CS2106 Introduction to OS

Office Hours, Week 12

Agenda

- Memory Management
  - Page Replacement Techniques
  - Frame Allocation
- File Systems
- Miscellaneous

- A few questions remained unanswered
  - Will cover them during revision

LRU Implementation (Page Replacement)

In approach A to implement LRU, we use a counter to track the last access time. Does this counter store the time when this page was last accessed (i.e., last accessed 10secs ago)? Or we store the number of times it was accessed (i.e., accessed 10 times)?

- Excellent questions!
- Generally, timestamps would be preferred.
  - That would give the OS the possibility to precisely rank all pages in the system by recency
- Counting the Accesses is the basis of another algorithm, called **LFU**
  - ***L****east* ***F****requently* ***U****sed*: the victim page is the one with the smallest access counter.
- Regardless of we would like to record timestamps or count the accesses, you would need hardware support.
  - Remember, ***no OS involvement in regular memory accesses!***
  - However, today's hardware does not provide such support. And even if it did, the OS would have hard time implementing precise LRU/LFU!

LRU Implementation (Page Replacement)

Follow-up questions (interesting for the final exam):

- What’s the hardware support needed for precise LRU?
- How can hardware communicate recency information to OS?
- Which component should be modified to support LRU?

Second Chance

For second chance page replacement, in step 3 of the algorithm, we reset the arrival time of the page. Does this mean that the page's position in the FIFO queue is changed as well? Do we have to move the pages around in the FIFO queue for this algo?

- No.
- All we do is clear the "access bit" in the page table, set by the hardware
- This will indicate that the page has not been accessed since the time of clearing the bit

Thrashing and Page Faults

- **Is thrashing related to page faults? Does it mean that if the process is constantly page faulting, then it is thrashing?**
- Yes, thrashing is a consequence of very high page-fault rate
- If the process is page-faulting enough to significantly degrade its performance, we say it's thrashing.
  - No clear definition
  - In some applications, the minimum memory capacity below which the process will thrash is well-defined (e.g., as soon as you don't capture the working set)
  - Other applications will see a more gradual performance degradation

Thrashing

You mentioned that to reduce thrashing, it's better to limit the number of processes. How can the OS limit this?

- The OS can easily count per-process (and total) page-faults per unit of time
- If the page-faults exceed some acceptable limit, the OS can refuse to start a new process
  - It’s conceptually easy to do
  - But I’ve never run into such a situation in modern Oses

Paging and I/O

Is accessing the disk to bring a memory page to the physical memory considered an I/O operation? If so, why is it an I/O operation?

- Disk is an I/O device
- Any device other than CPU and memory are I/O devices
- They are slow and rarely accessed compared to CPU/memory
  - → We can afford OS involvement
  - And be notified upon completion of I/O operation through interrupt
  - This relieves the CPU of having to waste time on slow devices

Non-volatility and Security

Does the secondary storage reside on disk which is non-volatile?

- Yes

If there is a sudden power outage, the OS gets shut down immediately and doesn't have time to clear the memory context of processes in secondary storage. Is this a security breach?

- Yes
- A simple solution is to encrypt the swap partition
  - or the entire file system

File System Requirements

Under the general criteria for file systems, could you explain self-contained again? I don't understand the part on "Information stored on a media is enough to describe the entire organization"

- Self-contained means that the file system should contain everything needed to read and operate on the data
  - This includes the data and any organizational/administrative metadata
  - E.g., there shouldn't be a case where your data is in the file system, but the instructions to access the data are elsewhere
  - Note that without self-containment you cannot have portability

Storage Media / Physical Media

There was mention of "storage media/physical media". What does this refer to? Does it refer to the hard disk? Or does it refer to files that processes can access/write/read/execute?

- Storage medium is a physical device that contains some storage cells
  - each cell stores 0 or 1, although multi-bit cells are possible
- Could be hard disk, or SSD, or something else
  - We assume hard disk in this module
  - (It’s like DRAM for memory)
- The job of the file system is map the user files into storage cells in a way that is efficiently retrievable, removable, updateable, and searchable, and protected from unauthorized access

File Extensions

Do you mind reexplaining the conversion of file extension again? What will happen if we convert doc to pdf in Linux vs Windows? How does the file opener (adobe reader for eg) handles this?

- Assume you have .docx file on windows
- If you try to rename to .pdf, Windows will reject
  - Because the type
  - If you manage to change the extension, then both Adobe and Word would complain if you try to open such a file
- In Linux, the extension is there only for user convenience
- If you have a .pdf in Linux, you can do anything you want with the extension (make it .docx, .jpg, .docx.pdf.jpg, …) and it will work

File Access Types

What is the difference between Random Access and Direct Access? Both are able to access data randomly…

- They are essentially the same thing.
- The key difference is that having a random access is much more powerful when the record is of fixed-size:
  - E.g., like a row in a database
  - You can directly access N^th^ row/record if you have random access
    - Similar to how we access the N^th^ entry in a page table
  - The random access is less useful if you have variable size records
    - You can't directly access the N^th^ record

Seek() vs. Read()

For random access of file data, there is seek() and read().

- Correct

Is it true that seek on its own does not actually read the data yet? Seek will only move to the right location without reading?

- Correct

So when Unix uses seek(), it has to perform read() as well to retrieve the data?

- Correct!

Seek() vs. Read()

For Random Access, does the seek(offset) operation move the cursor to that offset?

- Correct

If so, when we call read(offset), we don't need the parameter offset right?

- Correct!
- The cursor is already there and you don’t need the offset
- You DO need to indicate the number of bytes you want to read (regardless of whether you do seek() first or not)

File Records

Could you further explain the concept of records with regards to common file types? For C files, what are records inside these source code files?

- C file is a text file, each record is a character
- Consider a video file
  - Each record is a compressed frame (simplified)
  - They are of variable length depending on the compression rate of individual frames
- A table in an SQL database would be stored in a file with fixed records
  - Unless there is a variable-length string as an attribute
  - Each record is a row in the table

Also, what are file descriptors?

- File descriptors are “pointers” to files within the OS (recall the open file table)
- They are not related to the structure of the file

File Tables

Could you further explain Per-Process open file table? Is there one table per process or one table shared by all processes? When do we create an entry in this table and why does each entry point to a system wide table?

- Per-process table: one per process
  - Makes it easy to track open files of every process
  - Entry created:
  - 1) when you open a file 2) when you fork, it’s created in the child
- System-wide table: one per system
  - Entry created only when you open a file
  - Avoids duplication of certain entries

File Tables

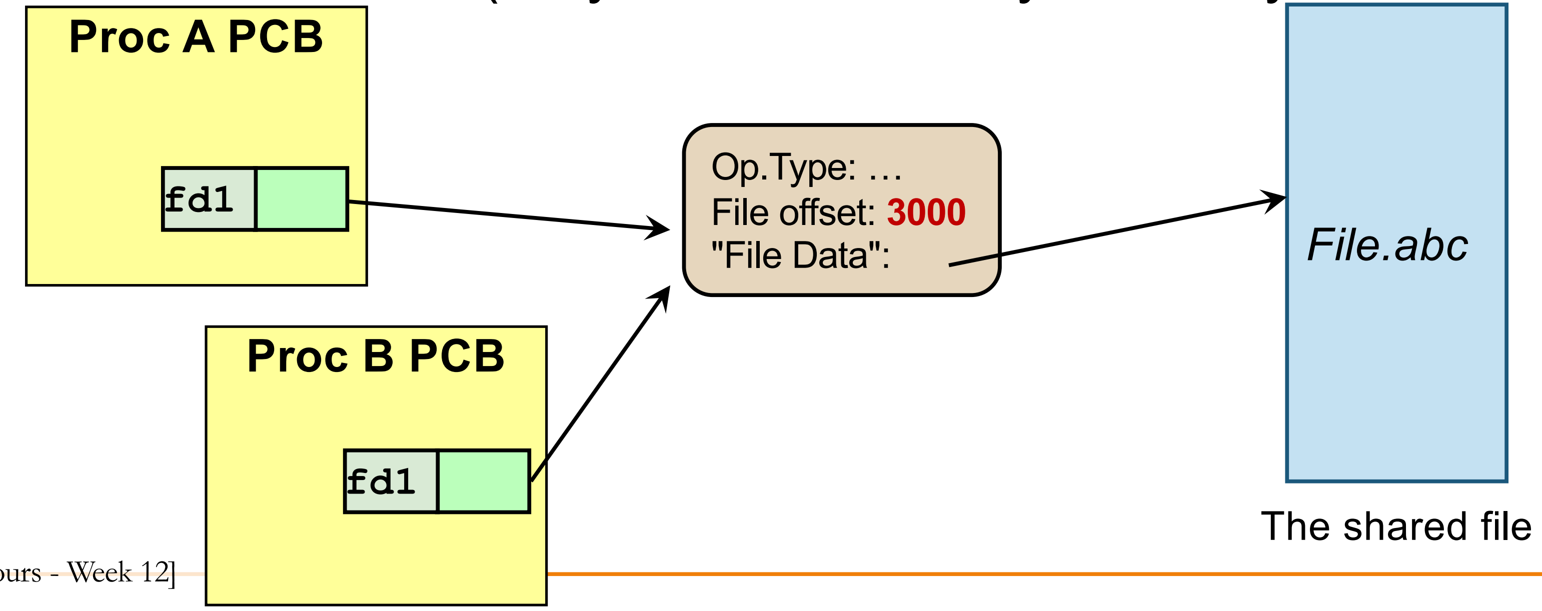
In Slide 29, you mentioned that there's 2 different entries in the system wide table. But in slide 27, it says there's only 1 entry per unique file.

- That was an error in the slide.
- The system-wide table in Linux has one entry per opening of a file.

File Tables

In slide30, there was mention of "only one offset->I/O changes the offset for the other process". What does this mean exactly and why does the I/O change the offset for the other process?

- Assume cursor is at the beginning, and the records are X, Y, Z
- If proc A reads one record, it will read X
- However, if B afterwards reads one record, it will read Y, because proc A advanced the cursor (they share the entry in the system-wide table)



File Tables

When the same process opens the same file twice, are there 2 entries in the process open-file table and each pointing to 2 entries in the system-wide table?

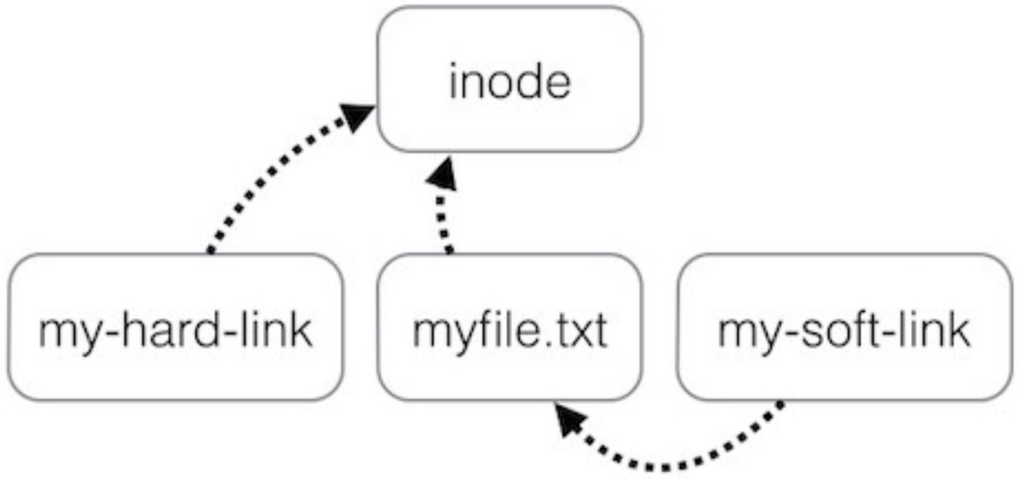
- Correct
- There will be two entries in both per-process and system-wide tables
- Check `man open` for more

'.' and '..'

Are the '.' and '..' when we do `ls -al` considered directories or files or special files?

- Directories:
- ‘.’ the current directory
- ‘..’ the parent directory

Hard Link vs. Soft Link



- **Hard link** points to the actual *i-node*
- i-node is the final pointer to file data
    - we will learn about them tomorrow
- As such, hard link to a file shares the same permission as a file
- If you delete one hard link, you can access the data via another
- There must be at least one hard link to a file

- **Soft links** point to a different i-node
    - That i-node is a simple pointer to the target location

Tut9 Q2

Can you go through modern hierarchical page table in more detail? Or go through Tut9 Q2

- The question is too general
  - If you have more specific questions about it, please ask
- Please check the W11 office hours
- Also, check the T9 recordings

Thank you!