事件驱动编程：编写一个视频游戏

视频游戏综合了一些基本的概念和原则，如空间、时间、中断等。本实验将学习屏幕管理、时间和信号。

1. 屏幕管理

1.1 curses 库

curses 库是一组函数，程序员可以用它们来设置光标的位置和终端屏幕上显示的字符样式。curses 库最初是由 UCB(加州大学伯克利分校)的开发小组开发的。大部分控制终端屏幕的程序使用 curses。

curses 将终端屏幕看成是由字符单元组成的网格，每一个单元由(行、列)坐标对标示。坐标系的原点是屏幕的左上角，行坐标自上而下递增，列坐标自左向右递增。

curses 具有的函数包括可以将光标移动到屏幕上任何行、列单元，添加字符到屏幕或者从屏幕上删除字符，设置字符的可视属性，建立和控制窗口以及其他文本区域。用户手册有 curses 的所有函数的详细描述。下面列出其中的 9 个。

基本 curses 函数：

initscr()　初始化 curses 库和 tty
endwin()　关闭 curses 并重置 tty
refresh()　使屏幕按照你的意图显示
move(r,c)　移动光标到屏幕的(r,c)位置
addstr(s)　在当前位置画字符串 s
addch(c)　在当前位置画字符 c
clear()　清屏
standout()　启动 standout 模式(使屏幕反色)
standend()　关闭 standout 模式

1.2 curses 内部：虚拟和实际屏幕

curses 设计成为能够在不阻塞通信线路的情况下更新文本屏幕。curses 通过虚拟屏幕来最小化数据流量。

curses 保留了屏幕的两个内部版本。一个内部屏幕是真实屏幕的复制。另一个是工作屏幕，其上记录了对屏幕的改动。每个函数，比如 move、addstr 等都只在工作屏幕上进行修改。工作屏幕就像磁盘缓存，curses 中的大部分的函数都只对它进行修改。

refresh 函数比较工作屏幕和真实屏幕的差异。然后 refresh 通过终端驱动送出那些能使真实屏幕与工作屏幕一致的字符和控制符。

1.3 curses 实例

curses 例 1：hello1.c

curses 的简单运用。

编译和运行：

```
$ gcc hello1.c –l curses –o hello1
$./hello1
```

curses 例 2：hello2.c

将 curses 函数与循环、变量和其他函数组合在一起会产生更复杂的显示效果。

2. 时间

为了写一个视屏游戏，需要把影像在特定的时间置于特定的位置。用 curses 把影像置于特定的位置。然后在程序中添加时间响应。第一步使用系统函数 sleep。

2.1 时钟编程：sleep

动画例子 1：hello3.c

当编译并运行这个程序的时候，将看到 hello 字符串在屏幕自上而下逐行显示，每秒增加一行，反色和正常显示交替出现。为什么在每次循环结束都要调用 refresh？如果不这样会有什么效果？

动画例子 2：hello4.c

hello4 制造移动的假象。字符串沿着对角线缓慢向下移动。秘诀是先在一个地方画字符串，睡眠 1 秒钟，然后在原来的地方画空字符串以删除原有影像，最后将输出位置推进。

动画例子 3：hello5.c

hello5 将字符串在屏幕两壁弹来弹去。变量 dir 用来控制字符串移动的速度。当 dir 是+1 时，字符串每一秒向右移动一列。当 dir 是-1 时，字符串每秒向左移动一列。改变 dir 的符号就改变了字符串移动的方向。

现在离目标还有多远？已经知道了如何在屏幕的任何地方画字符串，也知道了如何通过画、擦掉和重画之间插入时延来创造动画效果。只是：

(1) 一秒钟时延太长，需要更精确的计时器。

(2) 需要增加用户的输入。

上述问题引出新的话题：用时钟和高级信号编程。然后，将再次回到这个游戏。

2.2 时钟编程：alarm

程序可以以不同的方式使用时钟。一种方式用来在执行流中加入时延，另一种方式是调度一个将来要做的任务。

系统中的每个进程都有一个私有的闹钟(alarm clock)。这个闹钟很像一个计时器，可以设置在一定秒数后闹铃。时间一到，时钟就发送一个信号 SIGALRM 到进程。除非进程为 SIGALRM 设置了处理函数(handler)，否则信号将杀死这个进程。sleep 函数由 3 个步骤组成：

(1) 为 SIGALRM 设置一个处理函数;

(2) 调用 alarm(num_seconds);

(3) 调用 pause。

实例：sleep1.c

说明：调用 signal 设置 SIGALRM 处理函数，然后调用 alarm 设置一个 4 秒的计时器，最后调用 pause 等待。

调用 pause 的目的是挂起进程直到有一个信号被处理。当计时器计时 4 秒钟以后，内核送出 SIGALRM 给进程，导致控制从 pause 跳转到信号处理函数。在信号处理程序中的代码被执行，然后控制返回。当信号被处理完后，pause 返回，进程继续。

系统调用 alarm：设置发送信号的计时器。

alarm 设置本进程的计时器到 seconds 秒后激发信号。当设定的时间过去以后，内核发送 SIGALRM 到这个进程。

系统调用 pause：等待信号。

pause 挂起调用进程直到一个信号到达。如果调用进程被这个信号终止，pause 没有返回。如果调用进程用一个处理函数捕获，在控制从处理函数处返回后 pause 返回。

2.3 时钟编程：间隔计时器(interval timer)

Unix 很早就有 sleep 和 alarm 。它们所提供的时钟精度为秒，对于很多应用来说这个精度是不能让人满意的。后来一个更强大和使用更广泛的计时器被添加进来，即间隔计时器(interval timer)，它具有更高的精度，每个进程都有 3 个独立的计时器而不是原来的一个。每个计时器都有两个设置：初始间隔和重复间隔。

2.3.1. 三种计时器

进程可以以 3 种方式来计时：真实时间、用户时间、用户时间+系统时间。

考虑一个进程在运行了 30s 后结束。在一个分时系统中，这个进程不是一直在运行的，其他的进程与它共享处理器。如有这样一种可能性：从 0~5s 进程在用户模式运行，5~15s 睡眠，15~20s 在核心态运行，20~25s 睡眠，25~30s 在用户模式运行。则真实时间为 30s、虚拟 10s(用户态)、实用 15s(用户时间+系统时间)。

内核提供计时器来计量这 3 种类型的时间。3 种计时器的名字和功能如下：

(1) ITIMER_REAL

这个计时器计量真实时间，不管进程在用户态还是在核心态用了多少处理器时间它都记录。当这个计时器用尽，发送 SIGALRM 消息。

(2) ITIMER_VIRTUAL

这个计时器只有在用户态运行时才计时。当虚拟计时器用尽，发送 SIGVTALRM 消息。

(3) ITIMER_PROF

这个计时器在进程运行于用户态或由该进程调用而陷入核心态时计时。当这个计时器用尽，发送 SIGPROF 消息。

2.3.2. 两种间隔

每个间隔计时器的设置都有这样两个参数: 初始间隔和重复间隔。在间隔计时器用的结构体中初始间隔是 it_value，重复间隔是 it_interval 。

2.3.3. 用间隔计时器编程

在程序中使用间隔计时器，先要选择计时器的类型，然后需要选择初始间隔和重复间隔，还要设置在 struct itimerval 中的值。然后将这个结构体通过调用 setitimer 传给计时器。为了读取计时器设置，使用 getitimer。

间隔计时器例子：ticker_demo.c

程序 ticker_demo.c 演示了如何使用一个间隔计时器。

说明：程序流程。一开始使用 signal 设置函数 countdown 来处理 SIGALRM 信号。然后通过 set_ticker 来设置间隔计时器。

set_ticker 通过装载初始间隔和重复间隔设置间隔计时器。每个间隔是由两个值组成：秒数和微秒数。计时器开始计时，控制返回到 main。

回到 main，ticker_demo.c 进入一个无尽的循环，其间调用 pause。每过大约 500ms，控制跳转到 countdown 函数。countdown 将一个静态变量的值递减，打印一条消息，通常情况下返回调用者。当变量 num 达到 0 时，countdown 调用 exit。

间隔计时器的设置是通过 struct itimerval 来完成的。这个结构体包括初始间隔和重复间隔，两者存储在 struct timeval 中。

```
struct itimerval {
{
    struct timeval   it_value;   /* time to next timer expiration */
    struct timeval   it_interval; /*reload it_value with this */
}
```

```
struct timeval {
{
    time_t tv_sec;              /* seconds */
    suseconds_t tv_usec;        /* microseconds */
}
```

2.3.4 计时器小结

一个 Unix/Linux 程序用计时器来挂起执行和调度将要采取的动作。一个计时器是内核的一种机制。通过这种机制，内核在一定的时间之后向进程发送 SIGALRM。alarm 系统调用在特定的实际秒数之后发送 SIGALRM 给进程。setitimer 系统调用以更高的精度控制计时器，同时能够以固定的时间间隔发送信号。

现在已经知道如何在程序中计时了。要实现视频游戏还需要一项技术：管理中断。

3. 信号

3.1 概念

信号是硬件中断的软件模拟(软中断)。信号来自哪里？信号来自内核，生成信号的请求来自三个地方。

(1) 用户

用户能够通过输入 Ctrl+c、Ctrl+\等来请求内核产生信号。

(2) 内核

当进程执行出错时，内核给进程发送一个信号，例如，非法段存取、浮点数溢出 ，或是一个非法的机器指令。内核也利用信号通知进程特定事件的发生。

(3) 进程

一个进程可以通过系统调用 kill 给另一个进程发送信号。一个进程可以和另一个进程通过信号通信。

哪里可以找到信号的列表？信号编号以及它们的名字通常出现在 signal.h 文件中。每个信号用一个整型常量宏表示，以 SIG 开头，比如 SIGALRM、SIGINT 等。下面是文件的一部分：

```
#define SIGUP       1      /*hangup,generated when terminal disconnects */
#define SIGINT      2      /* interrupt,generated from terminal special char */
#define SIGQUIT     3      /* quit,generated from terminal special char */
……
#define SIGALRM    14      /* alarm clock timeout */
……
```

信号做什么？这要视情况而定。很多信号杀死进程。某时刻进程还在运行，下一秒它就消亡了，从内存中被删除，相应的所有文件描述符被关闭，并且从进程表中被删除。使用 SIGINT 中断一个进程，但是进程也有办法保护自己不被杀死。当进程接收到 SIGINT 时，并不一定非要消亡。

3.2 信号处理 1: 使用 signal

进程能够通过系统调用 signal 告诉内核，它要如何处理信号。

一个进程调用 signal 在以下 3 种处理信号的方法之中选择：

(1) 默认操作(一般是终止进程)，比如，signal(SIGALRM,SIG_DFL)

(2) 忽略信号，比如，signal(SIGALRM,SIG_IGN)

(3) 调用一个函数，比如，signal(SIGALRM,handler)。第三种方法是最强大的。

signal 用来设定某个信号的处理方法。

如果有多个信号到达会发生什么事情？用户可能通过按下 ctrl+c 来产生一个 SIGINT 信号，

或者是按下 ctrl+\产生 SIGQUIT 信号，或者计时器到时产生一个 SIGALRM 信号。在 Unix/Linux 系统中，一个进程如何响应多个信号？

编译并运行 sigdemo.c，测试多个信号。

程序运行时，输入以下几种情况(要输入得快)，看输出结果：

(1)[Ctrl+c][Ctrl+c][Ctrl+c] [Ctrl+c]

(2)[Ctrl+c][Ctrl+\]

(3)hello[Ctrl+c][Enter]

(4) hello[Enter] [Ctrl+c]

(5)[Ctrl+\][Ctrl+\]hello[Ctrl+c]

针对上面各种情况，不同版本的系统可能有不同的响应结果。

3.3 信号处理 2: 使用 sigaction

在 POSIX 中用 sigaction 替代 signal。参数非常相似。指定什么信号将被如何处理。还能得到这个信号上一次被处理时的设置。

4. 使用计时器和信号：视频游戏

现在回到视频游戏。游戏有两个主要元素：动画和用户输入。动画要平滑，用户输入会改变状态。

程序 bounce1d.c 将一个单词平滑地在屏幕上移动。当用户按下空格键，单词就向反方向移动。”s”键和”f”分别增加和减少单词的移动速度。按”Q”键就退出程序。

bounce1d 体现了两个重要的技术：状态变量和事件处理。记录位置、方向和延时的变量定义了动画的状态。用户输入和计时器信号是改变这些状态的事件。每次计时器到达信号就调用改变位置的处理函数。每次得到用户键盘输入信号就调用改变方向和速度变量的代码。

说明：

1. 做实验前认真阅读”事件驱动编程指导”。
2. 实验步骤中涉及到的源程序在”code”压缩文件中。
3. 仔细阅读源程序，包括注解行。