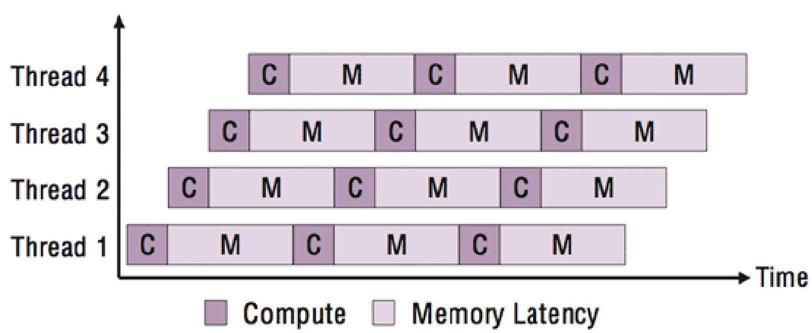
Programação Avançada (ProgA)

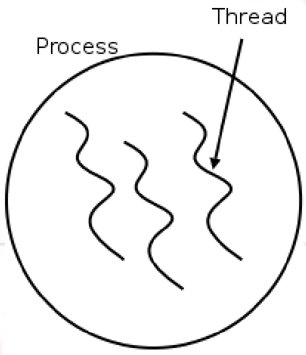
Threads

Patrício Domingues
ESTG/Politécnico de Leiria



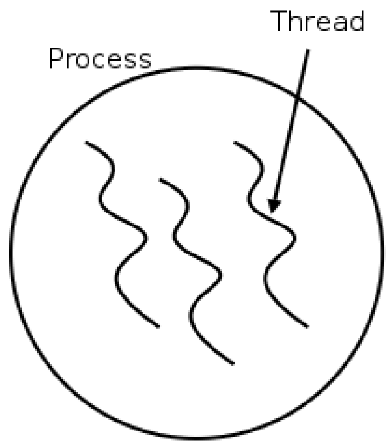
Definitions

- ✓ Binary (*executable*)
  - programs residing on a storage medium, compiled to a format accessible by a given operating system and machine architecture, ready to execute
- ✓ Process
  - Program/binary in execution
    - Loaded binary in memory controlled by the operating system
    - Abstraction of a running program
- ✓ Threads
  - Unit of execution within a process
    - Virtualized processor, stack and state



Processes and threads (1)

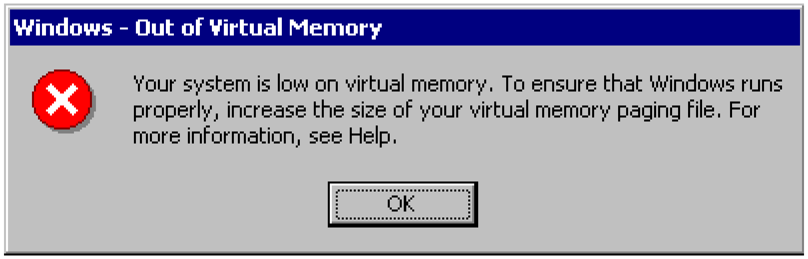
- ✓ A process contains one or more threads
  - – Process containing only one thread
    - Single threaded process
  - – Process containing several threads
    - Multithreaded process



- ✓ A thread only exists within the context of a process
  - – When a process ends, all of its threads also terminate
  - – A thread is the basic unit to which the OS allocates CPU time

Processes and threads (2)

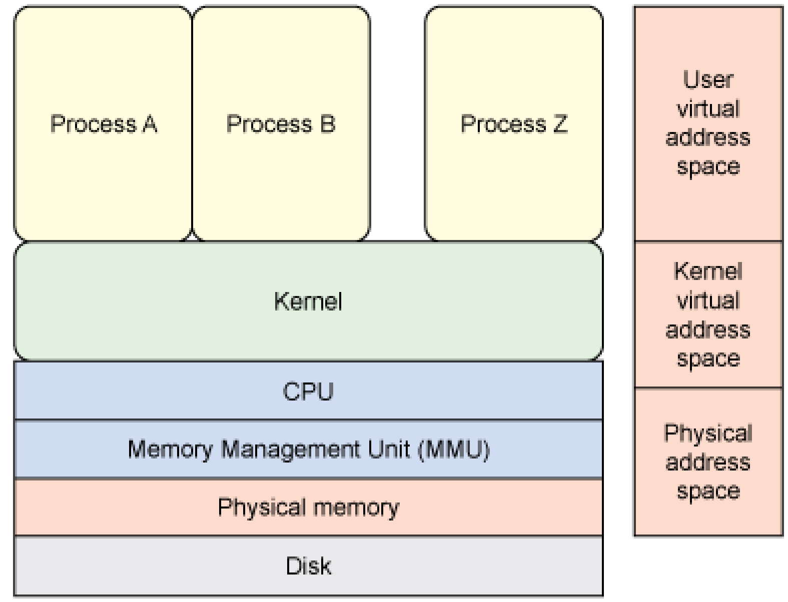
- An operating system provides two main abstractions for *user space*
  - Virtualized processor
  - Virtualized memory



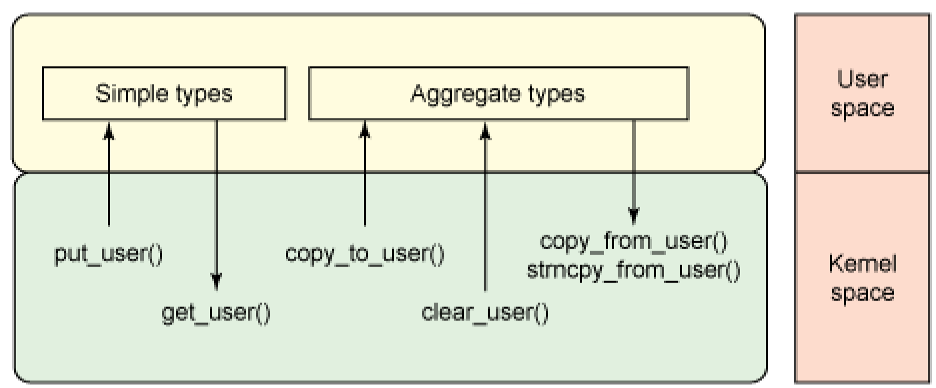
User space vs *kernel space* >>

User space vs. kernel space

- ✓ *User space*
  - *unprivileged mode*
  - *Process can only access memory from user space*
- ✓ *Kernel space*
  - Privileged mode
  - Used when a process makes a system call (e.g., `write`)
  - Runs kernel code



Source: http://ibm.co/1N0Jsl3



Source: http://ibm.co/1XrDdtN

IPL
escola superior de tecnologia e gestão
instituto politécnico de leiria

The `strace` tool

- ✓ strace – system call trace
  - Shows all the **system calls** performed by the execution of a program
    - Example: **`strace date`**

File Edit Tabs Help

```
execve("/bin/date", ["date"], [/* 44 vars */]) = 0
brk(NULL)                               = 0x8cab000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
74c000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=79027, ...}) = 0
mmap2(NULL, 79027, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7738000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320\207\1\0004\0\0\0"..
., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1786484, ...}) = 0
mmap2(NULL, 1792572, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb
7582000
mprotect(0xb7731000, 4096, PROT_NONE)   = 0
mmap2(0xb7732000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRI
TE, 3, 0x1af000) = 0xb7732000
mmap2(0xb7735000, 10812, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMO
US, -1, 0) = 0xb7735000
close(3)                                = 0
```

The `ltrace` tool

- ✓ The `ltrace` tool
  - – Shows all the calls of a process to functions of dynamic **libraries**
    - Example: `ltrace date`

```
user@ubuntu: ~
File  Edit  Tabs  Help
__libc_start_main(0x8049350, 1, 0xbfe1f984, 0x80520e0 <unfinished ...>
strrchr("date", '/')                              = nil
setlocale(LC_ALL, "")                             = "en_US.UTF-8"
bindtextdomain("coreutils", "/usr/share/locale") = "/usr/share/locale"
textdomain("coreutils")                           = "coreutils"
__cxa_atexit(0x804a610, 0, 0, 0xbfe1f984)         = 0
getopt_long(1, 0xbfe1f984, "d:f:I::r:Rs:u", 0x8053760, nil) = -1
nl_langinfo(0x2006c, 0xb766e2c0, 0x80484dc, 0xbfe1f898) = 0xb74a6295
getenv("TZ")                                      = nil
malloc(64)                                        = 0x94df860
clock_gettime(0, 0xbfe1f854, 0, 2)                = 0
localtime(0xbfe1f7e8)                             = 0xb76c4740
strftime(" Sun", 1024, " %a", 0xb76c4740)         = 4
fwrite("Sun", 3, 1, 0xb76c1d60)                   = 1
fputc(' ', 0xb76c1d60)                            = 32
strftime(" Sep", 1024, " %b", 0xb76c4740)         = 4
fwrite("Sep", 3, 1, 0xb76c1d60)                   = 1
fputc(' ', 0xb76c1d60)                            = 32
```

Virtualized processor

- ✓ Virtualized processor
  - – When executing, a process *thinks* that the CPU is solely for it
    - The process is not and needs not to be aware of *multitasking*
  - – The system can have 100 or more processes running, but each single process **needs not** to be aware of the other processes
  - – A programmer develops its application without the need to care for other processes

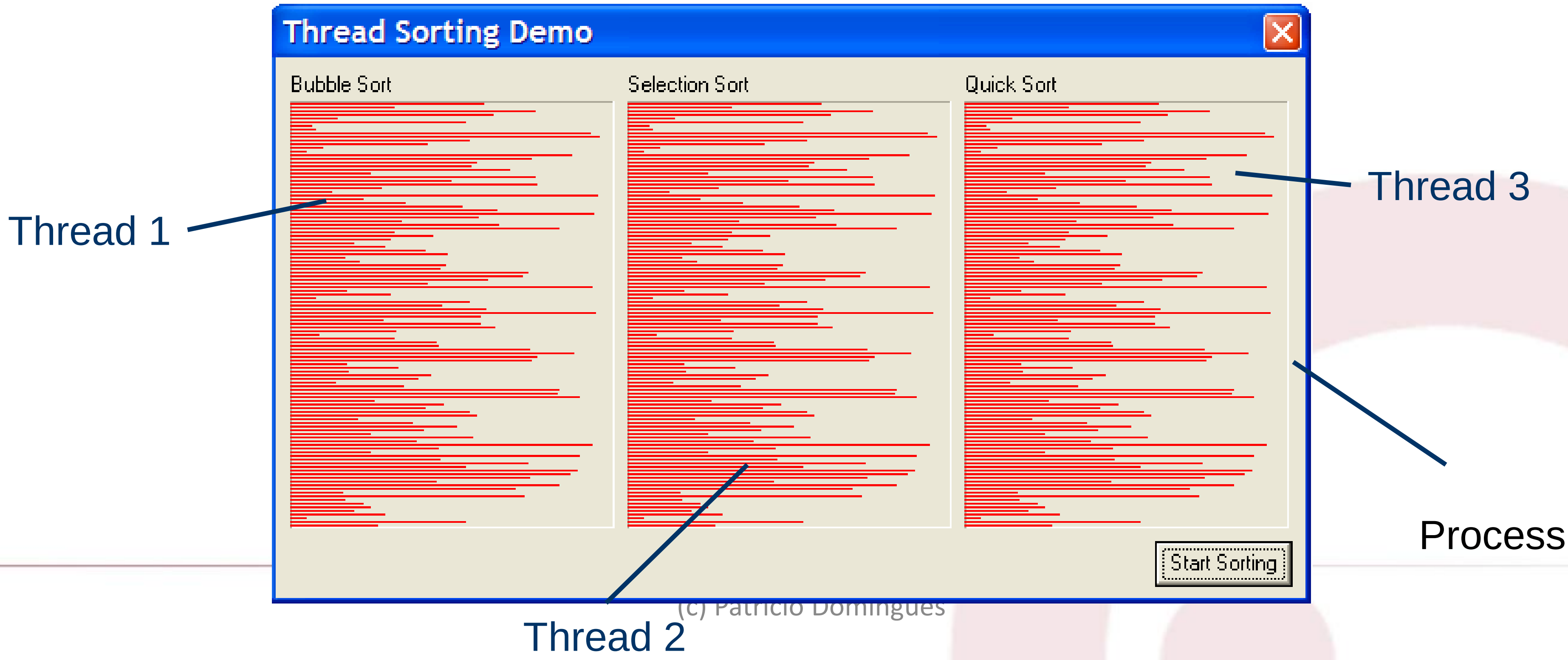
Virtualized memory >>

Virtualized memory

- ✓ Each process has a unique view of memory
  - – The system's RAM contains the data of many different running processes
  - – Each process sees virtual memory all of its
- ✓ Virtualized memory is associated with the process and not the thread
  - – Each process has a unique view of memory
  - – All of the threads of a given process share the process's memory space

What is a *thread*?

- ✓ Definition in the dictionary
  - "thread": a fine cord of twisted fibers (of cotton or silk or wool or nylon, etc.) used in sewing and weaving
  - Portugueses: «linha de cozer», etc.
    - Not the right definition for an operating system point of view



Why *multithreading*? (1)

- Five primary benefits of using threads
  - Programming abstraction
    - Dividing up work and assigning each division to a unit of execution (a thread) is a natural approach to many problems
  - Parallelism
    - In machines with multiple processors/cores, threads provide an efficient way to achieve *true* parallelism
  - Improving responsiveness
    - In a single-threaded process, a long-running operation can prevent an application from responding to user input. The application appears “frozen”
    - In a multi-threaded process, a specific thread can be set to deal with user input

Why *multithreading*? (2)

- Five primary benefits of using threads (continued)
  - Blocked I/O
    - Without threads, blocking I/O (e.g, waiting for user input or for data from a remote communication) halts the entire process
    - In a multithreaded process, individual threads may block, waiting on I/O, while other threads continue to make forward progress
  - Context switching
    - Cost of switching from one thread to a different thread within the same process is significantly cheaper than process-to-process context switching
      - Threads of a same process share the same memory address space

Examples of multithreading (#1)

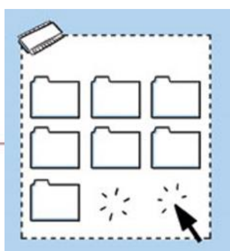
- ✓ Responsive application
  - Application which...
    - Performs intense and somewhat long CPU computation
      - Example: Image processing
    - While it is performing an image processing operation, it is welcome to update the user interface
      - Example: progress bar, responsive “cancel” button
    - This can be done with a specially dedicated thread
- ✓ Example: word processor
  - One thread to deal with user input
  - One thread to paginate the document
  - One thread to open/save a document
  - Another thread to run the spell checker and underline unknown words
  - ...

Examples of multithreading (#2)

- ✓ Mozilla firefox
  - Since version 48
    - 1 process for <u>u</u>ser <u>i</u>nterface (UI)
    - 1 process contains all tabs
  - Future version of firefox
    - 1 process for user interface
    - 1 process for <u>**each**</u> tab
  - Google Chrome already does this
    - Each tab is a process
    - Each process is isolated from the others

Application Basics

Name	Firefox
Version	54.0.1
Build ID	20170628075643
Update History	Show Update History
Update Channel	release
User Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:54.0)
OS	Windows_NT 10.0
Profile Folder	Open Folder
Enabled Plugins	about:plugins
Build Configuration	about:buildconfig
Memory Use	about:memory
Performance	about:performance
Registered Service Workers	about:serviceworkers
Multiprocess Windows	1/1 (Enabled by default)
Google Key	Found
Mozilla Location Service Key	Found
Safe Mode	false
Profiles	about:profiles

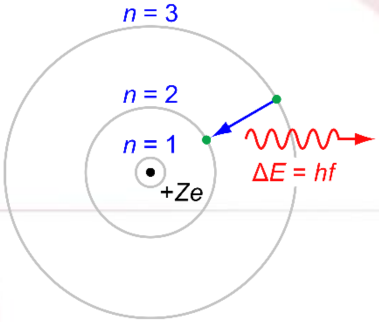


Challenges of multithreading

- ✓ Multithreading has its own challenges
  - – Multithread means to have several threads of the same process potentially accessing the same data
  - – Concurrency-related bugs

- *Heisenbugs* (from the quantum theory of W. Heisenberg)
  - – Bugs due to concurrency problems that manifest themselves in a non-deterministic way
    - » An heisenbug can manifest itself 1 out of 100 runs
    - » Hard to debug (it disappears or alter its behavior when one attempts to probe or isolate it)

- Bohrbugs (from Niels Bohr's atom model)
  - – Deterministic bugs that always occur when the program is run
- See: https://en.wikipedia.org/wiki/Heisenbug

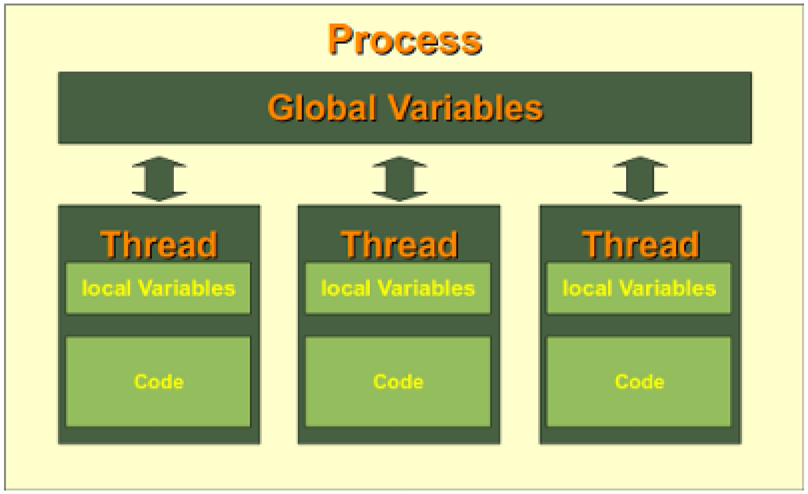


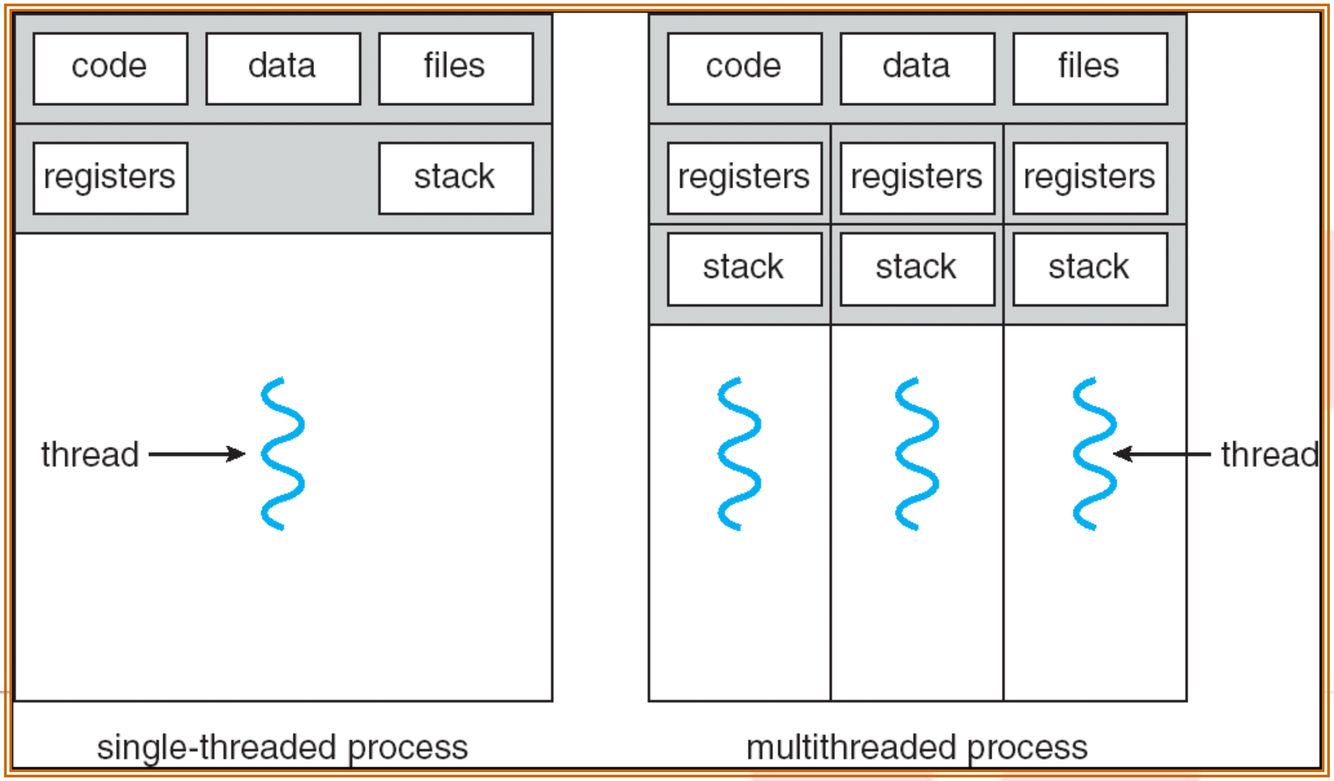
IPL escola superior de tecnologia e gestão instituto politécnico de leiria

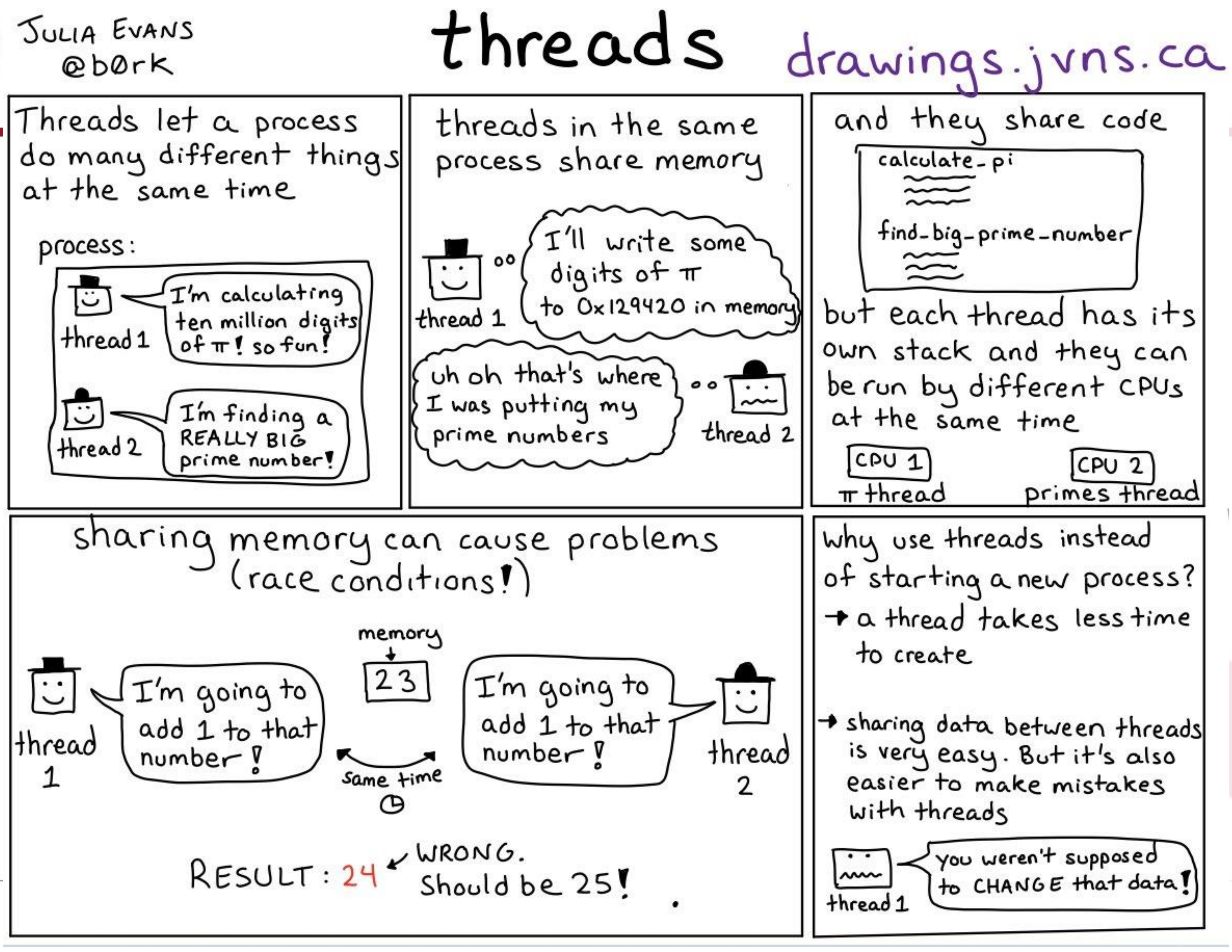
threads

- ✓ Private elements (each thread has its own)
  - Program counter (PC)
  - registers
  - Stack segment
- ✓ Shared elements (threads of a same process)
  - Text segment (code of the program)
  - Address space
  - Data segment (e.g., global variables)
  - Files, signal handlers, etc.

Processes and threads






JULIA EVANS
@b0rk
threads
drawings.jvns.ca
Threads let a process do many different things at the same time
process:
thread 1
I'm calculating ten million digits of π! so fun!
thread 2
I'm finding a REALLY BIG prime number!
threads in the same process share memory
thread 1
I'll write some digits of π to 0x129420 in memory
uh oh that's where I was putting my prime numbers
thread 2
and they share code
calculate_pi
find_big_prime_number
but each thread has its own stack and they can be run by different CPUs at the same time
CPU 1
π thread
CPU 2
primes thread
sharing memory can cause problems (race conditions!)
memory
23
thread 1
I'm going to add 1 to that number!
I'm going to add 1 to that number!
thread 2
same time
RESULT: 24
WRONG. Should be 25!
why use threads instead of starting a new process?
→ a thread takes less time to create
→ sharing data between threads is very easy. But it's also easier to make mistakes with threads
thread 1
you weren't supposed to CHANGE that data!

Threads in Linux

- ✓ Info about threads in Linux
  - `ps -eLf` or `ps axms`
    - `nlwp:` number of threads per process
      - lwp: lightweigth process

```
user@ubuntu: ~
File  Edit  Tabs  Help
user@ubuntu:~$ ps -eLf | grep -i "rsyslogd\|NetworkManager" | head -n 8
syslog     1023      1  1023  0      4 Oct24 ?        00:00:00 rsyslogd
syslog     1023      1  1024  0      4 Oct24 ?        00:00:03 rsyslogd
syslog     1023      1  1025  0      4 Oct24 ?        00:00:10 rsyslogd
syslog     1023      1  1026  0      4 Oct24 ?        00:00:00 rsyslogd
root       1157      1  1157  0      4 Oct24 ?        00:00:09 NetworkManager
root       1157      1  1159  0      4 Oct24 ?        00:00:00 NetworkManager
root       1157      1  1160  0      4 Oct24 ?        00:00:03 NetworkManager
root       1157      1  1161  0      4 Oct24 ?        00:00:00 NetworkManager
user@ubuntu:~$
```

nlwp

The *pthread* standard

✓ *pthread* standard - *POSIX thread*

```
void thread_slow(void){
    while(1){
        printf("Hi! I'm the slow thread…");
        sleep(3);
    }
}
void thread_fast(void){
    while(1)    {
        printf("Hi! I'm the fast thread…");
        sleep(1);
    }
}
int main(int argc, char **argv){
    pthread_t thr_slow, thr_fast;
    /* Create two threads */
    pthread_create(&thr_slow,NULL, thread_slow,NULL);
    pthread_create(&thr_fast,NULL, thread_fast,NULL);
   /* call pthread_join for each created thread */
}
```

Output of the example

```
$./thread_demo
```

Hi! I’m the slow thread...

Hi! I’m the fast thread...

Hi! I’m the fast thread...

Hi! I’m the fast thread...

Hi! I’m the slow thread...

Hi! I’m the fast thread...

Hi! I’m the fast thread...

Hi! I’m the fast thread...

Hi! I’m the slow thread...

Hi! I’m the fast thread...

Hi! I’m the fast thread...

pthreads

✓ PThreads

- POSIX standard (IEEE 1003.1c)
  - API (Application Programming Interface) for threads
- The API specifies the behavior of the functions
  - What each function is supposed to do
- The implementation is free
  - The how is done does not matter for the standard
- Pthreads is available in UNIX platforms
  - Linux, Mac OS X, BSD, Solaris
- It also exists for windows

Pthreads standard (1)

✓ Data types

- pthread_t: handle to a thread
- pthread_attr_t: thread attributes

✓ Handling functions (create, etc.)

- pthread_create(): create a thread
- pthread_exit(): terminate current thread
- pthread_cancel(): cancel execution of another thread
- pthread_join(): block current thread until another one terminates
- pthread_attr_init(): initialize thread attributes
- pthread_attr_setdetachstate(): set the detachstate attribute (whether thread can be joined on termination)
- pthread_attr_getdetachstate(): get the detachstate attribute
- pthread_attr_destroy(): destroy thread attributes
- pthread_kill(): send a signal to a thread

Pthreads standard (2)

✓ Synchronization functions

- pthread_mutex_init() initialize mutex lock
- pthread_mutex_destroy()
- pthread_mutex_lock(): acquire mutex lock (blocking)
- pthread_mutex_trylock(): acquire mutex lock (non-blocking)
- pthread_mutex_unlock(): release mutex lock
- pthread_cond_init()
- pthread_cond_destroy()
- pthread_cond_signal(): signal a condition
- pthread_cond_wait(): wait on a condition

fork vs. pthread_create

- ✓ Time need to create a process vs. a thread
  - Real –*elapsed time* or wallclock time
  - User – time spent by the CPU in ***user mode***
  - Sys – time spent in **system/kernel mode**

Time (seconds) in several machines for creating 50000 processes/threads

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
AMD 2.3 GHz Opteron (16cpus/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8cpus/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

source: https://computing.llnl.gov/tutorials/pthreads/fork_vs_thread.txt

C11 – most recent C standard (2011)

- ✓ O C11 () adds support for multithreading to the C language
- ✓ file .h **<threads.h>**
  - – Functions to handle threads (create, join,...)
  - – Mutexes and condition variables
  - – Qualifier "*_Atomic*"
  - – Qualifier "*_Thread_local*"
    - A variable declared as **_Thread_local** is private for the thread (i.e., it is not shared with the other threads of the process)
- ✓ Many compilers still do **<u>not</u>** support **<threads.h>** (gcc, etc.)

The OpenMP standard

- ✓ OpenMP standard for parallelization
  - – Goal is to provide for parallelization in an almost transparent way for the programmer
  - – The programmer annotates the code with #pragma instructions
- ✓ Example

```
void main()
{
  double r[1000];

  for (int i=0; i<1000; i++) {
    large_computation(r[i]);
  }
}
```

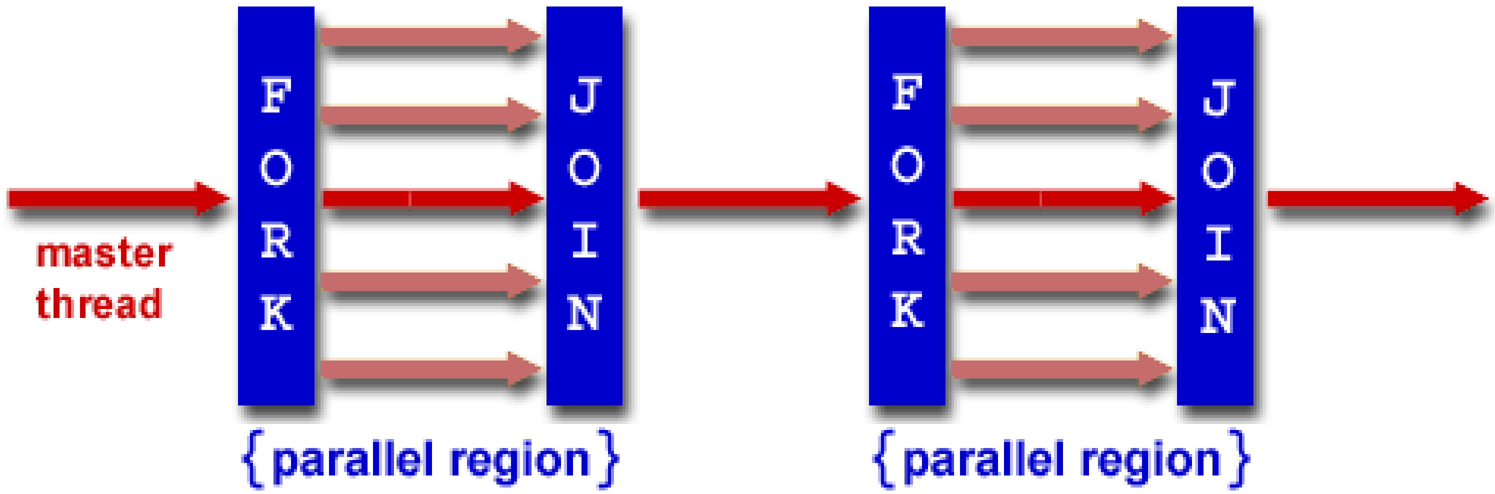
```
void main()
{
  double r[1000];

#pragma omp parallel for
  for (int i=0; i<1000; i++) {
    large_computation(r[i]);
  }
}
```

#pragma
The compiler performs the OpenMP parallelization (manages the threads)

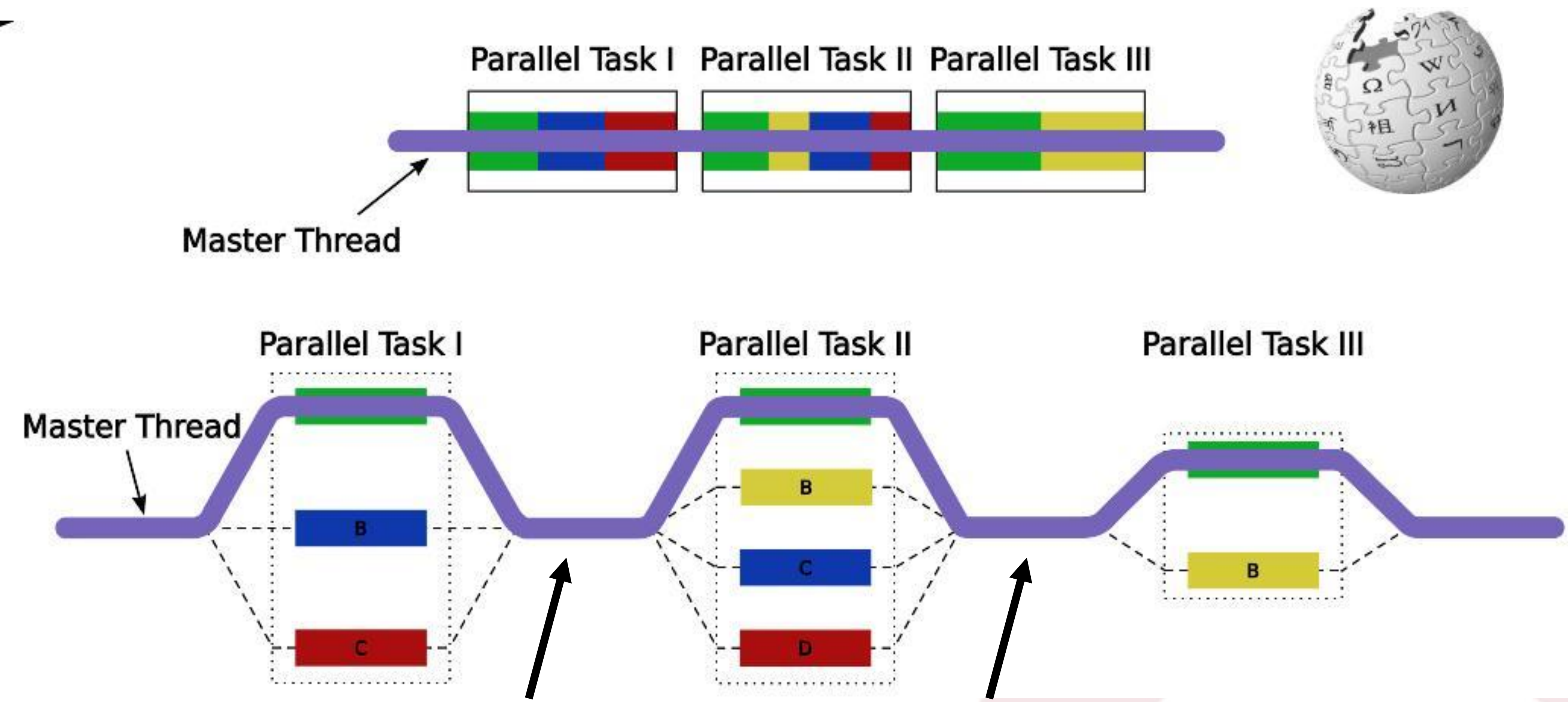
OpenMP Programming Model

- Fork-Join Model:
  - OpenMP uses the fork-join model of parallel execution:



  - All OpenMP programs begin as a single process: the **master thread**. The master thread executes sequentially until the first **parallel region** construct is encountered.

Master fork and join...



synchronization synchronization

OpenMP sequential and parallel

- ✓ Sequential

```
void main()
{
   double r[1000];

   for (int i=0; i<1000; i++) {
     large_computation(r[i]);
   }
}
```

- ✓ Parallel

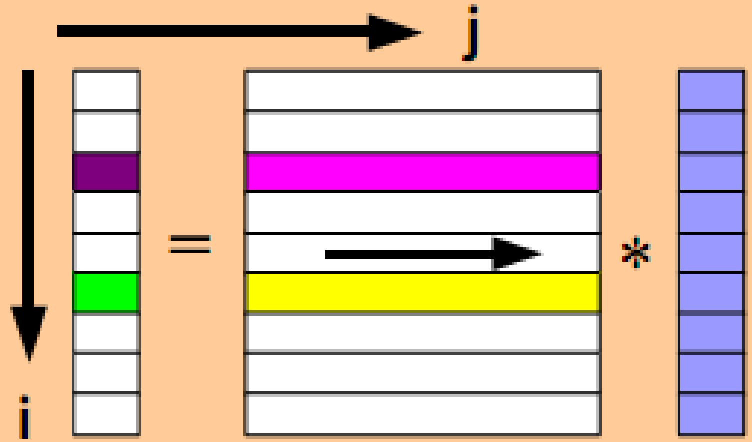
```
void main()
{
   double r[1000];

#pragma omp parallel for
   for (int i=0; i<1000; i++) {
     large_computation(r[i]);
   }
}
```

Another example

Example (source: “Introduction to OpenMP, SunMicrosystems”)

```
#pragma omp parallel for default(none) \
                private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
    sum = 0.0;
    for (j=0; j<n; j++)
      sum += b[i][j]*c[j];
    a[i] = sum;
 }
```



TID = 0 — Thread #1

```
for (i=0,1,2,3,4)
i = 0
    sum = Σ b[i=0][j]*c[j]
    a[0] = sum
i = 1
    sum = Σ b[i=1][j]*c[j]
    a[1] = sum
```

TID = 1 — Thread #2

```
for (i=5,6,7,8,9)
i = 5
    sum = Σ b[i=5][j]*c[j]
    a[5] = sum
i = 6
    sum = Σ b[i=6][j]*c[j]
    a[6] = sum
```

- ✓ Example of multithreaded applications
  - 7-zip
  - Imagemagick with OpenMP
    - SEE: https://imagemagick.org/discourse-server/viewtopic.php?t=20904
- ✓ Example of NON-multithreaded applications
  - Google chrome
  - Mozilla firefox (future version 50)
  - …

Imagemagick

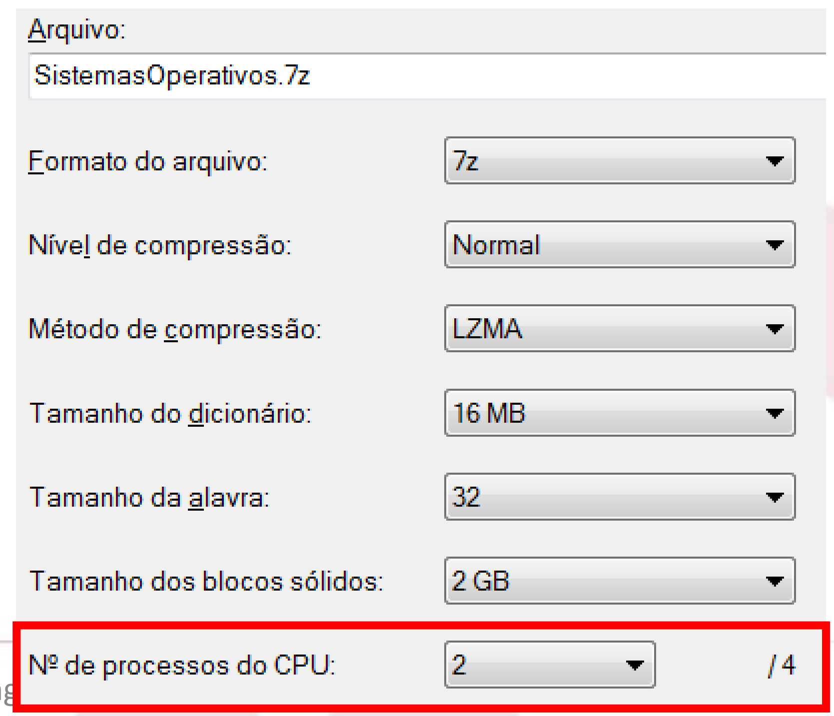
- ✓ Imagemagick
  - Software for image processing
  - Set of (very powerful) command line tools
    - convert, display, identify,…
  - Uses OpenMP to fully exploit the cores of the running machine
  - Example – channel.c:412

```
#if defined(MAGICKCORE_OPENMP_SUPPORT)
  #pragma omp parallel for schedule(static,4)
shared(progress,status) \
    magick_threads(image,image,image->rows,1)
#endif
```

Case study -7Zip (1/2)

- 7Zip
  - Data compressor (quite efficient)
  - By default, it uses all "CPUs" (cores and alike) of the computer

```
#ifdef COMPRESS_MT
    const UInt32 numProcessors =
  NSystem::GetNumberOfProcessors();
    _numThreads = numProcessors;
#endif
```

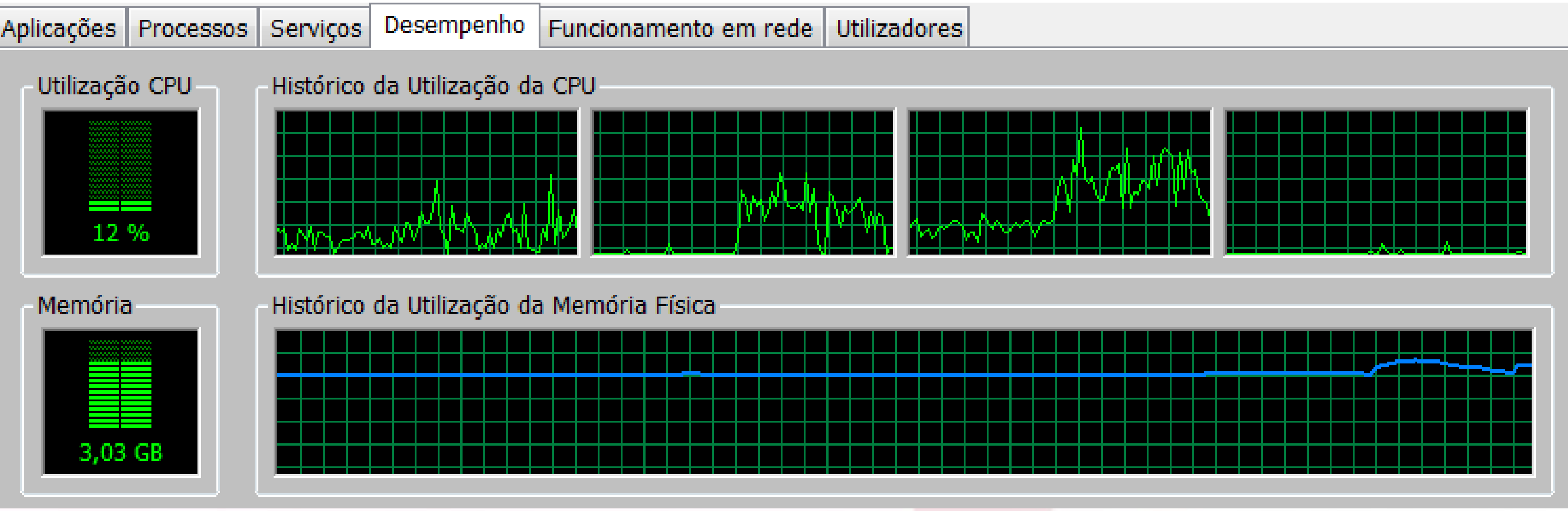


Case study -7Zip (1/2)

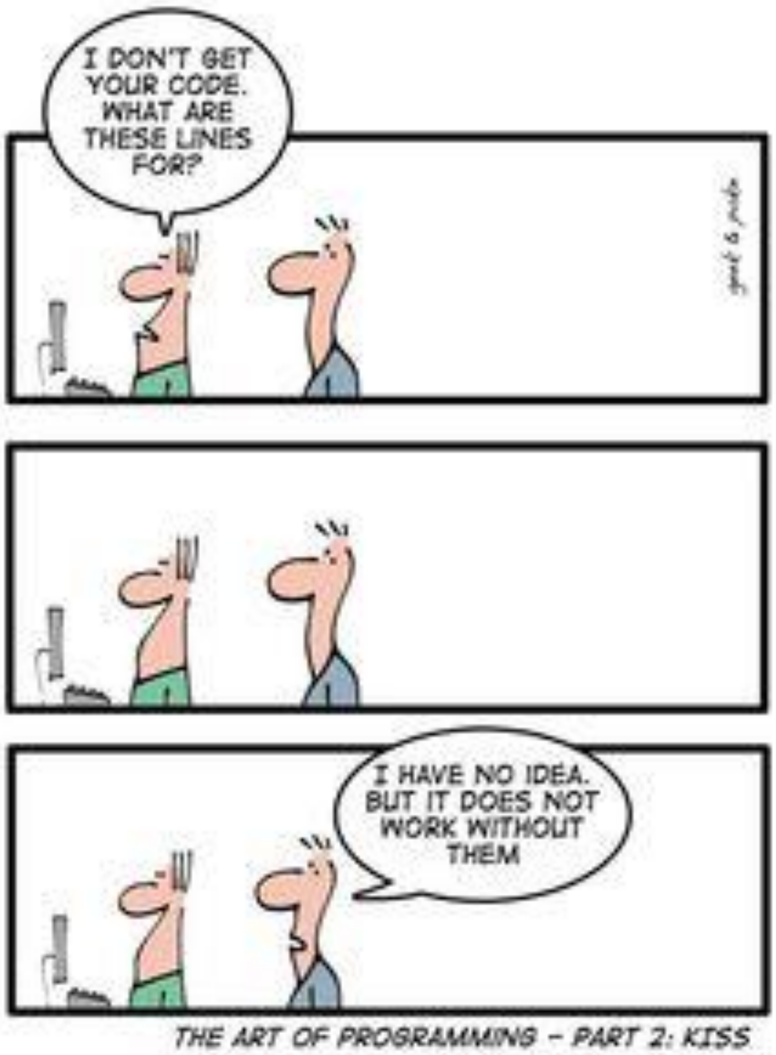
✓ 7Z in action

– Decompress the 7z file of the virtual machine

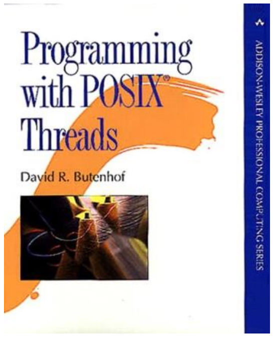
– Several cores are being used (see plot)



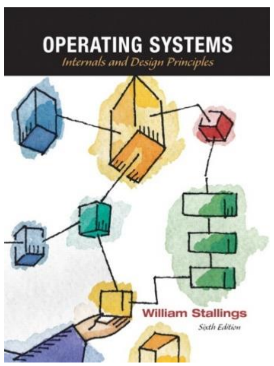
Multithread in practice...



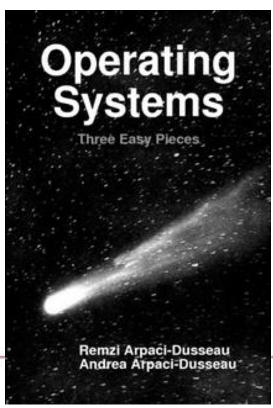
Bibliography #1



- ✓ David R. Butenhof, "Programming with POSIX Threads", Addison-Wesley, 1997, ISBN-13: 978-0201633924



- ✓ Chapter 4 - "Operating Systems – Internals and Design Principles", William Stallings, 7th edition, 2011 (ISBN : 013230998X)

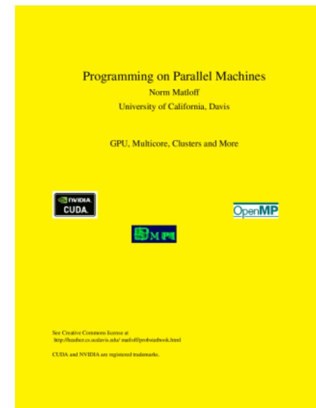


- ✓ Chapter 26 – Concurrency and threads. Arpaci-Dusseau, Remzi H., and Andrea C. Arpaci-Dusseau. Operating systems: Three easy pieces. Arpaci-Dusseau Books LLC, 2018.

Bibliography #2

- ✓ "Programming on Parallel Machines – University of California, Davis, 2012
  Open textbook: http://heather.cs.ucdavis.edu/parprocbook

- Chapter 1 (section 1.4): pthreads (example)
- Chapter 4: introduction to OpenMP", Norm Matloff,



- ✓ OpenMP
  - – http://openmp.org/wp/openmp-specifications/
  - – Source code do imagemagick (example)
    - http://www.imagemagick.org/script/download.php