Advances in Engineering Research (AER), volume 82
2016 International Conference on Engineering and Advanced Technology (ICEAT-16)

The Dynamic Multi-stack Storage Structure

Zhiguo Ren*[1, a], Wenjiao Da

[1] School of Electronic and Information Engineering, Lanzhou City University. No.11 Jiefang Road, Anning District, Lanzhou City, Gansu Province, 730070, P.R.China.

[a]ren_zhiguo@qq.com;

Keywords: Multi-stack; Dynamic Expansion; Dynamic Recovery

ABSTRACT

The multi-stack has been designed based on sequence storage in reference [1]. But there have two problems: 1.Usually allocate the storage space to the multi-stack in advance, but when one of the stacks has too much data elements, Then the push stack operation will cause the space insufficient, appear "overflow" phenomenon. 2. If one of the stacks to store the data element is too little, it will cause the waste of space. Here based on the above two problems, we design a dynamic multi-stack storage structure, and realizes the basic algorithms of it for the first time.

1. INTRODUCTION

The stack is widely used in system software and application software. In some software often appear the case, that using a number of stacks at the same time. At this time, multi- stacks need to share a block of storage space. In the reference [1-2], We discussed the multi-stacks characteristic, at the same time, we also designed the multi-stack storage structure and implemented basic operation on the structure.

Taking into account the following two problems which are not solved in the sequential multi-stack structure: 1) In general, multi-stack should be allocated storage space preferentially. However, if the multi-stack stored too many of data elements and to be carried on the push operation, it will result in a lack of space, the phenomenon is called "overflow"; 2) If a stack stored too little data elements, it will cause a waste of space.

In this paper, we designed a kind of dynamic multi-stack storage structure, which based on the above two problems and the dynamic shared stack structure inspired in reference [2]. And the basic algorithms of the structure are realized. The structure overcomes the lack of space which caused by the operation of push stack, and the waste of space caused by the operation of pop stack. The concepts and terms which are not mentioned in this article refer to the reference [3-6].

2. DESIGN OF DYNAMIC MULTI-STACK STRUCTURE

In the application of the sequential multi-stack, a certain storage space is given in advance. But when there are too many data elements to be stored in the multi-stack, when the push stack operation is occurred, then the space is not enough, which is called "overflow" phenomenon; If the data elements are too little in the sequential multi-stack, it will cause a waste of space.

Based on the above problems, the dynamic multi-stack technology is proposed. Here the dynamic storage structure is defined, which defines an initial space STACK_INITAL_SIZE for each single stack in the multi-stack. At the same time, each single stack is set up the increment STACK_INCREASE of its storage space, and all of the single stacks are stored in a table a[N] for the unified management, then composed of a dynamic multi-stack structure. The dynamic multi-stack storage structure can be defined as follows:

```
# define STACK_INITIAL_SIZE 5
# define STACK_INC REASE 3
# define STACK_PLANSIZE 10 / / Predetermined space size
# define STAKE_PERCENT 0.5 / / The ratio reached 50% when the recovery was carried out
#define N 4 /* Number of single stack in multiple stacks * /
typedef struct
{   ElemType * Stack;     int Stack_Size, top; }fnode;
typedef struct
{   fnode a[N];}MultipleStack;
```

In the definition of the above structure, N represents the number of single stacks, which contained in the multi-stack. Top represents the top pointer of the i stack, Stake_Size represents the current capacity of the i stack. If N assigned to 4, STACK_INITIAL_SIZE and STAKE_INCREASE assigned to 5 and 3 respectively. The structure of the multi-stack is defined as shown in Figure 1.

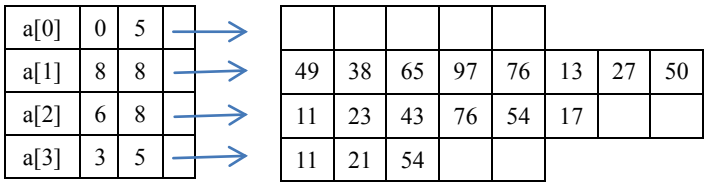

Fig.1 diagram of dynamic multi-stack

3. DESIGN AND IMPLEMENTATION OF BASIC OPERATIONS ON DYNAMIC MULTI-STACK STRUCTURE

3.1 Initialization Operation

The initialization multi-stack is that set each single stack are empty. The operation is as follows: First, the size of the storage space is allocated to each stack. If the stack is no space allocated, the error message is returned. The size of each stack is then assigned to an initial value of STACK_INITIAL_SIZE. Finally, multi-stack is set to empty, each stack's top pointer S->a[i].top is set 0. So the initialization operation is described as follow:

```
int Initial_Stack( MultipleStack * S)
{   int i,spacesize;
    for(i=0; i< N; i++)
    {   spacesize=STACK_INITIAL_SIZE* sizeof( ElemType);
        (* S).a[i].Stack=(ElemType *)malloc(spacesize);
        if((*S).a[i].Stack==NULL)    return ERROR;
         (*S).a[i].Stack_Size=STACK_INITIAL_SIZE;
         (*S).a[i].top =0;
    }
  return OK;
}
```

3.2 Push Stack Operation

In the operation of the general the sequential multi-stack, if the i stack which will push stack is full, it will appear abnormal and return error information. Dynamic multi-stack technology can effectively avoid the occurrence of such situation. If the i stack which is pushing is full, We can use the realloc function to increase the capacity of STACK_INCREASE units. If the space extended successfully.s->a[i].Stake_Size change to S->a[i].Stack_Size+STACK_INCREASE.

The push operation can be continued as the sequential multi-stack. After completing, the top of the stack pointer points to the next position. So the push operation is described as:

```
int PushStack( MultipleStack * S, int i, ElemType e)
{   /* Pressure x data elements into the I stack * / ElemType * New_Base; int xkjdx;
    if(i<1| |i>N)    return ERROR;
    if(S->a[i-1].top>=S->a[i-1].Stack_Size－1)    /* No. I stack is full, to expand * /
     {     xkjdx=S->a[i-1].Stack_Size + STACK_ INC ЯEASE* sizeof( ElemType);
           New_Base=(ElemType*)realloc(S->a[i-1].Stack, xkjdx);
           if(New_Base==NULL) return ERROR;
           (* S).a[i-1].Stack=New_Base;
           S->a[i-1].Stack_Size+=STACK_INC ЯEASE;
     }
     S->a[i-1].Stack[S->a[i-1].top]= e;
     S->a[i-1].top + +;
     return OK;
  }
```

3.3 Pop Stack Operation

Dynamic multi-stack technology not only can solve the "overflow" phenomenon caused by the space shortage, but also can solve the problem of space waste. In this process, we need to judge whether free storage space reached the set value STACK_PLANSIZE after pop stack operation. If free storage space reached a fixed value of STACK_PLANSIZE, we should accord the proportion of STACK_PERCENT to recover part of the free space. The size of the recovered space is: Hs_Size=STACK_PLANSIZE*STACK_PERCENT. Recovery by function “realloc”. When the recovery is complete, then modify the capacity of this stack.

```
int PopstackStack( MultipleStack * S, int i, ElemType * e)
{ /* Popstack up the top of the stack from the I stack and send it to e */
    ElemType * New_Base;
    int xkjdx,Hs_Size=( int)(STACK_PLANSIZE* STACK_PE ?CENT) ;
    if(i<1| |i>N) return ERROR;
    if(S->a[i-1].top<=0) return ERROR;
    *e=S->a[i-1].Stack[S->a[i-1].top];      S->a[i-1].top--;
    if(S->a[i-1].Stack_Size —S->a[i-1].top>= STACK_F ?REESIZE)
    /* Free space to reach a predetermined value, for recycling * /
    {     xkjdx=(S->a[i-1].Stack_Size —hsgs)*sizeof(ElemType);
          newbase=(ElemType*)realloc(S->a[i-1].Stack,xkjdx);
          if(!New_base)    return ERROR;
          S->a[i-1].Stack=New_Base;
          S->a[i-1].Stack_Size —= Hs_Size;
     }
     return OK;
 }
```

4. CONCLUSION

A two-dimensional array of Stack[N][M] is allocated to the sequential multi-stack in advance, It is likely to cause a waste of space and produce “overflow” phenomenon. Dynamic multi-stack adopted the method of dynamic allocation of storage space. We allocate an initial space at first, if the “overflow” phenomenon occurred, we should expand the storage space. If the stack’s free space is large, which causes the waste of space, we should recover part of the free space, which greatly improves the utilization rate of the space.

In this paper, analyzing the dynamic recovery algorithm and dynamic expansion algorithm of multi-stack, we know that dynamic multi-stack technology has the following advantages: 1) Only when storage space is full, we should expand the storage space. Also only when free storage space increases to a certain value, we should recover part of the free space. So, the stack’s free space will always remain in a certain range, which reduces the waste of space. 2) The method can achieve dynamic expansion and recovery storage space without stopping program executing. 3) When the number of data elements is not determined, we’d better use the dynamic multi-stack.

Zhiguo Ren. School of Electronic and Information Engineering, Lanzhou City University. No.11 Jiefang Road, Anning District, Lanzhou City, Gansu Province,730070, P.R.China. Supported by Association of Fundamental Computing Education in Chinese Universities(AFCECU:2016084);Supported by the Education and Teaching Research Foundation (2013-JY-25) and Doc’s Foundation (LZCU-BS2013-09;LZCU-BS2013-12).

5 REFERENCES

1. Zhiguo Ren,Yanan Li,Longzhong Zhang. Sequence storage and implementation technology of multi stack[J]. Automation and instrumentation, 2010(05):103-104.
2. Zhiguo Ren, Data Structure(C Language Description), Science Press, Peking China,2016.
3. Alfred V.Aho,John E.Hopcroft,and Jeffrey D.Ullman.Data structures and Algorithms. Addison-Wesley, 1983.
4. Donald E.Knuth.Fundamental Algorithms,volume 1 of The Art of Computer Programming. Addison-Wesley, 1968.Third edition,1997.
5. Donald E.Knuth.Sorting and Searching,volume 3 of The Art of Computer Programming. Addison-Wesley,1973.Second edition,1998.
6. Mark Allen Weiss.Data Structures and Algorithm analysis in Java. Addison-Wesley,third edition,2007.
7. Thomas H.Cormen,Charles E.Leiserson,Ronald L.Rivest,Clifford Stein. Introduction to 8. Algorithm, the third edition. The MIT Press,2009.
8. Srba I, Bielikova M. Why is Stack Overflow Failing? Preserving Sustainability in Community Question Answering[J]. IEEE Software, 2016, 33(4):80-89.
9. Kashyap B R K. The Double-Ended Queue with Bulk Service and Limited Waiting Space[J]. Operations Research, 1966, 14(5):822-834.
10. Ye D, Xing Z, Kapre N. The structure and dynamics of knowledge network in domain-specific Q&A sites: a case study of stack overflow[J]. Empirical Software Engineering, 2016:1-32.