Week 07M: Multithreaded Programming

Thuy Vu

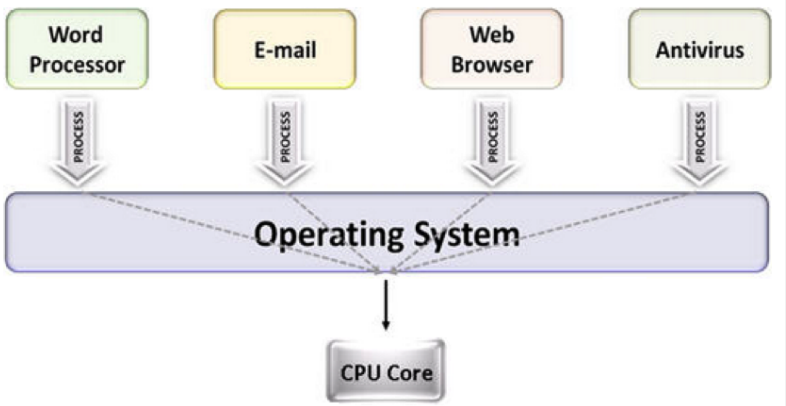
Reminders

- Assignment 6
  - `web.cs.ucla.edu/classes/winter17/cs35L/assign/assign6.html`
  - Time due 23:55 this Friday, February 24

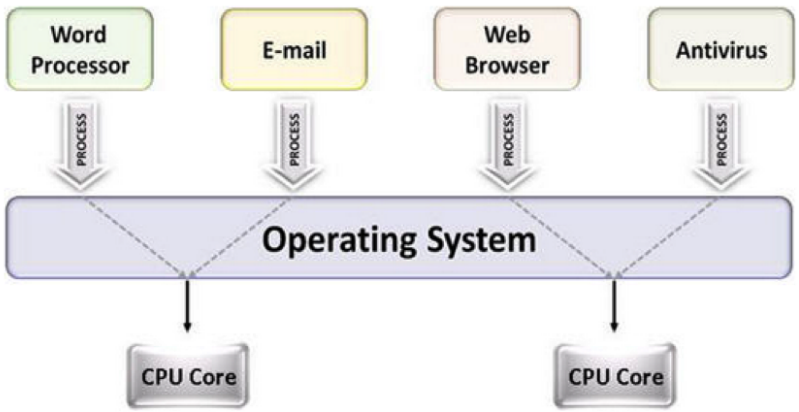
Week 7: Assignment #6 – Multithreaded Performance
Week 8: Assignment #7 – Dynamic Linking
Week 9: Assignment #8 – SSH Setup and Use in Applications
Week 10: Assignment #9 – Change management

Multi-processing vs Uni-processing

Parallelism is to execute several computations simultaneously



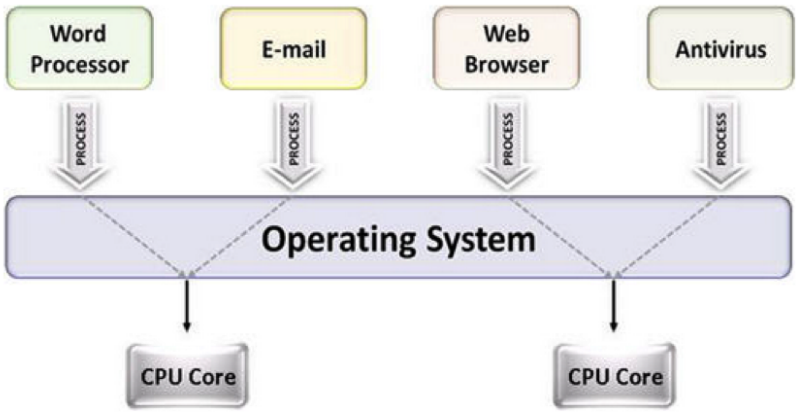
Uni-processing



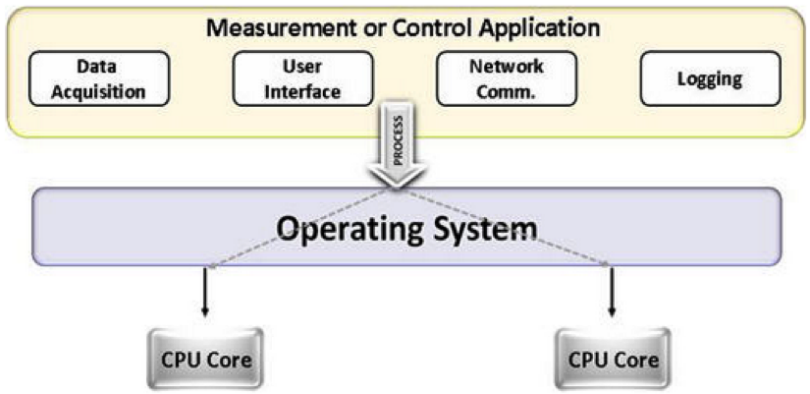
Multi-processing

Multi-tasking vs Multi-threading

Parallelism is to execute several computations simultaneously



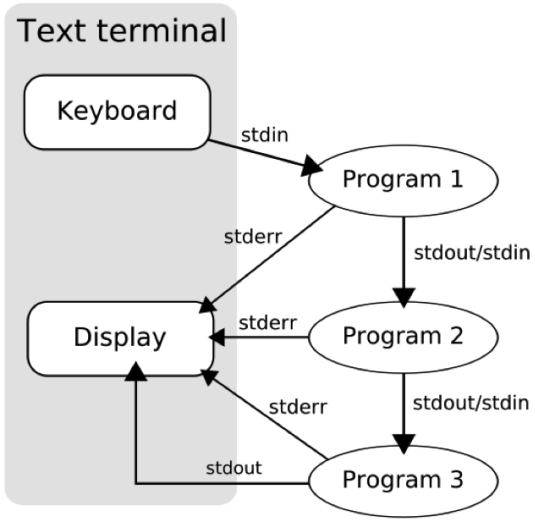
Multitasking – several processes are scheduled alternately or possibly simultaneously on a multiprocessing system



Multithreading – same job is broken logically into pieces (threads) which may be executed simultaneously on a multiprocessing system

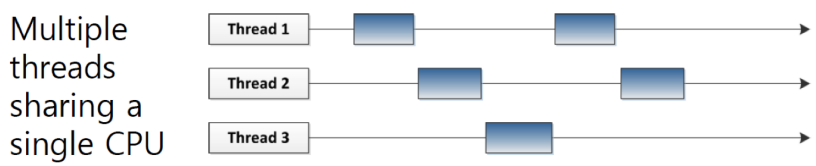
- **Process** – each process has it own address space

```
tr 'A-Z' 'a-z' | sort | comm -23 - f
```

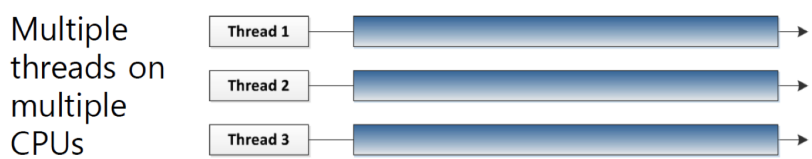


- Process 1: `tr`
- Process 2: `sort`
- Process 3: `comm`

- **Thread** – flow of instructions in a process
  - the smallest unit of processing scheduled by OS
  - a process = a main thread + other threads
  - multi threads can be running on:



one processor – switching between different threads → pseudo parallelism



multi processors – threads run on different cores → true parallelism

- all threads share the same address space, but have private stacks

shared memory in threads

- **powerful** – easily access and share among threads
- **efficient** – easy to spawn/kill, no need system calls when sharing data
- **non-trivial** – race condition $\rightarrow$ needs synchronization

thread APIs include

- `pthread_create` – creates a new thread within a process
- `pthread_join` – waits for another thread to terminate
- `pthread_equal` – compares thread ids
- `pthread_self` – returns the id of the calling thread
- `pthread_exit` – terminates the currently running thread

the lab

1. **check version**
   - `sort --version`
   - `export PATH=/usr/local/cs/bin/:$PATH`
2. **generate file** of 10M random double-precision floating point numbers
   - `/dev/urandom`
   - **od** – to write an input file to standard output in format
     - `-t f` – double-precision floating point
     - `-N <count>` – format no more than `<count>` bytes of input
   - **sed, tr** – clean up
3. `time -p` on command `sort -g` and send output to `/dev/null`
   - `-g` ∼ general numeric values
   - `--parallel=...` ∼ n-threads

the homework

- `int pthread_create(pthread_t *thread,`
  `const pthread_attr_t *attr,`
  `void *(*start_routine)(void*),`
  `void *arg);`
  - create a new thread with handle `thread` with attribute `attr` to execute `start_routine` with `arg` as its sole argument

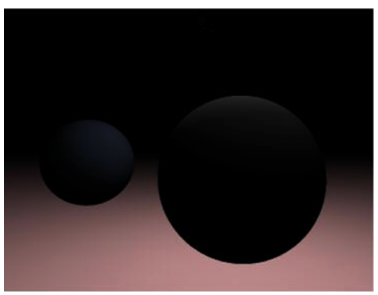
- `int pthread_join(pthread_t thread, void **value_ptr);`
  - suspend execution of the calling thread until the target thread terminates, unless the target thread has already terminated
  - what can go wrong without calling `pthread_join`?

POSIX Threads Programming – An Example

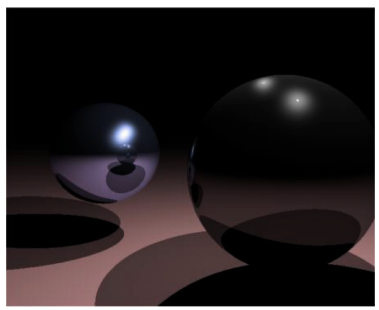
...

1. **Ray Tracing**

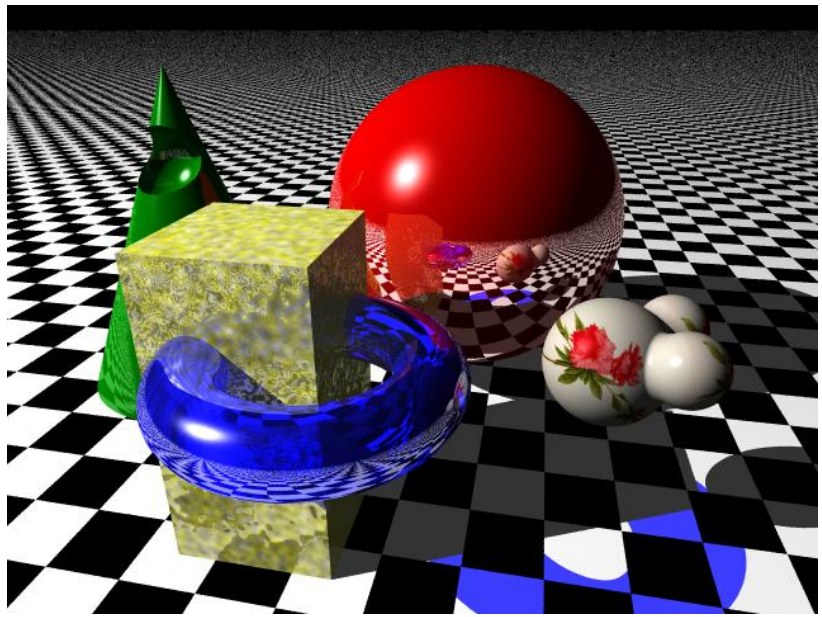
- a technique to render lights in 3D
- light rays to bounce from object to object



with-out ray-tracing



with ray-tracing



$\rightarrow$highly computationally intensive

2. **write a multi-threaded version!**