МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Отчет по игре

Студент: Волков И.К.
Группа: ККСО-01-18

Москва — 2019

СОДЕРЖАНИЕ

1. ОПИСАНИЕ ИГРЫ

Консольная игра написанная на **C** с использованием **ncurses**. Суть игры пролететь как можно дальше, не сталкиваясь с астероидами. Но сложность состоит в том, что за рулем космолета сидит самоуверенный в себе водитель. Чем дольше он летит, тем больше набирает скорость, не рассчитывая, что может столкнуться с астероидом. Для управления используются стрелки, для выхода из игры нужно нажать q.

Целью игры было:

1. Научиться разбивать однопоточную программу на многопоточную.(+)
2. Столкнуться с критическими секциями и научиться их обрабатывать.(+)
3. Научитсья создавать точку отката, чтобы в случае неверной работы системы откатиться в рабочее состояние.(-)
4. Создать образ диска на котором запускается игра.(+)

Игра соистоит из четырёх потоков. Один поток отвечал за отрисовку астероидов, Второй за отрисовку звезд, третий поток отвечал за прорисовку игрока в функции main(), ну а четвёртый отвечал за установку уровня сложности.

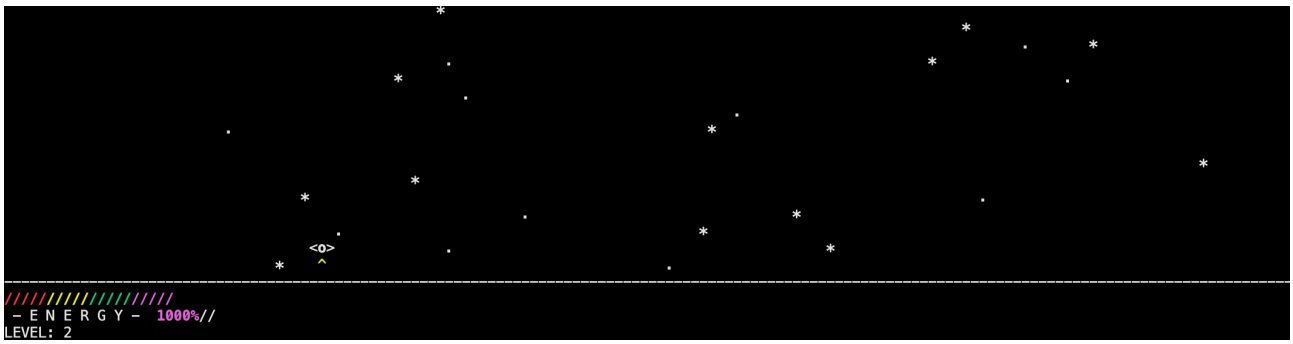

Figure 1.1: Space game

2. КОМПИЛЯЦИЯ ИГРЫ

На линуксе: gcc game.c list.c -lncurses -pthread -o game

На макосе: gcc game.c list.c -lncurses -o game

В list.c находятся функции для работы со списком, поскольку вся отрисовка заднего фона построена на списках.(Приложение №2)

3. ПОТОКИ

Как уже говорилось выше, игра состоит из четырёх потоков. Немножечко подробнее о каждом потоке.

Листинг 1: Создание потока: 1-Отрисовки звезд; 2-Отрисовки астероидов; 3-Установки уровня

```
1    pthread_create(&pid1, NULL, stars_update, NULL);
2    pthread_create(&pid2, NULL, asteroids_update, NULL);
3    pthread_create(&pid3, NULL, set_level, NULL);
```

Поток с функцией *stars_update()* создает список с координатами звезды и отрисовывает каждую звезду, пока игра не завершится. Координаты генерируются псевдослучайным способом при помощи функции *rand()*.

Поток с функцией *asteroids_update()* работает аналогично функции stars_update(), только отрисоывывает астероиды.

Поток с функцией *set_level()* работает независимо от всех потоков, просто отсчитывает 10 секунд и увеличивает уровень сложности.

Поток в *main()* ждет нажатия клавиши, чтобы перерисовать корабль.

Все потоки завершаются как только игрок проигрывает или нажимает клавишу выхода.

Figure 3.2: End of game

4. КРИТИЧЕСКИЕ СЕКЦИИ И ИХ ОБРАБОТКА

Критические секции возникали при обращении к общим данным(глобальные переменные) несколькими потоками и при отрисовке экрана. Для устранения непредвиденного поведения использовались алгритм взаимосключения и мьютексы.

4.1. Алгоритм Деккера

Чтобы согласовать поведение потока *stars_update()* и потока *asteroids_update()*, использовался алгоритм Деккера. Один поток ждет в бесконечном цикле пока другой работает в критической секции. То есть перед тем как зайти в критическую секцию поток проверяет, а не работает ли там другой поток, и если там занято, он будет в бексонечном цикле спрашивать: ”Ты закончил?”(Напомнило мне Осла из Шрека, когда он спрашивал приехали они или нет). И как только он получит положительный ответ, занимает кртическую секцию и теперь уже его будут спрашивать закончил он или нет.

Листинг 2: Алгоритм Деккера

```
1  /* DEKKER ALGORITHM */
2      thread2wantstoenter = true;
3      while (thread1wantstoenter == true)
4      {
5        if (favouredthread == 1)
6        {
7          usleep(10000 - (level * 1000)); // 10 ms
8          thread2wantstoenter = false;
9          while (favouredthread == 1) ;
10         thread2wantstoenter = true;
11       }
12     }
13 /* DEKKER ALGORITHM */
14 /* CRITICAL SECTION */
15 ...
16 /* CRITICAL SECTION */
17 favouredthread = 1;
18 thread2wantstoenter = false;
```

4.2. mutex

Для обработки кртических секций при отрисовке игрока, использовался *mutex*, который блокировался функцией *pthread_mutex_lock()* при отрисовке в одном из трех потоков, и разблокировался по завершению отрисовки функцией *pthread_mutex_unlock()*. То есть, если поток пытался заблокировать уже заблокированный mutex, он блокоировался до тех пор, пока mutex не становился доступным.

В потоках *stars_update()* и *asteroids_update()* mutex блокировался при заходе в критическую секцию, а в потоке *main()* перед тем как начать отрисовку корабля.

Не думаю, что мой подход является оптимальным, но хотелось попробовать различные метобы обработки.

Листинг 3: Использование *mutex* в main

```
1     /* CRITICAL SECTION */
2     pthread_mutex_lock(&mutex);
3     /* draw < > and engine */
4     wattron(game_wnd, A_ALTCHARSET);
5     mvwaddch(game_wnd, player_y, player_x - 1, ACS_LARROW);
6     mvwaddch(game_wnd, player_y, player_x + 1, ACS_RARROW);
7     wattron(game_wnd, COLOR_PAIR((tick % 10 / 3 % 2) ? 1 : 2));
8     mvwaddch(game_wnd, player_y + 1, player_x, ACS_UARROW);
9     wattroff(game_wnd, COLOR_PAIR((tick % 10 / 3 % 2) ? 1 : 2));
10    wattroff(game_wnd, A_ALTCHARSET);
11    /* draw o */
12    wattron(game_wnd, A_BOLD);
13    mvwaddch(game_wnd, player_y, player_x, PLAYER);
14    wattroff(game_wnd, A_BOLD);
15    wrefresh(game_wnd);
16    /* CRITICAL SECTION */
```

5. СБОРКА ОБРАЗА

1. С сайта kernel.org скачан исходник ядра и разархивирован.
2. Настройка сборки ядра через *make menuconfig*.
3. Сборка ядра с использованием *make bzImage*.
4. С сайта debian.org был взят initrd.gz и разархивирован.
5. После распаковки initrd.gz, в папку *bin* был перемещен исполняемый файл(*game*), который будет стартовать при запуске образа.
6. Так же с помощью команды *ldd* были узнаны библиотеки, необходимые для работы игры. Недостающие библиотеки были скопированы с */bin/lib* (системы) в */lib* (initrd).
7. После этих изменений все было обратно заархивировано в *initrd.gz*.
8. Был написан файл isolinux.cfg

```
1 default mini
2
3 label mini
4   kernel boot/bzImage
5   append initrd=boot/initrd.gz init=/bin/game quiet
```

9. После всех вышеперечисленных пунктов был собран game.iso-образ с помощью утилиты *mkisofs*.

С помощью программы *qemu* можно запустить образ game.iso. Команда запуска: qemu-system-x86_64 -cdrom ../Downloads/game.iso -m 1024. После параметра -m лучше указывать больше, потому что игра получилась требовательная, видимо(ну или я криво написал игру).

6. ПРИЛОЖЕНИЕ №1

Листинг 4: Исходный код игры

```
1  #include "game.h"
2
3  WINDOW*   game_wnd;
4  WINDOW*   info_wnd;
5  int       game_over = false;
6  int       exit_request = false;
7  int       player_y;
8  int       player_x;
9  const char *gv = "GAME OVER";
10 const char *ex = "Press any key to exit...";
11 const char *lvl = "You have reached level %d";
12 int       energy = 1000;
13 int       level = 1;
14
15 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
16
17 /* Dekker algorithm */
18 int       favouredthread = 1;
19 int       thread1wantstoenter = false;
20 int       thread2wantstoenter = false;
21 int       info_height = 4;
22
23 int init()
24 {
25     initscr();
26     cbreak();
27     noecho();
28     curs_set(0);
29   start_color();
30   game_wnd = newwin(LINES - info_height, COLS, 0, 0);
31   info_wnd = newwin(info_height, COLS, LINES - info_height, 0);
32   keypad(game_wnd, TRUE);
33   if(!has_colors())
34   {
```

```
35      endwin();
36      printf("ERROR: Terminal does not support color.\n");
37      exit(1);
38    }
39    init_pair(0, COLOR_WHITE, COLOR_BLACK);
40      wbkgd(info_wnd, COLOR_PAIR(0));
41      wbkgd(game_wnd, COLOR_PAIR(0));
42      init_pair(1, COLOR_RED, COLOR_BLACK);
43    init_pair(2, COLOR_YELLOW, COLOR_BLACK);
44      init_pair(3, COLOR_GREEN, COLOR_BLACK);
45      init_pair(4, COLOR_MAGENTA, COLOR_BLACK);
46    return 0;
47  }
48
49  void  *set_level(void *data)
50  {
51    int   tick;
52
53    tick = 0;
54    while (energy != 0 && exit_request == false)
55    {
56      usleep(1000000);
57      tick++;
58      if (tick == 10 && level < 11)
59      {
60        tick = 0;
61        level++;
62      }
63    }
64    return (NULL);
65  }
66
67  void  set_pair(int *pair)
68  {
69    int   rem;
70
71    rem = energy / 5;
```

```
72
73    if (rem >= 150)
74      *pair = 4;
75    else if (rem >= 100)
76      *pair = 3;
77    else if (rem >= 50)
78      *pair = 2;
79    else if (rem > 0)
80      *pair = 1;
81    else
82      *pair = 0;
83  }
84
85  /* User interface */
86  void  draw_info(void)
87  {
88    int   i;
89    int   pair;
90
91    wmove(info_wnd, 0, 0);
92    whline(info_wnd, '-', COLS);
93    wmove(info_wnd, 1, 0);
94    whline(info_wnd, ' ', 20);
95    set_pair(&pair);
96    for (i = 1; i <= pair; i++)
97    {
98      wattron(info_wnd, COLOR_PAIR(i));
99          wattron(info_wnd, A_BOLD);
100         waddstr(info_wnd, "/////");
101         wattroff(info_wnd, A_BOLD);
102         wattroff(info_wnd, COLOR_PAIR(i));
103   }
104   mvwprintw(info_wnd, 2, 0, " - E N E R G Y -       //");
105   wattron(info_wnd, A_BOLD);
106   wattron(info_wnd, COLOR_PAIR(pair));
107   mvwprintw(info_wnd, 2, 18, "%d%%", energy);
108   wattroff(info_wnd, COLOR_PAIR(pair));
```

```
109   wattroff(info_wnd, A_BOLD);
110   mvwprintw(info_wnd, 3, 0, "LEVEL: %d", level);
111   wrefresh(info_wnd);
112 }
113
114 void  *stars_update(void *data)
115 {
116   t_objects *tmp;
117   t_objects *stars = NULL;
118
119   while (energy != 0 && exit_request == false)
120   {
121     tmp = stars;
122
123     /* DEKKER ALGORITHM */
124     thread1wantstoenter = true;
125     while (thread2wantstoenter == true)
126     {
127       if (favouredthread == 2)
128       {
129         usleep(10000 - (level * 1000)); // 10 ms
130         thread1wantstoenter = false;
131         while (favouredthread == 2) ;
132         thread1wantstoenter = true;
133       }
134     }
135     /* DEKKER ALGORITHM */
136     /* CRITICAL SECTION */
137     pthread_mutex_lock(&mutex);
138     while (tmp)
139     {
140       tmp->y++;
141       if (!(((tmp->y - 1) == player_y && tmp->x == player_x)
142         ||((tmp->y - 1) == player_y && tmp->x == player_x + 1)
143         ||((tmp->y - 1) == player_y && tmp->x == player_x - 1)
144         ||((tmp->y - 1) == player_y + 1 && tmp->x == player_x)))
145         mvwaddch(game_wnd, tmp->y - 1, tmp->x, EMPTY);
```

```
146         if (!((tmp->y == player_y && tmp->x == player_x)
147           ||(tmp->y == player_y && tmp->x == player_x + 1)
148           ||(tmp->y == player_y && tmp->x == player_x - 1)
149           ||(tmp->y == player_y + 1 && tmp->x == player_x)))
150           mvwaddch(game_wnd, tmp->y, tmp->x, STAR);
151         wrefresh(game_wnd);
152         if (tmp->y == LINES - info_height)
153         {
154           erase_element(&stars);
155           tmp = stars;
156         }
157         else
158           tmp = tmp->next;
159       }
160       draw_info();
161       pthread_mutex_unlock(&mutex);
162       /* CRITICAL SECTION */
163       favouredthread = 2;
164       thread1wantstoenter = false;
165       push_back(&stars, 0, rand() % COLS);
166     }
167     clear_list(&stars);
168     return (NULL);
169 }
170
171 void    *asteroids_update(void *data)
172 {
173       t_objects *tmp;
174     t_objects *asteroids;
175     int     tick = 1;
176
177     asteroids = NULL;
178     while (energy != 0 && exit_request == false)
179     {
180       tick++;
181       tmp = asteroids;
182
```

```
183     /* DEKKER ALGORITHM */
184     thread2wantstoenter = true;
185     while (thread1wantstoenter == true)
186     {
187       if (favouredthread == 1)
188       {
189         usleep(10000 - (level * 1000)); // 10 ms
190         thread2wantstoenter = false;
191         while (favouredthread == 1) ;
192         thread2wantstoenter = true;
193       }
194     }
195     /* DEKKER ALGORITHM */
196     /* CRITICAL SECTION */
197     pthread_mutex_lock(&mutex);
198     while (tmp)
199     {
200       if (tick % 7 == 0)
201         tmp->y++;
202       mvwaddch(game_wnd, tmp->y - 1, tmp->x, EMPTY);
203       wattron(game_wnd, A_BOLD);
204       mvwaddch(game_wnd, tmp->y, tmp->x, ASTEROID);
205       wattroff(game_wnd, A_BOLD);
206       wrefresh(game_wnd);
207       if ((tmp->y == player_y && tmp->x == player_x)
208         ||(tmp->y == player_y && tmp->x == player_x + 1)
209         ||(tmp->y == player_y && tmp->x == player_x - 1))
210         energy -= 25;
211       if (tmp->y == LINES - info_height)
212       {
213         erase_element(&asteroids);
214         tmp = asteroids;
215       }
216       else
217         tmp = tmp->next;
218     }
219     pthread_mutex_unlock(&mutex);
```

```
220       /* CRITICAL SECTION */
221       favouredthread = 1;
222       thread2wantstoenter = false;
223       if (tick % 7 == 0)
224       {
225         tick = 1;
226         push_back(&asteroids, 0, rand() % COLS);
227       }
228     }
229     clear_list(&asteroids);
230     return (NULL);
231   }
232
233   void  player_clr(int y, int x)
234   {
235     mvwaddch(game_wnd, y, x + 1, EMPTY);
236     mvwaddch(game_wnd, y, x - 1, EMPTY);
237     mvwaddch(game_wnd, y, x, EMPTY);
238     mvwaddch(game_wnd, y + 1, x, EMPTY);
239   }
240
241   void  direction(int *y, int *x, int ch)
242   {
243     if (ch == KEY_UP)
244     {
245       player_clr(*y, *x);
246       (*y)--;
247       if (*y < 0)
248         (*y)++;
249     }
250     else if (ch == KEY_RIGHT)
251     {
252       player_clr(*y, *x);
253       (*x)++;
254       if (*x > COLS - 2)
255         (*x)--;
256     }
```

```
257   else if (ch == KEY_LEFT )
258   {
259     player_clr(*y, *x);
260     (*x)--;
261     if (*x < 1)
262       (*x)++;
263   }
264   else if (ch == KEY_DOWN)
265   {
266     player_clr(*y, *x);
267     (*y)++;
268     if (*y > LINES - info_height - 2)
269       (*y)--;
270   }
271   else if (ch == 'q' || ch == 'Q')
272     exit_request = true;
273 }
274
275 void  _game_over_()
276 {
277   clear();
278   attron(A_BOLD);
279   mvaddstr(LINES / 2, COLS / 2 - strlen(gv) / 2, gv);
280   mvprintw(LINES / 2 + 2, COLS / 2 - strlen(lvl) / 2, lvl, level
        );
281   mvaddstr(LINES / 2 + 4, COLS / 2 - strlen(ex) / 2, ex);
282   attroff(A_BOLD);
283   refresh();
284   getch();
285 }
286
287 int   main(void)
288 {
289   pthread_t   pid1;
290   pthread_t   pid2;
291   pthread_t   pid3;
292   int       ch;
```

```
293   long long int tick = 0;
294
295
296   init();
297   player_y = LINES - info_height - 2;
298     player_x = 1;
299   clear();
300   // nodelay(game_wnd, true);
301   srand(time(0));
302   pthread_create(&pid1, NULL, stars_update, NULL);
303   pthread_create(&pid2, NULL, asteroids_update, NULL);
304   pthread_create(&pid3, NULL, set_level, NULL);
305   while (energy != 0 && exit_request == false)
306   {
307     tick++;
308     /* CRITICAL SECTION */
309     pthread_mutex_lock(&mutex);
310     /* draw < > and engine */
311     wattron(game_wnd, A_ALTCHARSET);
312     mvwaddch(game_wnd, player_y, player_x - 1, ACS_LARROW);
313     mvwaddch(game_wnd, player_y, player_x + 1, ACS_RARROW);
314     wattron(game_wnd, COLOR_PAIR((tick % 10 / 3 % 2) ? 1 : 2));
315     mvwaddch(game_wnd, player_y + 1, player_x, ACS_UARROW);
316     wattroff(game_wnd, COLOR_PAIR((tick % 10 / 3 % 2) ? 1 : 2));
317     wattroff(game_wnd, A_ALTCHARSET);
318     /* draw o */
319     wattron(game_wnd, A_BOLD);
320     mvwaddch(game_wnd, player_y, player_x, PLAYER);
321     wattroff(game_wnd, A_BOLD);
322     wrefresh(game_wnd);
323     /* CRITICAL SECTION */
324     pthread_mutex_unlock(&mutex);
325     ch = wgetch(game_wnd);
326     pthread_mutex_lock(&mutex);
327     direction(&player_y, &player_x, ch);
328     pthread_mutex_unlock(&mutex);
329   }
```

```
330    pthread_join(pid1, NULL);
331    pthread_join(pid2, NULL);
332    pthread_join(pid3, NULL);
333    if (exit_request == false)
334      _game_over_();
335    delwin(game_wnd);
336    delwin(info_wnd);
337    endwin();
338    exit (0);
339  }
```

7. ПРИЛОЖЕНИЕ №2

Листинг 5: lists.c

```
1  #include "game.h"
2
3  void  clear_list(t_objects **objects)
4  {
5    t_objects *tmp;
6
7    while (*objects)
8    {
9      tmp = *objects;
10     *objects = (*objects)->next;
11     free(tmp);
12   }
13   (*objects) = NULL;
14 }
15
16 void  erase_element(t_objects **stars)
17 {
18   t_objects *clear;
19
20   clear = *stars;
21   (*stars) = (*stars)->next;
22   free(clear);
23   clear = NULL;
24 }
25
26 t_objects  *lstnew(int y_cord, int x_cord)
27 {
28     t_objects  *new;
29
30     if (!(new = (t_objects*)malloc(sizeof(t_objects))))
31         return (NULL);
32     new->y = y_cord;
33     new->x = x_cord;
34     new->next = NULL;
```

```
35      return (new);
36  }
37
38  void    push_back(t_objects **objects, int y_cord, int x_cord)
39  {
40      t_objects  *tmp;
41
42      if (*objects)
43      {
44          tmp = *objects;
45          while (tmp->next)
46              tmp = tmp->next;
47          tmp->next = lstnew(y_cord, x_cord);
48      }
49      else
50          *objects = lstnew(y_cord, x_cord);
51  }
```