Caching I/O Example worksheet

This worksheet is an example for how you can simulate operations on your Caching I/O implementation as you plan your design. See the handout for details on how to use it!

Here's the full example program (broken down step by step with template on the next page):

```
char buffer[5];
io300_file* testFile = io300_open("test_files/tiny.txt", "tiny!");
ssize_t r = io300_read(testFile, buffer, 5);
ssize_t w = io300_write(testFile, "aaa", 3);
r = io300_read(testFile, buffer, 2);
ssize_t s = io300_seek(testFile, 12);
w = io300_write(testFile, "aaa", 3);
r = io300_readc(testFile);
io300_close(testFile);
```

Line 2: `io300_file* testFile = io300_open("testfiles/tiny.txt", "tiny!");`

Cache (8 bytes):

t	h	i	s		i	s	

(SPACE)

Metadata:

```
field1:  value1
field2:  value2
...
```

File contents and OS read/write head (^): *[we've filled in this one for you as an example]*

t	h	i	s		i	s		a		t	e	s	t	\n

R/W HEAD AFTER FETCH (OPTIONAL)

System calls made (if any):

OPEN( ) => 3

Return value:

- At minimum, need to call open() to open the file => will return file descriptor for this file

- Could also fetch data into the cache at this point -- but depends on your design!

Line 3: `ssize_t r = io300_read(testFile, buffer, 5);`

Should return 5
buffer = {t,h,i,s,' '}

Cache (8 bytes):

t	h	i	s		i	s	

Metadata:

```
field1:  value1
field2:  value2
...
```

File contents and OS read/write head (^):

t	h	i	s		i	s		a		t	e	s	t	\n

System calls made (if any):

Return value:

(See hint in handout for what your file and cache should look like at this point!)

=> If previous step fetched data into the cache, no syscalls necessary here
=> Or, if no data in cache, need to fetch => would call read()

Takeaway: Need some way to track current position in the cache, to know which bytes to return on each read/write call
=> Next byte to be read/written

Line 4: `ssize_t w = io300_write(testFile, "aaa", 3);`

Cache (8 bytes):

t	h	i	s		a	a	a

Metadata:

```
field1:  value1
field2:  value2
...
```

File contents and OS read/write head (^):

t	h	i	s		i	s		a		t	e	s	t	\n

System calls made (if any): **Bytes to be written are in the cache => no syscalls necessary yet!** *(some designs might flush the cache now, since write changed last byte)*

Return value:

=> Returns 3 (number of bytes written)

Key takeaways
- "Next data to read/write" pointer is the same for both reads and writes (just like the OS read/write head)
- Not all writes need to make system calls => don't always need to change the file immediately

Line 5: `r = io300_read(testFile, buffer, 2);`

First: what should read() return?
=> 2 bytes read, buffer={a, ' '}

Cache (8 bytes):

a		t	e	s	t	\n	(undef)

Metadata:

```
field1:  value1
field2:  value2
...
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

File contents and OS read/write head (^):

t	h	i	s		a	a	a	a		t	e	s	t	\n

System calls made (if any):

Return value:

Next data to be read isn't in the cache, so need to fetch!
However: we made changes to cache, need to "flush" data out to disk. To do this:
- first need to reset OS read/write head such that we can write correct bytes
  => lseek() => Need to decide on position in file to seek to based on current cache
  => write(fd, ...) => Need to decide how much data to write

Then: need to fetch next data into cache
  => read(fd, ..., 8) =>7

=> Takeaway: need something to keep track of if there have been pending writes (if cache is "dirty")

Problem: When filling the cache, read() returns 7 instead of 8. What does this mean?
=> Only 7 bytes left in file! This means there's empty space in the cache (value undefined!)

=> Takeaway: need to keep track of how much data in cache is "valid" (ie, where to stop reading)

Line 6: `ssize_t s = io300_seek(testFile, 12);`

Cache (8 bytes):

a		t	e	s	t	\n	(undef)

Metadata:

```
field1:  value1
field2:  value2
...
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

File contents and OS read/write head (^):

t	h	i	s		a	a	a	a		t	e	s	t	\n

System calls made (if any):

Return value:

=> Seek should set the file offset to the specified position (called an "offset"), such that the next read/write happens from there (in this case, offset 12).

=> Note: for our project, io300_seek always specifies an "absolute" offset into the file. The system call version (lseek) makes this configurable--see `man 2 lseek` for details you may need

System calls: not strictly necessary here (though some designs might, to prepare for future reads/writes)

=> Should return 3

Line 7: w = io300_write(testFile, "aaa", 3);

Cache (8 bytes):

a		t	e	s	t	\n	(undef)

Metadata:

```
field1:  value1
field2:  value2
...
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

File contents and OS read/write head (^):

t	h	i	s		a	a	a	a		t	e	s	t	\n

System calls made (if any):

=> Bytes to be changed are in the cache--no need to make syscalls here!

Return value:

=> Read should return 3 (number of bytes written)

(See hint in handout for what your file and cache should look like at this point!)

Line 8: r = io300_readc(testFile);

Cache (8 bytes):

a		t	e	a	a	a	(undef)

Metadata:

```
field1:  value1
field2:  value2
...
```

```
0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
```

File contents and OS read/write head (^):

t	h	i	s		a	a	a	a		t	e	s	t	\n

END OF FILE! (EOF)

System calls made (if any):

Return value: -1

readc should return the next character, but there are no more characters to read!
=> This is called reaching the eof-of-file (EOF)

=> By convention, readc() should return -1 to indicate to the user that EOF has been reached

What if this were a read() call instead of readc()? By convention read() should return 0 to indicate EOF, in order to match the behavior of the read() system call

=> Be sure to remember this behavior when reading from files, and make sure that your implementation returns 0 at EOF too--need to make sure your library has the same outward behavior!

Line 9: io300_close(testFile);

Cache (8 bytes):

a		t	e	a	a	a	(undef)

Metadata:

```
field1:  value1
field2:  value2
...
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

File contents and OS read/write head (^):

t	h	i	s		a	a	a	a		t	e	a	a	a

System calls made (if any):

Return value:

What needs to happen on close?

When the file is closed, all pending changes need to be written to the file
=> we have pending changes, so need to flush => calls write()
=> Remember that not all bytes of the cache are valid! Need to only write up to the last valid byte (ie, the last 'a')

Closing thoughts
=> When you get stuck, try to think about how your program should operate in terms of the cache and disk, like you see here => this will help!
=> You got this!!!

Blank template (in case you need it)

<line of code here>

Cache:

***Metadata*:**

```
field1:  value1
field2:  value2
...
```

***File contents and OS read/write head (^)*:**

***System calls made (if any)*:**

***Return value*:**