Lab Report

Subject: OS Assignment 3 .

# College Liangjiang International college .

Major Computer Science and technology

# Class class 117197204

Stu.Name 刘建中

# Guidance teacher Pro.XiangNan

## 1. Requirement

1. Write a program that translate logical address into physical address
2. Program the following disk scheduling algorithms, and find out the average seek length of each algorithm. Design the main interface to let the user select an algorithm and input the track sequence.

## 2.Experimental content

1. Write a program that translate logical address into physical address using:

① address translation in paging mode

② address translation in segment mode

The demonstration is required to be correct and clear, and the programming tools are unlimited.

2. Program the following disk scheduling algorithms, and find out the average seek length of each algorithm. Design the main interface to let the user select an algorithm and input the track sequence.

① 1. First come first serve algorithm (FCFS)

② 2. Shortest seek time first algorithm (SSTF)

## Experimental environment

1. Ubuntu 20.04 -- The whole experiment was carried out on the platform.
2. GCC，GDB, --Debug Execution Code.
3. Vim, Vscode, gedit,Eclipse -- platforms for writing code

## Basis process

### Address translation

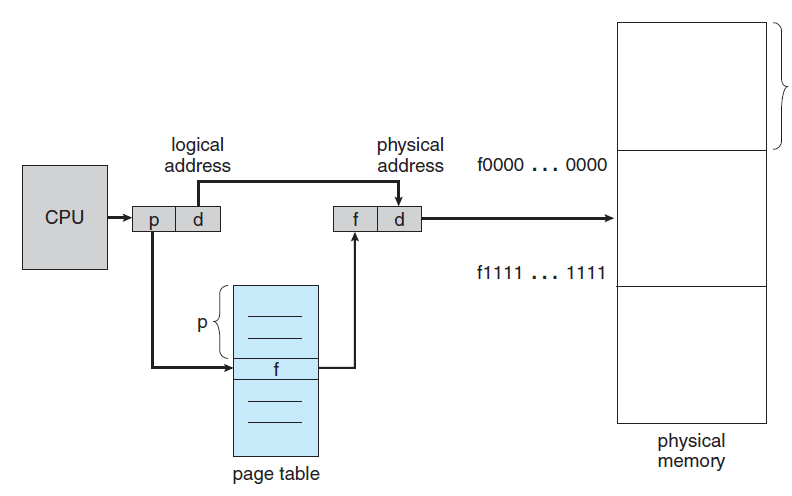
① Page mode

paging avoids external fragmentation and the need for compaction, whereas segmentation does not. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store. Most memory-management schemes used before the introduction of paging suffered from this problem. The problem arises because, when code fragments or data residing in main memory need to be swapped out, space must be found on the backing store. The backing store has the same fragmentation problems discussed in connection with main memory, but access is much slower, so compaction is impossible. Because of its advantages over earlier methods, paging in its various forms is used in most operating systems, from those for mainframes through those for smartphones. Paging is implemented through cooperation between the operating system and the computer hardware.

The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source (a file system or the backing store). The backing store is divided into fixed-sized

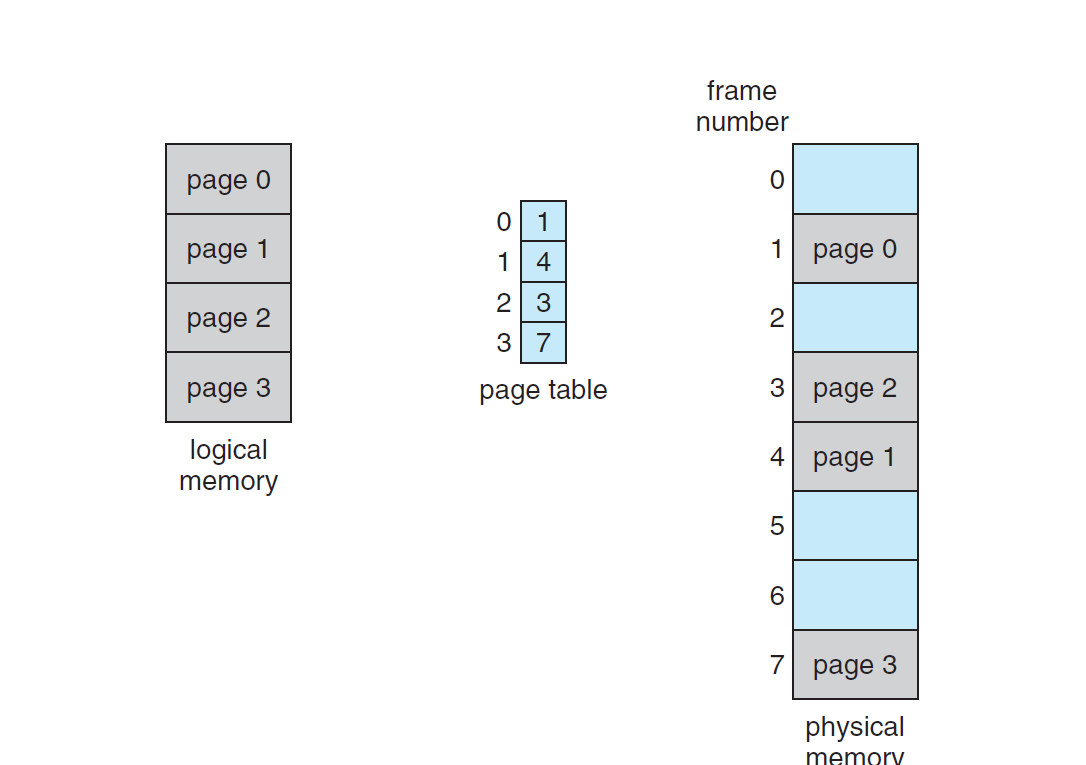
blocks that are the same size as the memory frames or clusters of multiple frames. This rather simple idea has great functionality and wide ramifications. For example, the logical address space is now totally separate from the physical address space, so a process can have a logical 64-bit address space even though the system has less than 264 bytes of physical memory.

The hardware support for paging is illustrated in Figure 1. Every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d).



Figure

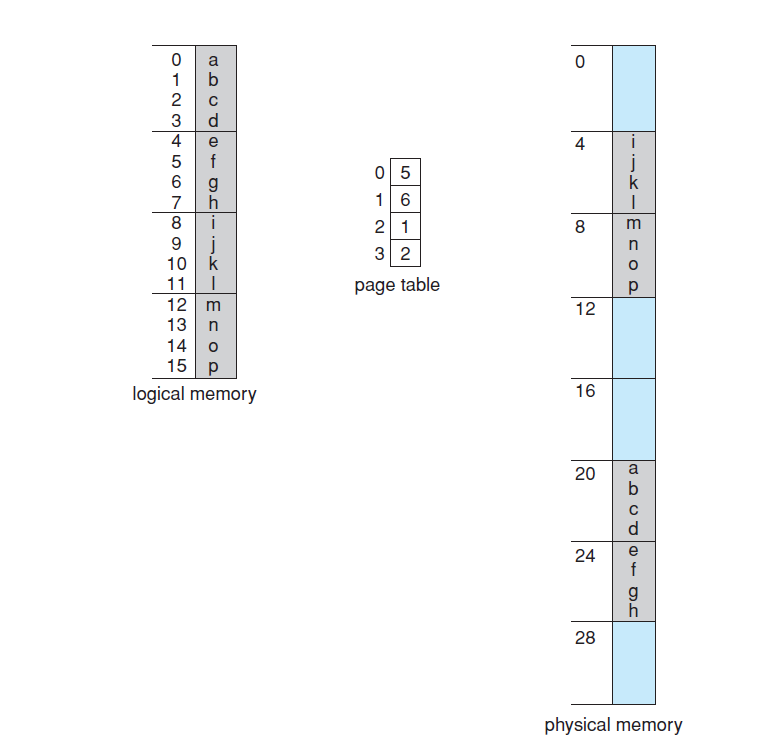
The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit. The paging model of memory is shown in Figure2.



Figure

The page size (like the frame size) is defined by the hardware. The size of a page is a power of 2, varying between 512 bytes and 1 GB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of the logical address space is 2m, and a page size is 2n bytes, then the high-order m− n bits of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:

where p is an index into the page table and d is the displacement within the page. As a concrete (although minuscule) example, consider the memory in Figure 3. Here, in the logical address, n= 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the programmer’s view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0.



Figure

is free or allocated and, if it is allocated, to which page of which process or processes. In addition, the operating system must be aware that user processes operate in user space, and all logical addresses must be mapped to produce physical addresses. If a user makes a system call (to do I/O, for example) and provides an address as a parameter (a buffer, for instance), that address must be mapped to produce the correct physical address. The operating system maintains a copy of the page table for each process, just as it maintains a copy of the instruction counter and register contents. This copy is used to translate logical addresses to physical addresses whenever the operating system must map a logical address to a physical address manually. It is also used by the CPU dispatcher to define the hardware page table when a process is to be allocated the CPU. Paging therefore increases the context-switch time.

② Segment mode

Each segment has a name and a length. The addresses specify both the segment

name and the offset within the segment. The programmer therefore specifies

each address by two quantities: a segment name and an offset.

For simplicity of implementation, segments are numbered and are referred

to by a segment number, rather than by a segment name. Thus, a logical address

consists of a two tuple:

<segment-number, offset>.\

Normally, when a program is compiled, the compiler automatically constructs

segments reflecting the input program.

A C compiler might create separate segments for the following:

1. The code

2. Global variables

3. The heap, from which memory is allocated

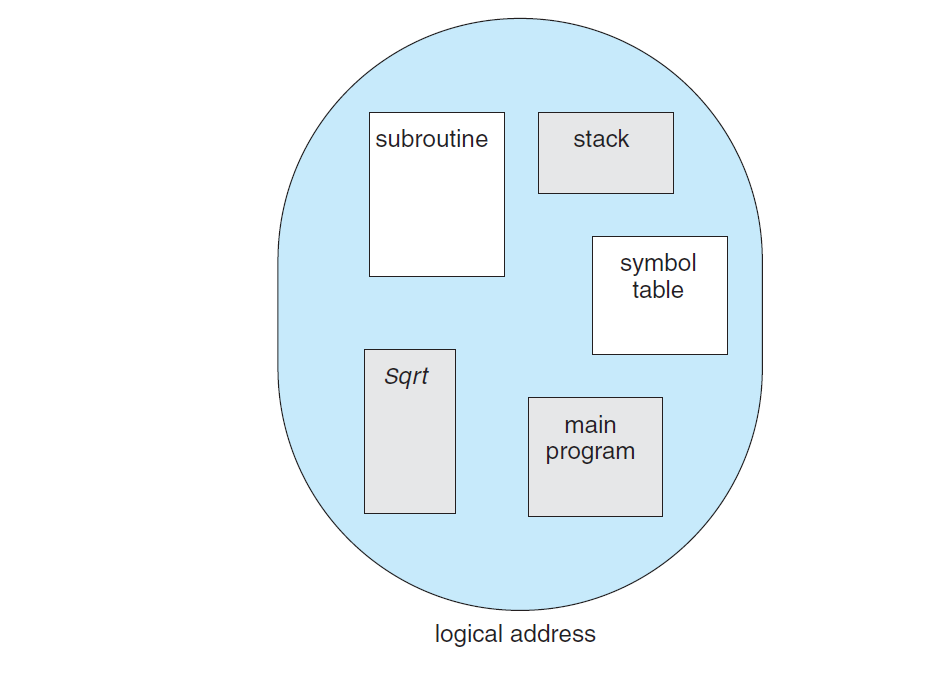
4. The stacks used by each thread

5. The standard C library

Libraries that are linked in during compile time might be assigned separate

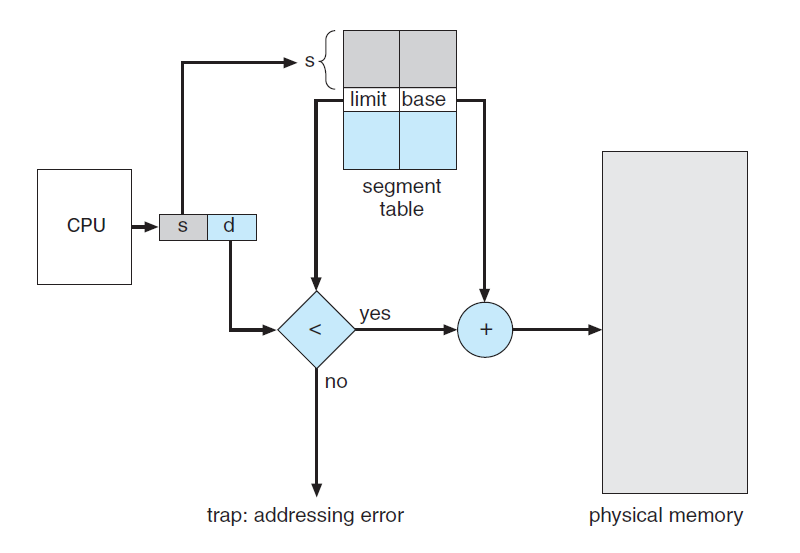
segments. The loader would take all these segments and assign them segment

numbers.



Figure

Although the programmer can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one dimensional sequence of bytes. Thus, we must define an implementation to map two-dimensional user-defined addresses into one-dimensional physical addresses. This mapping is effected by a segment table. Each entry in the segment table has a segment base and a segment limit. The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment. The use of a segment table is illustrated in Figure 5. A logical address consists of two parts: a segment number, s, and an offset into that segment, d. The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base–limit register pairs. As an example, consider the situation shown in Figure 8.9. We have five segments numbered from 0 through 4. The segments are stored in physical memory as shown. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit). For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.



Figure

1. **Disk scheduling algorithm**

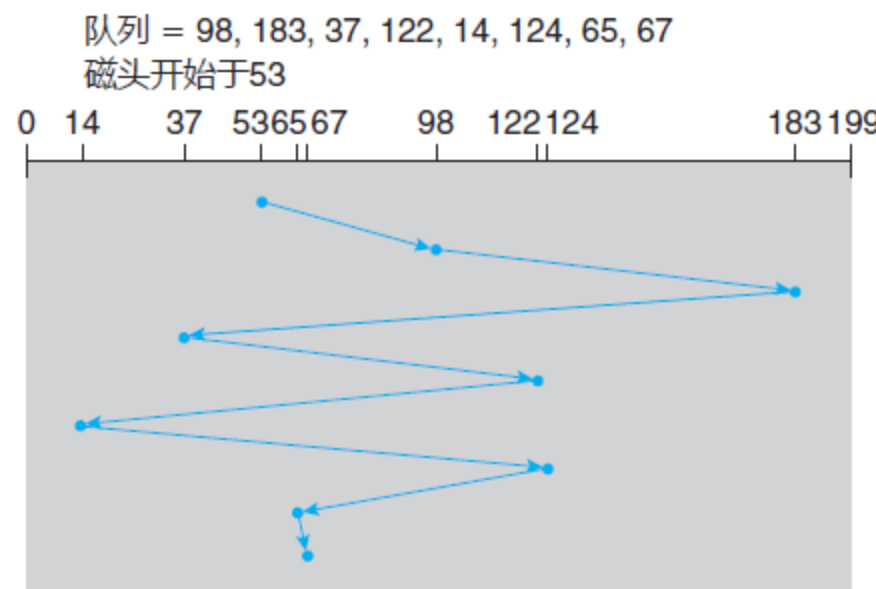
**①FCFS**

The simplest form of disk scheduling is of course the first come first served (FCFS) algorithm. Although this algorithm is relatively fair, it usually does not provide the fastest service. For example, consider a disk queue whose I/O request block cylinder sequence is as follows:

98,183,37,122,14,124,65,67

If the head starts at cylinder 53, then it first moves from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67. The total number of cylinders moved by the head is 640. This kind of scheduling is shown in Figure 6.

The big swing from 122 to 14 to 124 illustrates the problem of this kind of scheduling. If the requests for cylinders 37 and 14 are processed together, whether before or after 122 and 124, the total head movement will be greatly reduced, and the performance will be improved as a result.



Figure

**②SSTF**

It may be reasonable to process all requests close to the current head position before moving the head elsewhere to process other requests. This assumption is the basis of the shortest seek time first (SSTF) algorithm.

The SSTF algorithm selects and processes the request for the shortest seek time from the current head position. In other words, the SSTF selects the pending request closest to the head position.

For the request queue example above, the closest request to the starting head position (53) is on cylinder 65. Once at cylinder 65, the next most recent request is at cylinder 67. From there, since cylinder 37 is closer than 98, process 37 next time. In this way, requests located on cylinder 14 will be processed, followed by 98, 122, 124, and finally 183 (Figure 2).

SSTF disk scheduling

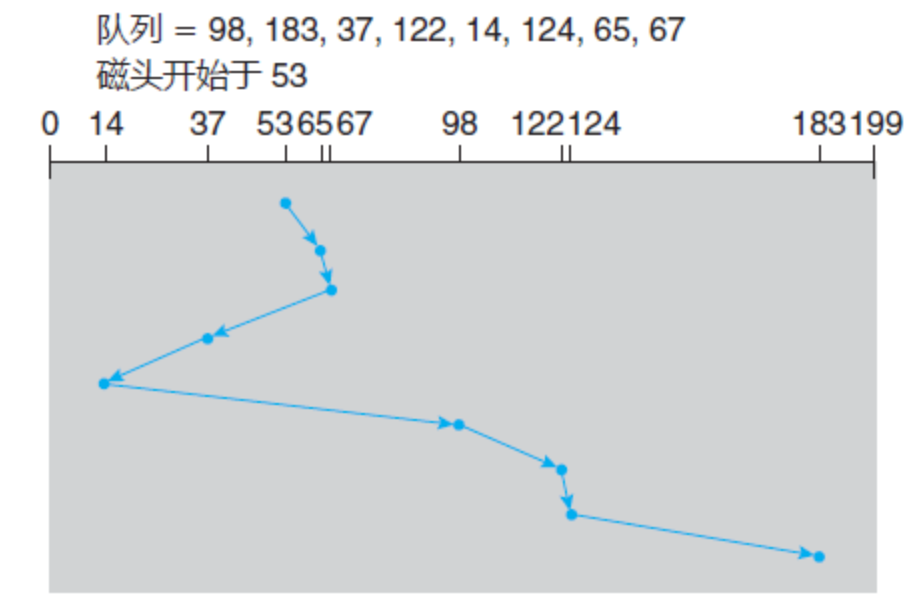
Figure 7 SSTF disk scheduling

The head movement of this scheduling algorithm is only 236 cylinders, which is a little more than one-third of the total number of head movements of the FCFS scheduling algorithm. Obviously, this algorithm greatly improves performance.

SSTF scheduling is essentially a shortest job first (SJF) scheduling; like SJF scheduling, it may cause starvation of some requests. Remember, requests may arrive at any time. Suppose there are two requests in the queue, respectively for cylinders 14 and 186, and when the request from 14 is processed, another request near 14 comes, this new request will be processed next time, so the request at 186 needs wait. When the request is processed, another request near 14 may arrive.

In theory, some requests that are close to each other will arrive continuously, so that the requests on 186 may never be served. This situation is likely to occur when the waiting queue for processing requests is long.

Although the SSTF algorithm is quite improved over the FCFS algorithm, it is not optimal. For this example, you can do better: move the head from 53 to 37 (although 37 is not the most recent), then to 14, and then to 65, 67, 98, 122, 124, 183. The total number of cylinders moved by the head of this strategy is 208.



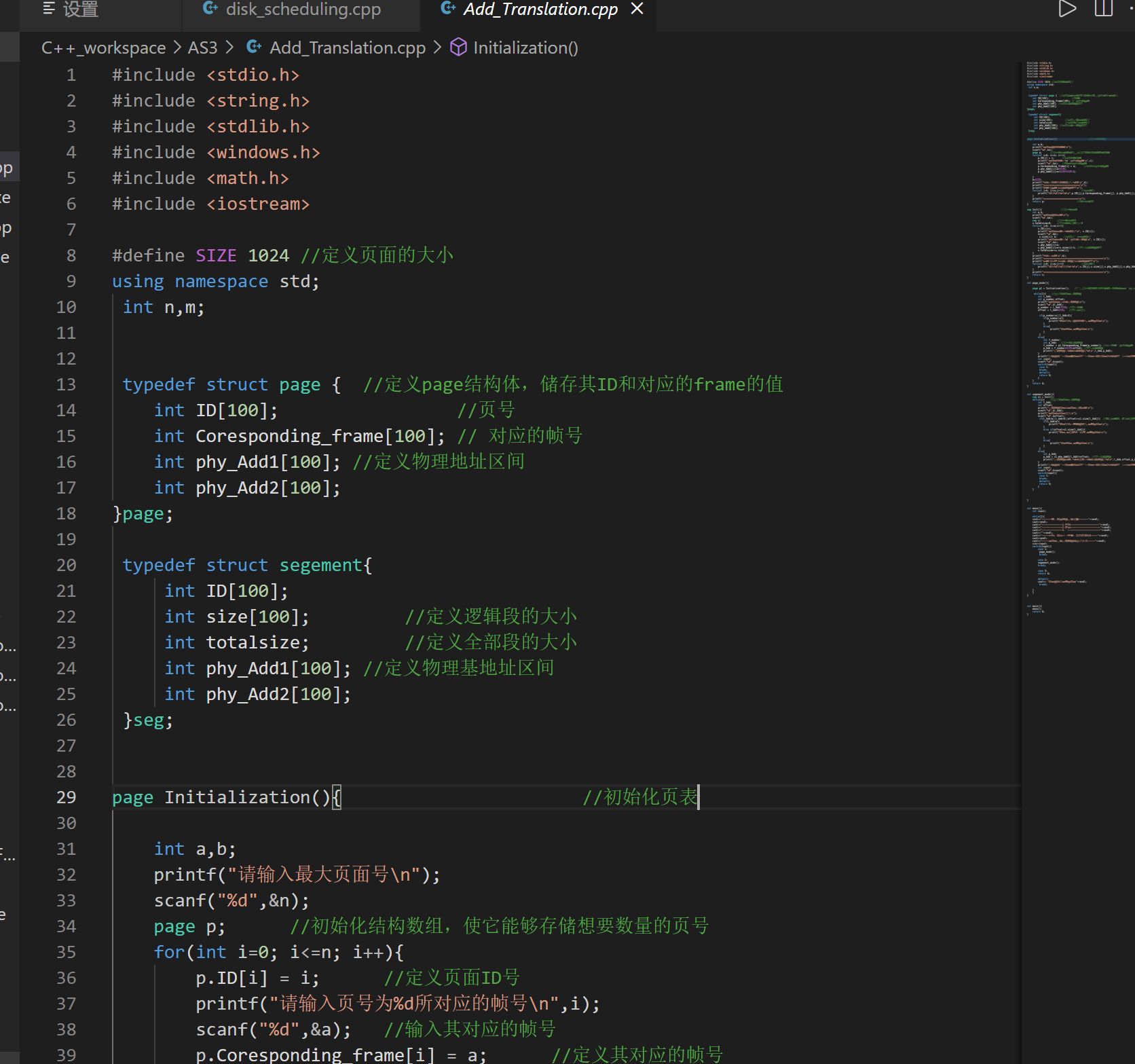
Figure

## Program results

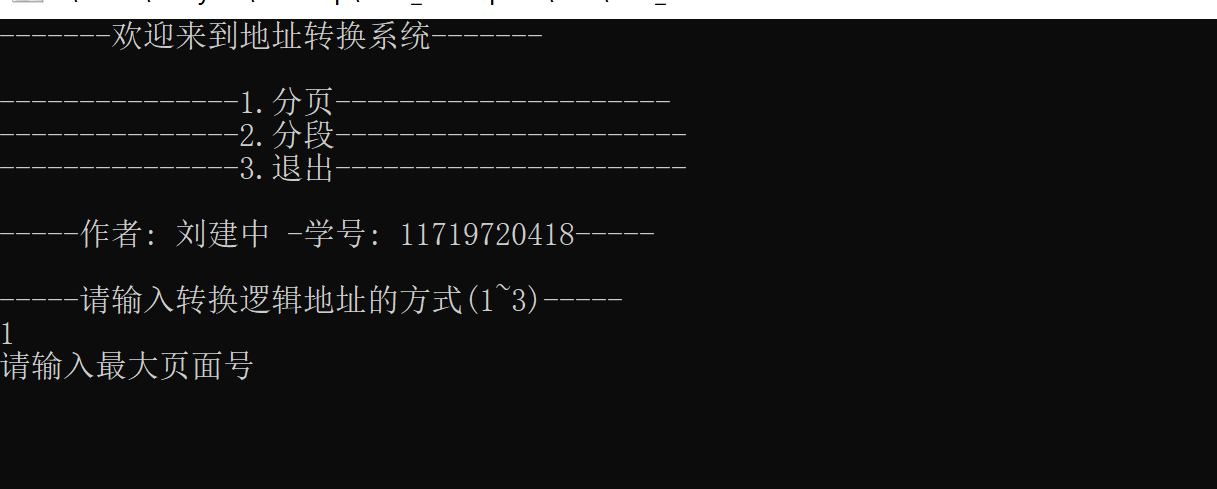
**(1).address translation(Page Mode)**

① Write the reference code on windows

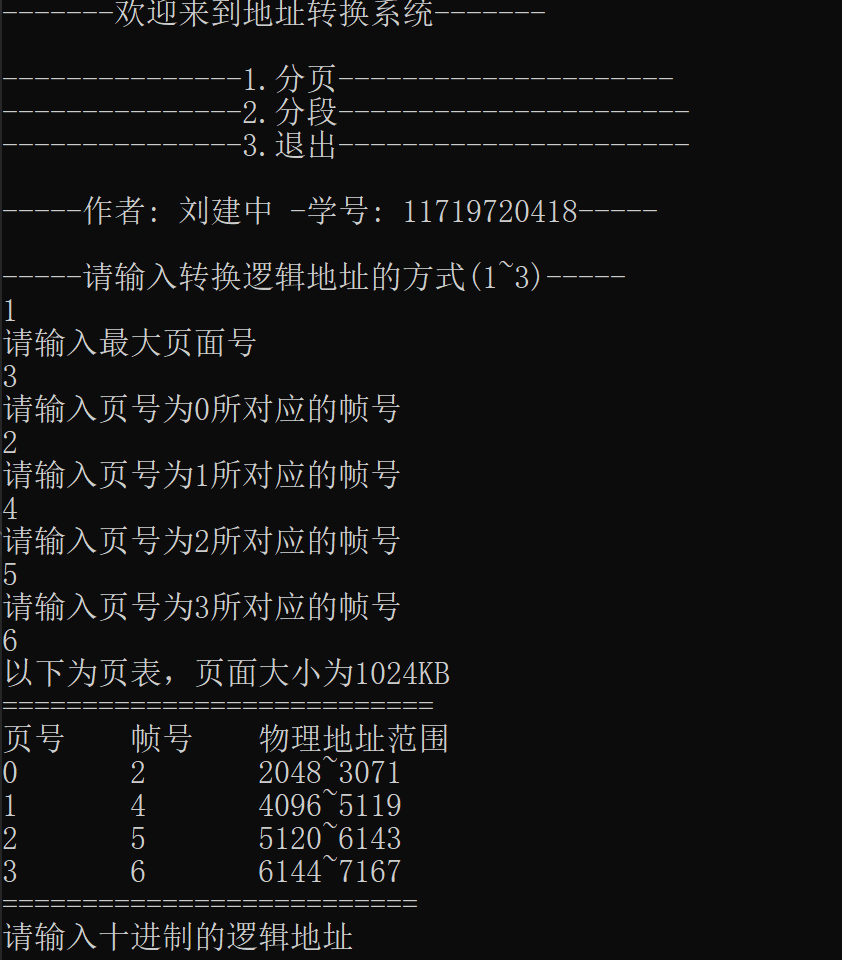
This is the code placed on Vscode



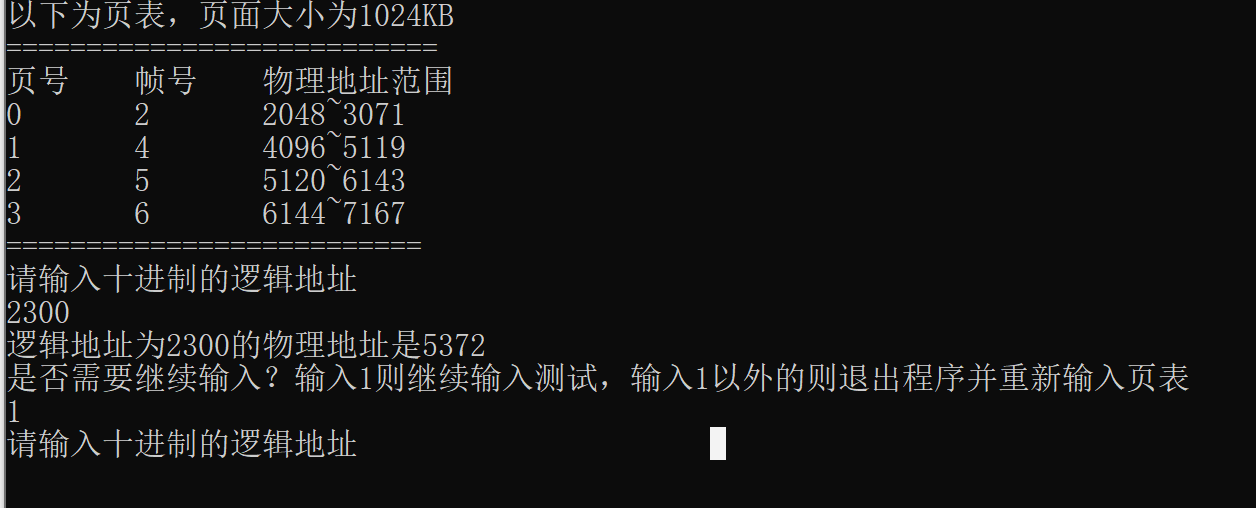
② Compile this program. We firstly test the page mode translation.



③The first required execution is that we should initialize the page table by inputting page numbers and their corresponding frame number. Following is the completed page table I input that can demonstrate the page number, frame number and the range of the physical address.

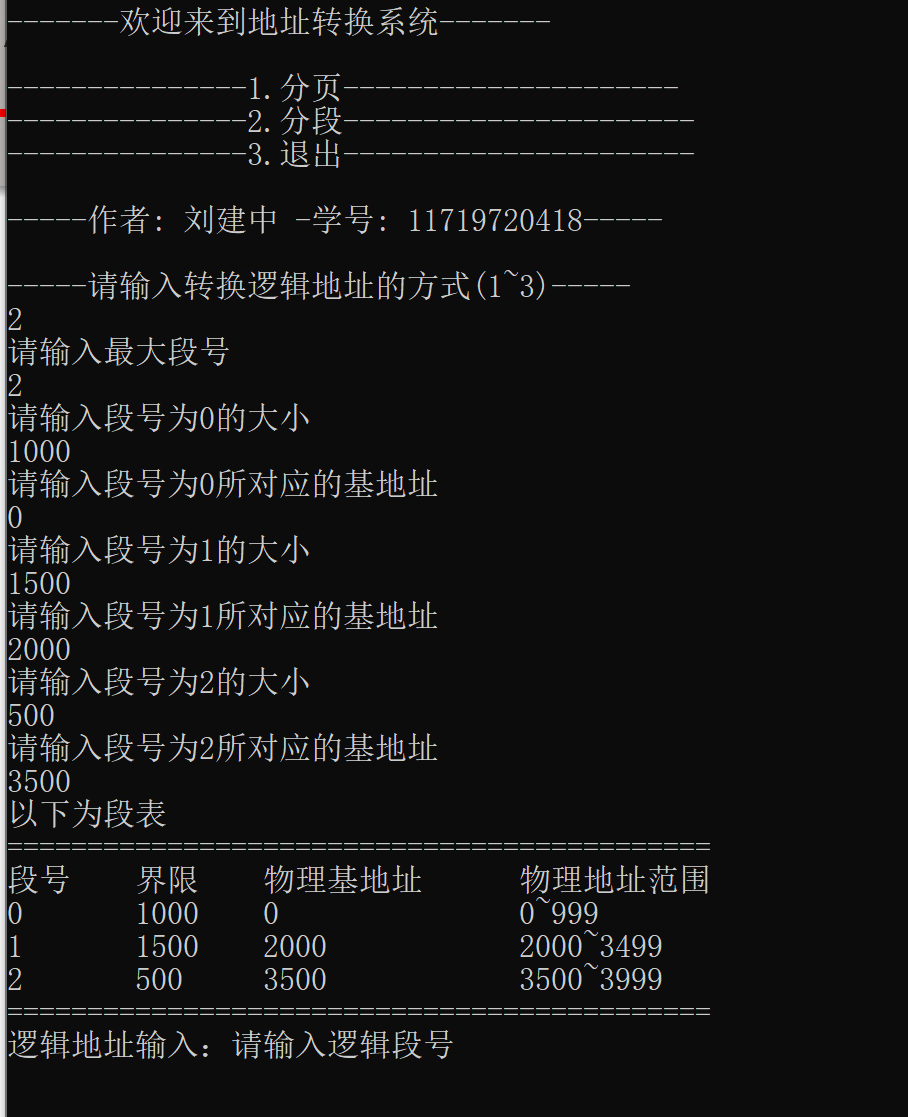


④ Finally, we are required to input a logical address. This case shows that the inputted address is 2300 whose page number should be 2 ,coz 2300/1024 = 2 with offset d = 2300%1024 = 252. Thus the physical address should be 5120 (since corresponding frame number is 5) +252 = 5372. Following is the screenshot of the result which proves that my simulation algorithm is correct. In this algorithm we can change the page table we expect, so try to test as many times as you can to familiar with the pattern of page mode translation.

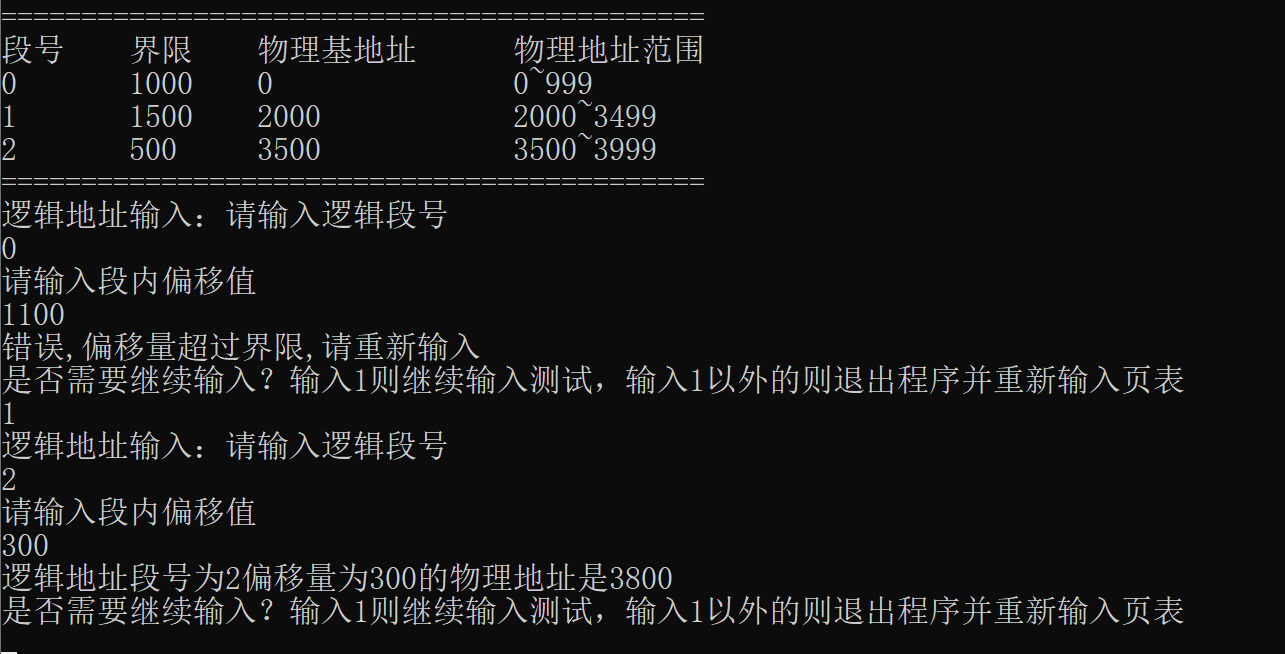


**2.address translation(Segment Mode)**

**①**Likewise, in segment mode, which is an another crucial method of memory management, we should also initialize the segment table including the number of segments, the size (Limit number) of segments you input, and their corresponding base address. Following is the picture illustrating the process of defining the segment table.



**②**Then, by contrast, we must input the segment number and its offset value separately so as to find the physical address. If the offset number exceed the limit value, the warning will be displayed. Following is the picture showing the whole process described above, including an error occurred.



③Okey, the result is what we expect, we input segment number of 2 whose base address is 3500 in segment table. Then the whole physical address should be 3500 + 300(offset value) =3800.

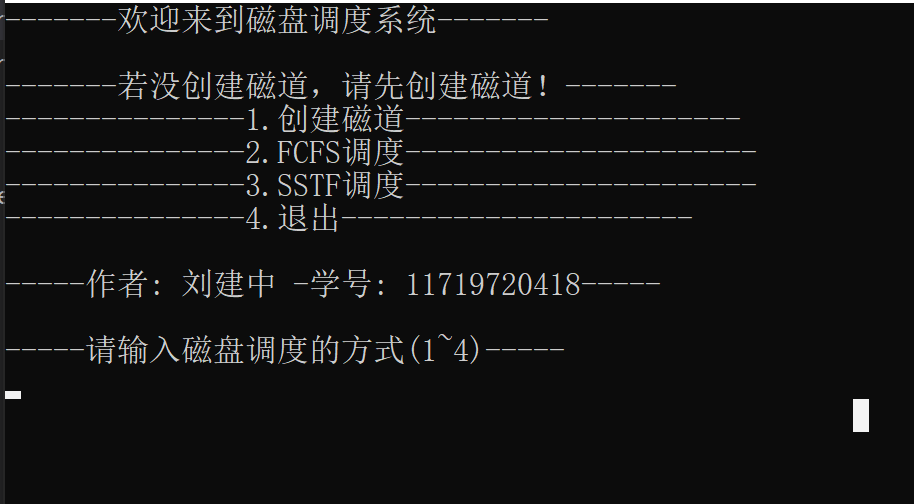
**3.Disk Scheduling(FCFS)**

* 1. Write the code in VScode

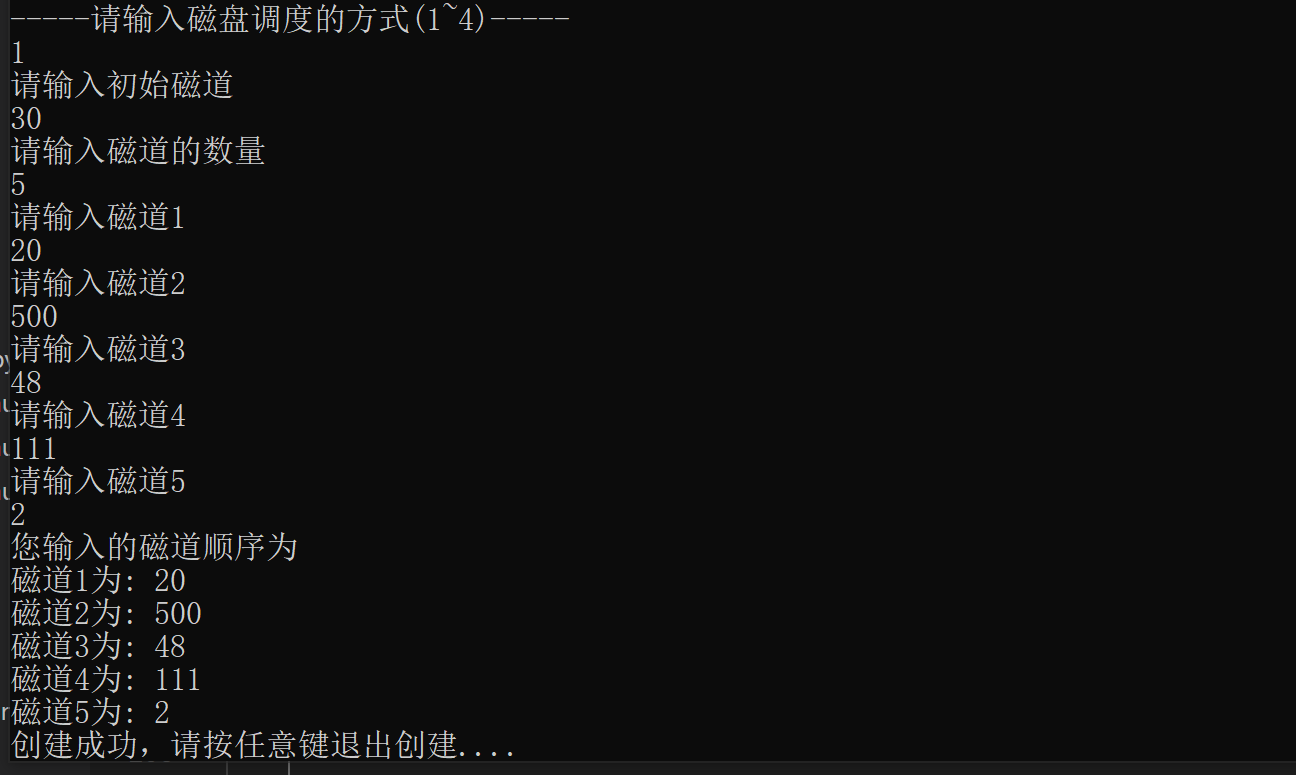


* 1. Run this program using debug

This is an interface, and please select the algorithm you want.



③ we must create the sequence before executing any scheduling method. Following is the process that I create a sequence of track. We should input the initial track pointer and the number of track we want before defining all tracks’ locations.

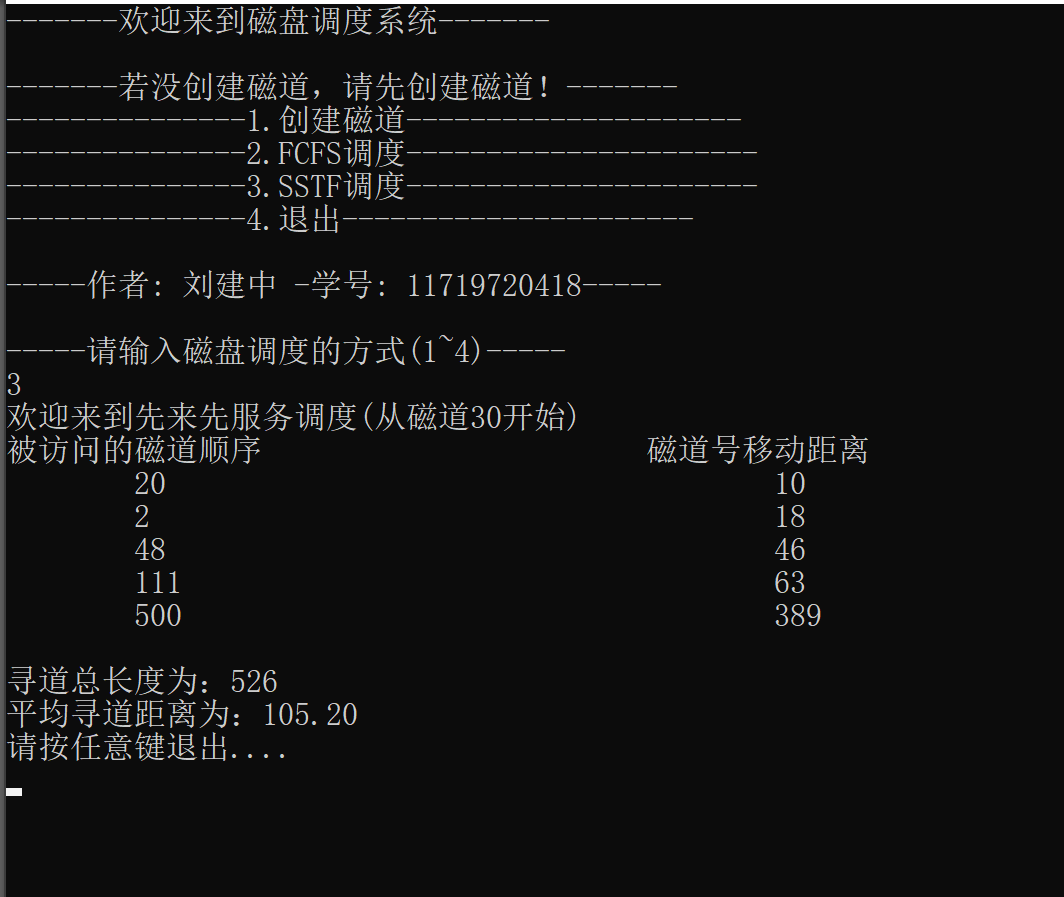


② Then we choose the scheduling(FCFS),the result will be displayed immediately. The interface can show the sequence of tacks according to the scheduling, as well as the length that the track has moved from the previous one. Finally we can correctly get the average seek length, which can be conduced by following： （10+480+452+63+109）/ 5=222.80



**4.Disk Scheduling (SSTF)**

**①**After quitting the previous algorithm, we can directly access the SSTF scheduling, since the created tracks have been saved in global array. Let’s the result a look, the algorithm can seek the shortest-length-track from the previous one. Thus this method can drastically decrease the average seek time comparing to that of FCFS.



## Conclusion (Learning and Thinking)

In this experiment, I simulated a lot of algorithms based on the knowledge of computer operating systems. Among them, the algorithm of paging and segment address conversion, the innovation I used is that it can tabulate according to the number of pages, paragraphs, and paragraph sizes entered by the user, and their corresponding base addresses and frame numbers. This kind of table can clearly show all the details to verify the accuracy of our algorithm, and to better understand the logical address and physical address conversion. In the disk scheduling algorithm, I deliberately designed the "create track" equation, which will be stored in global variables after it is created. In this way, users can directly access FCFS and SSTF to quickly compare the details of these two scheduling algorithms.

**Accessing my git hub link to view the Source code:**

https://github.com/512187207/AS3\_OS\_111719720418\_-\_CQUT.git