安检排队

2017年5月4日 8:56

安检大厅
- 排队缓冲区（蛇形）
  - 直队*MaxLines
  - 乘客*MaxCustSingleLine
- 安检口*MaxCheck （1~8）
  - 乘客*Num*MaxCustCheck

排队缓冲区
- 初始：直队
- Cust>Num*MaxCustSingleLine 增加队
- Cust>MaxCust 不允许进入缓冲区
- 首先进入小序号的安检口

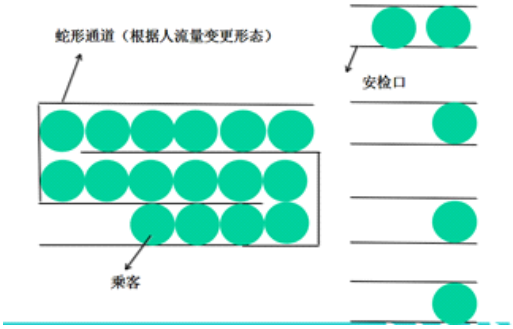
实现
- 循环队列？

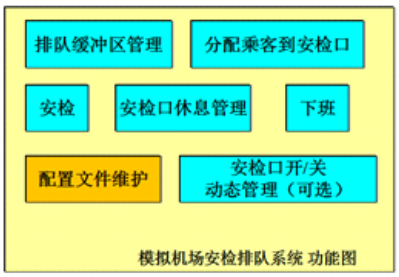
安检口（链表）
- 初始：MinCheck
- 缓冲区/安检口>=MaxSeqLen 开启
  < EasySeqLen
  关闭
- 暂停 休息时间由随机数（1~MaxSec）
- 办理业务 随机数（1~MaxSec）
- 状态显示： 暂停 关闭 服务中

实现
- 数组 int* check【max】
- 链表 队列

下班
- 缓冲区暂停
- 安检口处理完后关闭

为乘客编号 记录当天总数
*使用概率模型 正态分布模拟生成乘客的数量





预备工作（独立的随机模拟事件程序）：
- 用随机函数模拟乘客到达事件的时间间隔秒、人数；
- 保存在到达事件数据结构中，然后记录在输入文件（二进制文件input.dat）里。

正常运行：
- 读取输入文件（二进制文件input.dat），一次读取一条记录，保存在程序的到达事件数据结构中。
- 然后，启动时钟，等待乘客间隔时间到，才能将到达事件置为有效，通知控制模块可以读取。

针对错误进行调试：保留出错情况的那份输入文件，多次运行程序，直到错误修复。

文件输入 input.dat

```
struct entry {
int no; //事件序号
int sec;//事件发生时间间隔
char type;//事件类型：C-乘客到达，R-安检口申请暂停，Q-下班；
int mans;  //事件属性1-到达人数
int check;  //事件属性2-申请暂停的安检口编号
};
```

键盘输入

‘G’表示一个乘客申请进入“排队缓冲区”。
‘X’表示X号安检口请求暂停，X取值为1~MaxCheck。
‘Q’表示下班。

int ev_valid=0;/*事件生效标志,控制模块用,输入文件不用，0-未生效，1-生效;*/

文件输出 output.txt

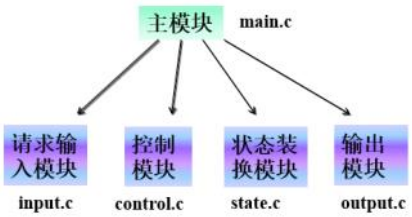
(1)当安检口、安检口队列和排队缓冲区状态有变化时进行记录
(2)每3秒周期性记录安检口、安检口队列和排队缓冲区的状态。
T=<当前时间（3位）>
OFFDUTY=Y/N（是否下班）
WIN1:State=状态(11位)，WinList=队列中乘客编号列表，WinListCustCount=安检口队列人数
WIN2:同上
...
ListLines=排队缓冲区队首乘客编号，队尾乘客编号。
ListCustCount=排队缓冲区总乘客数
当有乘客进入排队缓冲区时，输出“***乘客进入排队缓冲区”
当乘客想进入排队缓冲区但遭到拒绝时，输出“排队缓冲区已满”
当乘客进入安检队伍，输出“***乘客进入**安检口排队”
当乘客结束安检离开安检大厅，输出“***乘客完成安检离开”
当下班指令到达，输出“接收到下班指令”<\n>

配置文件 para.dat
启动后，进入配置信息维护界面，读取历史配置参数，提示用户进行修改，然后运行。

MaxCustSingleLine：单队列最大等待乘客数
MaxLines：蛇形缓冲区最多由MaxLines个直队组成
MaxSeqLen：最大允许等待长度
EasySeqLen： 短期等待长度
MinCheck：最少开放安检口数量
MaxCheck：最多开放安检口数量
MaxCustCheck：安检口队伍最大长度
MaxSec：安检口安检或休息最大时长，单位是秒。

输入和输出的方式和格式应当尽可能方便用户的使用

（1）对所有的输入数据都进行检验，从而识别错误的输入，以保证每个数据的有效性。
（2）检查输入项的各种重要组合的合理性，必要时报告输入状态信息。
（3）使得输入的步骤和操作尽可能简单，并保持简单的输入格式。
（4）输入数据时，应允许使用自由格式输入。
（5）应允许缺省值。
（6）输入一批数据时，最好使用输入结束标志，而不要由用户指定输入数据数目。
（7）在以交互式输入／输出方式进行输入时，要在屏幕上使用提示符明确提示交互输入的请求，指明可使用选择项的种类和取值范围。同时，在数据输入的过程中和输入结束时，也要在屏幕上给出状态信息。
（8）给所有的输出加注解，说明输出数据的含义




```
StartTimer(500); //每0.5秒产生一个周期事件
while ((ev = WaitForEvent())!= EXIT)
{    switch (ev) {
       case TIMER: //响应周期事件
           detectPoint();
           control();
           state_trans();
           print_message();
}
StopTimer();
```

```
main()
{
    init();/*初始化，读取配置信息*/;
    while (未结束 ){
          get_input(); /*输入：获取乘客到来事件*/
          control(); /*控制：排队缓冲区管理；分配乘客到安检口；休息管理；下班管理*/
          state_trans(); /*状态机：计算此刻各安检口的状态 */
          print_message();/*输出：输出此刻各安检口及排队队列、排队缓冲区的状态*/
          time_count(); /*获取程序时间，或者自己模拟计算时间*/
    }
    结束处理；//if any
}
```

Get_input()'

```
int get_input(FILE *fp){
   if (ev_valid){
/*若上次读取的事件已生效，读下一个事件。*/
   {    fread(&theEvent,sizeof(struct entry),1,fp);
   /*读取当前位置的到达事件记录到theEvent结构变量中*/
    ev_valid=0;        }
   if (clock()>=theEvent.sec+time_pre_ev){
/*当前时钟大于上事件时间+本事件间隔，发出本事件*/
    ev_valid=1;
    time_pre_ev=clock(); //记录发出事件时间
    }//end if clock
} //end get_input()
```

clock()/CLOCKS_PER_SEC
Time.h

```
srand(time(NULL)):
1+rand()%6
```