CS 35L

LAB 8,

TA: Sucharitha Prabhakar

EMAIL ID: prabhakarsucharitha@gmail.com

Outline

Multiprocessing

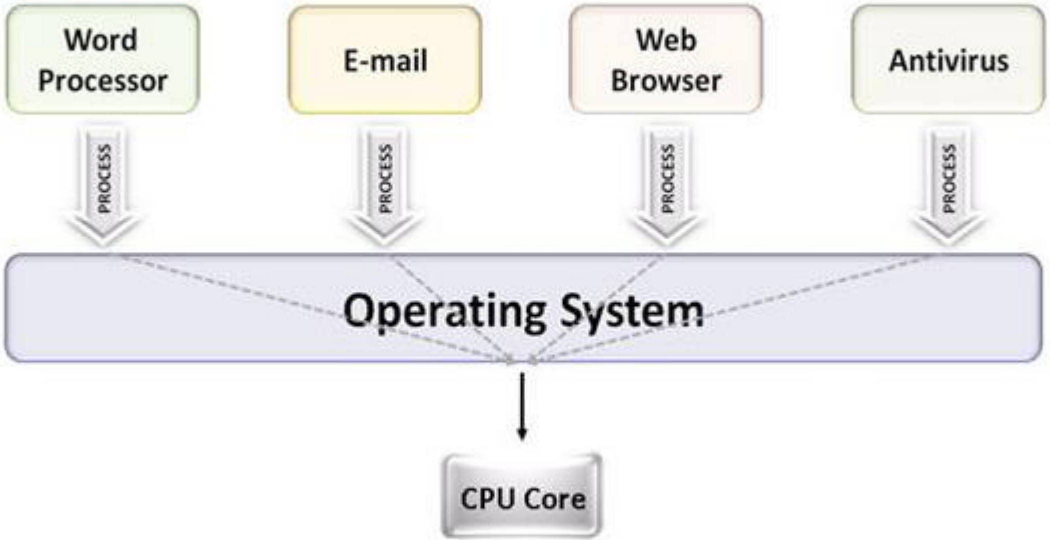
Parallelism

Multithreading

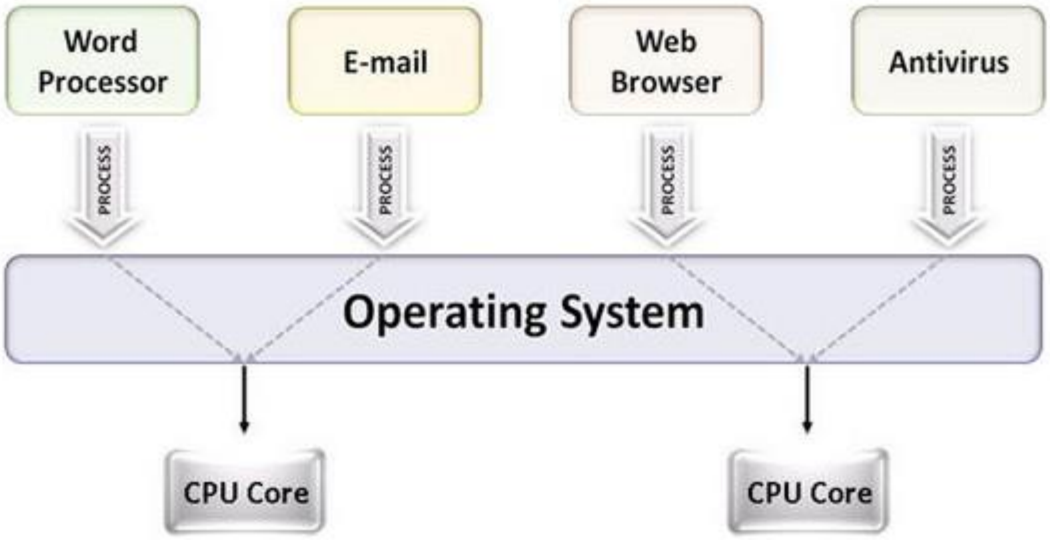
Multiprocessing

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

Uniprocessing



Multiprocessing



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

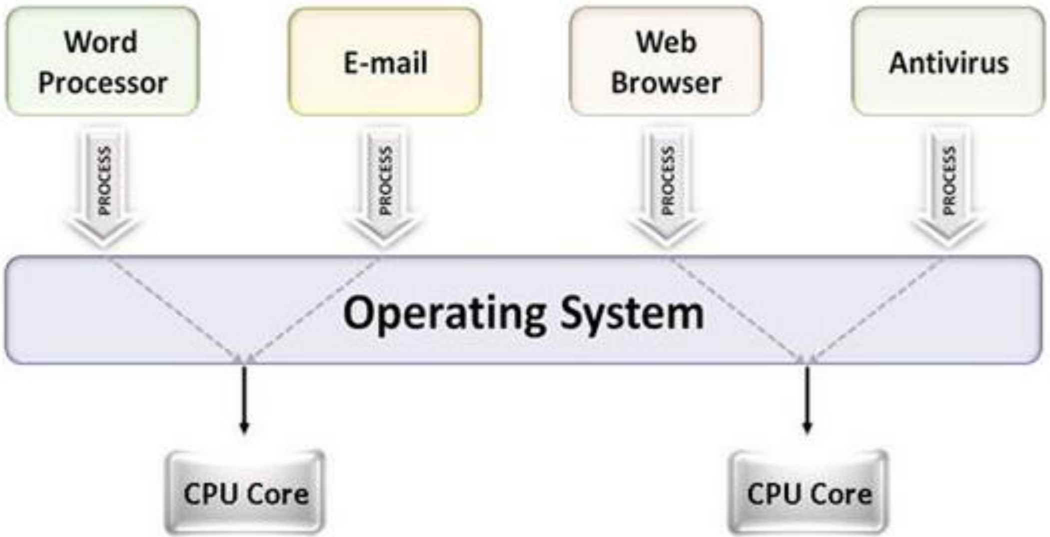
What is a thread?

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

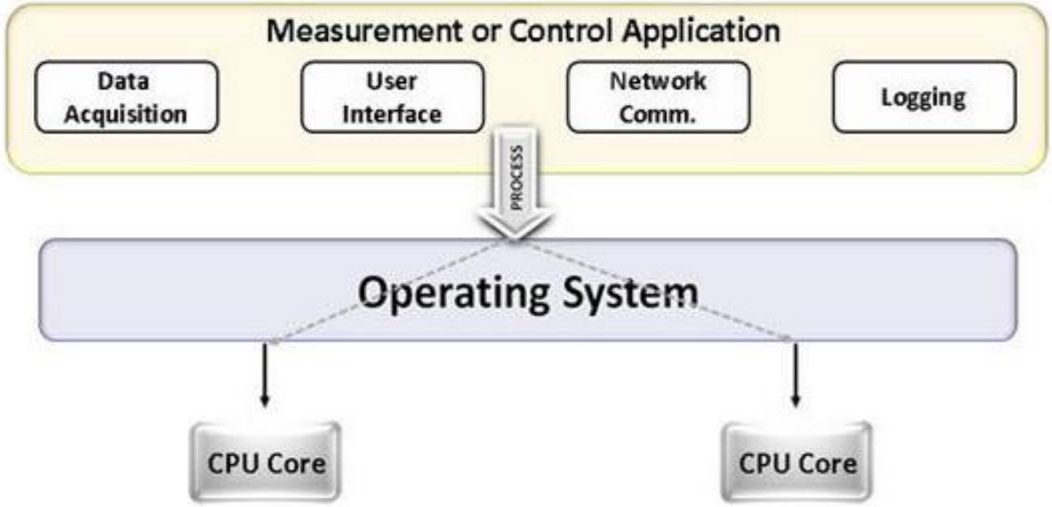
Multiple threads can be run on:

- 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

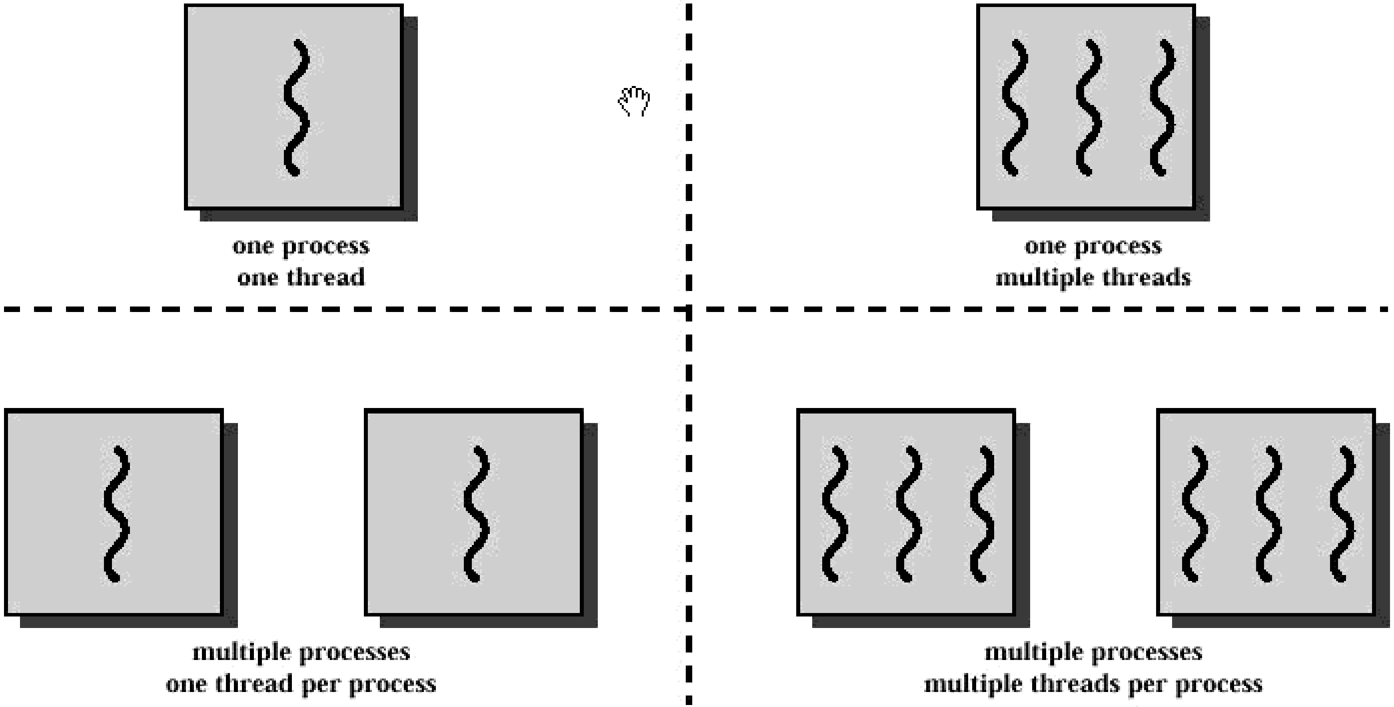
Multitasking



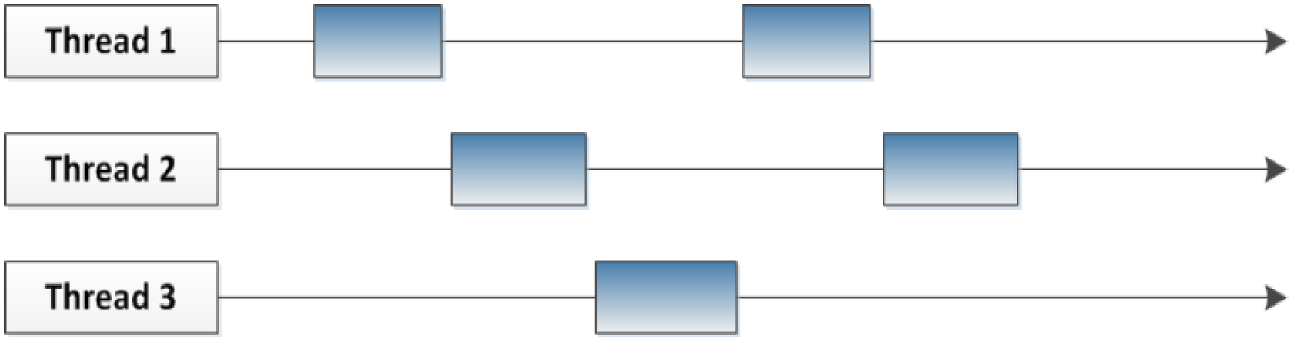
Multithreading



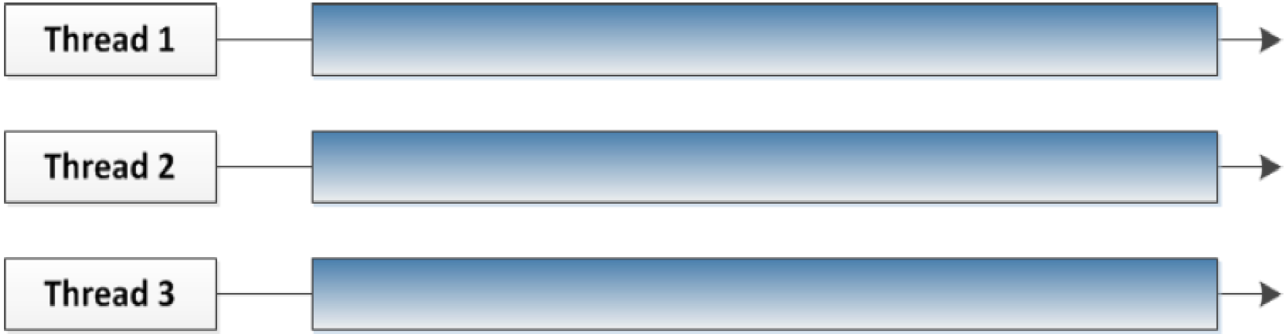
Multitasking and Multithreading



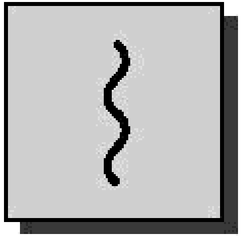
Multiple threads sharing a single CPU

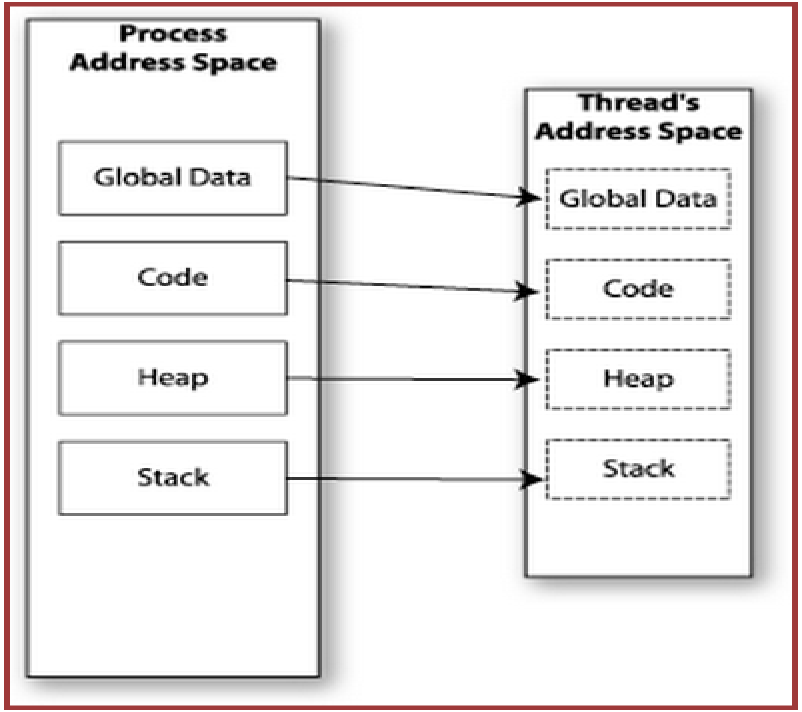


Multiple threads on multiple CPUs

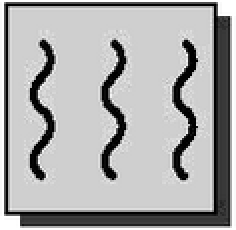


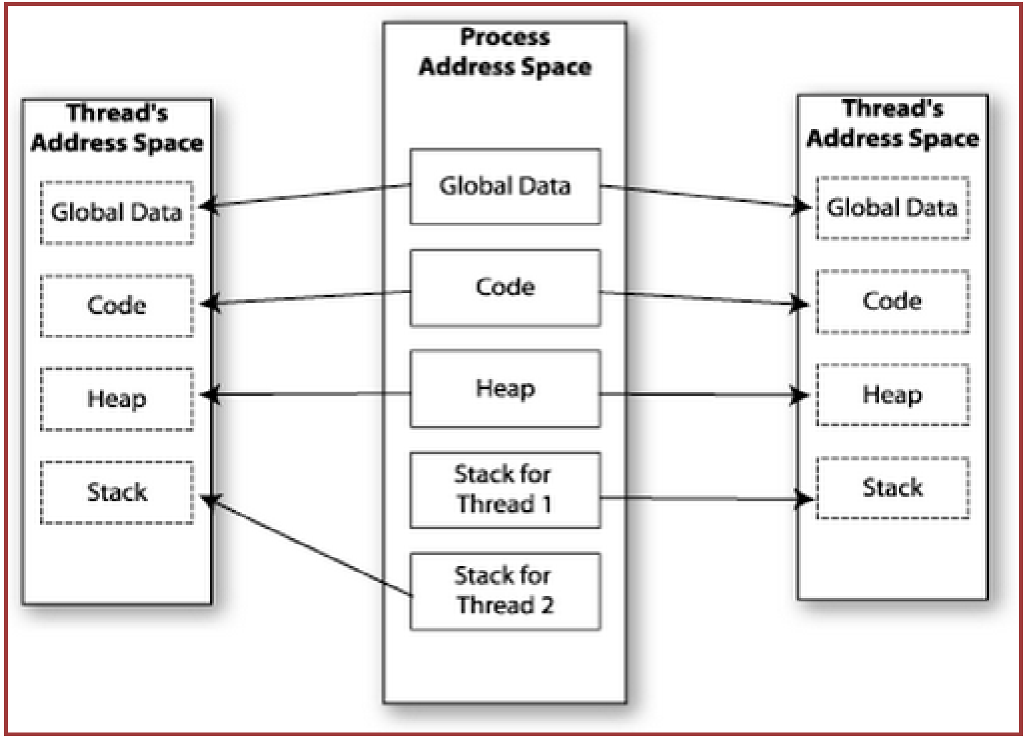
Memory Layout: Single-Threaded Program





Memory Layout: Multithreaded Program





Multithreading

Multithreading & Multitasking: Comparison

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

Communication in 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

Communication in Multithreading

Threads share all of the process's memory except for their stacks

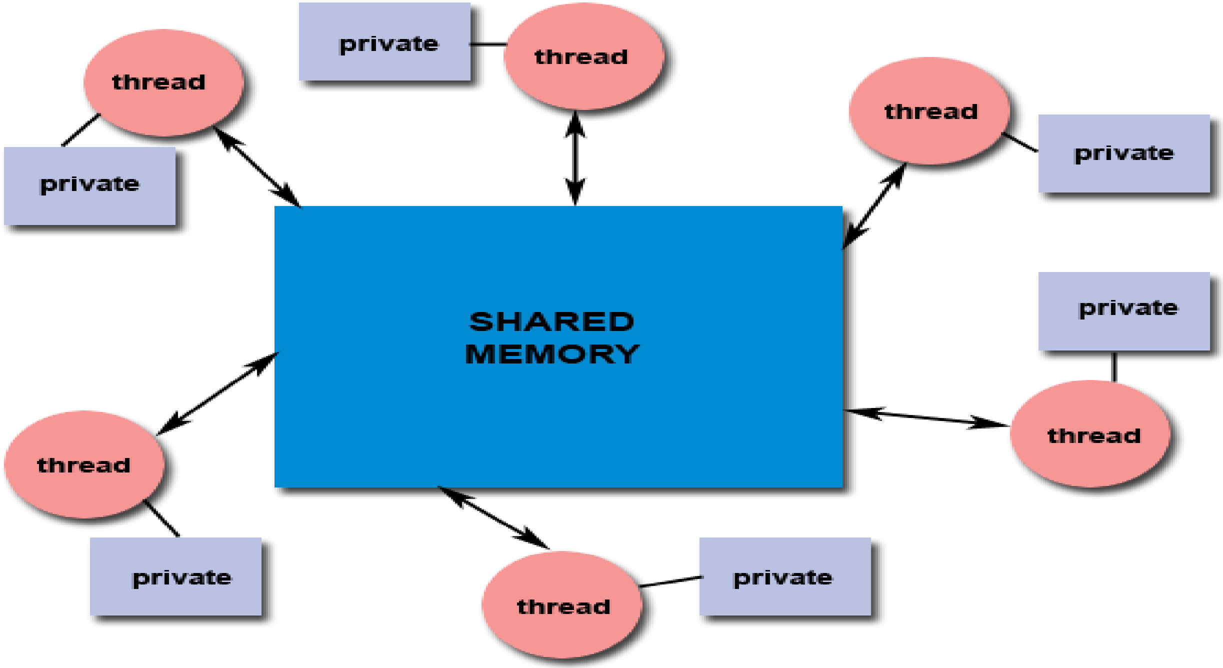
=> Data sharing requires no extra work (no system calls, pipes, etc.)

Shared memory

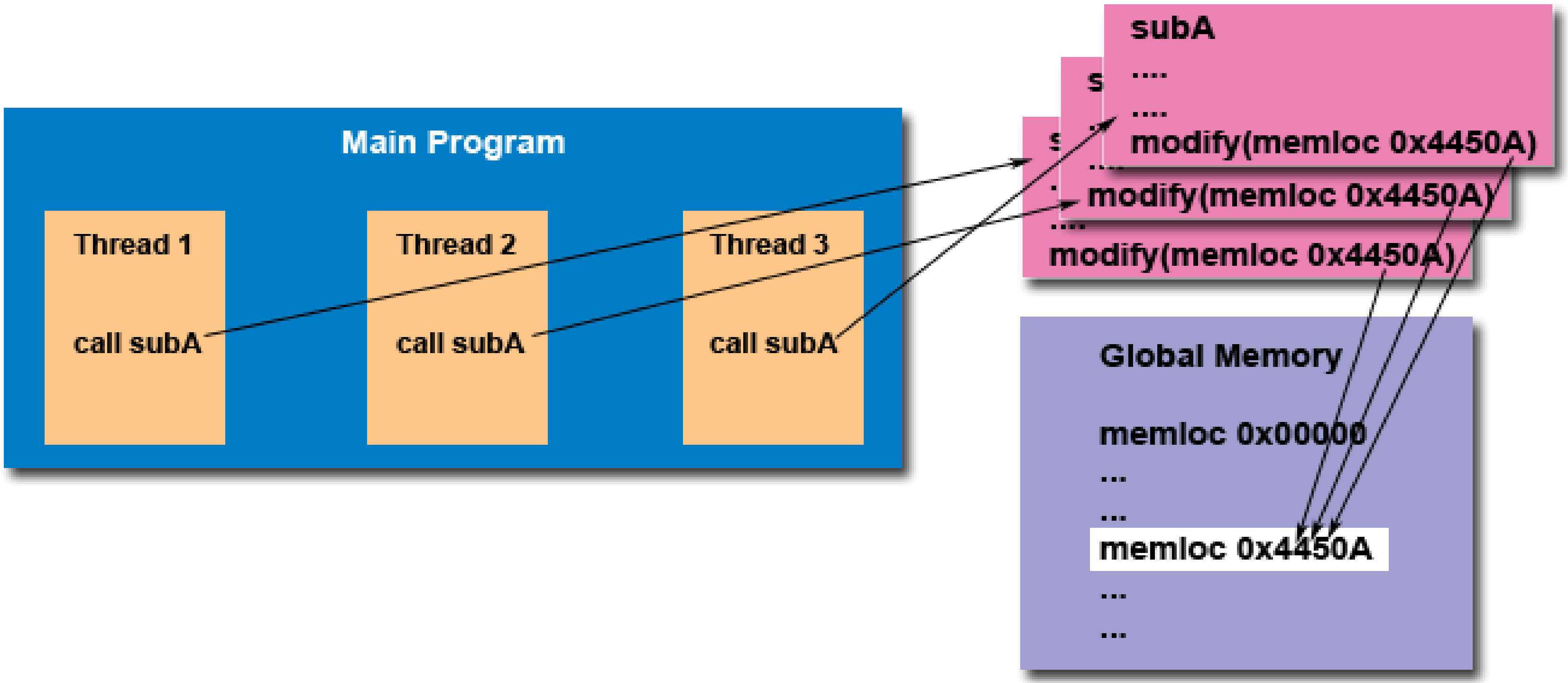
Makes multithreaded programming

- 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)

Shared memory

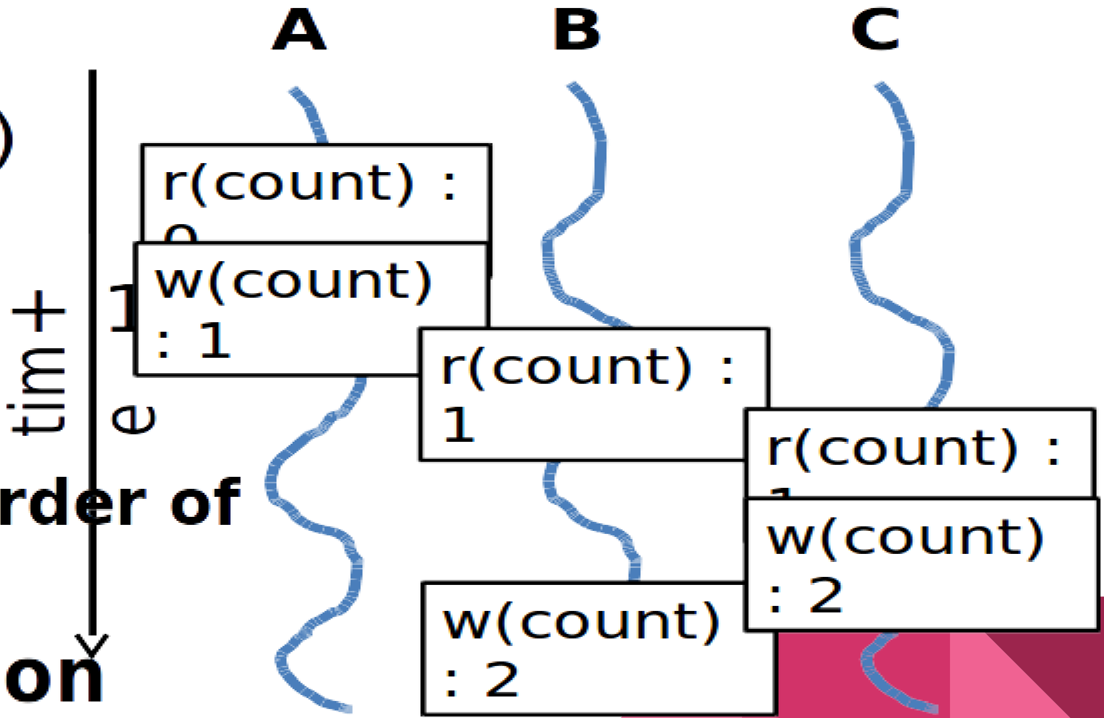


Thread safety



Race Condition

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



Result depends on order of execution

=> Synchronization needed

Mutex

mutexes - Mutual exclusion lock: Block access to variables by other threads. This enforces exclusive access by a thread to a variable or set of variables.

Mutexes are used for serializing shared resources.

Anytime a global resource is accessed by more than one thread the resource should have a Mutex associated with it.

When a mutex lock is attempted against a mutex which is held by another thread, the thread is blocked until the mutex is unlocked.

POSIX Threads

The POSIX thread libraries are a standards based thread API for C/C++.

PThread Creation

int pthread_create(pthread_t * thread, const pthread_attr_t * attr,

void * (*start_routine)(void *), void *arg);

Arguments:

thread - returns the thread id. (unsigned long int defined i bits/pthreadtypes.h)
attr - Set to NULL if default thread attributes are used. void * (*start_routine) - pointer to the function to be threaded. Function has a single argument: pointer to void.
*arg - pointer to argument of function. To pass multiple arguments, send a pointer to a structure.

Pthread exit

void pthread_exit(void *retval);

Arguments:

retval - Return value of thread.

This routine kills the thread.

Note: the return pointer *retval, must not be of local scope otherwise it would cease to exist once the thread terminates.

Pthread join

int pthread_join(pthread_t *thread*, void ***value_ptr*);

The *pthread_join()* function suspends execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated.

Mutex

int pthread_mutex_lock(pthread_mutex_t **mutex*);
int pthread_mutex_unlock(pthread_mutex_t **mutex*);

The mutex object referenced by *mutex* is locked by calling *pthread_mutex_lock()*. If the mutex is already locked, the calling thread blocks until the mutex becomes available. This operation returns with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

Lab

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

Lab

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

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 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 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