Assignment Algorithm of Computer Storage Management Partition and Its Implementation

Simin Mu,Weimin Li,Zhangang Zhao

Electronic Information Engineering College, Jinhua University of Technology, Zhejiang, China

Abstract: This paper is about the algorithm of computer storage management partition and its implementation. First, we introduced the background and significance of the project, as well as the principle of technology used by the author in this project. This includes the principles of storage management, partition allocation technology and recycling technology. Then, a detailed analysis of the operating system primary storage allocation of a simple algorithm and one of its implementation., which mainly is to achieve a variable partition of the storage management allocation and recycling technology. Variable partition management usually uses a variety of methods, and our design are the first adaptive algorithm and the use of the best adaptive algorithm. Then it is followed by the test results of the system and discussion of its advantages and disadvantages. The entire storage management partition allocation algorithm and its implementation is entirely in accordance with the primary storage management of the operating system and effective use, in order to achieve a certain level of economic and technical value.

Key words:storage management; allocation; recycling

Chapter 1 Introduction

1.1 Objectives

Von Neumann's storage mechanism requires that any program must be loaded into primary storage (3) to be executed, which modern computer systems are based on. In the computer system, primary storage management to a large extent affects the performance of the system, which makes storage management as one of the central issues of the study of the operating system.

Primary storage in modern computer systems is usually composed of primary storage and secondary storage. CPU direct access to primary storage for the instructions and data has fast primary storage access, but the capacity is small and expensive. Secondary storage does not interact with the CPU directly to store the implementation of the program and data, but able to start the corresponding I / O device for internal and external information exchange, though the access speed is slow, the capacity greatly exceeds the primary storage capacity and it is cheap. While the cost of primary storage is declining rapidly as hardware technology and production levels grow, primary storage capacity remains the most critical and tedious computer resource. Therefore, the effective management of primary storage is still a very important issue in modern operating systems. One of the most obvious distinguishing features between many operating systems is that the storage management methods used are different.

1.2 Important aspects of project development

The use of primary storage in the system is generally divided into two parts: system space, where the operating system itself and its associated system data is stored. The other part is the user space, storing (3) user program and data. In a single-channel system, the primary storage is only transferred to a user process once at a time, and the process can use up all the primary storage space occupied by the operating system. Storage management comes in in the department of the allocation and recycling of primary storage.

In a multi-channel system, multiple operation can be loaded into primary storage at the same time, thus providing a series of requirements for storage management i.e: how to effectively allocate primary storage to multiple operations, how to share and protect primary storage, etc. Storage management is necessary to improve storage and resource utilization efficiency, at the same time providing ease of use. Hence storage management are required to be have primary storage space management, address translation, primary storage expansion, primary storage sharing and protection functions.

1. Primary storage space management

Primary storage space management is responsible for recording the use of each primary storage unit, responsible for the allocation and recycling. Primary storage allocation has two modes: static allocation and dynamic allocation. Static allocation does not allow the operation to reapply primary storage space at run time and allocate all the space required for the one-time allocation when the target module is loaded into primary storage. Dynamic allocation allows the operation to be requested to allocate additional space at runtime, allocating only the basic primary storage required for the operation.

In systems with dynamic allocation methods, a method of combining free zones is often used to make a space as large as possible.

2. Address translation

The program corresponds to a primary storage address when the primary storage is loaded, and the user does not have to worry about the actual location of the program in primary storage. Once the user program is compiled, each target module is addressed with a base address of 0, which is called a relative address or a logical address, and the addresses of the individual physical primary storage cells in primary storage are addressed sequentially from a uniformed base address. Address are known as an absolute address or a physical address. When the primary storage allocation is determined, the logical address needs to be converted to a physical address. This conversion process is called address translation, also known as relocation.

3. Primary storage expansion

The primary storage capacity is limited by the actual storage unit, and running programs are not limited by the size of the primary storage, which requires effective storage management technology to achieve the logical expansion of primary storage. This expansion is not to increase the actual storage unit itself, but through virtual storage, coverage, exchange and other technologies. After primary storage expansion, allows performance that are more than the primary storage capacity of the program. Storage expansion needs to consider

strategies of placing, loading, and eliminating.

4. Primary storage sharing and protection

In order to use primary storage space more efficiently, it requires shared primary storage. Sharing refers to the programs or numbers of the programs in shared primary storage. For example, when the two processes have to call the C compiler, the operating system only allows a C compiler into primary storage, so that the two processes share the primary storage of the C compiler, which reduces primary storage space and improve primary storage utilization. Furthermore, primary storage sharing allows two simultaneous processes to access the primary storage area. As the multi-channel programs share primary storage, each one should run in its own storage area, each running without interfering with each other.

When the multi-channel program to share primary storage space, primary storage information is needed to be protected to ensure that each program run normally in their own primary storage space. During information sharing, protection of the shared area also prevent any processes to destroy any information the shared area. Commonly used primary storage protection methods are hardware methods, software methods and combination of both. The hardware method includes a boundary address protection method and a storage access key method. The boundary address protection method is set to the upper and lower bound addresses for each process. The stored access key method assigns a protection key to each protected primary storage block. Different processes have different privilege codes that are allowed only when the privilege code matches the storage protection key.

1.3 The main work of the author

The author participates in this project of computer storage management partition allocation algorithm and its implementation process. The research and development is carried out in accordance with the design process. The work of the author is mainly to familiarizing the data structure, C / C ++ development environment, operating system, simple management of the storage management of the operating system, and executing one of the applications. The most important application is to achieve variable partition storage management allocation and recycling technology.

Chapter 2 Overview of Storage Management

2.1 Basic concepts

CPU can directly store the commands and data in primary storage, also known as the main storage and real storage. Its structure and implementation will largely determine the performance of the entire computer system. Primary storage size is determined by the system hardware, it is the real storage capacity (3) limited by the actual storage unit. It is the center of modern computer system operation. As shown in Figure 2.1, both the CPU and the I / O system interacts with primary storage.

Figure 2.1 Position of primary storage in a computer system

Primary storage used to store the kernel, program command and data. All functional parts of the computer are stored in a specific unit of primary storage. Primary storage is a large one-dimensional array of words or bytes with each unit has its own address. Through specified address unit to read / write operations, primary storage is accessed.

2.2 The levels of primary storage

Although the primary storage access speed is much higher than the external primary storage, but it is unable to match with the high-speed CPU, thus affecting the entire system. Hence often the primary storage is divided into three levels using the cache, to store the CPU recent procedures and data, as shown in Figure 2.2. The cache consists of hardware registers that are faster than primary storage, but at a much higher cost than primary storage, hence capacity of the cache in an actual system is not large. Often a certain program or data from the primary storage is transferred to the cache for the CPU to directly access the cache. Thereby CPU access to primary storage is reduced, improving the system processing speed.

Figure 2.2 Level 3 primary storage structure

In the three-level circular primary storage structure shown in Figure 2.2, the capacity is growing from the cache to the external primary storage, the cache capacity can be 128KB-256KB up to 256MB while in the primary storage. Access to data are slower with the price is getting cheaper. IBM's cache maximum transfer rate is 120ns-225ns per word while its primary storage transfer rate is only 1us per word. As this chapter mainly describes the primary storage management, therefore description of the cache is not detailed.

2.3 Storage management

The main discussion of primary storage management discussion is at the main storage. Main storage is one of the important resources of the computer system, because any program and data, as well as any data structure for the use of control must occupy a certain storage space. Thus, the primary storage is a valuable and tight resource. Effectiveness of managing them not only directly affects the utilization of primary storage, but also have a significant impact on system performance.

2.3.1 Main goals of primary storage management

The main goal of storage management is to improve the efficiency of the main storage, so a better trade-off is achieved between in the cost, speed and scale.

2.3.2 Main functions of primary storage management

1. Distribution and Recycling of Main Primary storage

In the multi-channel programming environment, there are often multiple programs stored in the main primary storage at the same time. So the main function of the primary storage allocation is to use a certain data structure, with a certain algorithm and allocate each program its main primary storage space, at the same time recording the use of the main primary storage space and operation distribution.

Recycling of the main primary storage space means that when an operation is finished, it must be returned to the occupied main primary storage space. That is, the data structure in which the main primary storage usage is recorded, and the data structure of the operation assignment is deleted.

2. Address translation

The process of changing the logical address of a user program to a physical address that can be addressed directly by the machine at runtime is called address translation, also known as address mapping (i.e, program loading).

3. Shared protection of main primary storage space

In a multi-channel programming system, operations that enter the main primary storage at the same time may need to call same program or data, which is the sharing of main primary storage.

In the allocation and sharing of main storage, protect of information the storage area must be addressed. Storage protection work is generally achieved by hardware and software.

4. Expansion of main primary storage space

The provision of virtual primary storage allows the user to write programs without concerning to the actual capacity of the main primary storage, making the computer system seems to have a much larger main primary storage than the actual main storage capacity.

2.4 Loading and Linking of Programs

1. Implementation of the source program

In the multi-channel environment, the process for the program to run must be created (1) and first thing in the process creation is to

program and data into the main storage. The control diagram is shown in Figure 2.3 below, on how to change a user source into a program that can be executed in main storage, which usually through compiling, linking, and loading.

Figure 2.3 The execution of the source program

② compile: compile the user source code into a number of target modules.

② link: link program will be connecting compiled target module and the library function chain to form a load module.

③ Load: Load the module into the main storage by the loader.

Source program after compilation and link generation to load the program code is done by the compiler language. Loading into the main primary storage is completed by the operating system's loading control program.

2. Link to the program

The function of the linker is to assemble a set of target modules that have been compiled or assembled with the required library functions into a complete load module. There are three ways to implement a link: static links, dynamic links during loading, and dynamic links during running.

⑴ static link. If there are three compiled modules A, B, and C, with lengths L, M, and N, respectively, in module A, there is a statement CALL B for calling module B. In module B, there is a statement CALL C for calling module C. B and C belong to the external call symbols, to load several target module link into a load module, following two questions needed to be solved:

①To modify the corresponding address, usually generated by the compiler of all the target module, the starting address is 0 and each module address is relative to 0. After linking into a loader, the starting points of modules B and C are no longer 0, but L and L + M, where the relative addresses in B and C have to be modified, i.e all relative addresses in block B Plus L and the relative address in module C are added with L + M.

② Change of the external call symbol. The external call symbols used in each module are converted to relative addresses. For example, the starting point of B is transformed into L; the starting point of C is transformed into L + M. A complete load module is formed by the link, as an executable file. It is usually no need to be opened and capable to be run directly in the main storage. This kind of linking in advance with further opening up, is static link.

⑵ Dynamic link when loading. User source program compiled target module is loaded in the main storage, at the same time loading the link. That is, when a target module is loaded, an external module call will cause the loader to find the corresponding external target module, load it into main storage and modify the relative address in the target module. The dynamic link method when loading has the following advantages:

① to facilitate changes and updates of the software version. It is very easy to modify or update individual target modules by using the dynamic link method at load time. However, if the static link has been loaded, modifying or updating one of the membrane module modules, requires reopening the load module. This is not only inefficient, but sometimes impractical.

② easy to achieve target module sharing. If you use the dynamic link when loading, the operating system can link a target module to several application modules, that is, sharing of multiple modules across different applications. However, when using static linking, each application module must contain a copy of the target module and cannot be shared.

⑶ Runtime dynamic link. Although the dynamic link loading method allows a load module to be loaded into the main primary storage of any place, the structure of the module is static. This is mainly reflected in two aspects: First, in the entire implementation of the process (program), the module does not change the load. Second, every time the same load module is loaded. In fact, in many cases, time to run the module may not be the same. Because in advance it is not known which module to run, it can only be all possible to run the linked module together so that the execution module at each execution is the same although this is inefficient. Because of this, in the process of loading the module, often there will be some target modules simply do not run. A typical example is the error handling module. If the program in is running smoothly throught the process with no error, it will not load the module.

2.5 Achievements in storage management

Users need a computing environment to support component compilation and flexible use of data. System administrators need efficient and orderly storage control mechanisms. In order to meet these requirements, the operating system has the following five storage management principles:

⑴ process isolation. The operating system must prevent mutual interference between independent processes.

⑶ automatic distribution and management. The program should be able to be allocated to the required storage area under dynamic conditions. This process is transparent to the programmer. In this way, programmers do not have to worry about the storage area restrictions, according to the needs of the operating system the task allocation of primary storage is determined, improving its efficiency.

⑷ support component programming. The primary storage sharing allows a program to have the potential to address the storage space of another program. Sometimes this is needed and sometimes it poses a great threat to the vast majority of programs and even the operating system itself.

⑸ Long term storage. Many users and applications require long-term storage of information.

⑹ protection and access of space.

Often, the operating system uses virtual primary storage and file system to meet these needs. Virtual primary storage allows the program to be addressed in a logical way, regardless of the amount of primary storage that is physically available. When a program is executed, in fact, only part of the program and data can be stored in primary storage, the other part is stored on disk. This storage space is divided into physical space and logical space of the method, providing a strong support in the storage of data for the operating system.

The file system stores the information in a named object called a file for long-term storage. A file is a very convenient concept for programmers, and the file is the access control and protection unit for the operating system. Figure 2.5 describes two views on the storage system. From the user's point of view, the processor along with the operating system to provide users with a virtual processor, which is the access to virtual storage space. This type of storage can make a linear address space or a set of segments (segments of available length blocks). In either case, programming instructions can access programs and data in virtual primary storage. Process separation can be achieved by giving each process a unique, non-overlapping virtual primary storage space. Process sharing can be achieved by overlapping two virtual storage spaces. The file is stored on the storage medium and can be copied into the virtual primary storage. From the designer's point of view, the primary storage consists of an auxiliary storage that can be directly addressed into the main storage and supported by indirect accessed of loading the data block into primary storage. Address translation hardware - the mapper is located between the processor and primary storage. The program is accessed with a virtual

address, and the virtual address is mapped to the actual primary storage address. If an access is not in the actual primary storage, then a piece of primary storage in the actual primary storage will be exchanged to the auxiliary storage, which can exchange needed data into primary storage. In this process, the process of generating this address access will be suspended.

(A)User's perspective

(B) the operating system: designer's point of view

Figure 2.5 looks at two different views of the storage system

2.6 Fixed partition storage management

When the main primary storage is large and the operation is small, but the user (1) storage management of the main primary storage causing too much waste. So how can the main primary storage be loaded into multiple operations at the same time This produces two storage management methods that can be used in multiprogramming. That is, fixed partition storage management and variable partition storage management. This section mainly introduces the basic principle of fixed partition storage management, the distribution and recovery of main primary storage, address translation and storage protection, and its management characteristics.

Fixed partition storage management is the earliest use of a multi-channel operation running storage management. It requires all the work stored into the main primary storage into a continuous main primary storage space. In this management mode, the main primary storage allocated to the user area is divided into a number of fixed size of the region, every region into a partition, every partition loaded to an operation, while an operation can only be loaded In a partition. Multiple operations can be loaded so that they are executed concurrently. When there is a free partition, we can select a suitable size operation from the backup queue in the external primary storage. When the operation finishes, we can select another cell operation from the backup queue to move to that partition.

There are two ways to divide the main primary storage into several fixed-size partitions.

(1) Equal partition size. All the main partitions are equal in size with obvious drawbacks. That is, when the program is too small, it will cause the waste of the main primary storage space. When the program is too large, may be due to the size of the partition, it is not enough to load the program, leaving the program fail to run. In spite of this, equal partitioning is still used, mainly used for a computer to control multiple identical objects, because the main primary storage space required for these objects to be equal in size. For example, the furnace temperature control system uses a computer to control multiple identical smelting furnaces.

(2) Inequal partition size. In order to overcome the shortcomings of the partition size equal distribution method, in the main primary storage., a number of larger partitions are divided out and moderately assigning medium partition and giving a small amount of huge partitions. For small programs, you can assign a small partition, so that when the large and medium program comes, a large partition can be found for the program to be loaded into the main primary storage and run.

2.7 Variable partition storage management

As the fixed partition storage management mode partition size is fixed, the operation is likely to cause a lot of waste of the main storage space. In order to make the size of the partition consistent with the size of the operation, we can use variable partition storage management. Variable partition storage management, also known as dynamic partition storage management, is based on the size of user operations, dynamic division of the main storage. Variable partition (1) storage management compared to fixed partition storage management significantly improve the utilization of the main primary storage space. This section mainly introduces the basic principle of variable partition storage management, allocation/recovery of main primary storage space, address translation, storage protection, and its management characteristics.

Variable partition storage management dynamically partition according to the size of the operation when the operation requests to load the main primary storage, making the size of the partition suitable for operation's requirements. The size of the partition and the number of main primary storage in the partition is uncertain.

Variable partition storage management must handle three issues. Firstly, the partition used in the allocation of data structure, secondly, the partition allocation algorithm, and finally the partition of the distribution and recycling.

Chapter 3 Storage Management Partition Assignment Algorithm

In the initial use of partition management system, in addition to the operating system occupied by the partition, the remaining storage area is a large free area, an available space for partition allocation to refer to the system according to the user's request, providing space to meet the user requirements of the free area and assigned to the user. After a period of time of run, the system due to the continuous distribution and recycling, available space are partitioned into a number of different sizes of free areas, managed with a linked list.

The first adaptation algorithm organizes the free area by the order in which the first address is incremented into a free area table. When there is a space for the user to apply as size, the allocation program starts from the first free area in the free area table and finds the first free area of size equal to or greater than the size applied. A partition of size is allocated from the zone to the user, and the remainder remains in the free area table as a free area. When recycling, the free area table is queried at the first address of the recycle. If there is a free area adjacent to the recovery area, it is merged into the adjacent area. Otherwise, the recovery area is inserted into the free area table in the order of increasing size of the address.

The optimal algorithm is to retrieve all the nodes in the available space table, and to find the smallest one in the available blocks greater than or equal to the application space, in order to save searching time. According to the size of the free area from small to large organization of the free area table, when the user to applies, the search begins from the free area header. First search to meet the requirements of the free area, that is, the optimal free area, will intercept the size of the partition size and assigned to the user. The remaining part of size changes, hence it should be re-inserted in the size of the free area table. Recycling cannot simply insert the deleted node into the free area table, but to consider the recovery area and the address connected to the free area to merge, and to ensure that the recovery area is still arranging in the order of size.

Chapter 4 Detailed design of the system

4.1 the first strategy adapted for the recovery algorithm of acceptment 1()

The nodes of the first adaptation strategy are arranged in ascending order of the first address. Idle area table header are head, the first address of the recovery area are back1, before and after are two adjacent area pointers. The algorithm is as follows:

(1) set initial state before = head, after = head-> next.

(2) In accordance with the size of the address to find the insertion point of the node, that is, back1-> address> = before -> address, and back1-> address <= after-> address, insert it as shown in

Figure 4.1

Figure 4.1 Insert the collection node into the available space

(3) with the previous combining piece considered, if before-> address + before-> size == back1-> address, the recovery area are adjacent to the previous address, hence the recovery area is merged into the previous one. Modify the before the pointer and size, before -> size = before-> size + back1-> size, before-> next = back1-> next, release the space occupied by back1, aligning the pointer back1 to before, as shown in Figure 4.2.

Figure 4.2 merges with the previous block

If the back1-> address + back1-> size == after-> address, the recovery area is adjacent to the next block, so the next free area is merged into the recycle area. Modify the pointer and size to back1 , back1-> size = back1-> size + after-> size, back1-> next = after-> next, and finally release after the occupied space, as shown in Figure 4.3.

Figure 4.3 merges with the next block

⑸ Modify the maximum block and the largest number of blocks. If head-> size <back1-> size, the current maximum block value is back1-> size and the maximum number of blocks is 1. Otherwise, if head-> size == back1-> size, the maximum number of blocks is incremented by one.

The flow chart of the recovery algorithm for the first adaptive policy is shown in Figure 4.4.

Figure 4.4 Summary of the recovery algorithm for the first adaptation strategy by the acceptment1 () algorithm

4.2 recovery algorithm for the optimal adaptation strategy of acceptment2 ()

The nodes of the optimal adaptation strategy are arranged according to the size of the free area (the size of the assembly point) from small to large. Set the head node pointer as head, the recovery area as back1, before and after as two adjacent pointers, using the algorithm is as follows:

⑴ set initial state before = head, after = head-> next.

⑵ If the available area is empty, then the node will be recycled to the head after the head-> size = back1-> size, maxblock = 1.

⑶ If the available area is not empty, consider merging with the previous node. Loop to determine whether the node to be recovered back1 is adjacent to the previous block, the adjacent conditions are back1-> address == after-> size + after-> address. If adjacent merge, with the operation is to merge back1 to after, after-> size = after-> size + back1-> size, before-> next = after-> next. To release the space occupied by the back1 node, back1 = after, after = NULL, jump out of the loop. If not adjacent, move the pointer before and after. Repeat the above judgement until the merge or after pointer is empty. The node relationship is shown in Figure 4.5.

Figure 4.5 Recycling nodes are adjacent to the previous block

(4) If the available area is not empty, consider merging with the next node. Loop to determine whether the recovery node is adjacent to the next node, the adjacent conditions are back1-> size + back1-> address == after-> address. Merge if adjacent, with operation back1-> size = after-> size + back1-> size, before-> next = after-> next. Release the space occupied by the after node by operation after = NULL and jump out of the loop. If not adjacent, move the pointer before and after. Repeat the above judgement until the merge or after pointer is empty. The node relationship is shown in Figure 4.6.

Figure 4.6 Recycling nodes are adjacent to the next block

⑸ After the above steps, only the merge block is done. The recovery node has not yet been linked to the available partition. The pointer of the recovery node is still back1, so the entry of the recovery node is found from the free node header position, i.e back1-> size> before-> size and back1-> size <after-> size. Insert the back1 node. The pointer relationship shown in Figure 3.6.

Figure 4.7 Insert the Recycling Node into the appropriate position by size

⑹ modify the maximum block and the largest block number. If head-> size <back1-> size, the current maximum block value is back1-> size and the maximum number of blocks is 1. Otherwise, if head-> size == back1-> size, the maximum number of blocks is incremented by one.

The flow chart of the recovery algorithm for the optimal adaptation strategy of acceptment 1() is shown in Figure 4.8 (1) and 4.8 (2).

Figure 4.8 (1) flow chart of the optimal adaptation strategy for the recovery algorithm acceptment1 ()

Figure 4.8 (2) flow chart for the optimal adaptation strategy for the recovery algorithm acceptment1 ()

4.3 Assignment () allocation algorithm

Assignment () is the allocation of the partition to the user. In this process, the allocation of a partition to the user may result in the maximum free space available space size and number change, the variable maxblocknum is the largest number of idle area record variables. Hence during the change it is required to re-calculate the maximum free area size and the largest number of blocks. Even with the best adaptation, there is a need to adjust the arrangement of available areas of the list. The operation is as follows:

⑴ use malloc function to get a node, the size of the node stores the size of the application space application. The node address is assign. If the user application space size exceeds the maximum available space, it cannot be assigned, and returned to failure information.

(2) Set the move pointer before and after, with initial state before = head-> next, after = head-> next, this is used to find in order the free area equal to or greater than the user application space node, if found such nodes after, then:

① If the found node is equal to the size of the user's application, the node is assigned to the user. That is, if-> size == application, and if after-> size == head-> size, the number of maxblocknum blocks is decremented by one, the after node is assigned to the user, that is, the node is removed from the free area list and pointer is changed to before-> next = after-> next, assign-> address = after-> address, release after.

② If the node is found to be larger than the size of the user's application, appropriate length of the node is cut and saved. The remainder is still stored in the list of available space zones. If after-> size> application, and if after-> size = after-> next, the number of maxblocknum is decremented by 1, and perform after-> size = after-> size-application, assign-> address = after-> address + After-> size. The node is not completely allocated from the free zone table, it is only changing the size of it. Therefore, the node is not suitable at this position in the optimal adaptation algorithm and should be adjusted by picking the node out from the list as the remaining node, call function

acceptment2 in accordance with the recovery node into the appropriate location of the list.

(3) Modify the maximum number of blocks, because the distribution of nodes may lead to the change of the largest number of blocks and fast maximum. So from the header node, find the current available maximum area. If the maximum is not one, accumulate the number of blocks.

(4) return to free area address assigned by the called function.

The flow chart of the assignment () assignment algorithm is shown in Figure 4.9 (1) and 4.9 (2)

Figure 4.9 (1) assignment () allocation algorithm flow chart

Figure 4.9 (2) assignment () assignment algorithm flow chart

4.4 backcheck () checks the block function

The function of checks whether the recovery block is legal. As the program is simulating primary storage allocation, the allocation and recycling of primary storage area data is input by user, so it is necessary to check its legitimacy. The principle is that the size and address of the recovered block must be positive and the address cannot overlap with the existing block. The flow chart of the backcheck () check block function is shown in Figure 4.10.

Figure 4.10 backcheck () function of checking the block

4.5 print () output of available space

This is the output of the single node of the header node. When the format is set, the subsequent node of the slave node starts to output until the list is empty. The flow chart is shown in Figure 4.11.

Figure 4.11 print () flow chart of the output list

Chapter 5 System Testing

5.1 Results of the operation of the program

As shown in Figure 5.1 and Figure 5.2 shows the input and run results for selecting the optimal adaptation algorithm, it can be seen that the linked list is sorted by block size.

Figure 5.1 Initial interface and allocation results

Figure 5.2 Recall algorithm results

As shown in Figure 5.3 and Figure 5.4 shows the input and operation results of the first adaptation algorithm. We can see that the linked list is sorted by address size. When running the program, the data can be customized. It is important to pay special attention to whether the recovery is completed correctly with the merger of the adjacent block and to watch the arrangement of the list. Figure 5.5 shows the merging of the recovery and free zones.

Figure 5.3 Initial interface and distribution results

Figure 5.4 Recall algorithm results

Figure 5.5 Merging of Recycled and Free Zone

5.2 Advantages and disadvantages

The use of variable partition storage management is to allocate the main primary storage. It has the following advantages: the length of the partition is not pre-fixed, but according to the actual needs of the division. The number of partitions is not predetermined, but is determined by the number of operations loaded. The size of the partition is determined by the size of the operation, and the efficiency of the main storage is improved. The disadvantages are in the main primary storage allocation process, will produce many main primary storage 'fragments.' The so-called main primary storage 'debris' refers to the small and fragments inapplicable to the main primary storage space, causing certain amount of waste in the main storage space.

REFERENCES

[1] Lian Weimin, Xu Baomin. Tutorial of operating system principles l. Beijing: China Water Resources and Hydropower Press, 2004: 83-102
[2] Liao Lei. C language programming. Beijing: Higher Education Press, 2000
[3] He Yanxiang, Li Fei. Computer operating system. Beijing: Tsinghua University Press, 2003: 132-134
[4] Wang Qing. Computer Operating System. Beijing: Metallurgical Industry Publishing, 2003
[5] Lingyun Xiang, Shi Xiangning, Ge Xinhui. Operating system typical problem analysis and actual simulation. Changsha: National University of Defense Technology Press, 2001
[6] Liu Jiahai. Visual C ++ programming basics. Beijing: Science Press, 2003
[7] Tan Haoqiang. C Program Design. Second Edition. Beijing: Tsinghua University Press, 1999
[8] Li Qiang, Jia Yunxia. Visual C ++ Project Development Practice. Beijing: China Railway Publishing House, 2003