else if(state==GAME){
185             SemaphoreWait(game_full);
186             SemaphoreWait(mutex);
187                 local_dinosaur_row = dinosaur_row;
188                 localmap[0]= map[0];
189                 localmap[1]= map[1];
190             SemaphoreSignal(mutex);
191             SemaphoreSignal(game_empty);
192             EA = 0;
193             LCD_cursorGoTo(local_dinosaur_row ,0);
194             LCD_write_char('\1');
195             EA = 1;
196             for(i=14 + local_dinosaur_row  ;i>=1;i--){
197             if( (localmap[0]  << 15-i) >> 15 ){
198                 EA = 0;
199                 LCD_cursorGoTo(0,15-i);
200                 LCD_write_char('\2');
201                 EA = 1;
202             }
203             else{
204                 EA = 0;
205                 LCD_cursorGoTo(0,15-i);
206                 LCD_write_char(' ');
207                 EA = 1;
208             }
209             }
```



```
211             for(i=14 + 1- local_dinosaur_row;i>=1;i--){
212                 if( (localmap[1]  << 15-i) >> 15 ){
213                     EA = 0;
214                     LCD_cursorGoTo(1,15-i);
215                     LCD_write_char('\2');
216                     EA = 1;
217                 }
218                 else{
219                     EA = 0;
220                     LCD_cursorGoTo(1,15-i);
221                     LCD_write_char(' ');
222                     EA = 1;
223                 }
224             }
```

Line 185-191:

Use local variable to grab the game information. Since I/O takes time, it's not a good idea to hold critical section so long.

Line 192-195:

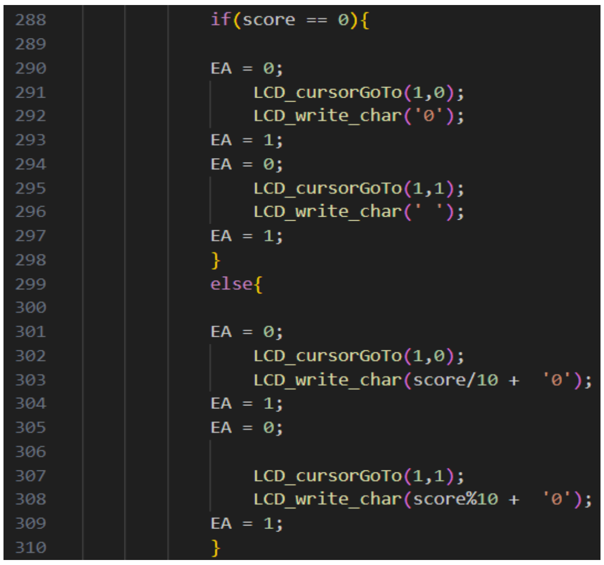
First column would be handled as a special case, since it has to print dinosaur first. It's not applicable to print dinosaur after the map had been printed, which would cause blinking image.

Line 199-224:

Print the map.

OVER

Code	Explanation



```
288             if(score == 0){
289 
290             EA = 0;
291                 LCD_cursorGoTo(1,0);
292                 LCD_write_char('0');
293             EA = 1;
294             EA = 0;
295                 LCD_cursorGoTo(1,1);
296                 LCD_write_char(' ');
297             EA = 1;
298             }
299             else{
300 
301             EA = 0;
302                 LCD_cursorGoTo(1,0);
303                 LCD_write_char(score/10 +  '0');
304             EA = 1;
305             EA = 0;
306 
307                 LCD_cursorGoTo(1,1);
308                 LCD_write_char(score%10 +  '0');
309             EA = 1;
310             }
```

Nothing special.

It print "gameover" and score.

Line 288-298:

Deal with 0 score situation.

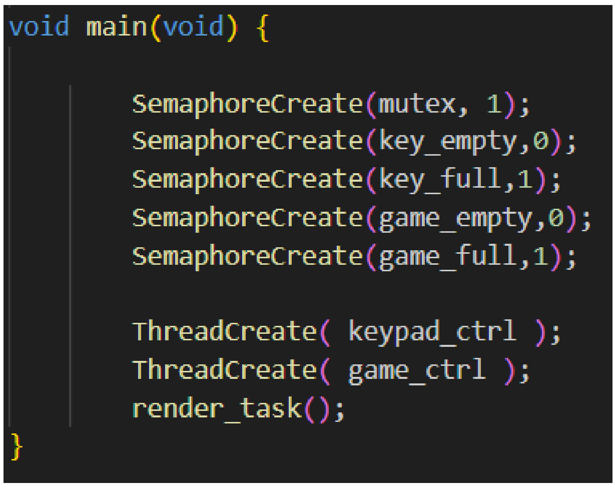
Line 299-309:

Transfer hex score to decimal by /and %.

110062118
CS25 Zi Yun, Lai

● main

The semaphores are set in this part. Mutex is 1, since no thread is active. Key_full and game_empty is 1, because it let the render work, even though there isn't any new game state update.



```
void main(void) {

        SemaphoreCreate(mutex, 1);
        SemaphoreCreate(key_empty,0);
        SemaphoreCreate(key_full,1);
        SemaphoreCreate(game_empty,0);
        SemaphoreCreate(game_full,1);

        ThreadCreate( keypad_ctrl );
        ThreadCreate( game_ctrl );
        render_task();
}
```

110062118
CS25 Zi Yun, Lai

CHECKPOINT5 PART1

I modified the dino game to complete this part. Since the original one had some bugs, but amazingly repaired when I constructed the dino game.

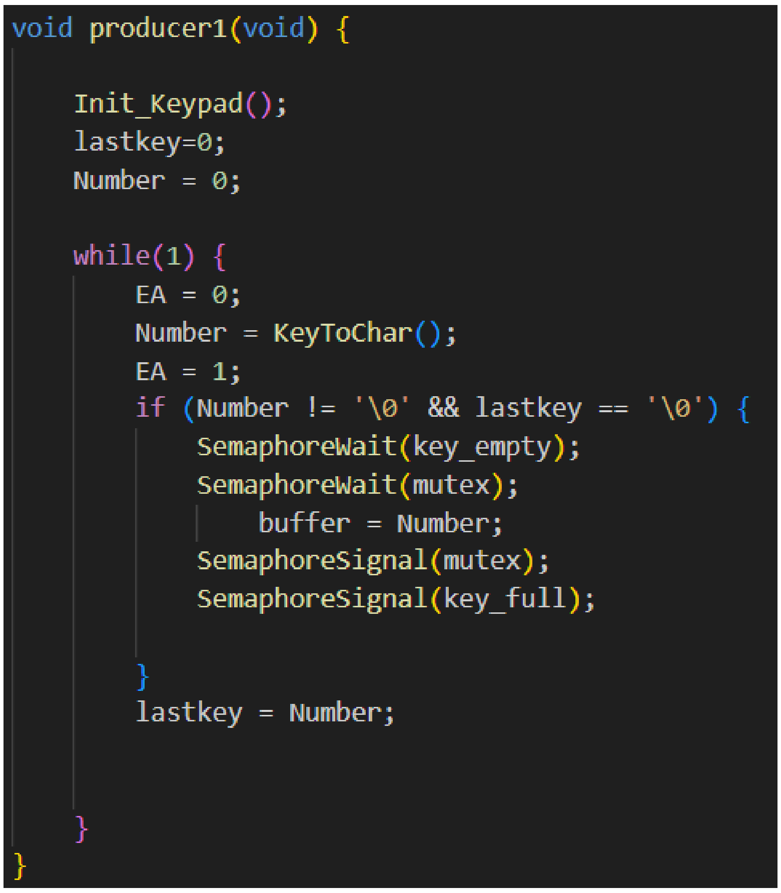
It contains three main threads: producer1, producer2, rendertask

Producer1

Producer1 is responsible for tracking keypad event. In the initialization part, it called Init_Keypad(). I use EA to envelope it, because it is not a good idea the operation be blocked in this condition, since the signal might be missing. I use lastkey to check whether key get pressed after all keypad have been cleaned. It can also handle the situation that the change happened when producer1 is not active.

I use Number to keep the information. Thus, even though it has to wait for the semaphore, the information will not disappear.

Buffer is for interthread data sharing.

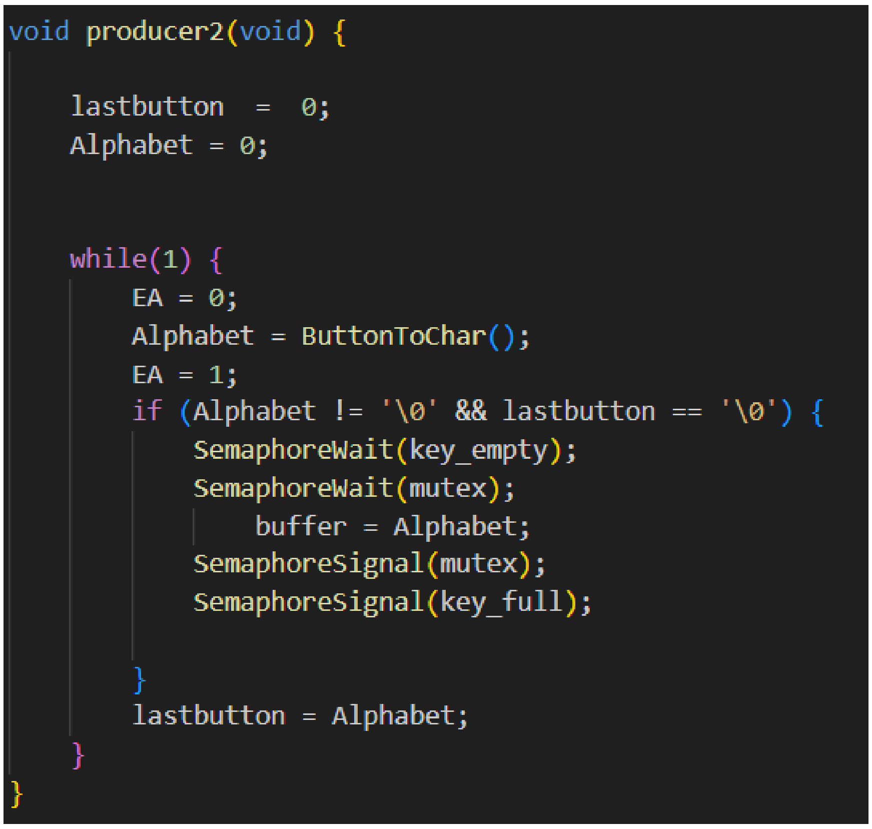


```
void producer1(void) {

    Init_Keypad();
    lastkey=0;
    Number = 0;

    while(1) {
        EA = 0;
        Number = KeyToChar();
        EA = 1;
        if (Number != '\0' && lastkey == '\0') {
            SemaphoreWait(key_empty);
            SemaphoreWait(mutex);
                buffer = Number;
            SemaphoreSignal(mutex);
            SemaphoreSignal(key_full);

        }
        lastkey = Number;


    }
}
```

110062118
CS25 Zi Yun, Lai

Producer2

It is basically same as Producer1. The only different is that button doesn't need to be initialized.

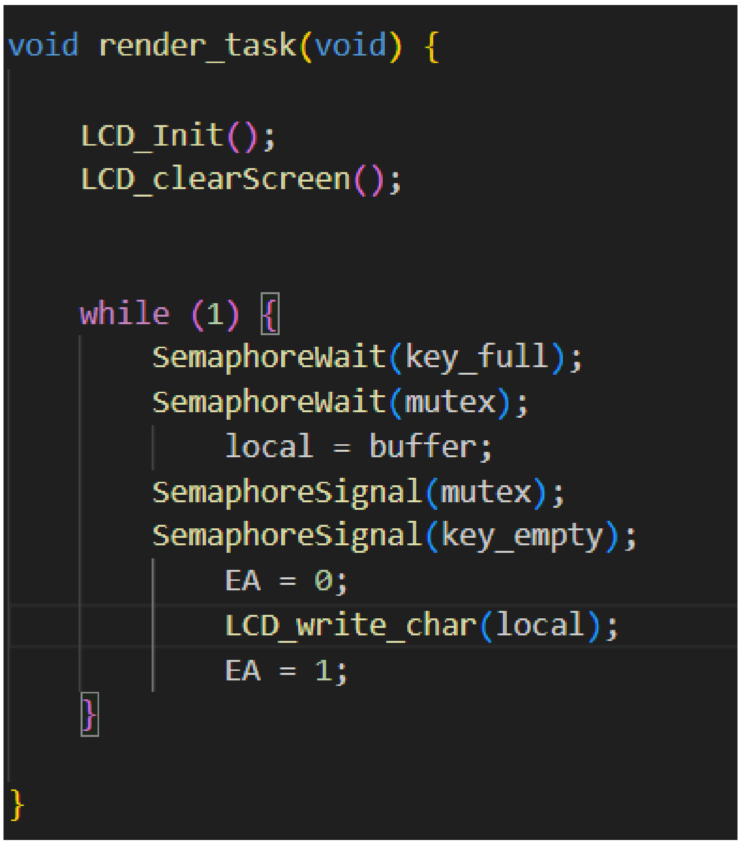


```
void producer2(void) {

    lastbutton  =  0;
    Alphabet = 0;


    while(1) {
        EA = 0;
        Alphabet = ButtonToChar();
        EA = 1;
        if (Alphabet != '\0' && lastbutton == '\0') {
            SemaphoreWait(key_empty);
            SemaphoreWait(mutex);
                buffer = Alphabet;
            SemaphoreSignal(mutex);
            SemaphoreSignal(key_full);

        }
        lastbutton = Alphabet;
    }
}
```

110062118
CS25 Zi Yun, Lai

render_task

render_task is responsible for printing numbers on LCD. I use LCD_init and clearscreen to initialize it. It will use local buffer to load the information and print it. An EA pair is applied since it is doing I/O. I didn't use LCD_ready, since it makes no different after I took it off.

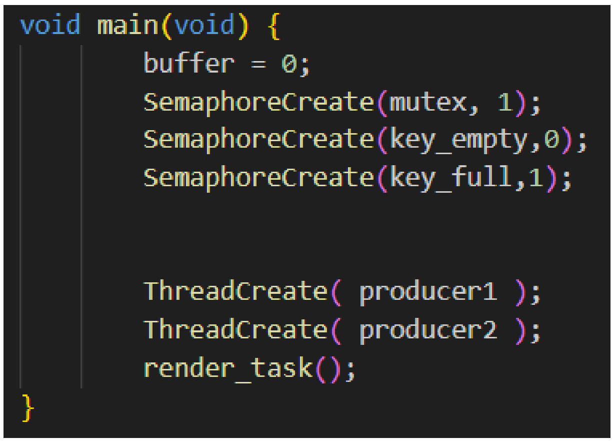


```
void render_task(void) {

    LCD_Init();
    LCD_clearScreen();


    while (1) {
        SemaphoreWait(key_full);
        SemaphoreWait(mutex);
            local = buffer;
        SemaphoreSignal(mutex);
        SemaphoreSignal(key_empty);
            EA = 0;
            LCD_write_char(local);
            EA = 1;
    }

}
```

110062118
CS25 Zi Yun, Lai

Main

It set the buffer and spawn two thread.



```
void main(void) {
        buffer = 0;
        SemaphoreCreate(mutex, 1);
        SemaphoreCreate(key_empty,0);
        SemaphoreCreate(key_full,1);


        ThreadCreate( producer1 );
        ThreadCreate( producer2 );
        render_task();
}
```