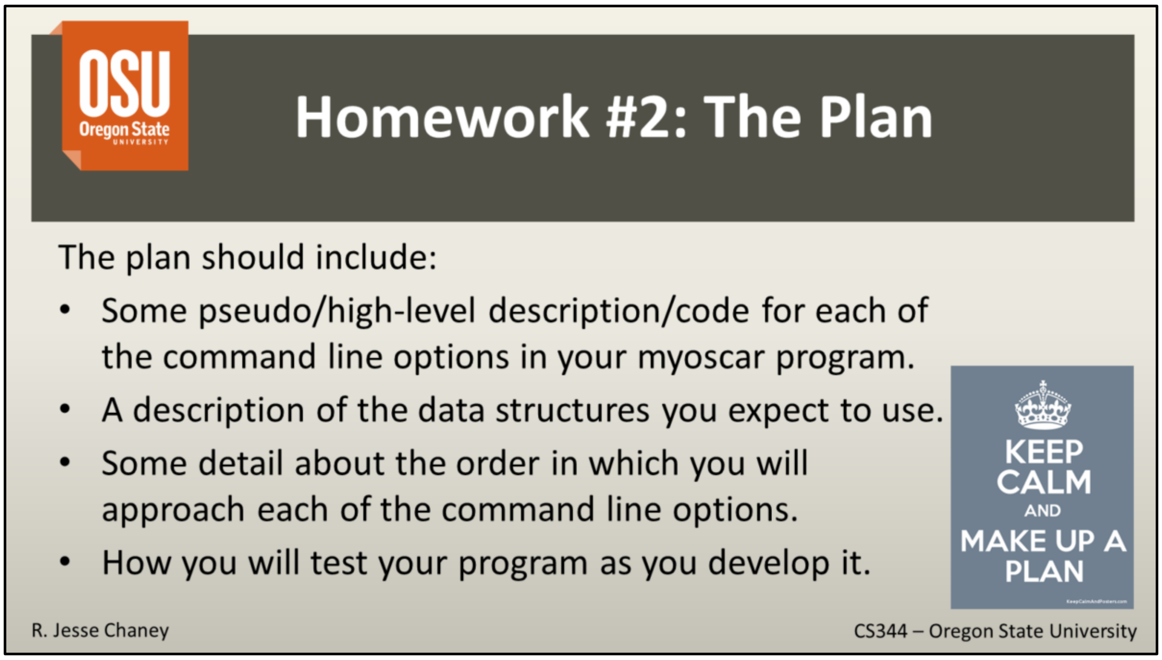


Homework #2: The Plan

The plan should include:

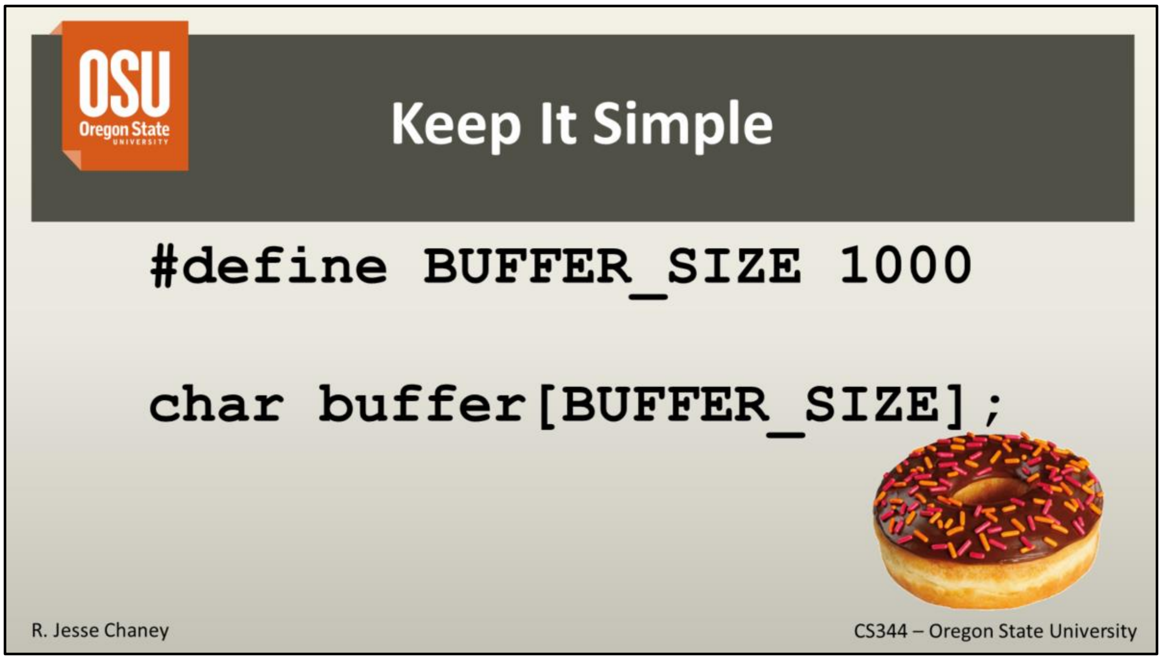
- Some pseudo/high-level description/code for each of the command line options in your myoscar program.
- A description of the data structures you expect to use.
- Some detail about the order in which you will approach each of the command line options.
- How you will test your program as you develop it.

KEEP CALM
AND
MAKE UP A PLAN

The plan for Homework #2 is due **Monday night** (Jan 26th). Section 2.1 in the assignment reads as…

Think about all the various command line options; there are quite a number of them. Spend some time thinking about the order in which you will begin to implement the command line options. Which one will be first? Which one will be last? Which ones will you never get to (like the extra credit)? Will you tackle the most difficult one first or the easiest one? How can you use the example oscar program I have on os-class to test your code?

I ask for a description of data structures you expect to use. Keep it simple. I know you've all had a data structures class and can create a doubly-linked-balanced-binary tree embedded in inside a Fibonacci hash map with double ninja escargot. Don't try and out smart yourself.



Keep It Simple

```
#define BUFFER_SIZE 1000

char buffer[BUFFER_SIZE];
```

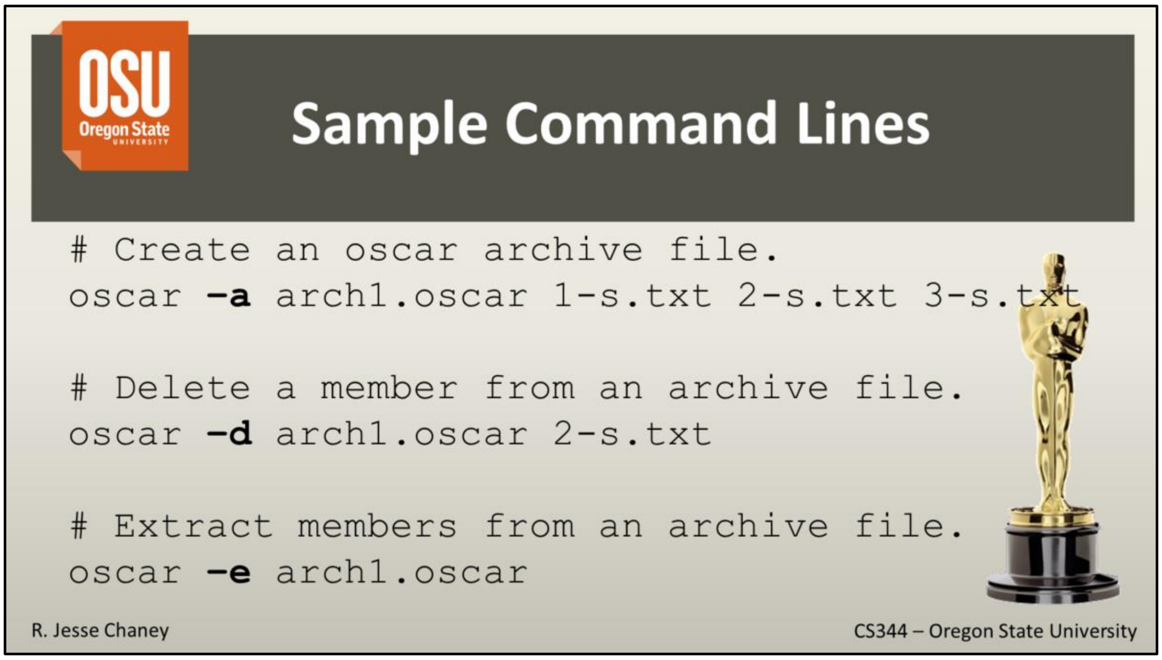
R. Jesse Chaney

CS344 – Oregon State University

For this assignment, your data structure might look a lot like this. Remember I said this assignment is about reading and writing files. I did not say anything about data structures that looks like a torus.

You should never need to read through an oscar archive file more than once to complete an operation. In fact, you will almost always need to read through the oscar archive file once.

Don’t assume that you can read the entire oscar archive file into memory all at once. You are not helping yourself if you go down that road.



Sample Command Lines

```
# Create an oscar archive file.
oscar -a arch1.oscar 1-s.txt 2-s.txt 3-s.txt

# Delete a member from an archive file.
oscar -d arch1.oscar 2-s.txt

# Extract members from an archive file.
oscar -e arch1.oscar
```

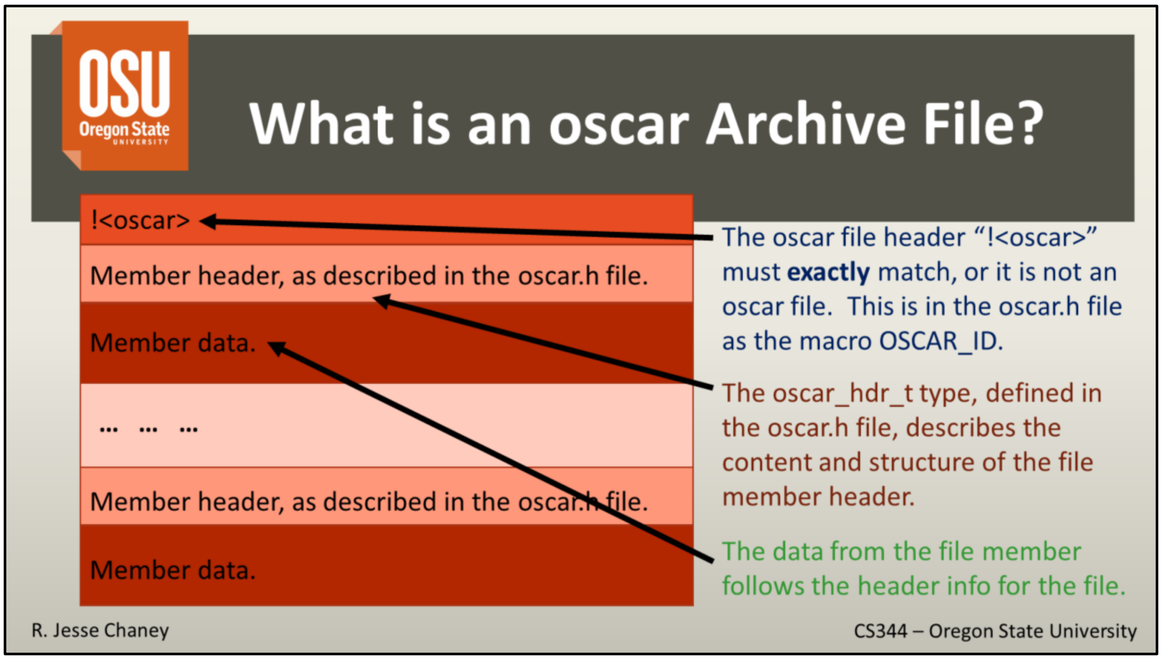
R. Jesse Chaney

CS344 – Oregon State University

The –a switch **creates** a new oscar archive file or **adds** an additional file member to the end of the oscar archive file. The a stands for append. A file member may exist more than once in an archive.

When you create a new a oscar archive file, what is the first thing you want to put into it? When you open an existing oscar file, what is the first thing you want to validate?

I really believe that getopt() is the best way for you to handle the command line. The rmchar.c code on Blackboard really is a good example of how to deal with this sort of command line.



What is an oscar Archive File?

!<oscar>
Member header, as described in the oscar.h file.
Member data.
… … …
Member header, as described in the oscar.h file.
Member data.

The oscar file header "!<oscar>" must **exactly** match, or it is not an oscar file. This is in the oscar.h file as the macro OSCAR_ID.

The oscar_hdr_t type, defined in the oscar.h file, describes the content and structure of the file member header.

The data from the file member follows the header info for the file.

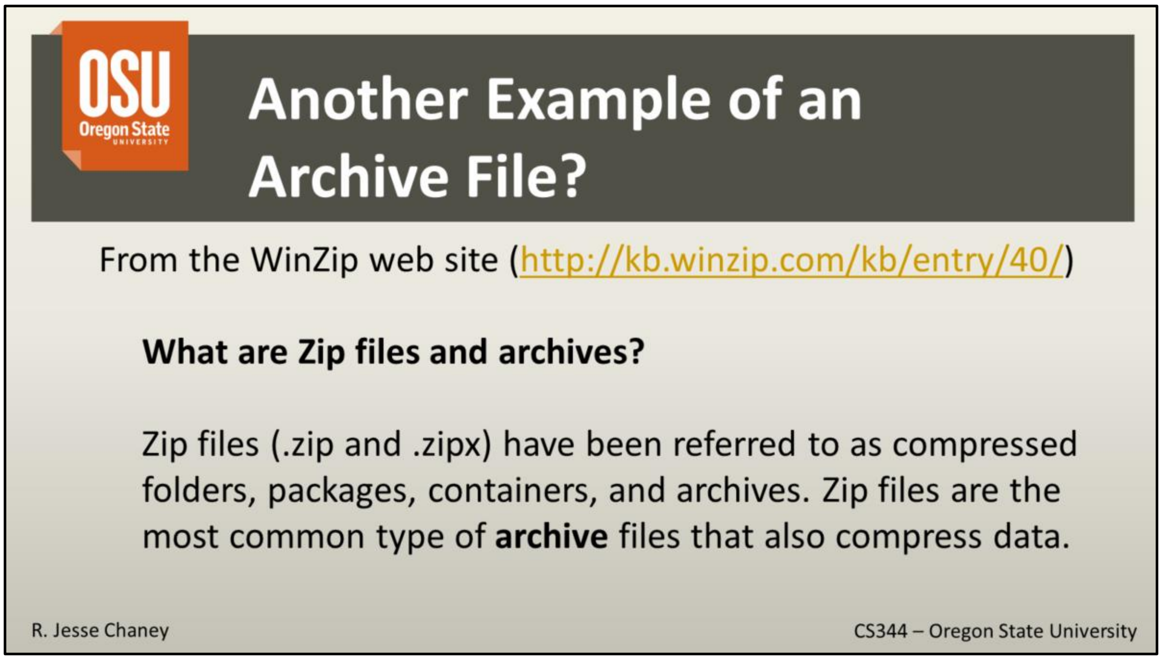
R. Jesse Chaney CS344 – Oregon State University

First, an oscar archive file is a file.

An oscar archive file is just a file that contains other files, like a zip file contains other files. The files that are contained within an oscar archive file are called file members. The file members in an oscar archive file are represented as the data from the inode (size, dates, mode, and such). We call the inode data meta-data for the file. The file member header is all ASCII text. The actual file member data follows the file member header. For most of our examples, the data for a member file is also ASCII data. An archive file can contain the meta-data (file member header) and data for many files.

The file header in an oscar archive file occurs only once and must exactly match the string from the oscar.h file that corresponds to the macro OSCAR_ID.

The file member header corresponds exactly to the type/struct you see in the oscar.h file, the oscar_hdr_t type or struct oscar_hdr_s.



Another Example of an Archive File?

From the WinZip web site (http://kb.winzip.com/kb/entry/40/)

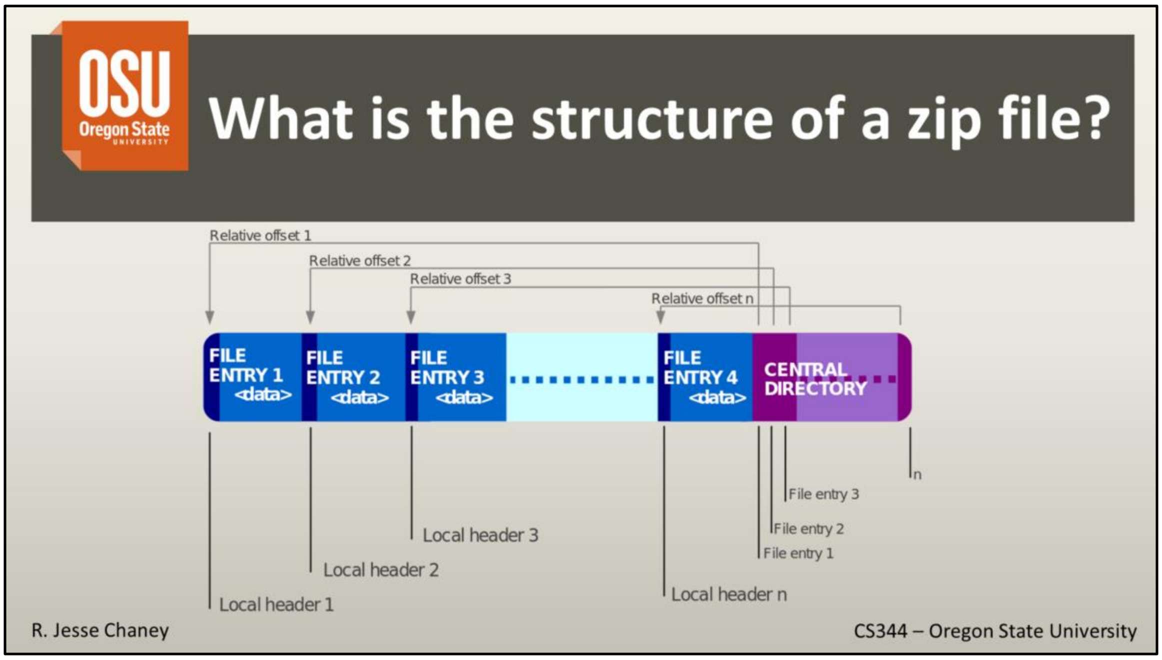
What are Zip files and archives?

Zip files (.zip and .zipx) have been referred to as compressed folders, packages, containers, and archives. Zip files are the most common type of **archive** files that also compress data.

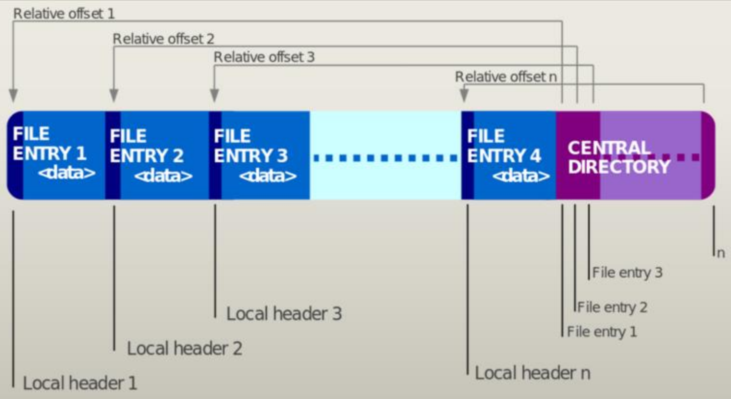
R. Jesse Chaney

CS344 – Oregon State University

This is a quote from the WinZip site. An oscar archive file does not compress data. Zip files are just another form of archive file.



What is the structure of a zip file?

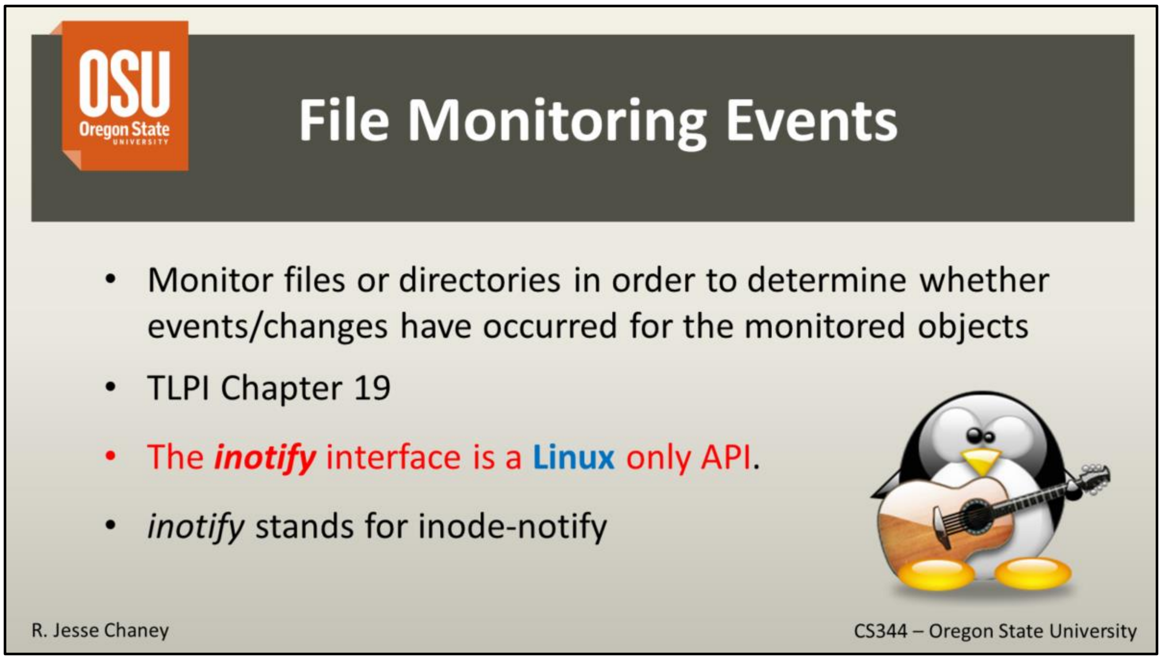


R. Jesse Chaney

CS344 – Oregon State University

From http://en.wikipedia.org/wiki/Zip_(file_format)

This does not look sooooo different from the structure of an oscar archive file.



File Monitoring Events

- Monitor files or directories in order to determine whether events/changes have occurred for the monitored objects
- TLPI Chapter 19
- The ***inotify*** interface is a **Linux** only API.
- *inotify* stands for inode-notify

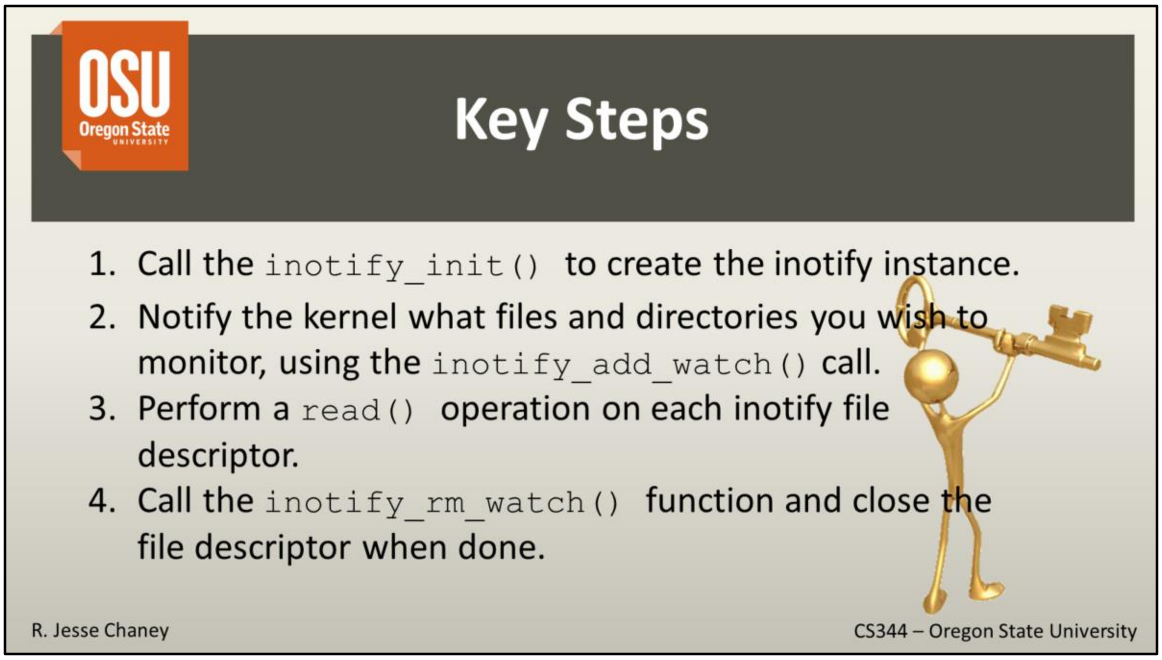
R. Jesse Chaney

CS344 – Oregon State University

- There is an older Linux file monitoring interface, dnotify, but it is not supported anymore.

- When monitoring a directory, the application will be informed about events for the directory itself and for files inside the directory.
- The inotify monitoring mechanism is not recursive. If an application wants to monitor events within an entire directory subtree, it must issue inotify_add_watch() calls for each directory in the tree.

- These are **kernel** events. The actual meaning of that will become evident when we move to looking at the code.



Key Steps

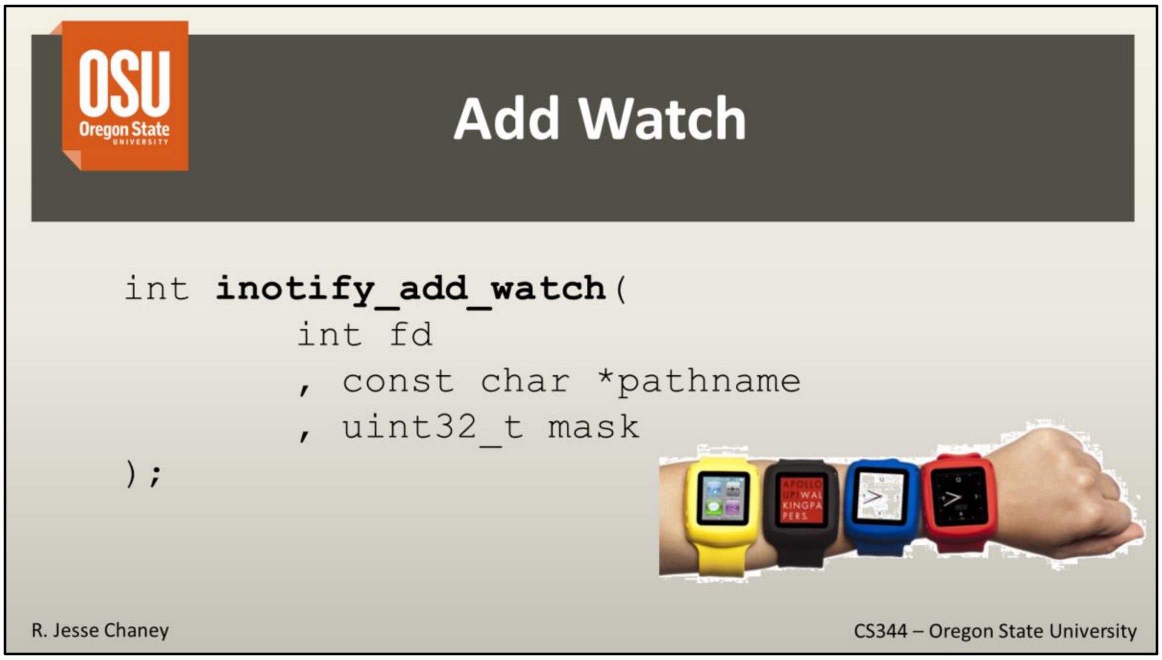
1. Call the `inotify_init()` to create the inotify instance.
2. Notify the kernel what files and directories you wish to monitor, using the `inotify_add_watch()` call.
3. Perform a `read()` operation on each inotify file descriptor.
4. Call the `inotify_rm_watch()` function and close the file descriptor when done.

R. Jesse Chaney   CS344 – Oregon State University

- When you call the *inotify_init*() function, it returns a **file descriptor**. It is a file descriptor just like the file descriptor returned by the open() call. File descriptors are things we will talk about all through the class. It gets back to the **Universality of I/O** that I mentioned the first week of class.

- Once you add files and directories with the *inotify_add_watch()* call, you get back an integer (int) called a **watch descriptor**.
- You can associate multiple watch descriptors with a single file descriptor.

- When you look at the code, we'll see how you could use the returned file descriptor with a select statement, just as you can with other file descriptors.



Add Watch

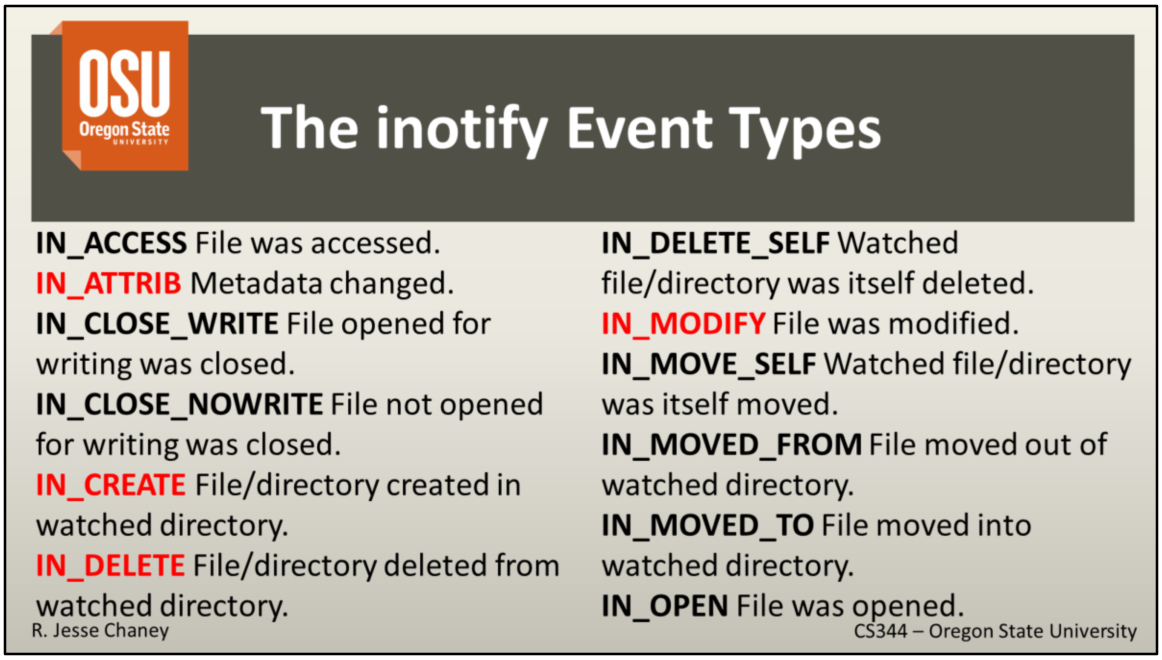
```
int inotify_add_watch(
        int fd
        , const char *pathname
        , uint32_t mask
);
```

R. Jesse Chaney

CS344 – Oregon State University

- The file descriptor is the fd returned from the inotify_inint() call.
- The path is a file or directory.
- The mask is an or collection of the types of events you wish to modify.

- If a new directory is created within a monitored directory, it is not automatically monitored. You have to add it list of monitored directories manually (or write code that does it). Finding some code that will traverse down a directory structure and add each directory to the watch list is pretty easy.



The inotify Event Types

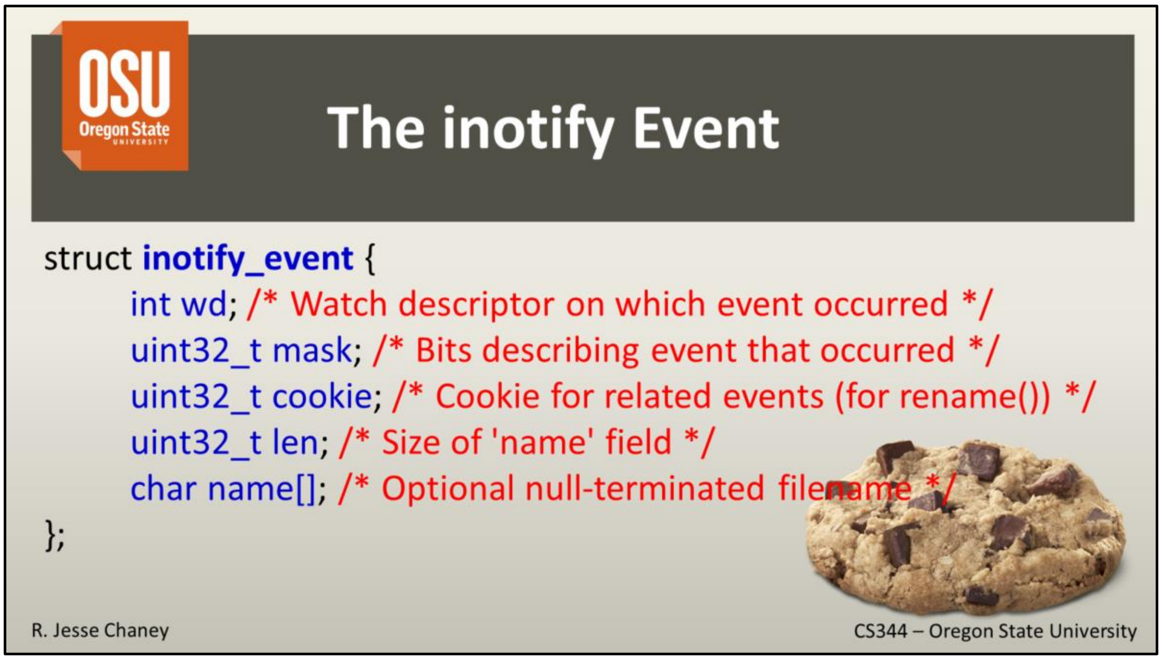
IN_ACCESS File was accessed.
IN_ATTRIB Metadata changed.
IN_CLOSE_WRITE File opened for writing was closed.
IN_CLOSE_NOWRITE File not opened for writing was closed.
IN_CREATE File/directory created in watched directory.
IN_DELETE File/directory deleted from watched directory.
IN_DELETE_SELF Watched file/directory was itself deleted.
IN_MODIFY File was modified.
IN_MOVE_SELF Watched file/directory was itself moved.
IN_MOVED_FROM File moved out of watched directory.
IN_MOVED_TO File moved into watched directory.
IN_OPEN File was opened.

R. Jesse Chaney

CS344 – Oregon State University

There are actually some additional events as well. They are listed in TLPI 19.3.

There is some overlap with a couple of these events. There are also come instances where you may receive multiple events for certain kinds of things.

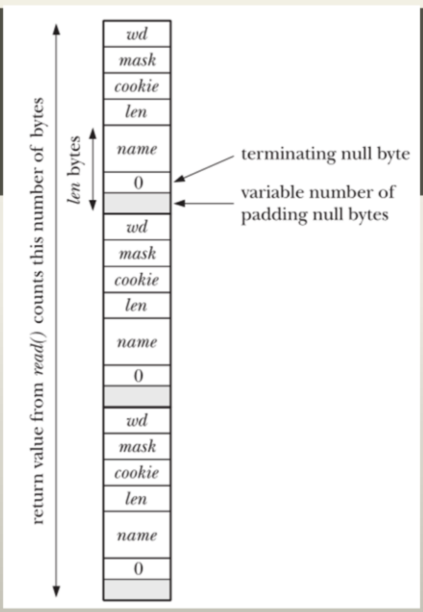


The inotify Event

```
struct inotify_event {
        int wd; /* Watch descriptor on which event occurred */
        uint32_t mask; /* Bits describing event that occurred */
        uint32_t cookie; /* Cookie for related events (for rename()) */
        uint32_t len; /* Size of 'name' field */
        char name[]; /* Optional null-terminated filename */
};
```

R. Jesse Chaney CS344 – Oregon State University

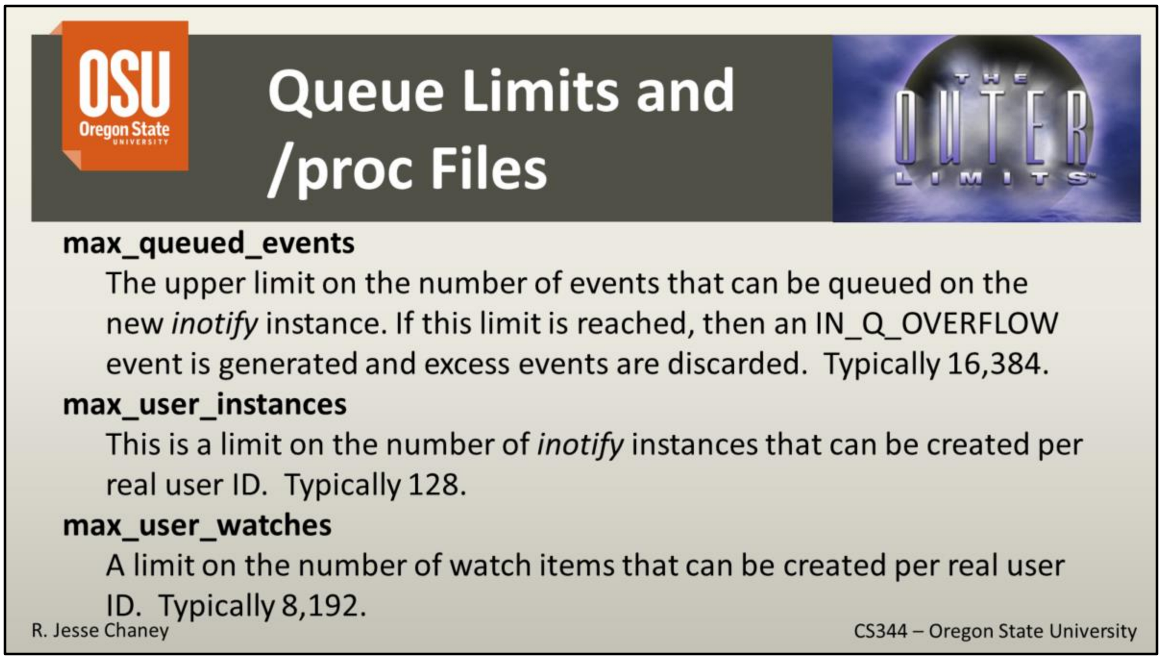
When you read from the file descriptor returned from inotify_init(), you'll get back a number of events of this structure.
The watch descriptor is what is returned from the inotify_add_watch() call.


return value from read() counts this number of bytes
len bytes
wd
mask
cookie
len
name
0
terminating null byte
variable number of padding null bytes

R. Jesse Chaney

CS344 – Oregon State University

- The **cookie** field is used to tie related events together.
- The **len** field indicates how many bytes are actually allocated for the name field. This field is necessary because there may be additional padding bytes between the end of the string stored in name and the. start of the next inotify_event structure contained in the buffer returned by read().
- The length of an individual inotify event is thus sizeof(struct inotify_event) + len.
- If the buffer passed to read() is too small to hold the next inotify_event structure, then read() fails with the error EINVAL to warn the application of this fact.
- A read() from an inotify file descriptor returns the minimum of the number of events that are available and the number of events that will fit in the supplied buffer.



Queue Limits and /proc Files

max_queued_events
The upper limit on the number of events that can be queued on the new *inotify* instance. If this limit is reached, then an IN_Q_OVERFLOW event is generated and excess events are discarded. Typically 16,384.

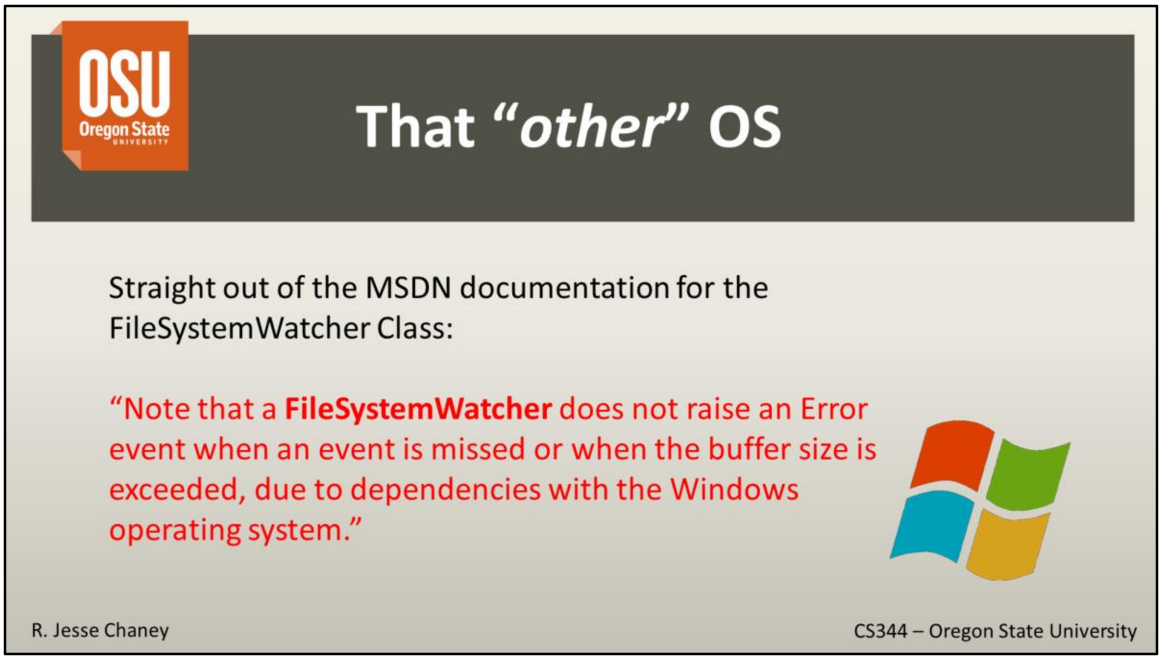
max_user_instances
This is a limit on the number of *inotify* instances that can be created per real user ID. Typically 128.

max_user_watches
A limit on the number of watch items that can be created per real user ID. Typically 8,192.

R. Jesse Chaney

CS344 – Oregon State University

The kernel places various limits on the operation of the inotify mechanism. The superuser can configure these limits via three files in the directory /proc/sys/fs/inotify



That "*other*" OS

Straight out of the MSDN documentation for the FileSystemWatcher Class:

"Note that a **FileSystemWatcher** does not raise an Error event when an event is missed or when the buffer size is exceeded, due to dependencies with the Windows operating system."

R. Jesse Chaney

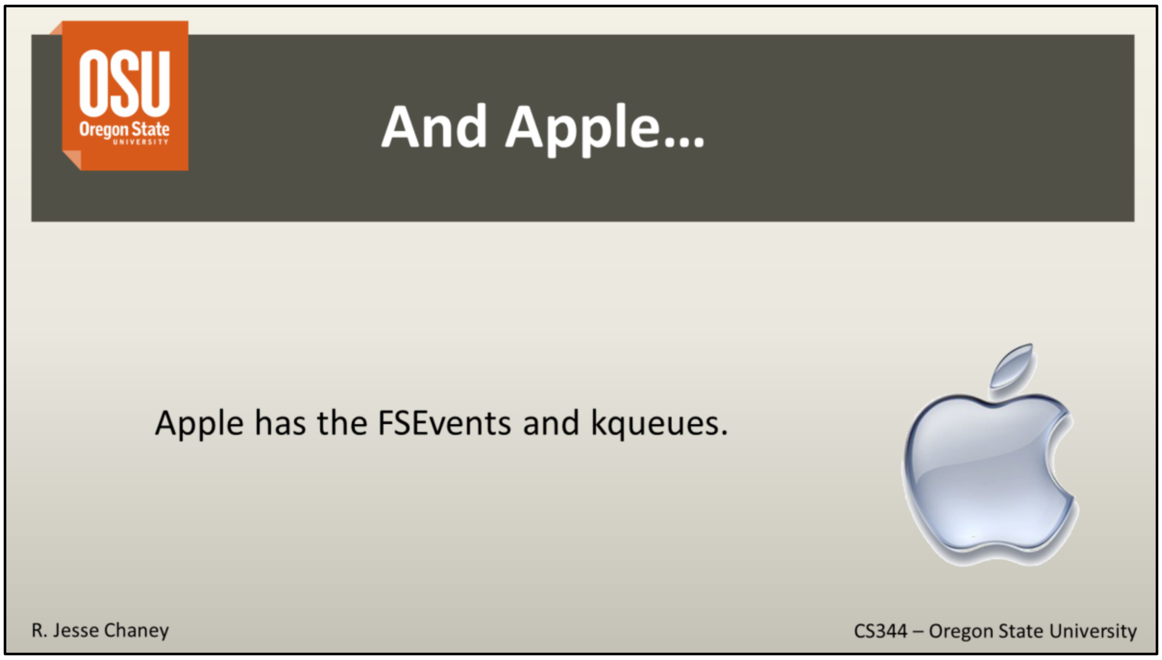
CS344 – Oregon State University

As a word of warning; Windows has a of couple similar file monitoring interfaces: FileSystemWatcher Class (which I have used) and FindFirstChangeNotification Win32 function (which I have not used).

However, the FileSystemWatcher Class has some issues. Having used that API in some development, it is reliable. If there are 1,000 file events, you'll receive 998, or 993, 987, or maybe 810, but receiving all 1,000 is not at all guaranteed. In fact, I found it unlikely.

MSDN does give you some guidelines to help prevent missing events, but they are not 100%.

- Increasing the buffer size with the **InternalBufferSize** property can prevent missing file system change events.
- Avoid watching files with long file names. Consider renaming using shorter names.
- Keep your event handling code as short as possible.



And Apple…

Apple has the FSEvents and kqueues.

R. Jesse Chaney

CS344 – Oregon State University

I’ve never use either of the Apple APIs for monitoring file and directory changes.