Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Лабораторная работа №4.
«Система ПВО и летающие тарелки»

по дисциплине
«Операционные системы реального времени»

Выполнил: студент 4 курса

ИВТ, гр. ИП-813

Бурдуковский И.А.

Выполнил с: Стояк Ю.К.

Проверил: преподаватель

кафедры ПМиК

Белевцова Екатерина Андреевна

Новосибирск, 2021 г.

Оглавление

Задание на лабораторную работу

Система ПВО состоит из четырех локаторов и главной ракетной установки (ГРУ). Локаторы определяют высоту и размеры объекта, когда он пролетает над ними. ГРУ может стрелять ракетами и запускать радио-управляемые снаряды (РУС). Количество ракет практически не ограничено, а РУСов всего 11. Вот как это выглядит: `plates-0.demo`. Чтобы выстрелить ракетой, нажмите пробел. Чтобы запустить РУС, нажмите его номер (0..9), Enter, затем пользуйтесь клавишами стрелок, пока он не улетит за пределы видимости. Потренируйтесь запускать несколько РУСов (хотя бы два) и управлять ими, нажимая соответствующий номер и клавиши стрелок.

Это был нулевой уровень. А теперь запустите программу первого уровня `plates-1.demo`. Здесь уже появляется летающая тарелка. Можете попытаться ее сбить. Это немного легче сделать с помощью РУСа. Но вообще, справиться с этой задачей может только счетная машина.

Дадим техническое описание системы. Система управляется и сообщает данные через регистры (доступные по определенным адресам). Мы будем использовать для работы с регистрами функции `int getreg (int reg)` и `putreg (int reg, int value)`, название и параметры которых говорят сами за себя.

ГРУ стреляет ракетой в результате записи команды `GUNS_SHOOT` в регистр `RG_GUNS`.

Управление РУСом производится следующим образом. В регистр `RG_RCMN` записывается номер снаряда (0, 1, ...). После этого команды, записываемые в регистр `RG_RCMC`, передаются (по радиоканалу) снаряду с этим номером. Вам рекомендуется использовать следующие команды:

- RCMC_START запуск снаряда и движение вверх
- RCMC_LEFT переключение направления движения влево
- RCMC_RIGHT переключение направления движения вправо
- RCMC_UP переключение направления движения вверх
- RCMC_DOWN переключение направления движения вниз

Программа первого уровня просто записывает определенную команду в определенный регистр при нажатии на определенную клавишу.

Дадим некоторые коментарии к этой программе.

```
#include "/home/adm/labs/plates.h"
```

включаем определения регистров, команд и др.

```
StartGame (int level);
```

запуск системы на уровне level. Различаются следующие уровни:

- 0 - нет тарелок
- 1 - одна тарелка, движущаяся слева направо
- 2 - две тарелки
- 3 - много тарелок

EndGame ();

завершение работы системы.

Трансляция программы осуществляется следующим образом:

```
cc myprog.cpp /home/adm/labs/plates.o -l vg
```

Ваша задача состоит в том, чтобы **программно** управлять ГРУ (вы видели, что вручную справиться с задачей уничтожения тарелок практически невозможно). Т.е. нужно просто записывать команду в регистр в нужный момент времени. И больше ничего. Все очень просто. **Рисовать на экране ничего не нужно**.

Тарелки движутся строго горизонтально с постоянной скоростью, причем всегда на разных высотах (т.е. высота однозначно идентифицирует тарелку). Для определения момента появления тарелки, ее высоты и скорости используются четыре локатора. Когда тарелка (точнее, ее центральная точка) пролетает над локатором, локатор формирует прерывание LOC_INTR. При этом регистры локатора содержат следующую информацию:

- RG_LOCN - номер локатора (1..4);
- RG_LOCY - высота цели (координата Y);
- RG_LOCW - размер цели.

Значение регистров сохраняется до выхода из обработчика прерывания (вне обработчика содержимое регистров меняется непредсказуемо и его целостность не гарантируется). Размер цели определяет ее область поражения. Для тарелки W=3. Кроме тарелок еще могут появляться перелетные птицы (воробьи) W=1. Стрелять по воробьям не рекомендуется.

Вы можете посмотреть, как идут прерывания от локаторов, запустив программу p4-3.demo.

Наконец, последние тактико-технические характеристики системы (эти данные секретны, их необходимо запомнить наизусть):

размер рабочего пространства

800 на 600 точек

×-координаты локаторов

10, 20, 780, 790

скорость полета ракеты

100 точек в секунду

начальная высота ракеты

570 (y-координата)

скорость полета управляемого снаряда

250 точек в секунду (в любом направлении)

начальная высота управляемого снаряда

570 (y-координата)

×-координата точки старта ракеты или управляемого снаряда

400

Теперь вы можете приступать к составлению программы. Реализация может выполняться на нескольких ступенях.

Выполнение

0) Написать программу, осуществляющую полет управляемого снаряда по квадрату 200x200, а затем по прямоугольнику 500x200 точек. Тарелок нет.

Для выполнения данного задания мы будем последовательно изменять значения регистров. Исходя из задания, что управляемый снаряд летит 250 пикселей в секунду, понимаем, что для пролета одного пикселя затрачивается 4000 микросекунд (4 миллисекунды). Для контроля за временем полета будем использовать функцию usleep(useconds_t usec) – usec – время задержки в микросекундах. За выбор номера снаряда используем регистр RG_RCMN, а за выбор направления полета регистр RG_RCMC. Запуск снаряда сделаем на клавишу пробел.

1) Написать программу, сбивающую одну тарелку с помощью ракеты (тарелка движется слева направо).

Для выполнения ступени № 1 необходимо определить подходящее время для запуска снаряда. Зная время пролета тарелки между двумя радарами, мы можем вычислить время, за которое тарелка достигнет середины экрана. Также не стоит забывать, что ракета должна успеть долететь с Земли до цели – необходимо брать упреждение, взяв необходимое время нам нужно вычесть из времени полета тарелки время полета нашего снаряда. Все временные отметки можно получить с помощью функции clock_gettime(clockid_t clock_id, struct timespec *t_time), где clock_id – идентификатор часов, а t_time – структура, в которую будет записано определенное время. Мы будем использовать реальное время, поэтому в качестве clock_id будем использовать флаг CLOCK_REALTIME. Чтобы произвести залп необходимо использовать регистр RG_GUNS, для точности задействуем залп из 10 выстрелов.

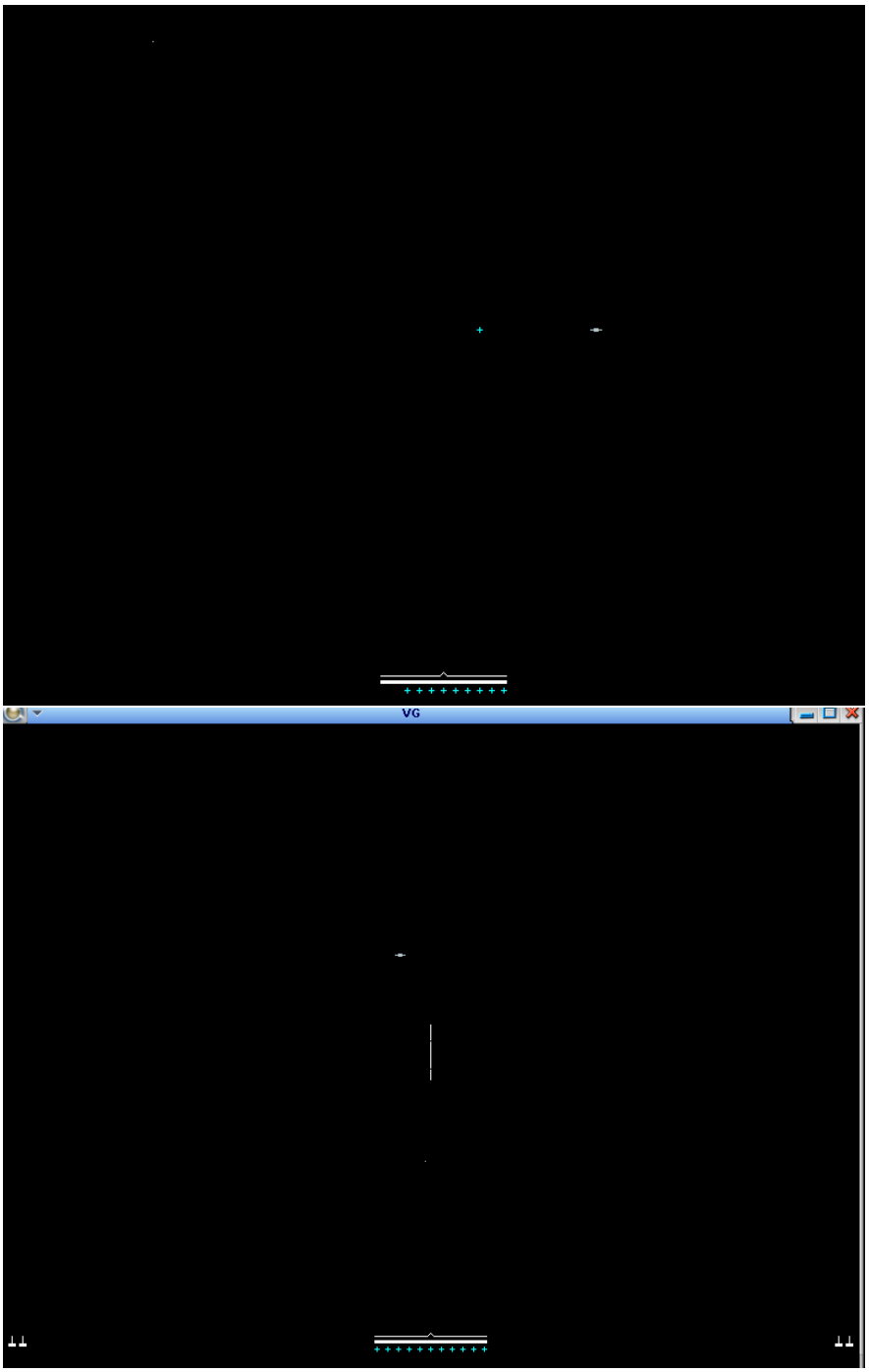
2) Написать программу, сбивающую несколько тарелок с помощью ракет (тарелки движутся в разных направлениях).

Для ступени №2 основной цикл был вынесен в две схожие нити (потока) из библиотеки pthread – в одной шел опрос правого локатора (номер локатора – 4), а во второй левого (номер локатора – 1) для своевременного обнаружения тарелок по обоим сторонам.

3) Написать программу, сбивающую медленные тарелки ракетами, а быстрые -- управляемыми снарядами.

Для ступени №3, на основе программы для ступени №2, была реализована стрельба управляемыми снарядами через специальную переменную ammo, которая находится в разделяемой памяти (shared memory) для обозначения номера, выпускаемого снаряда, такая реализация обусловлена тем, чтобы оба потока могли обратиться к этой переменной. Чтобы управляемый снаряд был выпущен, была добавлена проверка, что ракета не успеет долететь – это когда время до выстрела менее 200мс. Чтобы произвести стрельбу применяются изменения в регистрах RG_RCMN – выбор снаряда, RG_RCMC – направления полета и в зависимости от функции снаряд будет лететь либо влево, либо вправо.

Результат работы программы:



Код программы

File1.cpp:

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <vingraph.h>
#include <sys/neutrino.h>
#include "/root/labs/plates.h"

int main(){
    char key;
    StartGame(1);
    while(1)
    {
        key = InputChar();
        printf("%d \n", key-'0');
        if(key == '0')
        {
            putreg(RG_RCMN, key-'0');
            putreg(RG_RCMC, RCMC_START);
            usleep(800000);
            for(int i =0; i<3; i++)
            {
                putreg(RG_RCMC, RCMC_RIGHT);
                usleep(800000);
                putreg(RG_RCMC, RCMC_UP);
                usleep(800000);
                putreg(RG_RCMC, RCMC_LEFT);
                usleep(800000);
                putreg(RG_RCMC, RCMC_DOWN);
                usleep(800000);
            }
        }
        if(key  == '1')
        {
            putreg(RG_RCMN, key-'0');
            putreg(RG_RCMC, RCMC_START);
            usleep(800000);
            for(int i =0; i<3; i++)
            {
                putreg(RG_RCMC, RCMC_LEFT);
                usleep(1000000);
                putreg(RG_RCMC, RCMC_UP);
                usleep(800000);
                putreg(RG_RCMC, RCMC_RIGHT);
                usleep(1200000);
                putreg(RG_RCMC, RCMC_DOWN);
                usleep(800000);
                putreg(RG_RCMC, RCMC_LEFT);
                usleep(20000);
            }
        }
    }
    EndGame();
    return 0;
}
```

File2.cpp:

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <vingraph.h>
#include <sys/neutrino.h>
#include <time.h>
#include <pthread.h>
#include <sys/mman.h>
#include "/root/labs/plates.h"

using namespace std;

static int *ammo;

void *right_locator_thread(void *args)
{
    struct timespec start, stop;
    while(1)
    {
        usleep(1);
        if(getreg(RG_LOCN) == 4 && getreg(RG_LOCW) == 3)
        {
            clock_gettime(CLOCK_REALTIME, &start);
            while(1)
            {
                usleep(1);
                if(getreg(RG_LOCN) == 3)
                {
                    clock_gettime(CLOCK_REALTIME, &stop);
                    break;
                }
            }
```

```
            long nsecs = stop.tv_nsec - start.tv_nsec;
            long secs = stop.tv_sec - start.tv_sec;
            if(secs > 0 && nsecs < 0)
            {
                nsecs += 1000000000;
                secs--;
            }
            else if(secs < 0 && nsecs > 0)
            {
                nsecs -= 1000000000;
                secs++;
            }
            double usec = (secs * 1000000000 + nsecs) / 1000.0;
            int HEIGHT = getreg(RG_LOCY);
            double target_speed = 10.0 / usec;
            double target_center = 380.0 / target_speed;
            double shot_time = ((570 - getreg(RG_LOCY)) / 100.0) * 1000000;
            double shot_fire = target_center - shot_time - 300000;
            if(usec <= 60000 || shot_fire > 1100000000 || shot_fire < 0.3)
            {
                if(*ammo <= 9)
                {
                    long flight_time = 4000 * (570 - HEIGHT);
                    putreg(RG_RCMN, *ammo);
                    putreg(RG_RCMC, RCMC_START);
                    usleep(flight_time);
                    putreg(RG_RCMC, RCMC_RIGHT);
                    *ammo += 1;
                }
            }
            else
            {
                usleep(shot_fire);
```

```
                for(int i = 0; i < 10; i++)
                {
                    usleep(50000);
                    putreg(RG_GUNS, GUNS_SHOOT);
                }
            }
        }
    }
}


void *left_locator_thread(void *args)
{
    struct timespec start, stop;
    while(1)
    {
        usleep(1);
        if(getreg(RG_LOCN) == 1 && getreg(RG_LOCW) == 3)
        {
            clock_gettime(CLOCK_REALTIME, &start);
            while(1)
            {
                usleep(1);
                if(getreg(RG_LOCN) == 2)
                {
                    clock_gettime(CLOCK_REALTIME, &stop);
                    break;
                }
            }
            long nsecs = stop.tv_nsec - start.tv_nsec;
            long secs = stop.tv_sec - start.tv_sec;
            if(secs > 0 && nsecs < 0)
            {
                nsecs += 1000000000;
                secs--;
```

```
            }
            double usec = (secs * 1000000000 + nsecs) / 1000.0;
            int HEIGHT = getreg(RG_LOCY);
            double target_speed = 10.0 / usec;
            double target_center = 380.0 / target_speed;
            double shot_time = ((570 - getreg(RG_LOCY)) / 100.0) * 1000000;
            double shot_fire = target_center - shot_time - 300000;
            if(usec <= 60000 || shot_fire > 1100000000 || shot_fire < 0.3)
            {
                if(*ammo <= 9)
                {
                    long flight_time = 4000 * (570 - HEIGHT);
                    putreg(RG_RCMN, *ammo);
                    putreg(RG_RCMC, RCMC_START);
                    usleep(flight_time);
                    putreg(RG_RCMC, RCMC_LEFT);
                    *ammo += 1;
                }
            }
            else{
                usleep(shot_fire);
                for(int i = 0; i < 10; i++)
                {
                    usleep(50000);
                    putreg(RG_GUNS, GUNS_SHOOT);
                }
            }
        }
    }
}
```

```
int main(int args, char **argv){
    int level = 1;
    if(args > 1){
        level = argv[1][0] -'0';
    }
    printf("level - %d \n", level);

    ammo = static_cast<int*>(mmap(NULL,sizeof *ammo,
                                  PROT_READ|PROT_WRITE,
                                  MAP_SHARED|MAP_ANONYMOUS,
                                  -1, 0));
    *ammo = 0;

    pthread_t left_locator, right_locator;
    StartGame(level);
    pthread_create(&left_locator, NULL, &left_locator_thread, NULL);
    pthread_create(&right_locator, NULL, &right_locator_thread, NULL);
    pthread_join(left_locator, 0);
    pthread_join(right_locator, 0);
    EndGame();
    return 0;
}
```

Вывод

Мы реализовали логическую часть систему ПВО. Значительно упростило работу то, что уже были реализованы регистры и графика. В процессе выполнения заданий мы узнали, как работать с регистрами. Из пояснения к заданию нам удалось подчерпнуть информацию об устройстве системы ПВО. Самое сложной частью работы оказалось вычисление времени выстрела после того как радар засек тарелку.