一种基于结构体的文件有序存取算法

A file sequential accessing algorithm which based on structure

胡修林　唐信忠　(武汉 华中科技大学 430074)

摘要:在嵌入式软件的设计中,资源的限制是需要考虑的问题。同样,嵌入式系统外存中可以存储的文件大小和数目也会受到可用资源的限制。本文介绍一种在这种限制下用于实际嵌入式系统中的文件有序存取算法。

关键词:嵌入式系统 结构体 文件有序存取 DeltaFILE

1 引言

大多数嵌入式系统都需要将一些数据存储到文件中。而嵌入式系统本身资源的限制影响了能够存储的文件大小和数目。笔者在参与的某军用系统嵌入式软件的开发过程中便遇到了这样的问题。随着数据的不断更新,由于文件数目的限制,存储的旧文件将会不断的被新的文件覆盖;同时系统要求数据读取显示时按照新旧顺序排列。因此需要一个合适的文件有序存取算法。本文将介绍在该系统中使用的文件存取算法。

2 DeltaFILE 文件系统简介

在该系统嵌入式软件的开发中,我们选择的是国内的 DeltaOS 嵌入式操作系统。该操作系统以可选组件的形式提供了 DeltaFILE 文件系统。该文件系统将文件操作以一套基于 POSIX 规范的文件 API 的形式提供给开发者(编程语言为 C\C++)。图1便是该文件系统的体系结构。

下面是一些在本文的程序中使用到的该文件系统的 API 函数的简介。

T_WORD fnDFS_creat (T_BYTE * path, mode_t mode);

创建文件。参数 path 指定文件的全路径文件名,mode 在当前版本中始终置位 0;返回值为被创建且被打开的文件的文件句柄。

T_WORD fnDFS_open (T_BYTE * path, T_WORD flags);

打开文件。参数 path 指定文件的全路径文件名,flags 指定打开的模式(如只读、只写等);返回值为被打开的文件的文件句柄。

ssize_t fnDFS_read (T_WORD fildes, T_VOID * buf, size_t nbyte);

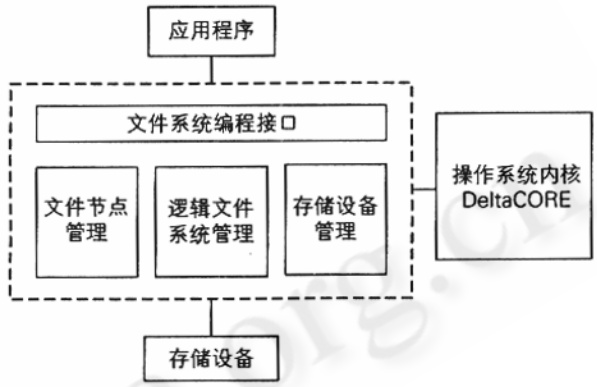

图 1　DeltaFILE 文件体系结构

读取文件。参数 fields 指定文件句柄,buf 指定输入数据缓冲区,nbyte 指定读取的字节数;返回值为实际读出的字节数。

ssize_t fnDFS_write (T_WORD fildes, T_VOID * buf, size_t nbyte);

写入文件。参数 fields 指定文件句柄,buf 指定输出数据缓冲区,nbyte 指定写入的字节数;返回值为实际写入的字节数。

off_t fnDFS_lseek (T_WORD fildes, off_t offset, T_WORD whence);

设置文件的读写位置。参数 fields 指定文件句柄,offset 指定偏移量,whence 指定基准点(包括文件开始、文件当前位置、文件末尾);返回值为文件当前的偏移量。

上述 API 函数当操作失败时，返回值为 -1。

3 算法描述

该嵌入式系统需要存储的数据为结构体类型的数据，假设该结构体如下：

```
struct DATA
{
  int Number1;
  int Number2;
  int Number3;
  int Number4;
}DATA;
```

由于资源的限制，系统只允许存储的文件大小为 100 个该结构体。在数据不足 100 个时，最新的数据是追加到文件末尾进行存储的；随着数据不断的存储，文件大小达到上限以后，由于需要保持文件中的数据尽量新，因此当前文件中最旧的数据便会被最新的数据所覆盖。这样文件内部数据的新旧顺序便发生了变化，不再是起初的按序排列了。系统设计要求按照数据的新旧顺序读取数据和显示数据。因此该算法的目标便是按照正确的新旧顺序读取和写入数据。

该算法实现的关键是一个用于存储文件信息的结构体，该结构体如下：

```
struct FileInfo{
  int Count;
  int Head;
  int Tail;
}FileInfo;
```

FileInfo 结构非常简单，有三个域。在初始化时 Count 被初始化为 0，Head 和 Tail 被初始化为 1。其中 Count 用于记录数据文件大小（以前面的结构体 DATA 为单位，因此取值范围是 0 ~ 100），Head 用于记录数据文件中旧的数据的位置，Tail 则用于记录最新的数据在文件中的位置。图 2 ~ 图 4 是三种典型的情况：

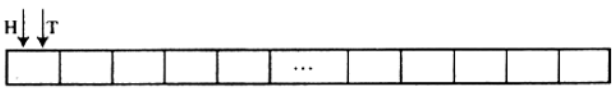

图 2　初始状态

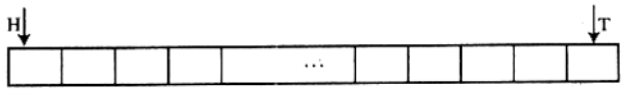

图 3　写入了第 100 个数据

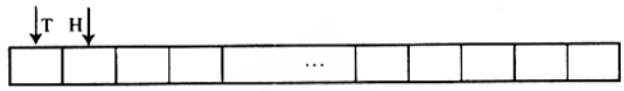

图 4　写入第 101 个数据

有序存取算法便是基于结构体 FileInfo 的。在该算法中，使用了两个文件来分别存储数据和文件信息。其中 File1 用于存储实际的数据，即前面提到的 DATA 类型的数据；而有关 File1 的文件信息 FileInfo 则存储于另一个文件 File2 中。因此每次操作时都会先从 File2 中读取文件信息 FileInfo，根据 FileInfo 便可以知道文件 File1 中的记录的个数，以及队首（对应最旧的数据）和队尾（对应最新的数据）在文件中的偏移量。下面从数据存入和读取两个方面分别说明该算法的原理。

3.1 数据存入

（1）当有数据需要写入文件时，首先会从 File2 中读出文件信息。并根据 Count 的值进行判断。若 Count = =100，说明文件存满，因此内部可能是无序的，这时转入（2）；若 Count <100，说明文件未存满，转入（3）。

（2）将 Head 和 Tail 分别模除 100 再加 1，这样就将这两个文件指针分别循环后移了一次。这时的 Tail 和 Head 就分别对应着最新的数据应该写入的位置以及写入数据后最旧的数据所在文件中的位置（如图 3 ~ 图 4）。将新的文件信息写入 File2；然后打开 File1，根据 Tail 值移动文件指针到最旧的数据处，写入最新的数据覆盖它。写入完成。

（3）打开 File1，移动文件指针到文件末尾，写入最新的数据到 File1 中；然后将 Count 和 Tail 的值分别加 1，接着将新的文件信息写入 File2。写入完成。

3.2 数据读取

（1）当读取数据时，仍然首先从 File2 读取文件信息，并根据 Count 值进行判断。若 Count = =100，则转入（2）；若 Count <100，则转入（3）。

（2）首先将 Tail 的值赋给偏移变量 Offset。然后

在 100 次循环中：首先根据 Offset 的值移动文件指针，从 File1 中读取数据；接着将 Offset 循环前移一次，这样进入下一次循环时就可以按照新旧顺序依次读取数据。读取完成。

（3）首先将 Tail 的值赋给偏移变量 Offset。然后在 Count 次循环中：根据 Offset 的值移动文件指针，从 File1 中读取数据；接着将 Offset 前移一位以准备下次循环时能正确读取数据。读取完成。

4 程序实现

下面给出在实际的嵌入式系统中使用该算法实现文件有序存取的关键源代码。笔者在代码中给出了详细的注释。

文件写入部分：

```
int fd;//文件句柄
int Offset;//文件偏移量
char *FileName;//文件名
FileName = "A:\File2.dat";
fd = fnDFS_open(FileName, O_RDWR);//以可读可写模式打开文件 File2.dat
if(fd != -1)//判断是否成功打开该文件
{
  fnDFS_read(fd, &FileInfo, sizeof(FileInfo));//将文件信息读入结构体变量 FileInfo 中
  if(FileInfo.Count == 100)//文件已经存满
  {
    FileInfo.Head = (FileInfo.Head % 100) + 1;//队头模除 100 加 1
    FileInfo.Tail = (FileInfo.Tail % 100) + 1;//队尾模除 100 加 1
    fnDFS_lseek(fd, 0, SEEK_SET);//重新定位到信息文件 File2.dat 的文件首
    fnDFS_write(fd, &FileInfo, sizeof(FileInfo));//将新的文件信息写入 File2.dat
    fnDFS_close(fd);//关闭 File2.dat
    FileName = "A:\File1.dat";
    fd = fnDFS_open(FileName, O_WRONLY|O_CREAT);//以只写或创建模式打开
  //数据文件 File1.dat
    if(fd != -1)//确认打开操作无误
    {
      Offset = (FileInfo.Tail - 1) * sizeof(DATA);//根据 Tail 算出偏移量
      fnDFS_lseek(fd, Offset, SEEK_SET);//根据偏移量 Offset 移动文件指针
      fnDFS_write(fd, &DATA, sizeof(DATA));//写入最新的数据到 File1.dat
      fnDFS_close(fd);//关闭 File1.dat
    }
  }
  else//文件未存满
  {
    FileInfo.Count++;//将文件记录数加 1
    FileInfo.Tail = FileInfo.Count;//指向最新的数据
    fnDFS_lseek(fd, 0, SEEK_SET);//重新定位到信息文件 File2.dat 的文件首
    fnDFS_write(fd, &FileInfo, sizeof(FileInfo));//写入新的文件信息到 File2.dat 中
    fnDFS_close(fd);//关闭 File2.dat
    FileName = "A:\File1.dat";
    fd = fnDFS_open(FileName, O_WRONLY|O_CREAT);// 以只写或创建模式打开
  //数据文件 File1.dat
    if(fd != -1) //确认打开操作无误
    {
      fnDFS_lseek(fd, 0, SEEK_END);//定位到数据文件尾
      fnDFS_write(fd, &DATA, sizeof(DATA));//向 File1.dat 文件尾追加数据
      fnDFS_close(fd);//关闭 File1.dat
    }
  }
}
```

文件读取部分：

```
int i;
int fd;
int Offset;
char *FileName;
FileName = "A:\File2.dat";
```

```
fd = fnDFS_open(FileName, O_RDONLY);//以只读方式打开 File2.dat
if(fd != -1)//确认打开操作无误
{
  fnDFS_read(fd, &FileInfo, sizeof(FileInfo)); //将文件信息读入结构体变量 FileInfo 中
  fnDFS_close(fd);//关闭 File2.dat
  if(FileInfo.Count == 100)//文件已存满
  {
    FileName = "A:\File1.dat";
fd = fnDFS_open(FileName, O_RDONLY);//以只读方式打开 File1.dat
    if(fd != -1)//确认打开无误
    {
      Offset = FileInfo.Tail;//初始化偏移量到最新数据处
      for(i = 1; i <= 100; i++)//循环 100 次,逐个读出数据
      {
        fnDFS_lseek(fd, (Offset - 1) * sizeof(DATA), SEEK_SET);//根据偏移量
//移动文件指针
        fnDFS_read(fd, &DATA, sizeof(DATA));//读取数据到 DATA 中
        ...
        Processing DATA...//处理数据
        ...
        Offset --;//前移一次偏移量以便读取前一个数据
        if(Offset <= 0)
          Offset = Offset + 100;//实现循环前移
        }
        fnDFS_close(fd);//关闭数据文件 File1.dat
      }
    }
    else//文件未存满
    {
      FileName = "A:\File1.dat";
      fd = fnDFS_open(FileName, O_RDONLY);//以只读方式打开 File1.dat
      if(fd != -1)//确认打开操作无误
      {
        Offset = FileInfo.Tail; //初始化偏移量到最新数据处
        for(i = 1; i <= FileInfo.Count; i++)//循环 Count 次,将所有数据读出
        {
          fnDFS_lseek(fd, (Offset - 1) * sizeof(DATA), SEEK_SET);//根据偏移量
//移动文件指针
          fnDFS_read(fd, &DATA, sizeof(DATA)); //读取数据到 DATA 中
          ...
          Processing DATA...//处理数据
          ...
          Offset --;//前移一次偏移量以便读取前一个数据
        }
        fnDFS_close(fd); //关闭数据文件 File1.dat
      }
    }
  }
```

5 结束语

本文介绍的文件存取算法已经运用到实际嵌入式系统的软件开发中。在嵌入式系统资源有限的情况下,该算法满足了系统设计的要求。并且经过实践验证,该算法成功的实现了文件的有序存取与更新。

参考文献

1 DeltaGUI 编程手册, 北京科银京成技术有限公司, 2002。

2 DeltaFILE 编程手册, 北京科银京成技术有限公司, 2002。

3 谭浩强著,C 程序设计, 清华大学出版社, 1999-12。