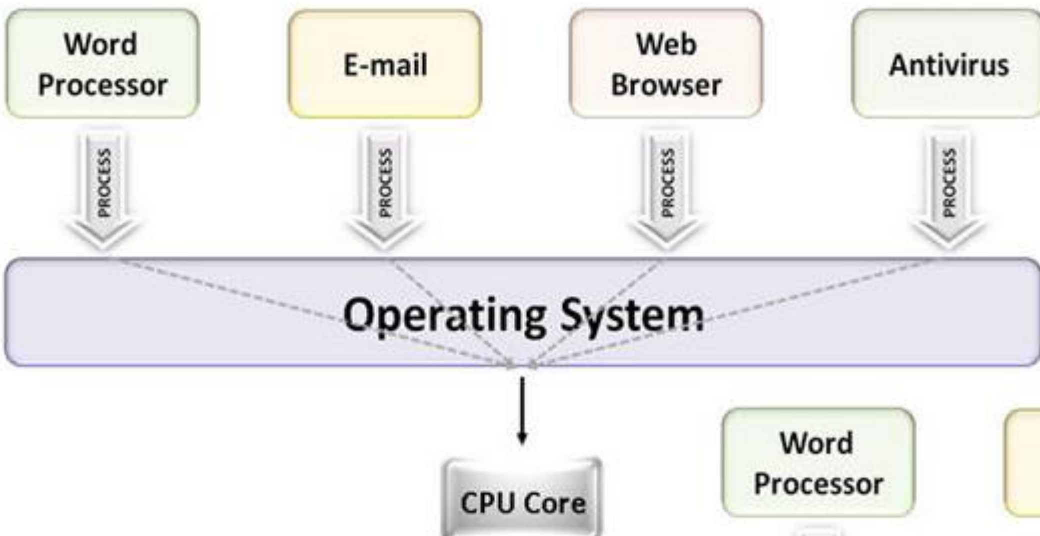
Multithreaded Performance

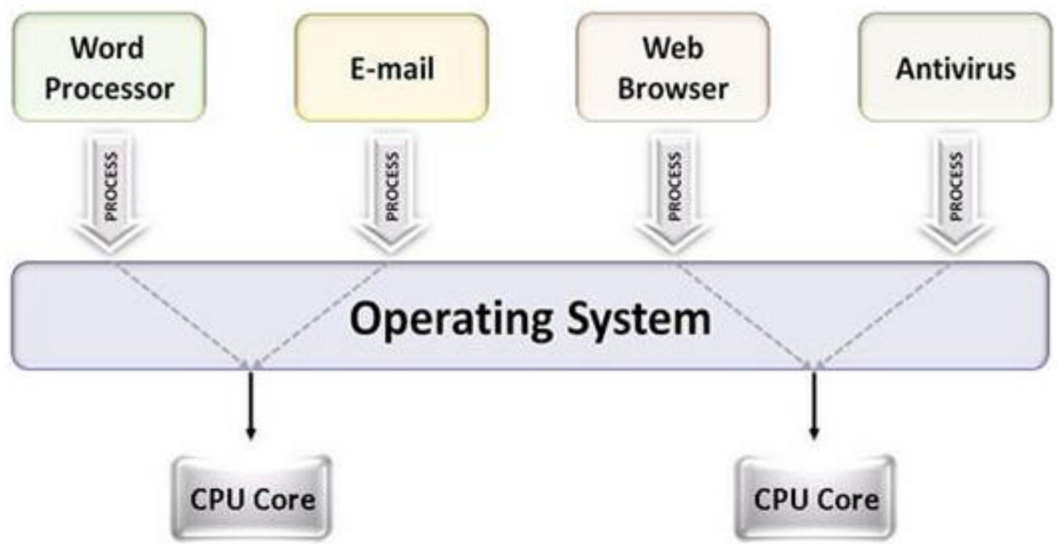
Week 8

Multiprocessing

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



Uniprocessing system

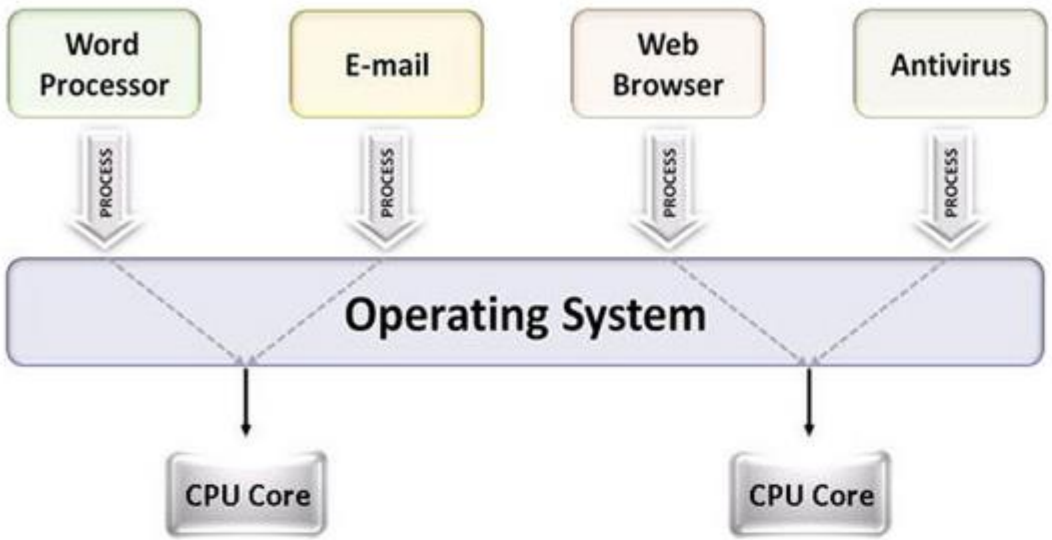


Multiprocessing system

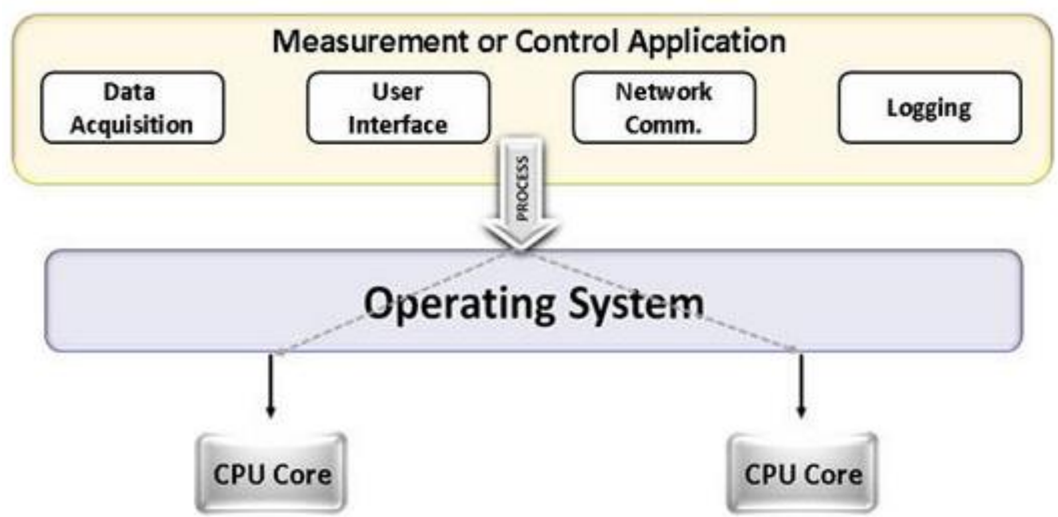
Parallelism

- 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

Multitasking vs. Multithreading

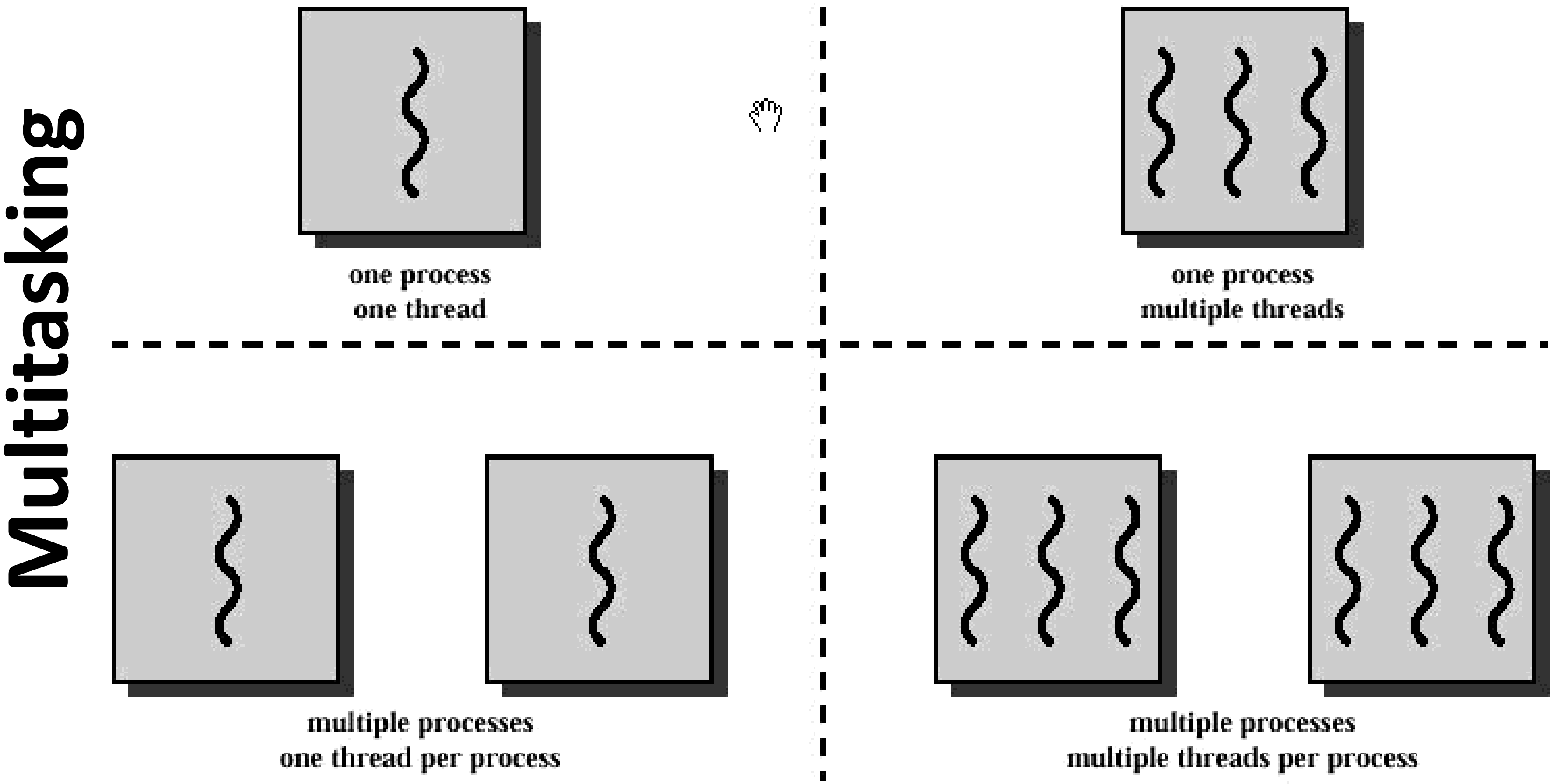


Multitasking



Multithreading

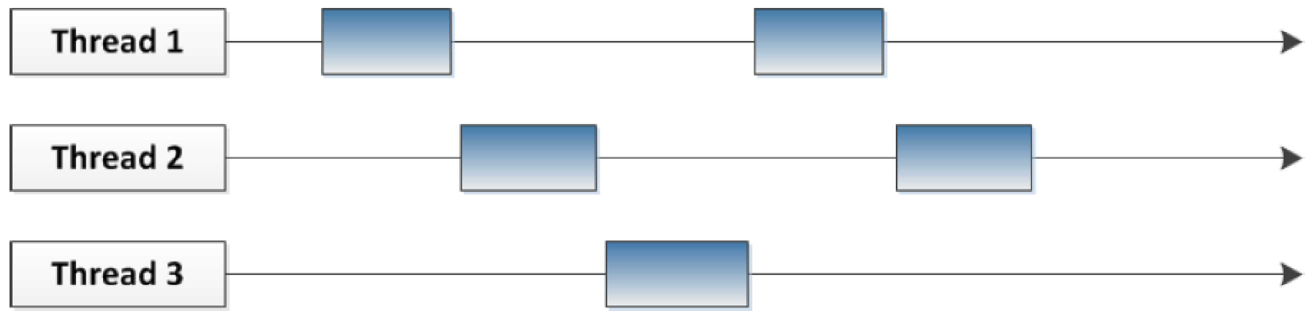
Multithreading



What is a thread?

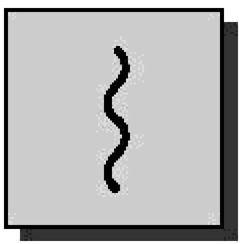
- A flow of instructions, path of execution within a process
- The smallest unit of processing scheduled by OS
- A process consists of at least one thread
- Multiple threads can be run on:
  - **A uniprocessor (time-sharing)**
    - Processor switches between different threads
    - Virtual parallelism (concurrency)
  - **A multiprocessor**
    - Multiple processors or cores run the threads at the same time
    - True parallelism

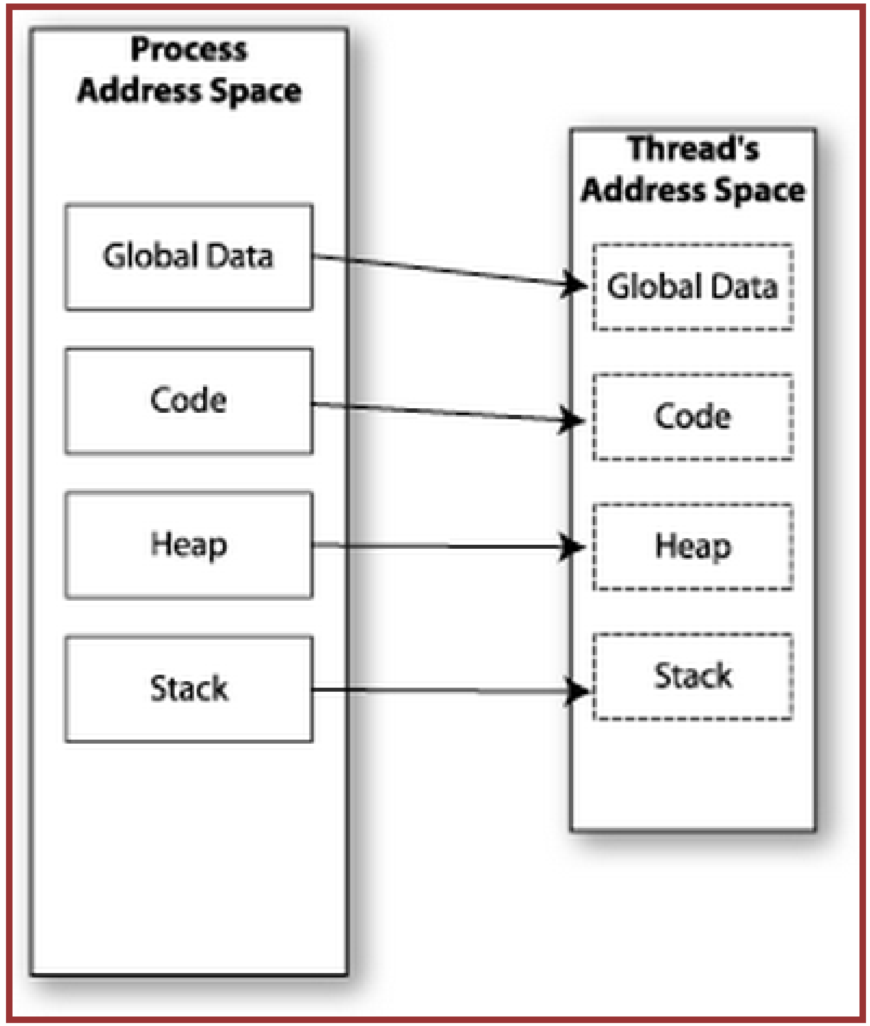
Multiple threads sharing a single CPU



Multiple threads on multiple CPUs

Memory Layout: Single-Threaded Program

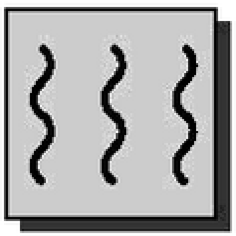


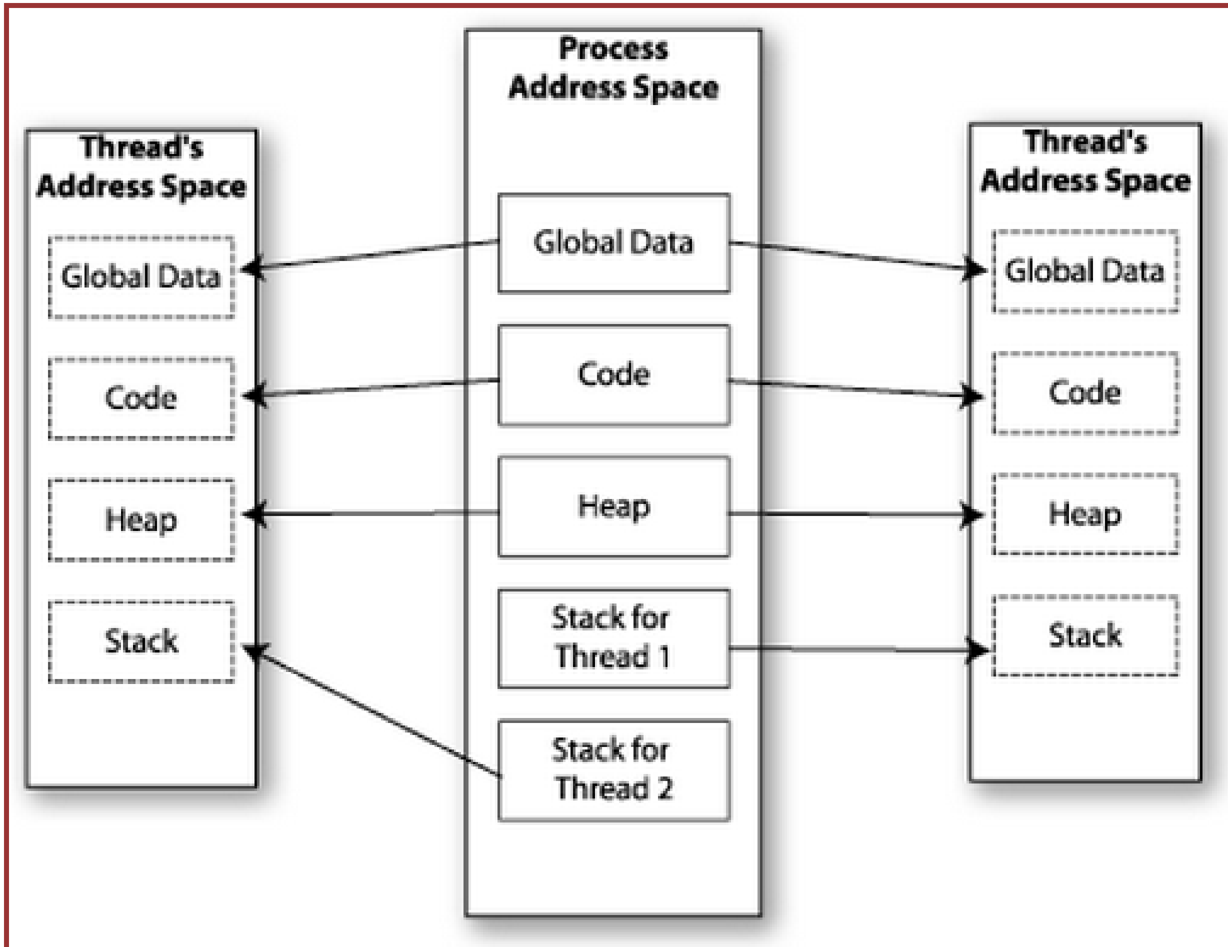


Multitasking

- $ 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

Memory Layout: Multithreaded Program





Multithreading

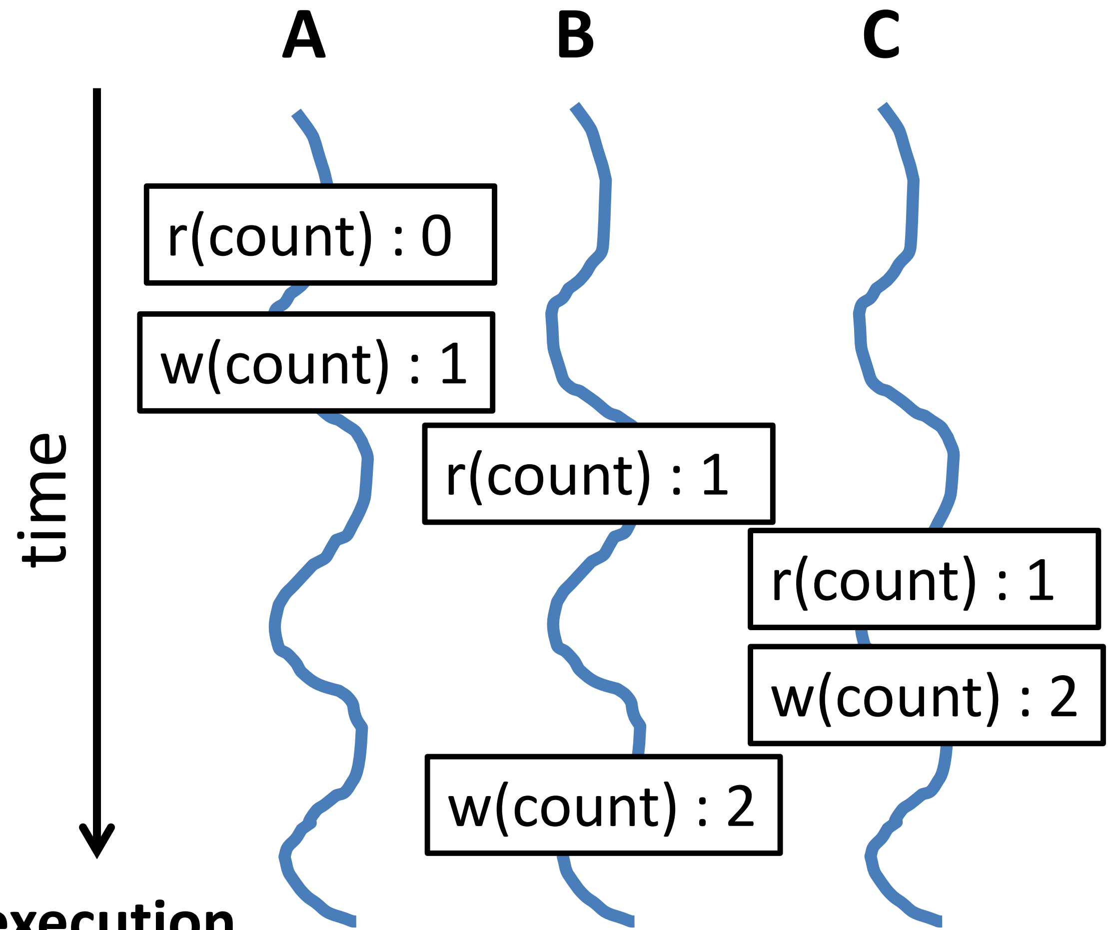
- Threads share all of the process's memory except for their stacks
- => Data sharing requires no extra work (no system calls, pipes, etc.)

Shared Memory

- Makes multithreaded programming
  - **Powerful**
    - can easily access data and share it among threads
  - **More efficient**
    - No need for system calls 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;
}
```



Result depends on order of execution

=> Synchronization needed

Multithreading & Multitasking: Comparison

- **Multithreading**
  - Threads share the same address space
    - Light-weight creation/destruction
    - Easy inter-thread communication
    - An error in one thread can bring down all threads in process
- **Multitasking**
  - Processes are insulated from each other
    - Expensive creation/destruction
    - Expensive inter-process communication (IPC)
    - An error in one process cannot bring down another process

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 byte generator

Lab 8

- `od`
  - write the contents of its input files to standard output 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

Lab 8

- 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: general numeric value
  - Use `time` command to record the real, user and system time when running sort with 1, 2, 4, and 8 threads
    - $ time –p sort –g --parallel=[1, 2, 4, 8] 10M_file > /dev/null
  - Record the times and steps in log.txt