$ man lsck
$ man read
$ man write
$ man open

file offset:

view file as a stream of bytes

struct, {
  // i-node.
}

};

myinode_type Sample_inode;
from user

\[ \text{fd} = \text{open}(\text{filename}, 2); \ // \text{read \& write} \]

\[ \text{llseek}(\text{fd}, 4096, 0); \ // \text{0 = beginning?} \]

\[ \text{if} (\text{read}(\text{fd}, , 128) < 128) \{ \]

\[ \text{problem} \.
\]

\[ \}

\[ \text{// manipulate Sample_inode} \]

\[ \text{// write updated Sample_inode to fd} \]

\[ \text{llseek}(\text{fd}, 4096, 0); \]

\[ \text{if} (\text{write}(\text{fd}, \text{Sample_inode}, 128) < 128) \{ \]

\[ \text{problem} \.
\]

\[ \}

---

Device Management

Disk drive

Disk I/O

Time to access block \( b \) of a hard drive
1. Move r/lw head to r/lw arm, correct track #, [seek]
2. Latency: Wait till r/lw arm is directly above the block b
   \[ t = \frac{1}{2} \text{ time for one read} \]
3. Transfer time:

Example: File stored in contiguous blocks.

3600 RPM
Average Seek time = 20 ms.
Block size = 512 bytes; 32 blocks/track

File Size: 256 blocks.
Time for 1 revolution = \( \frac{1}{60} = 16.67 \) ms.
Time to read 1st track
- Seek: latency + transfer
  \[ = 20 \text{ ms} + 8.33 + 16.67 \]
  \[ = 45 \text{ ms} \]
Time to read each successive track
  \[ = 0 + 8.33 + 16.67 \]
  \[ = 25 \text{ ms} \]

\( \frac{256}{32} = 8 \)
Time to read 8 tracks

= 45 + 7 \times 25 = 220\text{ms}.

File Stored Non-Continuously.

Time to read one block

\begin{align*}
= 20\text{ms} + 8.33 + \frac{1}{32} \times 16.67 \\
&= 20 + 8.33 + 0.5 \\
&= 28.83\text{ms}.
\end{align*}

Time to read the whole file

\begin{align*}
= 256 \times 28.83 = 7373\text{ms}.
\end{align*}

Disk head scheduling.

We have a series of requests for disk accesses.

How [what order?] to serve requests?
A. FIFO
Zig Zag movement of RLW head.
No starvation

B. Greedy - Shortest Seek Time First
Not much Zig Zag
Starvation

C. Scan / Elevator

11-14-13
RLW head has a direction of movement [inward or outward]
Continue in same direction if there are pending requests that can be served by proceeding in same direction.
Example

Tracks

move towards center → track #5
decrease

Current direction: inward, at track # 75
Pending requests at

100 150 175 119 80
59 39 12 112

Maintain 2 Sorted list

Forward list: Sorted list of pending requests that can be served by continuing in same direction

Reverse list: Sorted list of pending requests that can be served only after reversing direction

New request: for track number:
Insert into correct position of appropriate list
The head done with current request:
if (forward list != empty) {
    n = next item from forward
    list;
    delete n from the list;
    if (reverse list != empty) {
        n = next item from
        reverse list;
        delete n from the
        list;
        if (reverse list != empty) {
            else "...
            change direction:
            Swap two lists;
    ...
    else {
        wait till next request;}

Example

The head at track # 53 (inward)

Requests: 98, 183, 37, 122, 14, 124, 65, 67, ...

(a) FIFO: schedule: 53 → 98 → 183:
   → 67 27 → 122 → 14 → 124 → 65:

Total # of tracks moved = (98-53) +
(183-98) + (183-37) + (122-37)
= 640 Tracks

(B) Shortest Seek Time First

Schedule: 53 → 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183

# of tracks moved = (65-53) + (67-65) + 236 tracks

(C) Elevator

Schedule: 53 → 37 → 14 → Change direction → 65 → 67 → 98 → 122 → 124 → 183

# of tracks moved = (53-37) + (37-14) + (65-14) + 208 tracks
int fd = open(pathname, flags, mode)

name of file

create if file does not exist

for pid = 17

user file descriptor table

one / process

for pid = 18

File table

1 per system

i-node table

1 for system
Process with PID 17 performs open system call on `/home/venky/behappy.cc`

1. Translate `/home/venky/behappy.cc`
   into inode #.
   say 101 is inode #.

2. Fetch inode #101 from hard drive
   & put it in i-node table [if not in i-node table already]

3. Check permission
   if ok to open,
   (i) Create a (new) entry in file table,
   has offset [=0 initially]
   rw in both
   has a pointer to i-node entry [copy] in i-node table

Create an entry in user file descriptor table for this process
and make it point to file table entry
int read(fd, target_address, nbytes)

1. mode = R
2. Count = nbytes
3. find offset i from file table
4. store target address
   kernel or user mode?
5. (a) check if r is allowed.
   (b) convert offset into logical block #
   (c) logical block # into physical block # using
       addr[] of i-node.
6. fetch the [physical] data block
   into "Cache Buffer"
7. copy bytes from data block
   in cache buffer to target address.
8. update offset in file table
9. return number of bytes read.
Q4: What material?

LRU page replacement algorithm

Saturday Nov 23
10 Am in class room