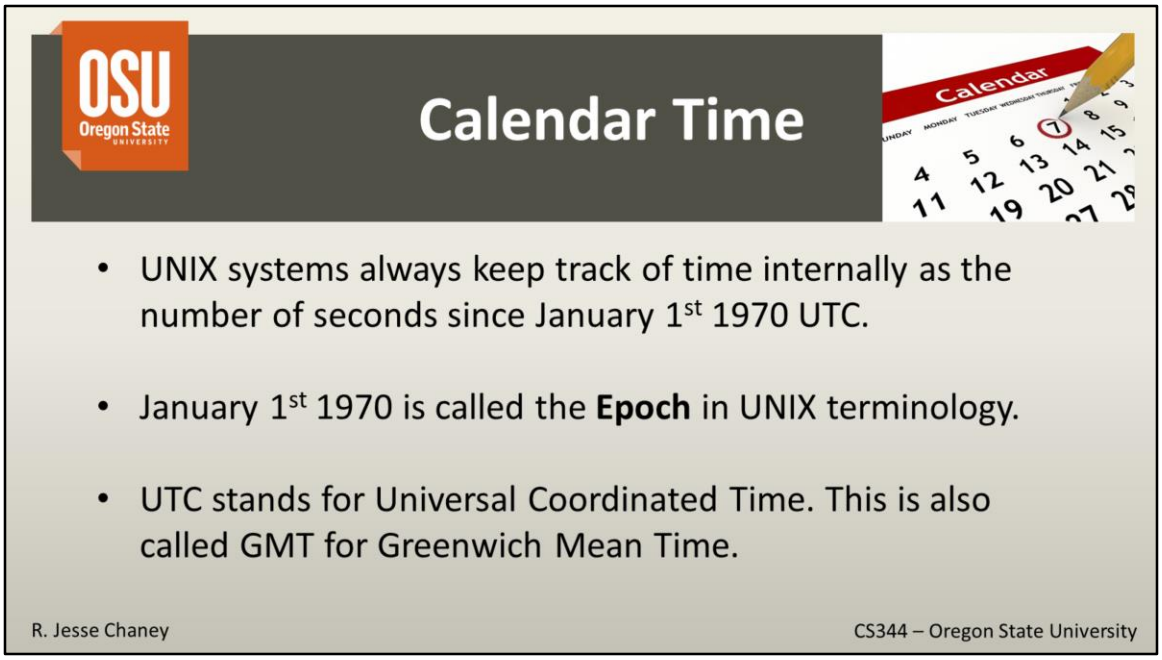
Unix Time

Within a program, we may be interested in two kinds of time:

- **Real time**: This is the time as measured either from some standard point (calendar time) or from some fixed point (typically the start) in the life of a process (elapsed or wall clock time). Obtaining the calendar time is useful to programs that, for example, timestamp database records or files. Measuring elapsed time is useful for a program that takes periodic actions or makes regular measurements from some external input device.
- **Process time**: This is the amount of CPU time used by a process. Measuring process time is useful for checking or optimizing the performance of a program or algorithm.

R. Jesse Chaney CS344 – Oregon State University

Part of every computer architecture is a built-in hardware clock that allows the kernel to keep track of real time and process time.



Regardless of geographic location, UNIX systems represent time internally as a measure of seconds since the Epoch; that is, since midnight on the morning of 1 January 1970, Universal Coordinated Time (UTC, previously known as Greenwich Mean Time, or GMT).

UTC just represents a standard against which offset are measured.

January 1st 1970 is approximately when UNIX came alive.

Calendar Time

On 32-bit Linux systems, **time_t**, which is a **signed** integer, that can represent dates in the range 13 December 1901 20:45:52 to 19 January 2038 03:14:07.

Many current **32-bit** UNIX systems face a theoretical **Year 2038 problem**, which they may encounter before 2038, if they do calculations based on dates in the future.

time_t is a system data type. Notice that it is **signed**.

This problem will be significantly alleviated by the fact that by 2038, probably all UNIX systems will have long become **64-bit** and beyond. However, **32-bit embedded systems**, which typically have a much longer lifespan than desktop hardware, may still be afflicted by the problem. Furthermore, the problem will remain for any legacy data and applications that maintain time in a 32-bit **time_t** format.

gettimeofday()

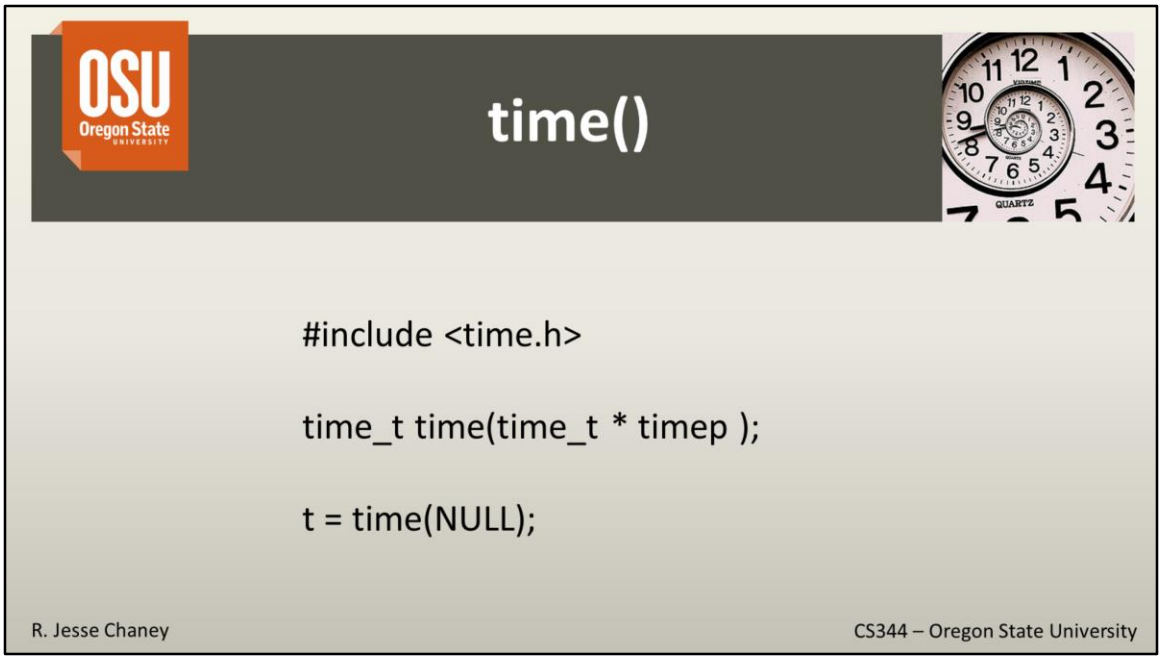
```
struct timeval {
    time_t tv_sec;          /* Seconds since 00:00:00, 1 Jan 1970 UTC */
    suseconds_t tv_usec; /* Additional microseconds (long int) */
};

#include <sys/time.h>
struct timeval tv;

gettimeofday(&tv, NULL);
```

R. Jesse Chaney

CS344 – Oregon State University

The gettimeofday() system call returns the **calendar** time in the buffer pointed to by tv.

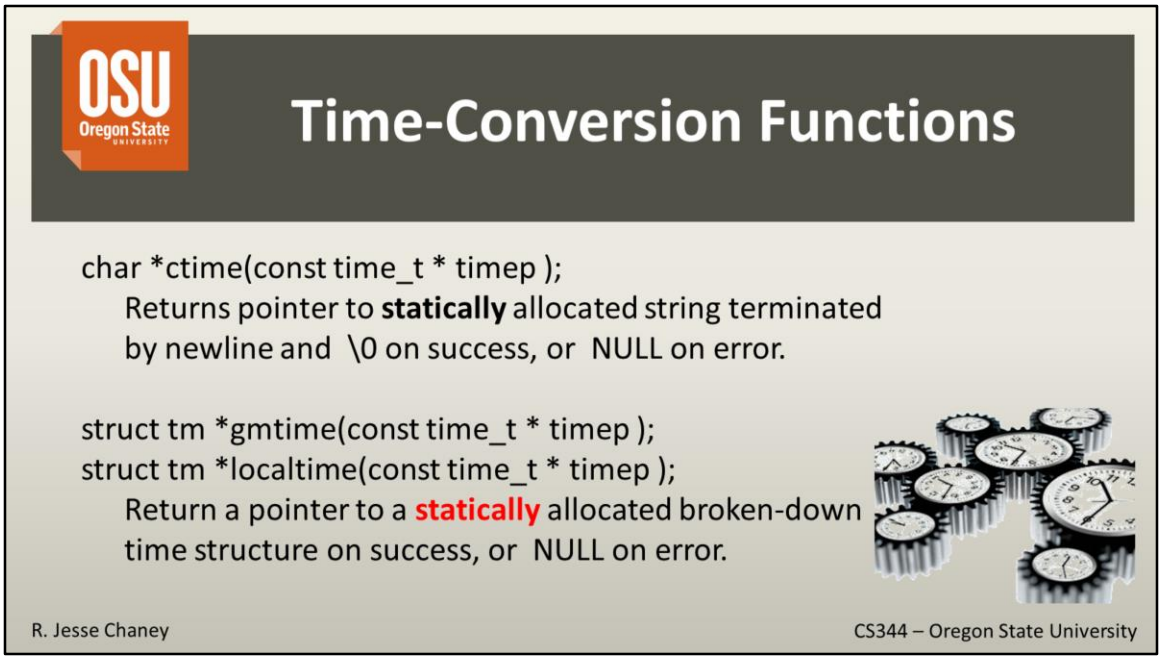
The **tz** argument to **gettimeofday()** is a historical artifact. In older UNIX implementations, it was used to retrieve timezone information for the system. This argument is now obsolete and should always be specified as NULL.



The time() system call returns the number of seconds since the Epoch (i.e., the same value that gettimeofday() returns in the tv_sec field of its tv argument).

If the timep argument is not NULL , the number of seconds since the Epoch is also placed in the location to which timep points.

The reason for the existence of two system calls (time() and gettimeofday()) with essentially the same purpose is historical. Early UNIX implementations provided time(). 4.3BSD added the more precise gettimeofday() system call.



The **gmtime**() function converts a calendar time into a broken-down time corresponding to **UTC**. (The letters gm derive from Greenwich Mean Time.) By contrast, **localtime**() takes into account **timezone and DST** settings to return a broken-down time corresponding to the system's **local** time.

Broken-down Time

```
struct tm {
    int tm_sec;     /* Seconds (0-60) */
    int tm_min;     /* Minutes (0-59) */
    int tm_hour;    /* Hours (0-23) */
    int tm_mday;    /* Day of the month (1-31) */
    int tm_mon;     /* Month (0-11) */
    int tm_year;    /* Year since 1900 */
    int tm_wday;    /* Day of the week (Sunday = 0)*/
    int tm_yday;    /* Day in the year (0-365; 1 Jan = 0)*/
    int tm_isdst;   /* Daylight saving time flag
                    > 0: DST is in effect;
                    = 0: DST is not effect;
                    < 0: DST information not available */
};
```

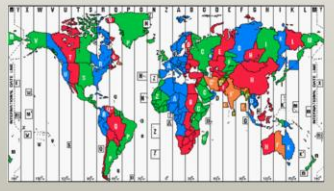
A leap second will be inserted at the end of June 30, 2015 at 23:59:60 UTC.

Timezones

To specify a timezone for running a program, we set the **TZ** environment variable to a string consisting of a colon (:) followed by one of the timezone names defined in /usr/share/zoneinfo

You get a default timezone value from the /etc/sysconfig/clock file.

```
$ cat /etc/sysconfig/clock
ZONE="America/Los_Angeles"
```



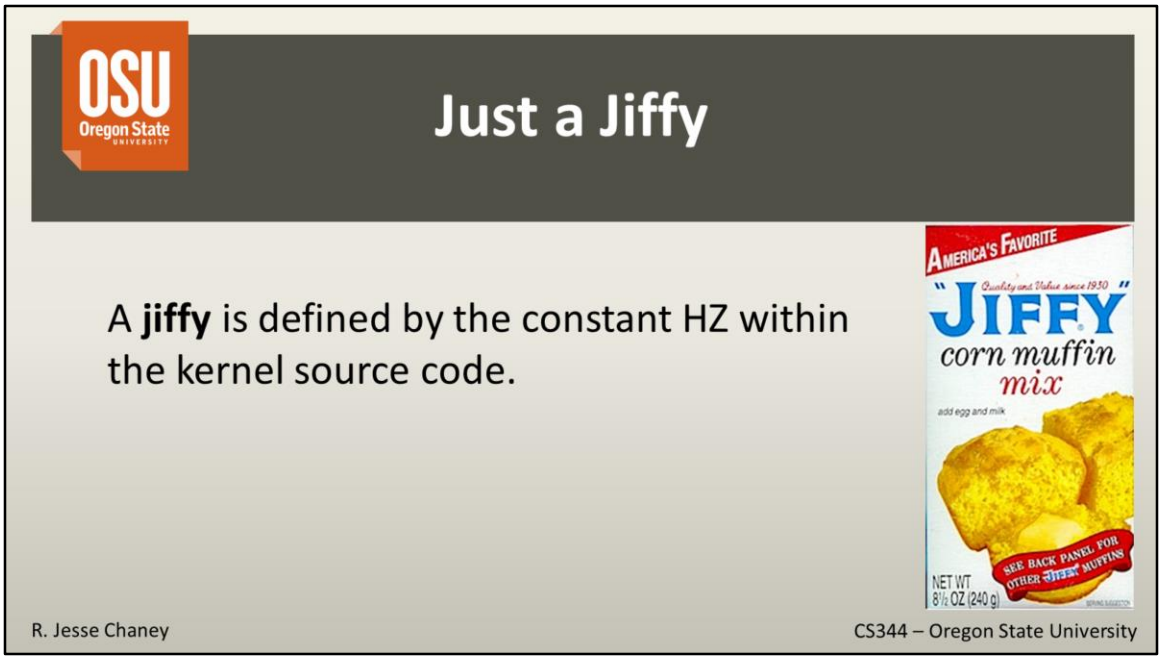
Updating the System Clock

settimeofday()
Sets the system's calendar time to the number of seconds and microseconds specified in the **timeval** structure pointed to by tv.

adjtime()
Make small incremental changes to the system clock, to ovoid the big changes that can be bad for a system.

We now look at two interfaces that update the system clock: settimeofday() and adjtime(). These interfaces are rarely used by application programs (since the system time is usually maintained by tools such as the Network Time Protocol daemon), and they require that the caller be privileged.

Abrupt changes in the system time of the sort caused by calls to settimeofday() can have deleterious effects on applications (e.g., make(1), a database system using timestamps, or time-stamped log files) that depend on a monotonically increasing system clock. For this reason, when making small changes to the time (of the order of a few seconds), it is usually preferable to use the adjtime() library function, which causes the system clock to gradually adjust to the desired value.

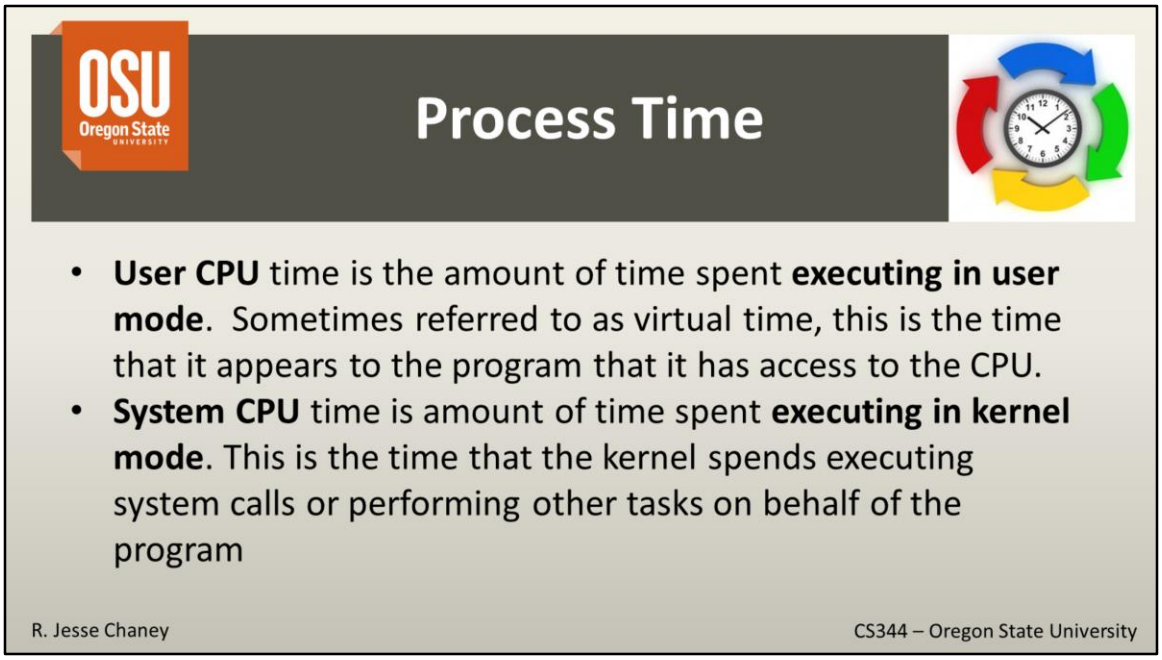


The accuracy of various time-related system calls described in this book is limited to the resolution of the system software clock, which measures time in units called **jiffies**. The size of a jiffy is defined by the constant HZ within the kernel source code. This is the unit in which the kernel allocates the CPU to processes under the round-robin time-sharing scheduling algorithm.

On Linux/x86-32 in kernel versions up to and including 2.4, the rate of the software clock was 100 hertz; that is, a jiffy is 10 milliseconds.

Debate among kernel developers eventually resulted in the software clock rate becoming a **configurable** kernel option.

The earliest technical usage for jiffy was defined by Gilbert Newton Lewis (1875–1946). He proposed a unit of time called the "jiffy" which was equal to the time it takes light to travel one centimeter (approximately 33.3564 picoseconds). It has since been redefined for different measurements depending on the field of study.



Process time is the amount of CPU time used by a process since it was created. For recording purposes, the kernel separates CPU time into the following two components:

- User CPU time is the amount of time spent executing in user mode. Sometimes referred to as virtual time, this is the time that it appears to the program that it has access to the CPU.
- System CPU time is amount of time spent executing in kernel mode. This is the time that the kernel spends executing system calls or performing other tasks on behalf of the program (e.g., servicing page faults).

Sometimes, we refer to process time as the total CPU time consumed by the process.