CS 35L

Software Construction Laboratory

Lecture 6.1

12th February, 2019

Logistics

- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
- Presentations for Assignment 10
  - https://docs.google.com/spreadsheets/d/1o6r6CKCaB2du3klPflHiquymhBvbn7oP0wkHHMz_q1E/edit?usp=sharing
- Assignment 5 is due on 16th Feb, 2018 at 11:55pm

Review - Previous Lab

- System Call Examples
  - Open, Create, Close
  - Read, Write
- Why System Calls?

Assignment 5 - Laboratory

- Review:
  - ch = getchar()
  - putchar(ch)
  - int numRead = read(STDIN_FILENO, ch, size)
  - int numWritten = write(STDOUT_FILENO, ch, size)

Assignment 5 - Laboratory

- Write tr2b and tr2u programs in 'C' that transliterates bytes. They take two arguments 'from' and 'to'. The programs will transliterate every byte in 'from' to corresponding byte in 'to'
  - ./tr2b 'abcd' 'wxyz' < bigfile.txt
    - Replace 'a' with 'w', 'b' with 'x', etc
  - ./tr2b 'mno' 'pqr' < bigfile.txt
- tr2b uses getchar and putchar to read from STDIN and write to STDOUT.
- tr2u uses read and write to read and write each byte, instead of using getchar and putchar. The nbyte argument should be 1 so it reads/writes a single byte at a time.
- Test it on a big file with 5,000,000 bytes
  - $ head --bytes=# /dev/urandom > bigfile.txt

Tr2b.c

- Write a main function which accepts arguments
  - main(int argc, const char* argv[])
- Check for the length of arguments
  - Retrieve first argument in char * from, second argument in char * to
  - Compare the lengths of from and to; If not same, throw an error and exit
  - You can use strlen to get lengths
- To throw an error, write to stderr
- To exit, write exit(1)
- Check if ‘from’ has duplicates
  - In a loop, take input from stdin (till you reach eof of stdin) using getchar()
  - Check if the character you just retrieved is a part of from; if yes then put the corresponding character in stdout with putchar()

Tr2u.c

- Repeat the same procedure as in tr2b.c except replace:
  - getchar() with read
  - putchar() with write

Time and *strace*

- **time [options] command [arguments...]**
- Output:
  - -real 0m4.866s: elapsed time as read from a wall clock
  - -user 0m0.001s: the CPU time used by your process
  - -sys 0m0.021s: the CPU time used by the system on behalf of your process
- **strace: intercepts and prints out system calls.**
- -$ strace -c ./tr2b 'AB' 'XY' < input.txt
- -$ strace -c ./tr2u 'AB' 'XY' < input.txt

Additional Information

- www.cs.uregina.ca/Links/class-info/330/SystemCall_IO/SystemCall_IO.html
- courses.engr.illinois.edu/cs241/sp2009/Lectures/04-syscalls.pdf
- www.bottomupcs.com/system_calls.xhtml

Assignment 5 - Homework

- Rewrite sfrob using system calls (sfrobu)
- sfrobu should behave like sfrob except:
  - If stdin is a regular file, it should initially allocate enough memory to hold all data in the file all at once
  - It outputs a line with the number of comparisons performed
- Functions you'll need: *read*, *write*, and *fstat* (read the man pages)
- Measure differences in performance between sfrob and sfrobu using the *time* command
- Estimate the number of comparisons as a function of the number of input lines provided to sfrobu

Assignment 5 - Homework

- Write a shell script “sfrobs” that uses tr and the sort utility to perform the same overall operation as sfrobu (support –f option as well)
- Use pipelines (do not create temporary files)
- Encrypted input -> tr (decrypt) -> sort (sort decrypted text) -> tr (encrypt) -> encrypted output

Assignment 5 - Homework

- Measure any differences in performance between sfrob and sfrobu using the time command.
- Run your program on inputs of varying numbers of input lines, and estimate the number of comparisons as a function of the number of input lines
- Use the time command to compare the overall performance of sfrob, sfrobu, sfrobs, sfrobu –f and sfrobs -f

Assignment 5 - Homework

- Run your program on inputs of varying numbers of input lines, and estimate the number of comparisons as a function of the number of input lines.
- Varying number of input lines => number of words
- Number of comparisons => keep a counter in the frobcmp() function to check how many times it is being called

Assignment 5 - Homework

- Refer *to Read, Write, Open, Close* System Calls
- Reserved File Descriptors
  - 0 – stdin
  - 1 – stdout
  - 2 – stderr
- int fstat(int fd, struct stat *buf)
  - Returns information about the file with the descriptor fd into buf

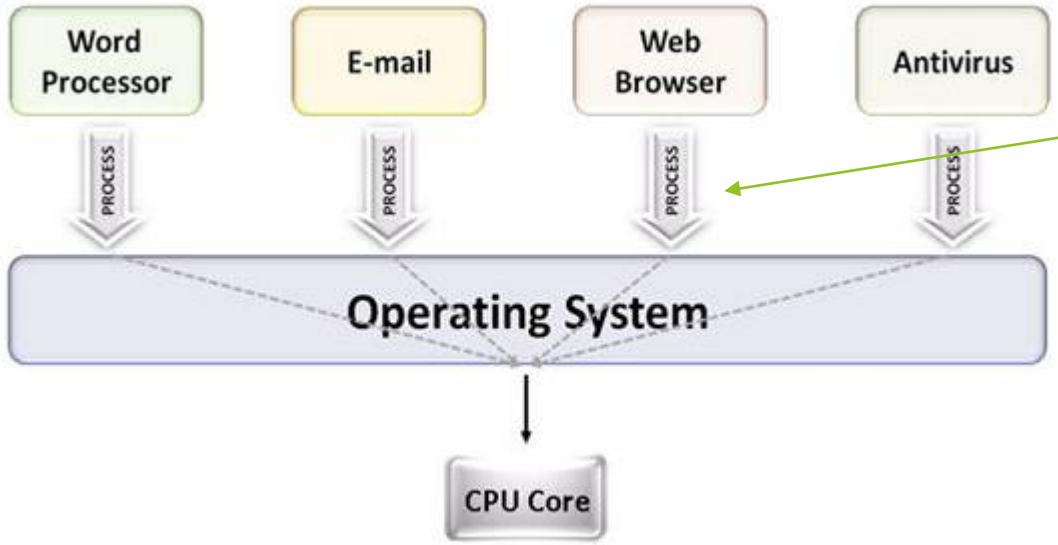
Parallelism and Multi Threading

Types of Resources

- CPU
  - It is an active resource
  - Can be used by only one runtime entity
  - Can be multiplexed in time (time sharing)
- Memory
  - Passive resource
  - Can be shared among multiple runtime entities
  - Can be multiplexed in space (allocated)

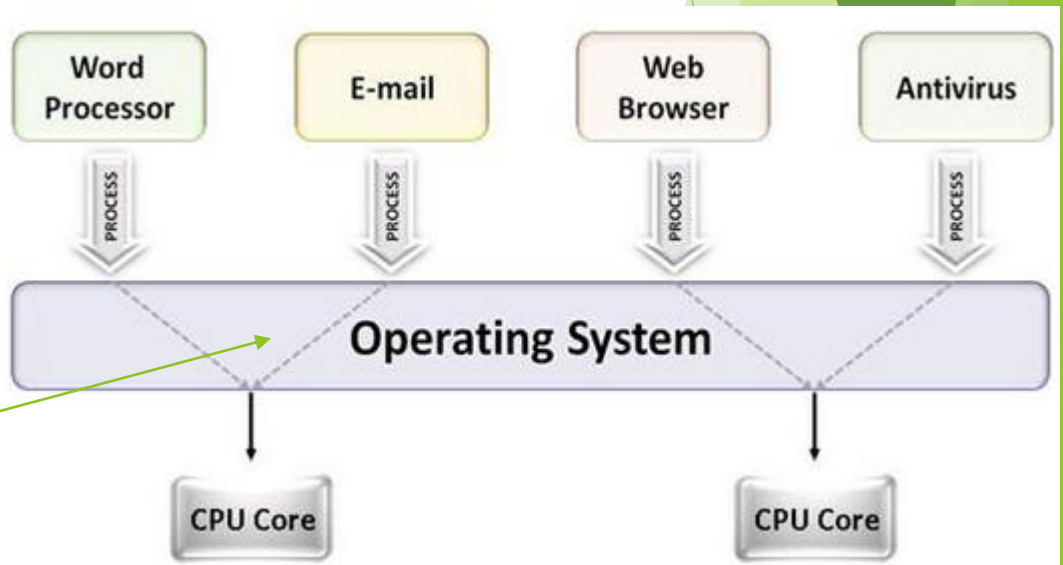
Multiprocessing

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



Uni-processing 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

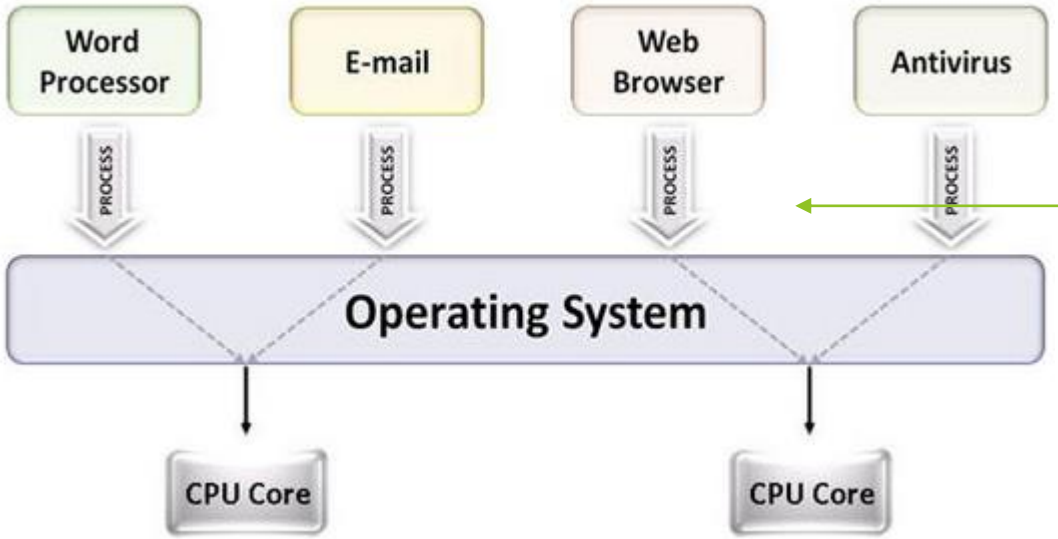
What is a Thread?

- 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
    - Parallelism is an illusion
- A multiprocessor
  - Multiple processors or cores run the threads at the same time
  - True parallelism

Process vs Threads

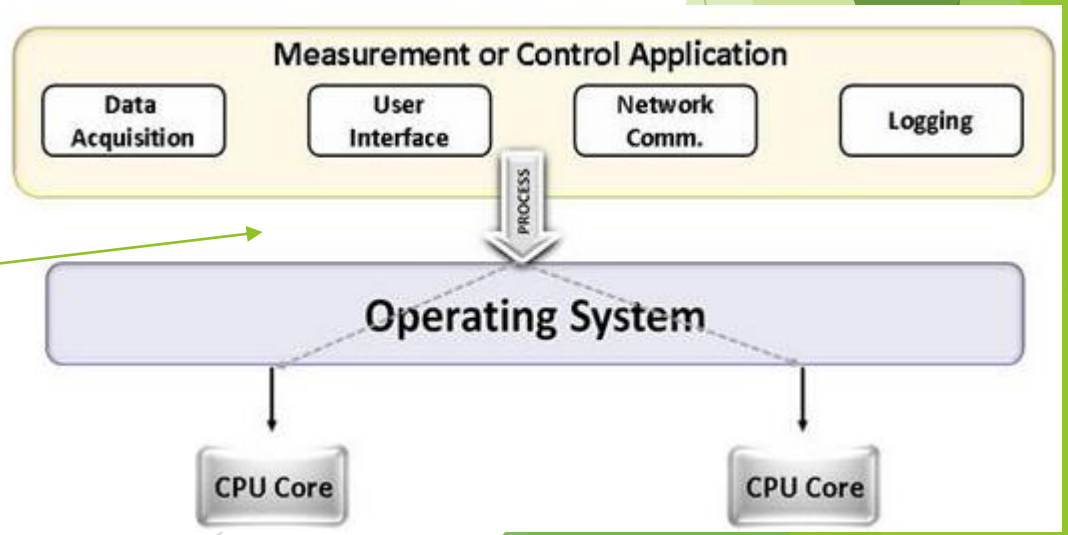
- Different processes see separate address spaces
  - good for protection, bad for sharing
- All threads in the same process share the same memory (except stack)
  - good for sharing, bad for protection
  - each thread can access the data of other thread

Multitasking vs. Multithreading

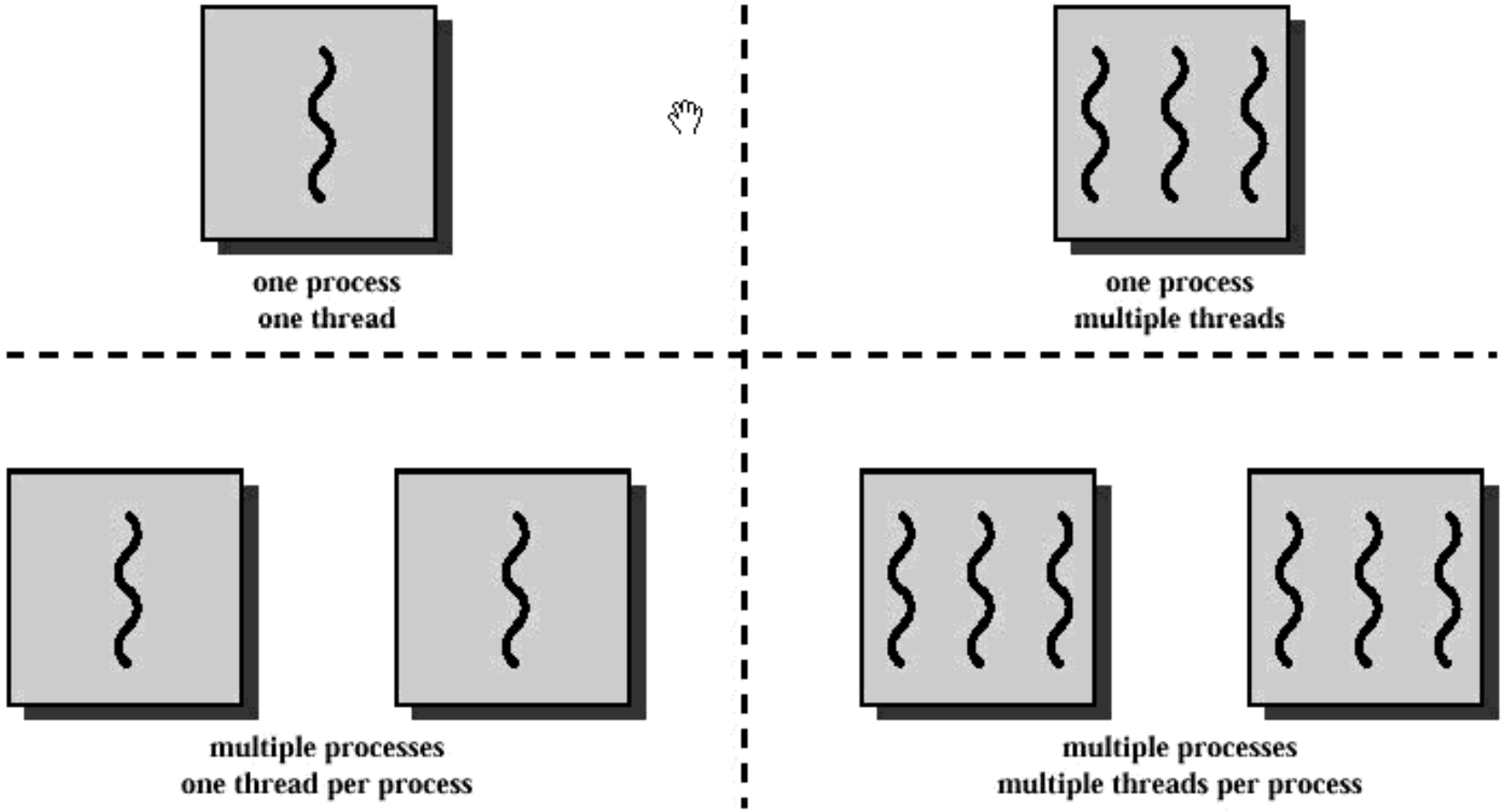


Multitasking

Multithreading



Multithreading

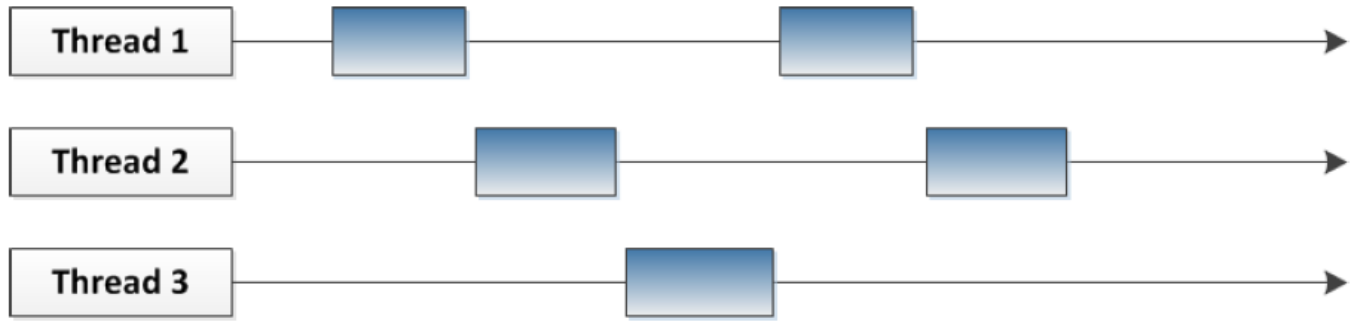


Multithreading & Multitasking: Comparison

- 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
  - An error in one process cannot bring down another process

Multithreading & Multitasking

Multiple threads sharing a single CPU



Multiple threads on multiple CPUs

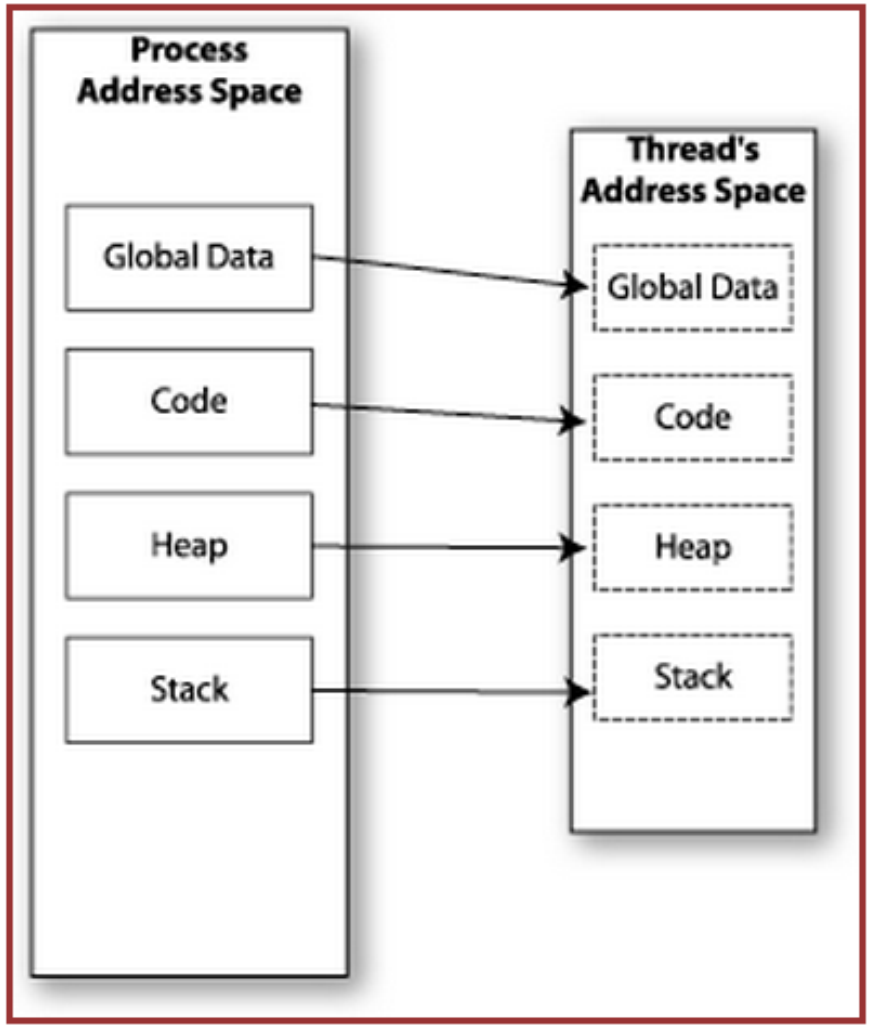
Multitasking

- $ tr ’abc’ ‘xyz’ | sort -u | comm -23 file1 -
  - Process 1 (tr)
  - Process 2 (sort)
  - Process 3 (comm)
- Each process has its own address space
- How do these processes communicate?
  - Pipes/System Calls

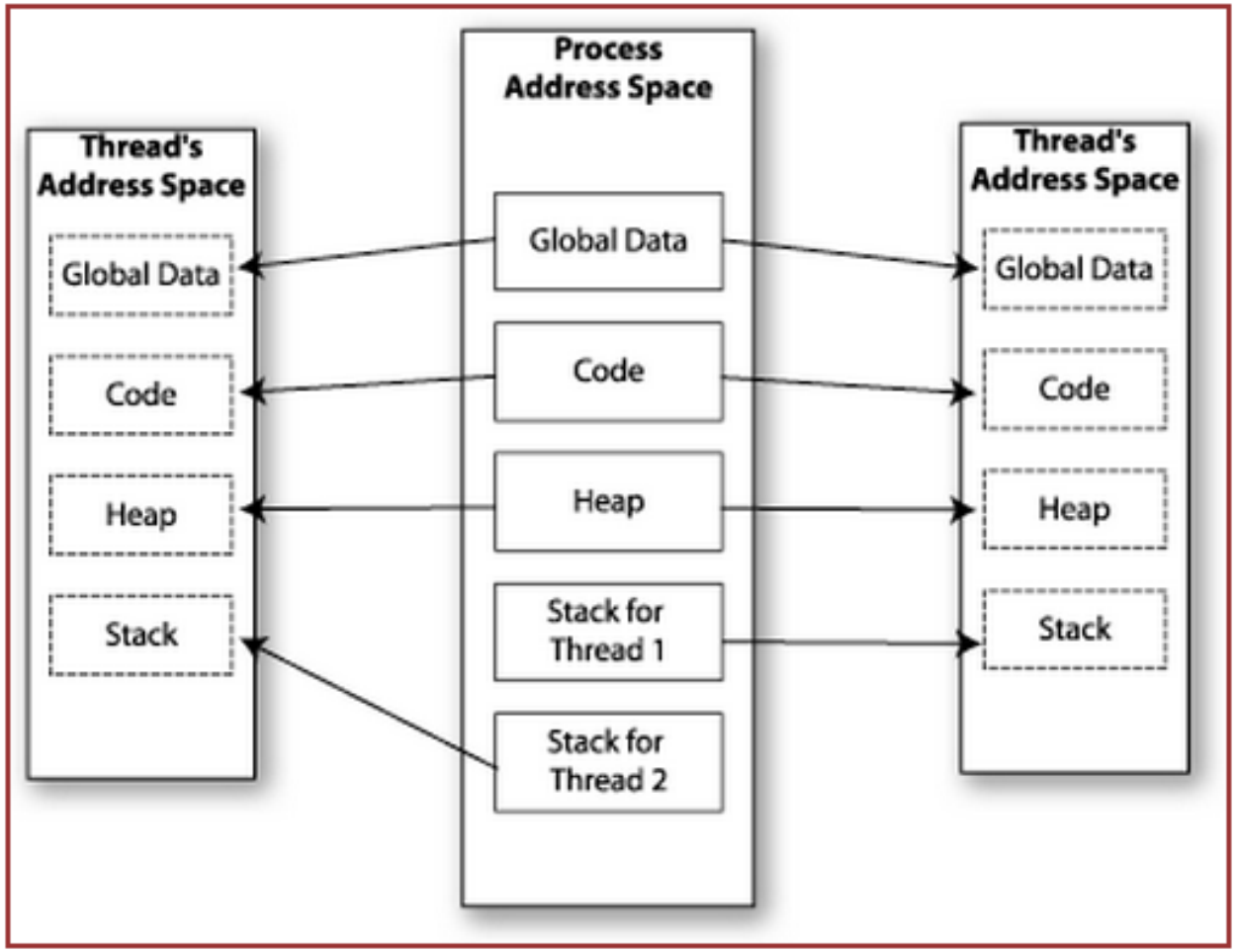
Multithreading

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

Memory Layout: Single-Threaded Program



Memory Layout – Multi Threaded Program



Shared Memory

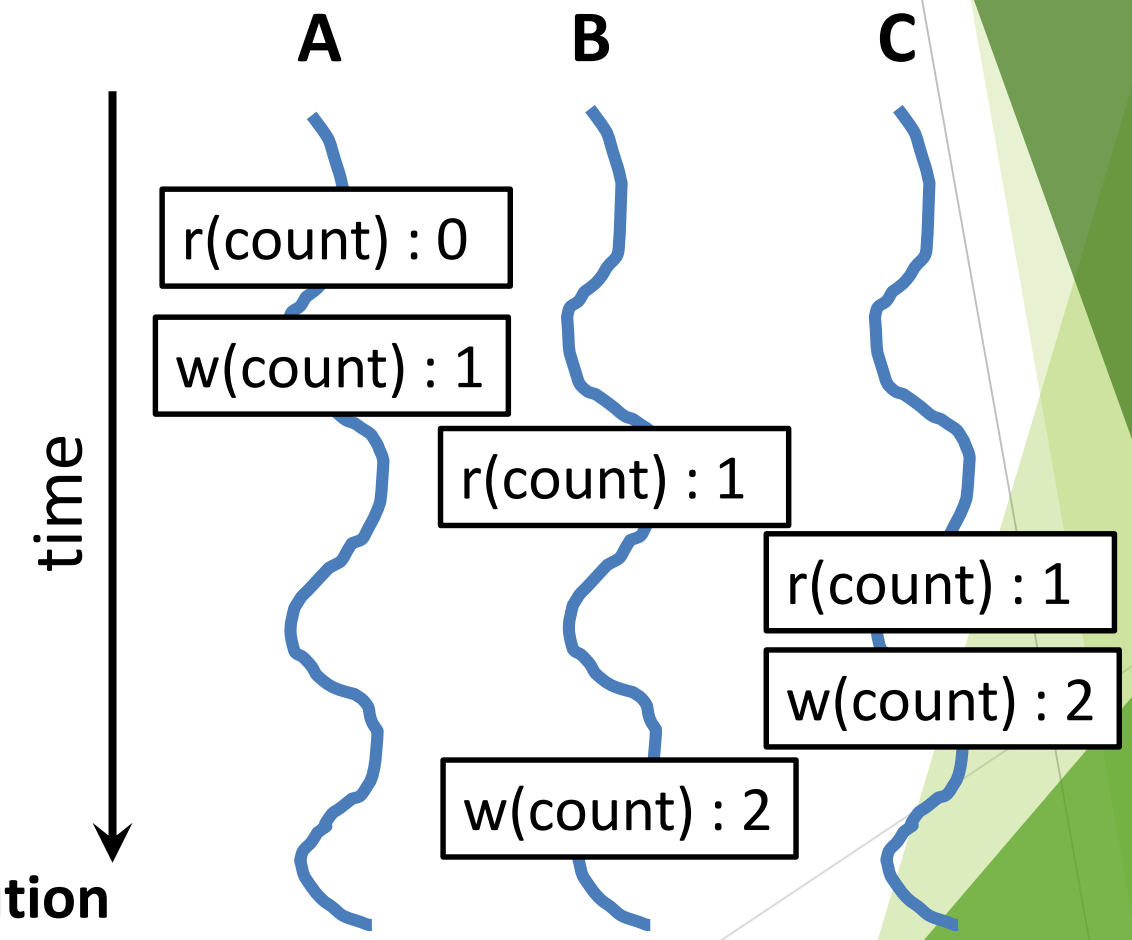
- Makes multithreaded programming
- Powerful
  - can easily access data and share it among threads
- More efficient
  - No need for system calls when sharing data
  - Thread creation and destruction less expensive than process creation and destruction
- Non-trivial
  - Have to prevent several threads from accessing and changing the same shared data at the same time (synchronization)

Process/Thread Synchronization

- Why is it needed?
- Because threads share the same resources, we need synchronization
- To prevent inconsistency

Race Condition

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



Result depends on order of execution

=> Synchronization needed

Presentations

- Todays Presentation:
  - Richard Cheng
  - Ethan Kwan
- Next up:
  - Niyant Narang
  - Rio Sonoyama

Questions?