文章编号:1001 - 9081(2007)06 - 1521 - 03

动态数组在系统可靠性与维修性仿真中的应用

余 愚,邓志平
(西华大学 机械工程与自动化学院,四川 成都 610039)
(yogu. xhu@163. com)

摘 要:系统可靠性与维修性仿真属于离散事件仿真的范畴。针对可靠性与维修性仿真过程中可能涉及大量的随机变量的问题,论述了利用动态数据结构提高仿真运行的效率和扩大仿真解题规模的可能性。并以 Turbo C 为背景,讨论了动态数组的建立、动态内存的管理以及动态数组的使用方法,给出了动态数组在系统可靠性与维修性仿真中的应用。

关键词:动态数组;仿真;可靠性;维修性;Turbo C;动态内存;离散事件
中图分类号: TP337;TP391.9 **文献标识码**:A

Application of dynamic array in reliability and maintainability simulation

YU Yu, DENG Zhi-ping
(*College of Mechanical Engineering & Automation, Xihua University, Chengdu Sichuan* 610039, *China*)

Abstract: The system Reliability and Maintainability (R&M) simulation belongs to the discrete event simulation. There may be a large number of random variables in system R&M simulation, so the possibility of using dynamic arrays to greatly enhance the efficiency and solving capability of simulation was described in this paper. The establishment and usage of dynamic arrays and management of dynamic memory in simulation were also discussed in detail and examples in turbo C were given in the programs. Finally the application of dynamic array in system R&M simulation was presented.

Key words: dynamic array; simulation; reliability; maintainability; Turbo C; dynamic memory; discrete event

0 引言

系统仿真是系统动态模型的实验[1],现在已成为研究、分析复杂系统的重要工具。系统仿真可以分为连续系统仿真和离散事件仿真,离散事件仿真所涉及的对象是离散事件系统模型所描述的系统,这种系统通常是用逻辑、数学的关系式或者是用描述系统进程的流程图来表示的,其主要的特点是随机性、离散性和动态性。

离散事件仿真所涉及到的系统是一个随机系统,在仿真进程中将产生大量的随机数据。这些随机数据除了数据的数值是随机的,此外,一簇数据的长度也是随机的。系统可靠性仿真属于离散事件仿真的范畴,在系统可靠性仿真过程中,将产生系统完好—故障—完好的时间历程,该时间历程的时间点是随机的,而且其时间的长度也是不确定的[2]。对于这种类型的数据的描述,可以利用数组的方式来实现,最为常用的数组形式是定长度的固定数组。如果采用固定数组来描述随机长度的数组的话,其数组的长度应定义为该数组数据可能出现的最大个数。由于事前无法预知这个最大个数,数组的长度可能被定义得很大。实际上,从概率的观点出发,一簇数据的个数要达到它可能出现的最大值的可能性非常小,由于每一数据组长度都定义得大于或等于每簇数据的最大值,其结果必然会造成内存资源的浪费进而影响仿真运行的效率。而且,随着仿真系统的复杂程度的增加,因中间数组的增多而造成内存的不足,从而限制仿真解题规模。因此,选择合适的数据结构对于离散事件仿真具有重大意义,显然,如果将数组长度构造成可以跟随离散事件各数据簇长度而动态修改的话,那么上述问题就可以得到很好的解决。

1 动态数组的建立与动态数组的使用方法

下面以 Turbo C 为背景来讨论动态数组的建立和动态数组的使用。C 语言使用的内存空间按照地址由低到高的顺序,可以分为系统内存空间、程序代码空间、全局变量空间、堆空间、栈空间等[3,4]。静态数组存放于“栈”空间(定义为局部变量时)或全局变量空间(定义为全局变量时),因而其大小将受到限制。动态数组可以通过内存分配函数动态地存放于“堆”空间中。“堆”空间是一个自由内存区域,其大小由机器的整个剩余空间长度决定,因此动态数组可以定义得很大,从而提高解题规模。

1.1 Turbo C 的动态内存管理函数

C 语言的动态内存管理函数有:

Void * malloc(unsigned int size):分配大小为 size 字节的内存空间。若能成功,返回分配空间首地址指针,否则返回 NULL。

Void * “calloc(unsigned int num, unsigned int size):分配大小为 num * size 字节的内存空间。如果成功,返回分配空间的首地址指针,否则返回 NULL。

Void free(void * ptr):释放先前由 malloc()或 calloc()所分配及 ptr 所指向的内存空间。

收稿日期:2006 - 12 - 21;**修订日期**:2007 - 03 - 14

作者简介:余愚(1959 -),男,四川蓬溪人,副教授,主要研究方向:机械设备状态监测与故障诊断技术、机电一体化技术、智能测控与智能仪器; 邓志平(1956 -),男,四川巴中人,副教授,主要研究方向:机电一体化技术、数控技术与装备.

1.2 动态数组的建立和使用

动态数组是利用指针与数组的灵活机制来建立的。下面以一个二维随机数组为例说明动态数组的建立。该随机数组的大小是随机变化的。设用 $a[i][j]$ 来表示数组元素，i 的取值范围为 $[0,n]$。当 i 取值不同时，j 的取值范围为 $[0,m_i]$。n 与 $m_i(i=0,1,2,\cdots,n)$ 是在仿真运行过程中产生的随机数。可以用一个二维指针 * * a 来指示该数组，并需用一个一维指针 * Length 来指示存放 m_i 的内存区域。以下程序段定义了二维随机数组 $a[i][j]$（假设该数组元素为实数）。

```
Int i, n, * Length;
/ * Length 为存放 mi的数组指针 * /Floa. ot * * a;
.../ * 产生随机数 n * /
a = (float * * ) calloc( n + 1, sizeof( floao. t * ) );
if ( a = NULL) {
  printf( "Out of memory! \n");
  …                                          / * 分配失败的处理 * /
}
Length = ( int * ) calloc( n + 1, sizeof( int) );
if ( Length = NULL) {
  printf( "Out of memory! \n");
  …                                          / * 分配失败的处理 * /
}
…                        / * 产生随机数 Length[ 0]  ~  Length[ n]  * /
/ * 其中 Length[ i] = mi * /
for( i =0; i < =n; i + + ) {
  a[ I] = ( float * ) calloc( Length[ I] + 1, sizeof( float)
  if ( a = NULL) {
    printf( "Out of memory! \n");
    …                                        / * 分配失败的处理 * /
  }
}
free( Length);
```

定义好后即可以 $a[i][j]$ 的形式使用数组了。

当使用完该数组后，应该将分配的内存空间释放，归还系统。以下程序段完成了动态随机数组 $a[i][j]$ 的释放：

```
for( i =0, i< =n, i + + ) free( a[ i]);
free( a);
```

当数组 $a[i][j]$ 的大小恒定（即 n 和 m_i 为非随机数）时，定义动态数组就不需要引入指示一维数组 $a[i]$ 长度的指针 * Length 了。设 i 和 j 的取值范围为 $[0,n]$ 和 $[0,m]$，其中 n 和 m 为确定值，则 $a[i][j]$ 的定义如下：

```
int   i, m, n;
float * * a
…
a = (float * * ) calloc( n + 1, sizeof( float * * ))
if ( a = = NULL) [
  printf( "Out of memory! \n");
  …                                          / * 分配失败的处理 * /
}
for( i =0, i < =n; i + + ) {
  a[ i] = ( float * ) calloc( m + 1, sizeof( ( float) )
  if ( a = = NULL) {
    printf( "Out of memory! \n");
    …                                        / * 分配失败的处理 * /
  }
}
```

使用完 $a[i][j]$ 后，同样要释放分配的内存空间，即：

```
for( i =0, I < =n, i + + ) free( a[ i]);
free( a);
```

一般说来，n 维动态数组的定义需要 n 重指针。当维数较大的时候，要用多重循环分配数组空间，这样就较为麻烦。为解决这一问题，可将 n 维数组转化为一维等价数组。例如，上例二维动态数组 $a[i][j]$ 可用一个等价的一维数组 $b[k]$ 表示，$b[k]$ 与 $a[i][j]$ 的关系如下：

当数组 a 的大小不确定时，$a[i][j]=b[k],k=\sum_{n=0}^{i-1}(m_i+1)+j$

当 a 的大小确定时，$a[i][j]=b[k],k=(m+1)\times i+j$

其中 m_i，m 与前面定义的动态数组 a 时的意义相同。

以上动态数组的定义与使用方法同样适合于定义结构数组。

1.3 动态内存操作的管理

由于在仿真过程中要产生大量的随机数据，需要用大量的动态数据，这将涉及频繁的内存分配与释放操作，有可能产生诸如使用随机指针、释放已释放的内存区域，释放指向静态内存区域的指针等错误。这些错误可能导致计算错误、程序紊乱、死机等现象，因此有必要进行动态内存操作的合理管理。为此，本文设置了动态内存管理链表，已分配的动态数组的有关信息都以结构的形式作为结点排入该链表中。当进行动态数组的分配的时候，首先在链表中查询该动态数组是否存在，若已存在则不分配；若不存在再按预定的要求分配内存，然后将该动态数组信息插入链表末尾。当释放动态数组时，首先检查链表中是否有此数组，若有，则释放该动态数组，然后删除链表中的有关信息；若不存在，表明试图释放一个非动态内存指针，则不释放该动态内存指针。经过这种处理，就可以保证正确建立和安全使用动态数据。

2 应用实例

系统可靠性与维修性仿真在已知系统元件无故障工作时间分布、维修时间分布、维修运行费用、系统运行特征等参数的基础上，通过模拟系统在一个给定的时间内所有状态变化（完好工作状态与维修状态间的相互转移），获得系统的各种可靠性特征量（可靠度、有效度、平均无故障工作时间等）以及资金在系统服役期内的流动情况，为系统的评估、维修、更新决策提供依据。其中确定系统完好—故障—完好的时间历程是仿真最为关键的一步。

系统完好—故障—完好时间历程定义如下：

```
structs SYSTIME{
  float time;                              / * 系统状态改变的时间 * /
  int state;                               / * 系统在 time 时刻的状态 * /
  / * 若完好: state = 1; 若故障: state = 0 * /
  int main_num;                / * 系统在 time 时刻已维修的次数 * /
  int fail_num;                / * 系统在 time 时刻已故障的次数 * /
  int k;                 / * 系统完好—故障—完好时间历程的序号 * /
  struct SYSTIME * next;     / * 指向时间历程的下一点的指针 * /
};
```

系统组成元件已知信息结构如下：

```
struct COMPONENT{
```

```
    int ftype;                    /*无故障工作时间分布类型*/
    float fpara1;                 /*无故障工作时间分布参数1*/
    float fpara2;                 /*无故障工作时间分布参数2*/
    int mtype;                    /*维修时间分布类型*/
    float mpara1;                 /*维修时间分布参数1*/
    float mpara2;                 /*维修时间分布参数2*/
    …                             /*其他信息*/
};
```

设系统组成元件的个数为 numcomp,则动态分配 numcomp 个元件已知信息结构数组,该数组由二重指针 * * comp - data 指示,其定义为:

```
struct COMPONENT * * comp_data;
```

分配其空间的程序语句为:

```
comp - data = ( struct COMPONENT * * ) calloc ( numcomp, sizeof
  ( struct COMPONENT * ) );
```

接下来就可将元件的已知信息读入该数组中,并可使用该数组抽样产生元件的无故障工作和维修时间(元件状态改变时间点)。对于元件 i(0≤i≤numcomp)可以用以下方式使用其已知信息:

```
comp_data[ i]  - > ftype /*元件 i 无故障工作时间分布类型*/
comp_data[ i]  - > fpara1
                   /*元件 i 无故障工作时间分布参数 1 */
comp_data[ i]  - > fpara2
                   /*元件 i 无故障工作时间分布参数 2 */
comp_data[ i]  - > mtype…    /*元件 i 维修时间分布类型*/
comp_data[ i]  - > mpara1… /*元件 i 维修时间分布参数 1 */
comp_data[ i]  - > mpara2… /*元件 i 维修时间分布参数 2*/
```

在各个元件的状态改变时间点上依据系统组成关系可以计算出在该点系统的状态。然后按一定法则判断该点是否为系统状态改变点。若是则写入系统完好一故障一完好时间历程。该时间历程用链表形式表示。链表结点的定义形式就是上面的结构 struct SYSTIME。

用定义为 struct SYSTIME * p 的指针 p 来指示系统完好一故障一完好时间历程,其表头指针 head 定义为 struct SYSTIME * head。获得完好一故障一完好时间历程的算法如下:

```
head = ( struct SYSTIME * ) malloc( sizeof( struct SYSTIME * ) );
/*表头指针*/
head  - > next = NULL;
p = ( struct SYSTIME * ) malloc( sizeof( struct SYSTIME * );
/*链表第一点指针*/
p - > next = head - > next
/*设置链表第一点参数*/
p - > time = 0;
p - > state = 1;
p - > main - num = 0
p - > fail - num = 0
for( ; ; ) {
  …   /*得到系统状态改变时间点 time 和系统状态 state*/
  if( time > TE) break;
  /*若 time 大于预定仿真时间 TE 则推出*/
  p = ( struct SYSTIME * ) mallocf( sizeof( struct SYSTIME) );
  /*建立新结点*/
  p - > next = head - > next;
  /*将系统改变信息插入链表*/
  p - > time = time;
  p - > k + = 1;
  if( state = = 1) p - > main - num + = 1;
  /*若系统完好则维修次数 main__num 加 1*/
  else p - > fail__num + = 1;
  /*若系统故障则故障次数 fail__num 加 1*/
}
```

经以上运算就得到了系统的完好一故障一完好时间历程,接下来就可以对时间历程进行统计计算,获得系统的运行参数累积值。经多次仿真即可得到系统的可靠性特征量。

3　结语

在离散事件仿真中采用动态数据结构,可以节省内存空间,提高仿真效率,扩大解题规模。利用 C 语言可以容易地实现动态数据结构。本文方法成功地用到了系统的可靠性数字仿真系统 SRDS 中,使得整个仿真系统的数据处于浮动状态,解题规模仅与机器的空间有关,有效地提高了仿真的效率,从而在微机上实现了复杂系统的可靠性仿真。

参考文献:

[1] 吴重光. 仿真技术[M]. 北京: 化学工业出版社, 2000.

[2] DU XP, MA DK, CUI L. System Reliability Simulation of Mechanical System[M]. Proceeding of International Symposium on Intelligence. Knowledge and Interaction for Manufacturing. Southeast University Press, 1995.

[3] 杜小平. C 语言动态内存的安全操作[J]. 电脑编程技巧与维护, 1995, (11): 21 - 22.

[4] 汤颢. C 语言动态内存操作错误的自动检测[J]. 计算机世界, 1993, (10): 22 - 22.

[5] 杜小平. 系统可靠性数字仿真系统 SRDS[J]. 计算机系统应用, 1994, (12): 24 - 27.

(上接第 1520 页)

参考文献:

[1] LORENSEN WE, CLINE HE. Marching cubes: A high resolution 3D surface const ruction algorithms[J]. Computer Graphics, 1987, 21(4): 163 - 169.

[2] NIELSON GM , SUNG J. Interval volume tetrahedronization[A]. In: Proceedings of the 8th IEEE Visualization' 97 Conference[C]. Phoenix, 1997. 221 - 228.

[3] GOLIAS NA , TSIBOUKIS TD. An approach to refining three-dimensional tetrahedral meshes based on Delaunay transformation[J]. International Journal for Numerical Methods in Engineering, 1994, (37): 793 - 812.

[4] 黄伟, 高隽. VTK: 一个面向对象的可视化类库[J]. 中国图象图形学报, 2001, 1(5): 25 - 29.

[5] 黄志聪, 庄天戈. DICOM 标准的发展及最新的变化[J]. 中国医疗器械杂志, 2004, 28(3): 203 - 207.

[6] 李华, 蒙培生, 王乘. 医学图像重建 MC 算法三角片的合并与实现[J]. 计算机应用, 2003, 23(6): 104 - 106.