MP4 Report

t11902210 張一凡

Part 1: Briefly answer each problem. If your solutions are reasonable and correct, you can get points. Note that you can get points even if you fail to solve the problems. (10 points)

(a) Problem 1: Large Files (4 points)

Please draw your inode block structure and explain how to access 66666th block in your implementation.

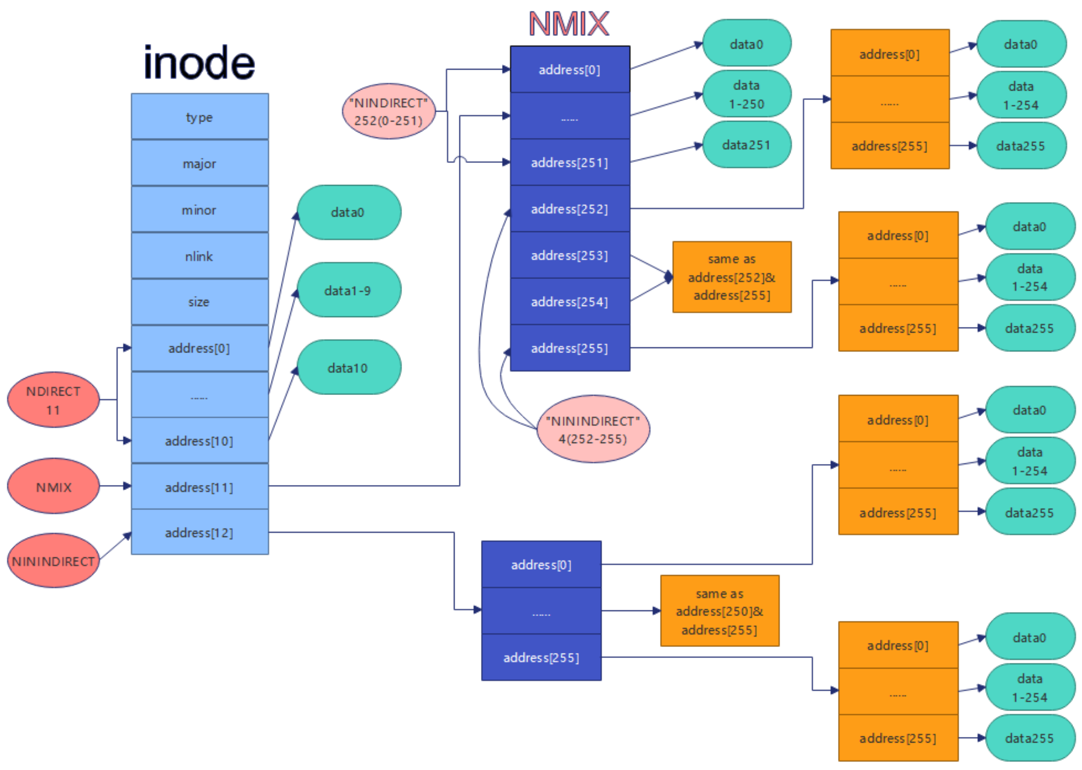
Answer:

Firstly, I will briefly introduce the inode block structure I designed here, and display the code and structure diagram mentioned in the title. The storage structure I designed includes secondary indirect addressing (including my newly created two structures) for accessing data blocks outside of the direct block address. The structure is as follows:

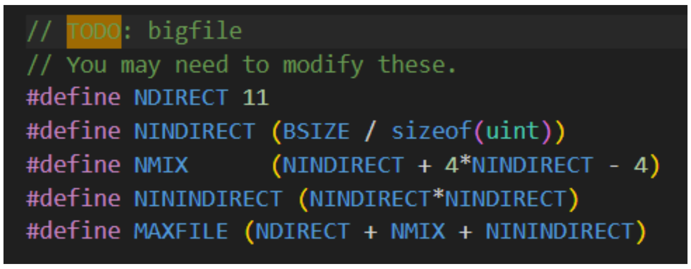
(1) Direct Block Address (NDIRECT): Due to providing two entries for indirect blocks, it has been reduced to 11 in this example

(2) Mixed Block Address (NMIX): 1 block, with an address calculation formula of NINDIRECT+4 * NINDIRECT-4 (where NINDIRECT=BSIZE/sizeof (uint)), referenced by ip ->addrs [NDRECT]. This part combines the ideas of single indirect blocks and double indirect blocks, hence it is called a mixed block

(3) Double Indirect Block Address (NINIDIRECT): 1 block, the address calculation formula is NINIDIRECT * NINDIRECT, referenced by ip ->addrs [NDIRECT+1]



The above is a schematic diagram that I designed, showing the structure of inode blocks, including direct blocks, mixed blocks, and dual indirect blocks. The direct block contains addresses that directly point to the data block. Mixed blocks and double indirect blocks store block addresses that point to more data block addresses, respectively. Next is my code structure design.



```
// TODO: bigfile
// You may need to modify these.
#define NDIRECT 11
#define NINDIRECT (BSIZE / sizeof(uint))
#define NMIX       (NINDIRECT + 4*NINDIRECT - 4)
#define NININDIRECT (NINDIRECT*NINDIRECT)
#define MAXFILE (NDIRECT + NMIX + NININDIRECT)
```

Next, I will describe in the text how to access 66666 data blocks. The previous NDIRECT blocks (11 blocks) can directly access their respective entries through direct block addresses. Therefore, blocks 0 to 10 can be accessed using ip ->addrs [0] to ip ->addrs [10]. The next level of direct blocks involves mixed blocks. This intermediate block stores the NINDDIRECT (256 in this case) block address. The addresses from the 0th block to the 251st block (totaling 252) in this continuous block all point to a set of address blocks that can be used to access the data block, which is the "direct" part of the mixed block; The remaining four blocks from 252 to 255 are designated as "indirect" by me, and their addresses each point to the block addresses of a new set of NINDIRECT indirect blocks, which in turn refer to the actual data blocks. Therefore, data blocks 11 to 1287 (11+256+4 * 256-4) can be accessed using ip ->addrs [NDIRECT] (mixed block) and appropriate addresses in the mixed block. Next is the dual indirect block, which allows for two levels of indirect addressing. The double indirect block is referenced by ip ->addrs [NDRECT+1] and stores the block address pointing to the second indirect block. Each auxiliary indirect block also stores NINDRIRECT block addresses, which refer to the actual data blocks and are processed in the same way as the last four blocks in the mixed block. Therefore, blocks 1287 (11+256+4 * 256-4) to 66823 (11+256+4 * 256-4+256 * 256) can be accessed using ip ->addrs [NDIRECT+1] (dual indirect block), appropriate auxiliary indirect block addresses, and addresses in auxiliary indirect blocks. At this point, the maximum value has exceeded 66666, completing the final task. In fact, in my initial plan, I originally used triple indirect blocks or two double indirect blocks to complete the task, which was easier to achieve the standard of 66666. However, I later believed that these two methods wasted space. In order to improve efficiency, I decided to use a combination of mixed blocks and double indirect blocks to solve the problem.

In the specific implementation of bmap code, we need to sequentially check which block bn belongs to (especially the processing in the mixed block), and then locate the mapping. Specifically, when storing the corresponding blocks in memory, it is necessary to first calculate the blocks that have already been sealed by bn. If these blocks are not allocated, then they also need to be allocated. Finally, when finding the blocks of bn, it is necessary to check if they have been allocated. If not, they also need to be assigned. Finally, the address of the block is returned as a function to return the result. The itruc code that also needs to be completed also needs to traverse all the involved blocks and release them one by one in a hierarchical order from low to high.

(b) Problem 2: Symbolic Links to Files (3 points)

Function symlink(”/b”, ”/a”) creates a symbolic link /a links to file /b, explain how to open symbolic link /a.

Answer:

When calling symlink ("/b", "/a") to create a symbolic link/a to point to a file/b, the following is the explanation steps for opening a symbolic link/a. I summarize them as steps and show them in the following table. It involves the code already provided in the question and the code we need to implement (specifically in step 2, I set font emphasis, which is the core of answering this question).

Step	Interpretation and Functions
1	Firstly, call the beginning of the sys_open function to open the file/a, passing the file path "/a" and open mode as parameters. In the sys_open function, in the sys_open function, we first obtain the path/a of the file to be opened and the opening mode from the user mode. If the open mode contains O_The CREATE flag indicates that if a file needs to be created, use the create function to create a file of type T_File inode of FILE and associate it with the path/a.
2	**Secondly, check if the open mode contains O_NOFOLLOW flag to determine whether to track symbolic link. Without this flag, we enter a loop, which is the main part we need to execute. The following steps need to be performed: (1) Use the namei function to find the inode corresponding to the path/a. If the type of inode found is T_SYMLINK, which means that/a is a symbolic link, then proceed to the next step, otherwise it means that the target exits directly due to error (2) We read the target path from the symbolic link inode through the break function and replace it with a new path (3) Repeat the above steps until the target file inode of the non symbolic link is obtained, that is, the type of inode directly found is no longer T_SYMLINK. However, if a circular symbolic link is encountered in the parsing process, or the depth of the loop exceeds the limit (to prevent the formation of a closed loop infinite loop), an error code will be returned and the function will exit directly. This part is what we need to implement.**
3	Once we have found the final inode, we call the namei function again to obtain the inode corresponding to path/a, which is the target file we are looking for/b. We will continue the open operation and set the corresponding file type and read/write flags based on the inode type and open mode. If the symbolic link/a points to a device file/b, we will set the file type to FD_Device and store the main device number in the main field of the file structure. If the symbolic link/a points to a normal file/b, we will set the file type to FD_INODE and initialize the file offset to 0. This also includes various forms of document judgment and processing.
4	Finally, we will check if the open mode includes O_TRUNC flag, and whether the type of the target file is T_FILE. If so, the itrunc function will be called to truncate the file to 0 bytes, which is also the part we implemented in the first

	question.
5	At the end of the function, you need to unlock the target file inode (iunlock (ip)) and return the file descriptor as the result of the end operation.

To sum up, the process of opening the symbolic link/a and accessing the file/b includes two steps: first, track the symbolic link/a until the final inode is resolved to obtain the target path/b, and then parse the inode of the target file/b again according to the target path/b to facilitate subsequent operations on the target file. Based on the problem set of this question, I will also sort out the functions. In the process of sys_symlink generating a symbolic link, you must first create a file. During the process of creating files, it is also necessary to check whether there are other files or other illegal operations in the path; If an error occurs at this time, returning -1 indicates an error; If not, the corresponding information is written to the inode through the writei function, which is the opposite of the open function's break implementation process, but the other logical judgment process and program execution process are similar.

(c) Problem 3: Symbolic Link to Directories (3 points)

Function symlink(”/y/”, ”/x/a/”) creates a symbolic link /x/a links to directory /y/, explain how to write to /x/a/b.

Answer:

In the previous question, we explained how to generate symbolic link and open files through symbolic link. This problem will involve how to program/x/a/b when calling the symlink function to create a symbolic link/x/a and link it to the directory/y/. The key to answering this question is the sys_chdir and namex functions, which I emphasized in steps 2 and 3 respectively. However, the main focus is on the namex function, as there were not many substantive modifications made to the sys_chdir function in my implementation. The font is also highlighted in bold during these two instructions.

Step	Interpretation and Functions
1	Firstly, according to the premise requirements in the question, we use the second question to complete the use of sys_symlink function creates a symbolic link, links the directory/y/to the path/x/a/, and generates a symbolic link/x/a, which paves the way for answering this question; Note that all paths in this are absolute paths, so a complete expression is required.
2	**The second step is to explain the sys_chdir function, which is mainly implemented because the symbol directory can be a part of the path, so it needs to be redirected to the content it links to. The logical form and implementation of this function are similar to sys_symlink: in the sys_chdir function, we call the namei function to find the inode corresponding to path/x/a to enter the target index node. Secondly, it parses and checks the index nodes. If path resolution fails or the target directory does not exist, it will exit directly. After success, it will unlock the index nodes of the target directory to allow other processes to access the directory. Then, we unlock the target inode and set the current working directory of the current process to that inode to achieve directory switching. In this way, we have switched the current working directory to the/y/directory. In**

	summary, the sys_chdir function is responsible for changing the working directory of the current process to the specified path and performing necessary error checks and operations. I did not make any substantive modifications in this function, just read and use it.
3	**The third step is mainly related to the implementation of namex. According to the implementation requirements of the symbol directory in the title, the actual path of/x/a/b should be/y/b. Therefore, if you write/x/a/b, it is actually writing/y/b, which is also a function of namex. At the beginning, we passed the path "/x/a" as a parameter to the namex function and set the nameiparent parameter to 0. This will return the inode of the directory "/x/a", or if the path is a symbolic link, return the inode of the final resolved target directory. This is achieved through a loop with a structure similar to open. Specifically, check whether the obtained inode is a symbolic link (T_SYMLINK). If it is a symbolic link, you need to parse the symbolic link to obtain the actual target directory. If the obtained inode is a symbolic link, use the break function to read data from the first block of the symbolic link inode (ip ->addrs [0]). These data contain the path of the actual target directory, which is to append "/b" to the obtained target directory path. In this case, the target directory path is "/y/", so the new path will be "/y/b. In this process, we will repeatedly parse the target path, which is achieved through the string processing function. At the same time, when the loop reaches the maximum recursion depth, it will automatically exit to prevent infinity. Therefore, we will parse the final target path/y/b. Once we have parsed the final inode, we will determine whether the type of inode is Directory (TDIR). Since the a in/x/a/b is a symbolic link and the target path after resolution is/y/b, we need to ensure that/y/b is a directory. This is the implementation method and purpose of it in namex, which is also the main part of this topic in my idea.**
4	Finally, if/y/b is a directory, we pass the new path ("/y/b") as a parameter to obtain the inode of the file or directory "/y/b" (while setting the nameiparent parameter to 0). After locating/y and determining/y/b in the above two steps, the subsequent data processing for writing is also output by the given code and testing program.

To sum up, to write data to/x/a/b, we need to switch to/y/directory through symbolic link/x/a (it is important to determine whether the given working path is a file or a symbolic link before this), and then write b in the current working directory/y/.

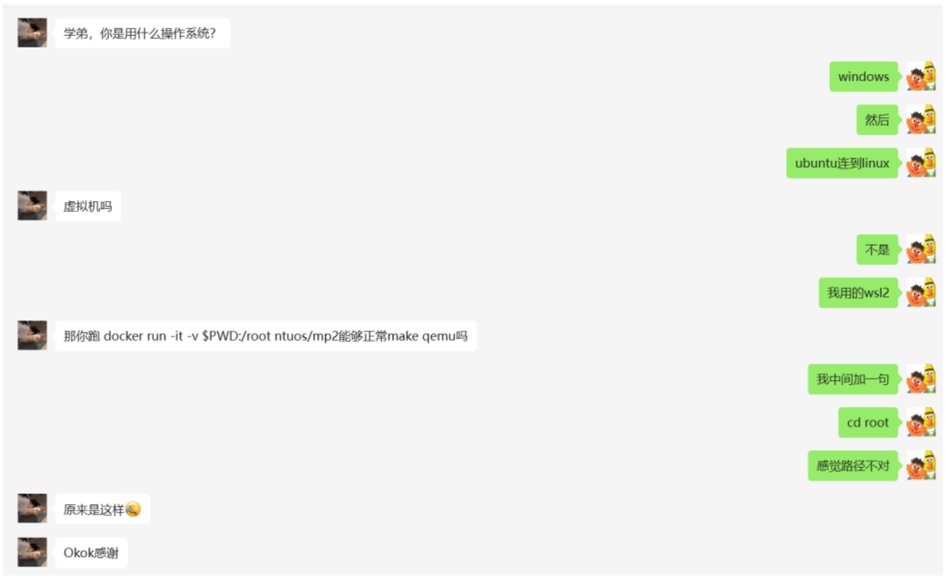
(I have also used this symbolic link application many times, especially in the csie workstation. Since the given home space is only 2GB, it is necessary to establish a symbolic link in the personal folder of tmp2, and place the downloaded files required by the configuration environment in tmp2 through the symbolic link to prevent the home from displaying that the disk is full. This is the application of this homework function in my daily study and life.)

Part 2:

We encouraged you to help other students. Please describe how you helped other students here. You should make the descriptions as short as possible, but you should also make them as concrete as possible (e.g., you can screenshot how you answered other students' questions on NTU COOL). Please note that you will not get any penalty if you leave it empty here. Please also note that this bonus is not for you to do optimization, so we will not release the grading criteria and the grades. Regarding the final letter grades, it is very likely that this does not help — you will get promoted to the next level only if you are near the boundary of levels and you have significant contributions. (0 points)

我是 t11902210 張一凡，是本學期來台大資工系的交換生，為期一個學期（本人來自於上海市同濟大學數據科學與大數據科技專業大學部三年級），選擇了林老師教授的作業系統這一門課。 感謝助教老師們和林老師的講授和作業說明講解。 對於工作中的幫助，我主要是與一比特研究生學長（r10922161 黃正輝）進行互幫互助，具體的線上聊天解答問題過程見下方截圖（右側是我）。其中包含了工作方面和考試複習時的問題，例如環境配寘、工作中一些過不去的 bug，一些需要使用到的工具函數的解釋。

mp2 的 docker 環境配寘



学弟，你是用什么操作系统?

windows

然后

ubuntu连到linux

虚拟机吗

不是

我用的wsl2

那你跑 docker run -it -v $PWD:/root ntuos/mp2能够正常make qemu吗

我中间加一句

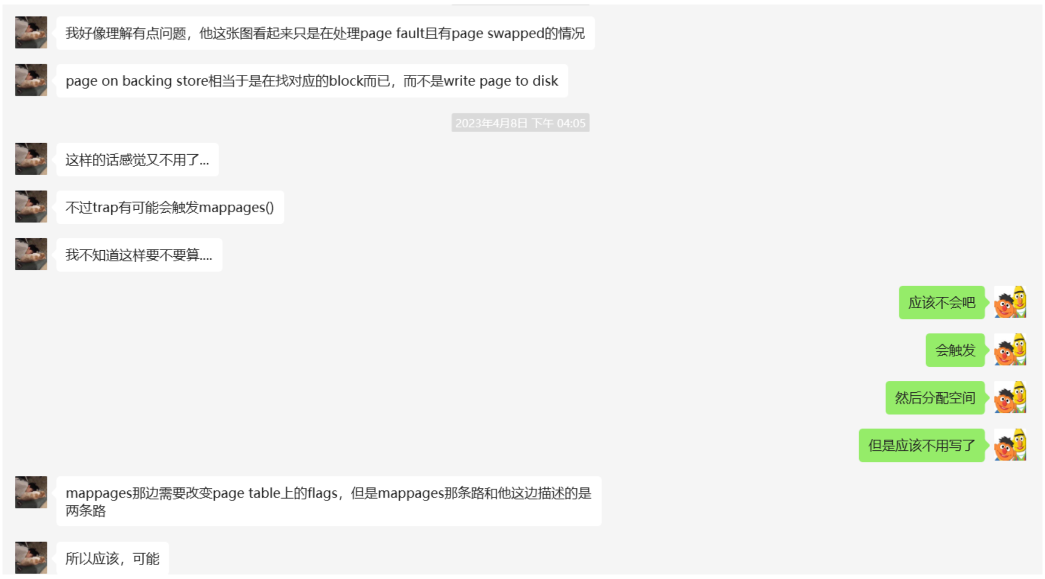
cd root

感觉路径不对

原来是这样

Okok感谢

mp2 的報告部分處理



我好像理解有点问题，他这张图看起来只是在处理page fault且有page swapped的情况

page on backing store相当于是在找对应的block而已，而不是write page to disk

2023年4月8日 下午 04:05

这样的话感觉又不用了...

不过trap有可能会触发mappages()

我不知道这样要不要算....

应该不会吧

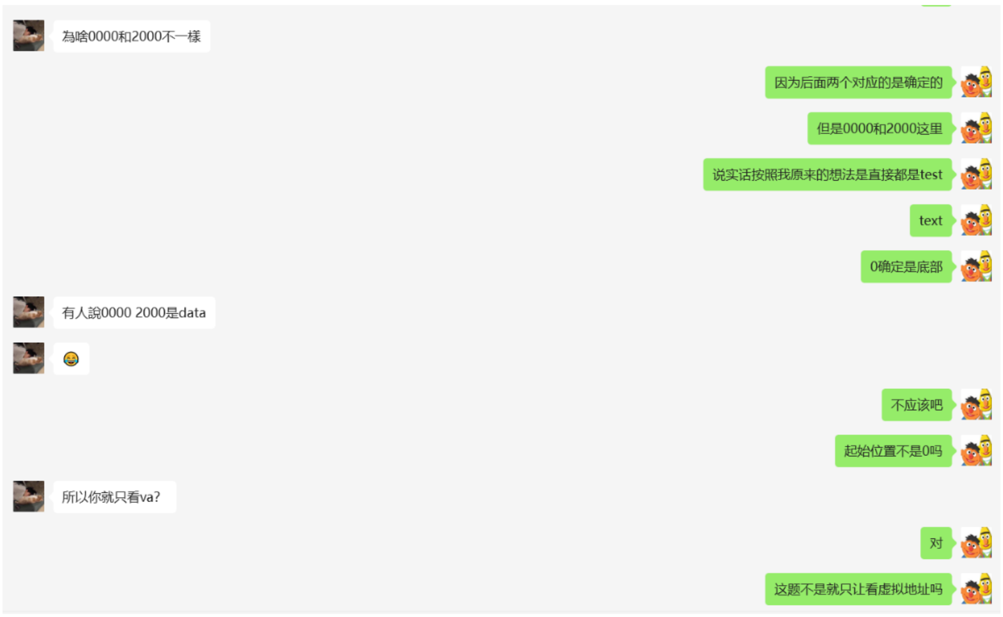
会触发

然后分配空间

但是应该不用写了

mappages那边需要改变page table上的flags，但是mappages那条路和他这边描述的是两条路

所以应该，可能



為啥0000和2000不一樣

因为后面两个对应的是确定的

但是0000和2000这里

说实话按照我原来的想法是直接都是test

text

0确定是底部

有人說0000 2000是data

😂

不应该吧

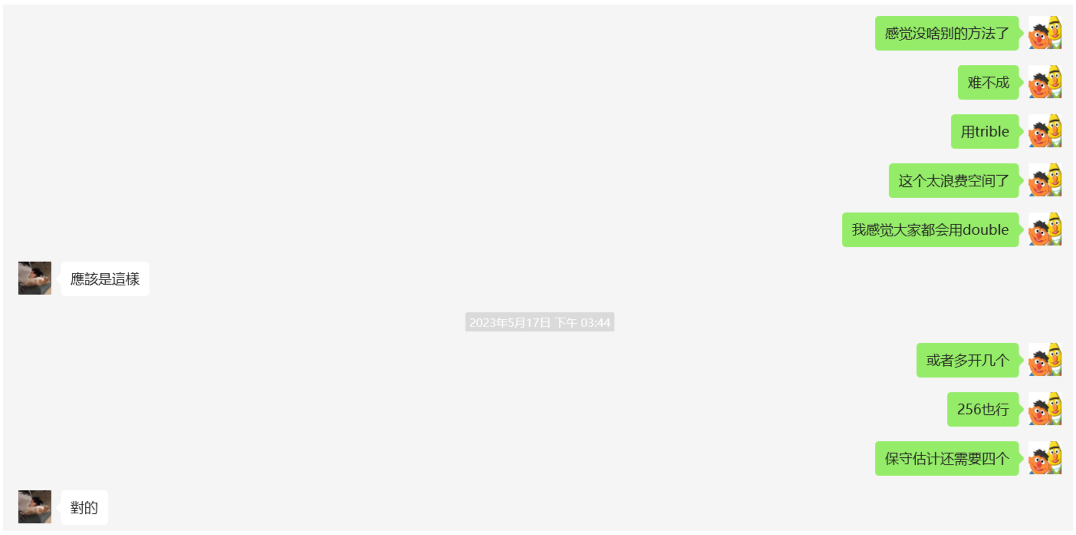
起始位置不是0吗

所以你就只看va?

对

这题不是就只让看虚拟地址吗

Mp4 想法的簡單討論



感觉没啥别的方法了

难不成

用trible

这个太浪费空间了

我感觉大家都会用double

應該是這樣

2023年5月17日 下午 03:44

或者多开几个

256也行

保守估计还需要四个

對的

其他的討論還有很多，總的來說相對於我幫助學長，學長幫助我的地方更多（畢竟學習

的知識和理解工作的能力强過我)。 其餘的我都是在討論區看到大家的問題從而對自己的想法加以修改，感謝一個學期各個學長以及助教老師和林老師對我這個交換生的幫助!

　　再次表達由衷的感謝!