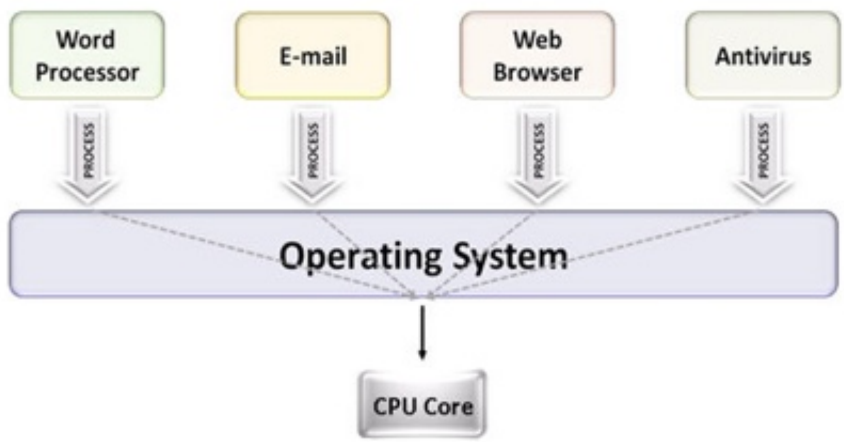
Week 8: Multithreaded Performance

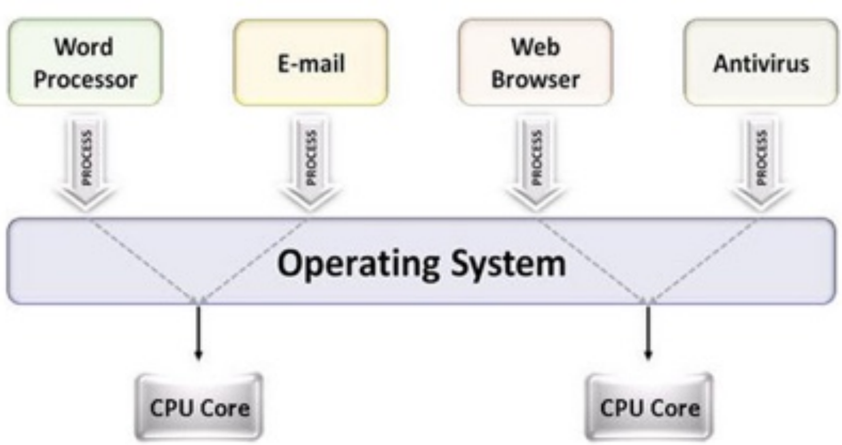
Multiprocessing

- use of multiple CPUs/cores to run multiple tasks simultaneously

Uniprocessing System



Multiprocessing System

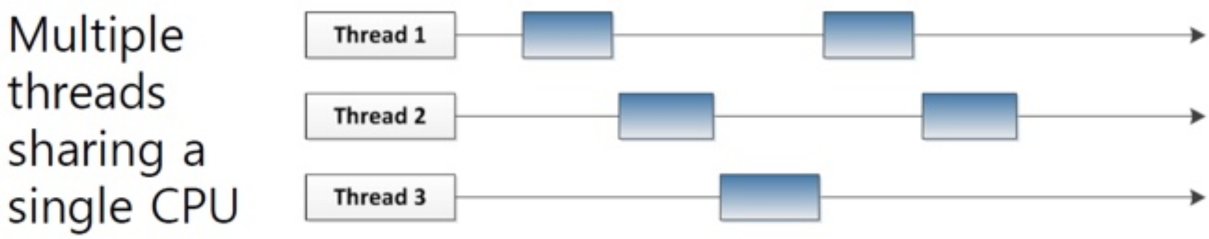


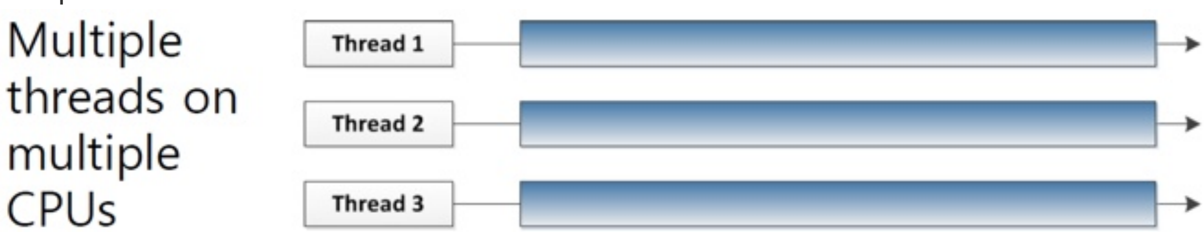
Parallelism

- executing several computations simultaneously to gain performance
- different forms of parallelism
  - **multitasking**
    - several processes are scheduled alternately or possibly simultaneously on a multiprocessing system
  - **multithreading**
    - same job is broken logically into pieces (threads) which may be executed simultaneously on

a multiprocessing system

What is a thread?

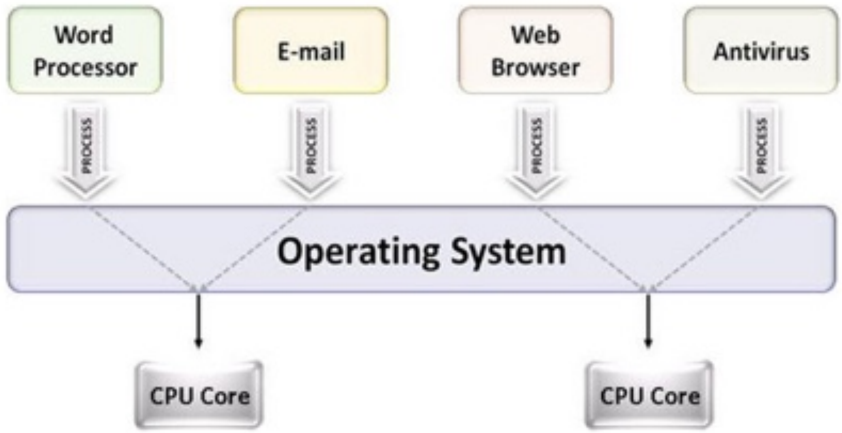
- a flow of instructions, path of execution within a process
- the smallest unit of processing scheduled by OS
- a process consists of *at least one* thread
- multiple threads can be run on:
    - **a uniprocessor (time-sharing)**
        - processor switches between different threads
        - switch back and forth, giving each thread a certain amount of time or CPU clock cycles to run
        - parallelism is an illusion
        - 

    - **a multiprocessor**
        - multiple processors or cores run the threads at the same time
        - true parallelism
        - 

Multitasking vs. Multithreading

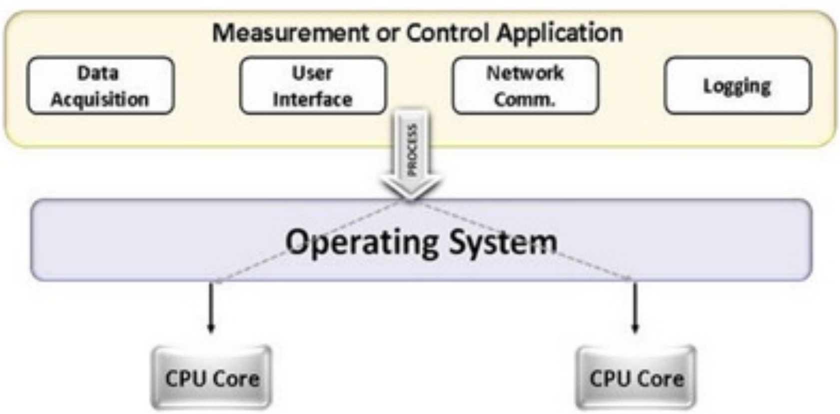
Multitasking

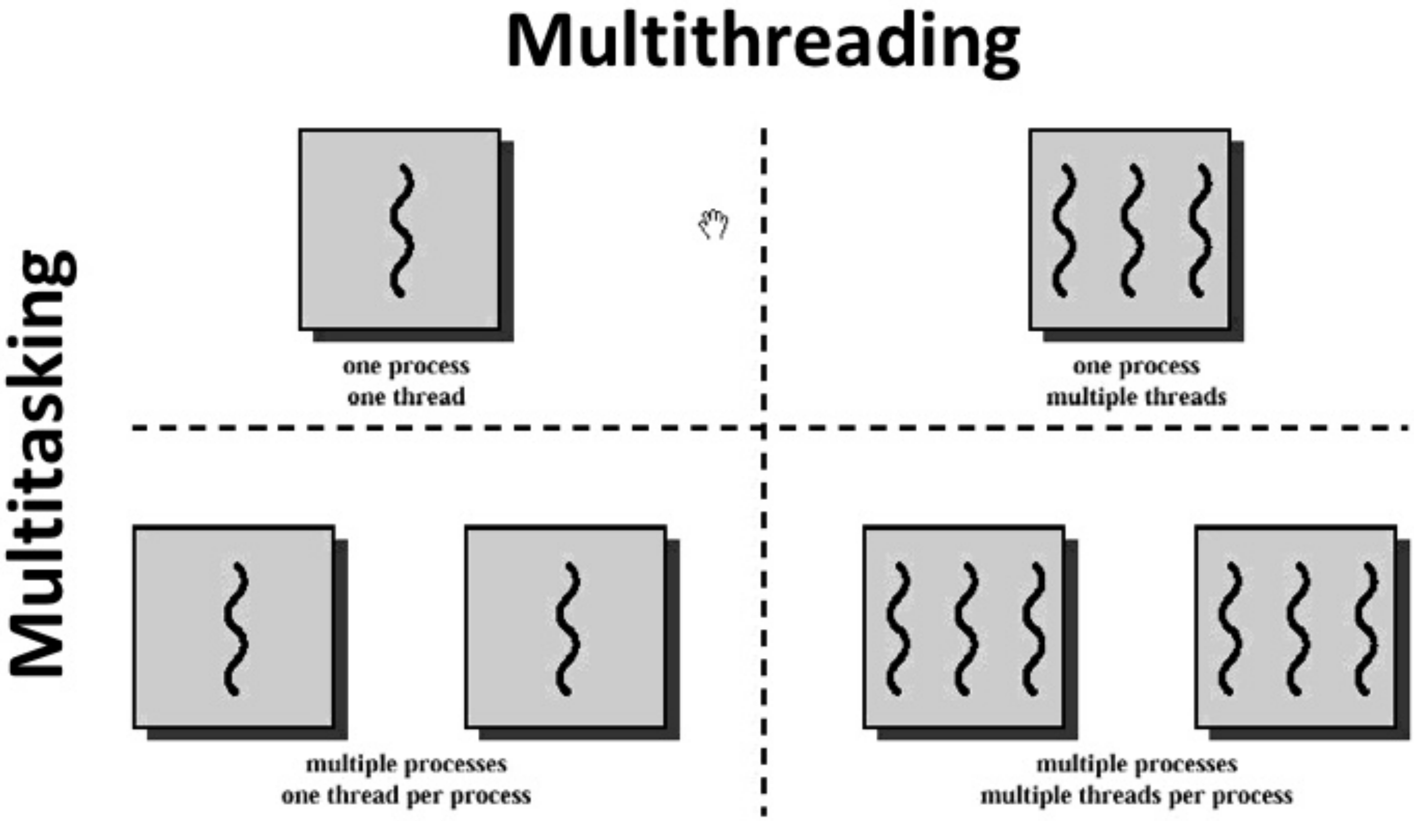
- multiple applications using multiple cores



Multithreading

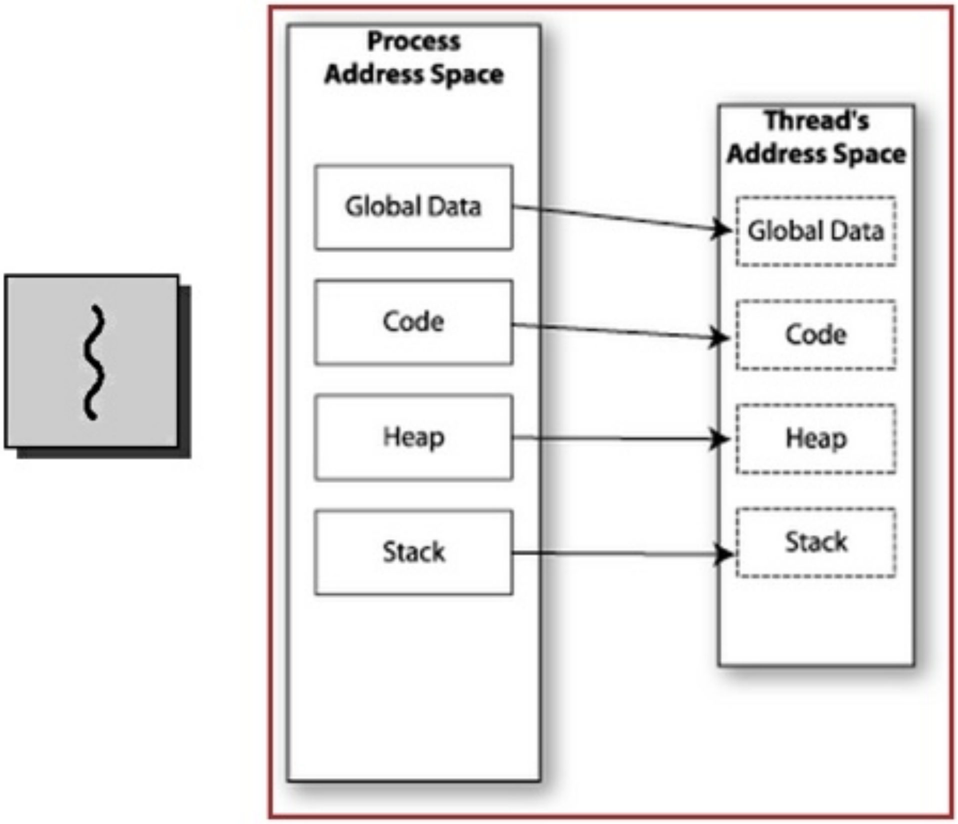
- single application using multiple cores for different threads



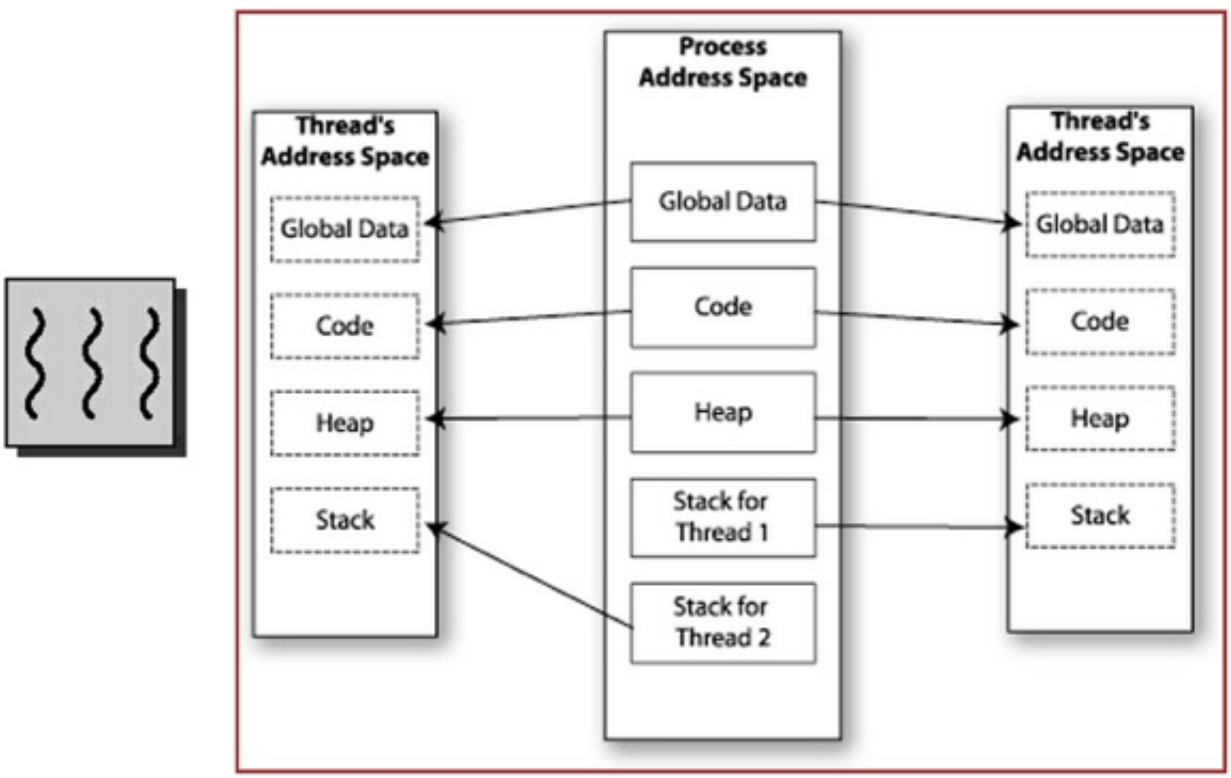


Memory Layout

Single-Threaded Program



Multithreaded Program



Multitasking

- `$ tr −cs ‘A−Za−z’ ‘[\n∗]’ sort −u | comm −23 − words`

    - process 1 (tr)
    - process 2 (sort)
    - process 3 (comm)
- each process has its own address space
- how do these processes communicate?
    - pipes/system calls

Multithreading

- threads share all of the process’s memory except for their stacks
- data sharing requires no extra work (no syscalls, pipes, etc.)

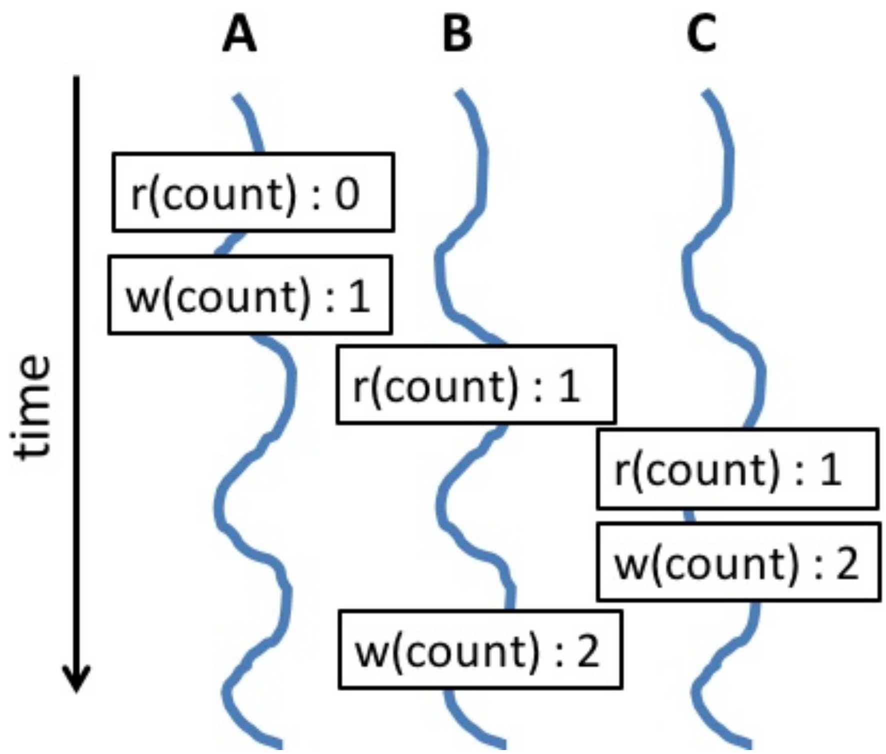
Shared Memory

- makes multithreaded programming…
    - **powerful**
        - can easily access data and share it among threads
    - **more efficient**
        - no need for syscalls when sharing data
        - thread creation and destruction less expensive than process creation and destruction
    - **non-trivial**
        - have to prevent several threads from accessing and changing the same shared data at the same time (synchronization)

Race Condition

```
int count = 0;
void increment() {
    count = count + 1;
}

int main() {
    increment();
    increment();
    increment();
}
```



- if we multithread this program across 3 threads, how the OS schedules the calls to increment makes a huge difference
  - if **thread C** executes between **thread B**'s read from count and its write to count, **thread B** will incorrectly "increment" the value of `count` to the same number it already was
- **synchronization needed**
  - each thread will "lock" each variable it modifies, preventing other threads from modifying the variable while it is being modified by the original thread
  - the thread will "unlock" that variable when it is done with it, freeing the variable to be modified by other threads

Multithreading & Multitasking: Comparison

Multithreading

- threads share same address space
  - light-weight creation/destruction
  - easy inter-thread communication
  - an error in one thread can bring down all threads in process
    - e.g. accessing a nullptr in one thread brings down entire process

Multitasking

- processes are insulated from each other
  - expensive creation/destruction
  - expensive IPC (inter-process communication)
  - an error in one process cannot bring down another process

        - but results may not be as expected

Lab 8

- evaluate the performance of multithreaded sort
- add `/usr/local/cs/bin` to `PATH`
    - `$ export PATH=/usr/local/cs/bin:$PATH`
- generate a file containing 10M random **double-precision floating point numbers**, one per line with no white space
    - `/dev/urandom`: pseudo-random number generator
- `od`
    - write the contents of its input files to stdout in a user-specified format
    - options
        - `-t f`: double-precision floating point
        - `-N <count>`: format no more than count bytes of input
- `sed, tr`
    - remove address, delete spaces, add newlines between each float
- use `time -p` to time the command `sort -g` on the data you generated
- send output to `/dev/null`
- run sort with the `--parallel` option and the
    - `-g` option: compare by general numerical value
    - use time command to record the real, user and system time when running sort with 1, 2, 4, 8 threads

```
$ time -p sort -g file_name > /dev/null (1 thread)
$ time -p sort -g --parallel=[2, 4, or 8] file_name > /dev/null
```

- record the times and steps in `log.txt`