CSCE 313-200
Introduction to Computer Systems
Spring 2024

File System II
Dmitri Loguinov
Texas A&M University

March 22, 2024
Chapter 11: Roadmap

11.1 I/O devices
11.2 I/O function
11.3 OS design issues
11.4 I/O buffering
11.5 Disk scheduling
11.6 RAID
11.7 Disk cache
11.8-11.10 Unix, Linux, Windows
Background on Files

- Asking the kernel for chunk of data
  - How large should the chunk be?
- Clearly not too small, otherwise many kernel-mode transitions, which are costly
- Some wrapper libraries (FILE and STL streams) have yet another buffer to avoid kernel-mode switching
  - Also needed if they perform text-mode pre-processing
- **OS buffering** can be disabled
  - Disk driver directly DMAs data into your program’s buffer
  - Caveat: buffer size must be a multiple of sector size (512 bytes)
APIs

• CreateFile is the most flexible and high-performance method of doing I/O
  - Treats the memory as a sequence of bytes
  - Operates in binary mode and gives you the native representation of RAM data structures

• Read MSDN about access (read, write, both), sharing, and disposition (e.g., open existing, create new)

• The flag field sets the attributes (e.g., hidden, encrypted, read-only, archived, system)
  - Also can be used to disable OS buffering (FILE_FLAG_NO_BUFFERING) or enable overlapped operation (FILE_FLAG_OVERLAPPED)

HANDLE WINAPI CreateFile(
    __in LPCTSTR lpFileName,
    __in DWORD dwDesiredAccess,
    __in DWORD dwShareMode,
    NULL, // security
    __in DWORD dwCreationDisposition,
    __in DWORD dwFlagsAndAttributes,
    NULL // template
);
APIs

• Some functions take two DWORDs instead of one uint64
  – How to convert?

```c
DWORD low = GetFileSize(HANDLE hFile, LPDWORD high);
```

• Overlapped I/O allows multiple outstanding requests

```c
DWORD WINAPI SetFilePointer(__in HANDLE hFile, __in LONG lDistanceToMove, __inout_opt PLOONG lpDistanceToMoveHigh, __in DWORD dwMoveMethod );
```

```c
OVERLAPPED ol;
memset (&ol, 0, sizeof (OVERLAPPED));
ol.hEvent = CreateEvent (NULL, false, false, NULL);
ReadFile (hFile, buf, len, NULL, &ol);
// if error == ERROR_IO_PENDING, continue
WaitForSingleObject (ol.hEvent, INFINITE);
GetOverlappedResult (hFile, &ol, &bytesRead, false);
```

Note: each pending request must have its own struct ol
**APIs**

- The FILE stream is the classical C-style library
  - Portable to Unix and most other OSes

```c
char buf [BUF_SIZE];
// open for reading in binary mode
FILE *f = fopen ("test.txt", "rb");
if (f == NULL) {
    printf ("Error %d opening file\n", errno);
    exit (-1);
}
// read up to one full buffer
// native representation
int bytesRead = fread (buf, 1, BUF_SIZE, f);
fclose (f);
```

```c
FILE *f = fopen ("test.txt", "rb");
// seek to the end
_fseeki64 (f, 0, SEEK_END);
// get current position
uint64 fileSize = _ftelli64 (f);
// return to beginning
_fseeki64 (f, 0, SEEK_SET);
printf ("file size %llu\n", fileSize);
```

```c
int a = 5;
double b = 10;
// open for writing in binary mode
FILE *f = fopen ("test.txt", "wb");
// ASCII representation
fprintf (f, "a = %d, b = %f\n", a, b);
fclose (f);
```

```c
int a;
double b;
// ASCII decoding of numbers
int ret = fscanf (f, "%d %f", &a, &b);
if (ret == 0 || ret == EOF)
    printf ("Hit error or EOF\n");
else
    printf ("Obtained %d, %f\n", a, b);
// %s gets one word and NULL terminates it
// note: potential buffer overflow
fscanf (f, "%s", buf);
// recommended to specify buf length
fscanf (f, "%32s", buf);
```
If an entire line is needed, a faster alternative to fscanf is fgets()

STL streams are similar

```c
char buf [BUF_SIZE];
FILE *f = fopen ("test.txt", "rb");
while (!feof (f)) {
    // read one line at a time
    if (fgets (buf, BUF_SIZE, f) == NULL)
        break; // EOF or error
    printf ("Line '%s' has %d bytes
", buf, strlen(buf));
    printf ("Position in file %d
", ifs.tellg());
}
fclose (f);
```

Q: using Windows APIs, how to print contents of a text file?

```c
// assume file is small and fits in RAM
// allocate the buffer
char *buf = new char [fileSize + 1];
ReadFile (... buf, fileSize, &bytes, ...);
// TODO: error checks
buf[bytes] = NULL;
printf ("%s", buf);
```
• Dual RAID controllers, each with 12 disks in RAID-5
  - Speed given in MB/s, CPU utilization = fraction of 16 cores

<table>
<thead>
<tr>
<th></th>
<th>Text mode</th>
<th>Binary mode</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Debug</td>
<td>Release</td>
<td>Debug</td>
</tr>
<tr>
<td></td>
<td>1.8</td>
<td>12</td>
<td>1.8</td>
</tr>
<tr>
<td>ifs &gt;&gt; s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fscanf (f, “%s”, buf)</td>
<td>6</td>
<td>19</td>
<td>7.5</td>
</tr>
<tr>
<td>fgets (buf, BUF_SIZE, f)</td>
<td>26</td>
<td>50</td>
<td>39</td>
</tr>
<tr>
<td>ifs.read w/32MB buffer</td>
<td>90</td>
<td></td>
<td>360</td>
</tr>
<tr>
<td>fread w/32MB buffer</td>
<td>90</td>
<td>144</td>
<td>503</td>
</tr>
<tr>
<td>ReadFile w/32MB buffer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadFile + no OS buffering</td>
<td></td>
<td></td>
<td>982</td>
</tr>
<tr>
<td>ReadFile + no buf + overlapped</td>
<td>1923</td>
<td></td>
<td>2500</td>
</tr>
</tbody>
</table>

• Modern PCI-e 4.0 m.2 drives
  - Up to 7 GB/s; multiple in RAID configuration up to 30 GB/s