

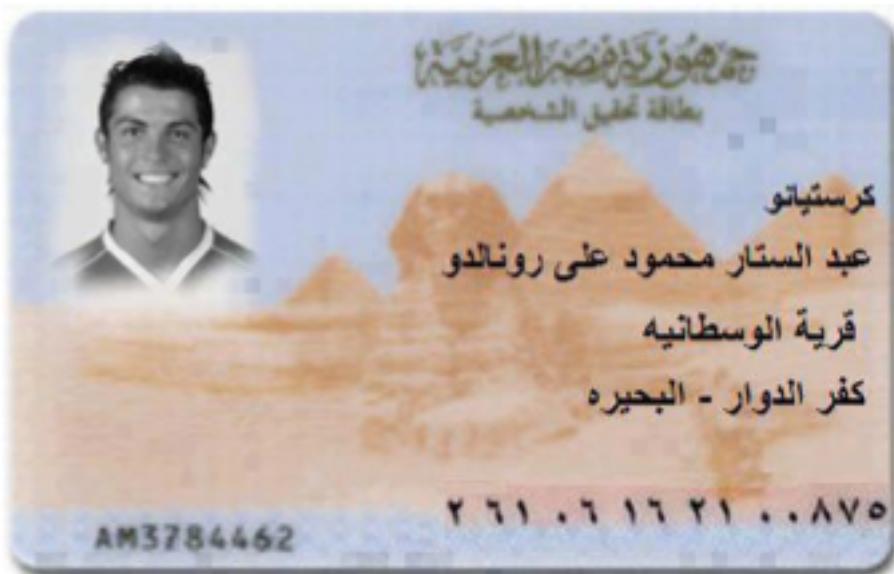
CPE 460 Operating System Design

Chapter 6: A Thread Story

Ahmed Tamrawi

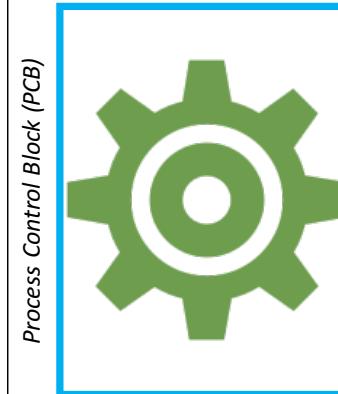
Any program to run must be loaded in memory





OPERATING SYSTEM KINGDOM

PROCESS



Process Control Block (PCB)

PROCESS STATE:

PROGRAM COUNTER:

CPU REGISTERS:

CPU SCHEDULING INFO:

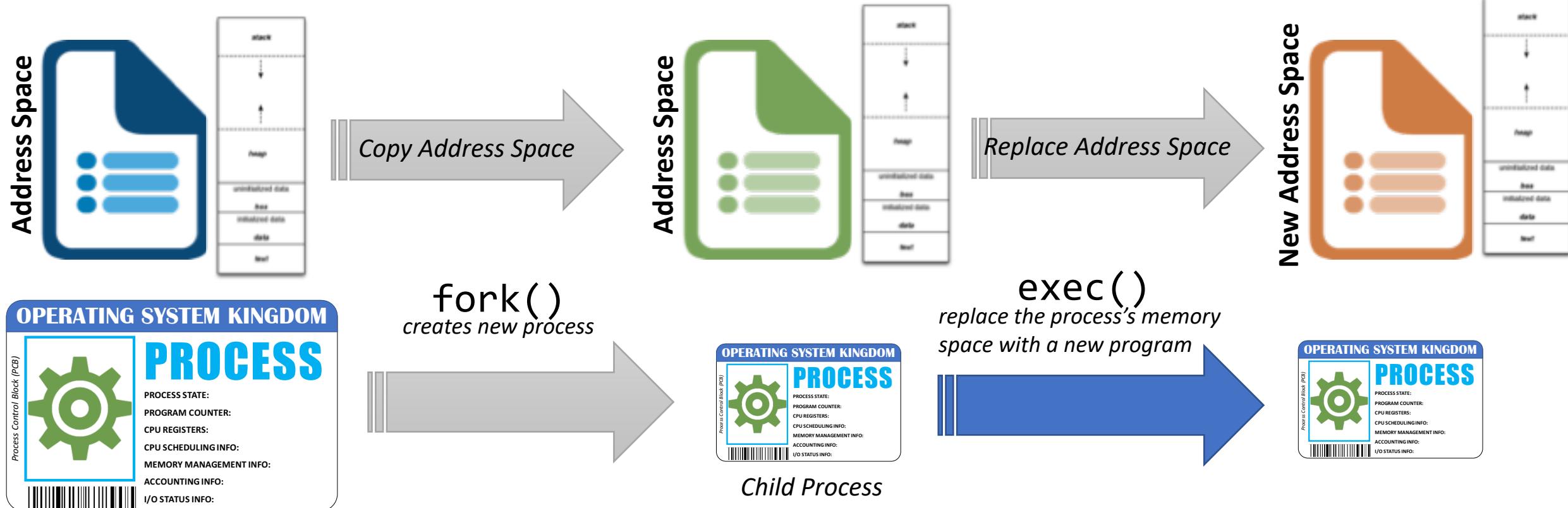
MEMORY MANAGEMENT INFO:

ACCOUNTING INFO:

I/O STATUS INFO:

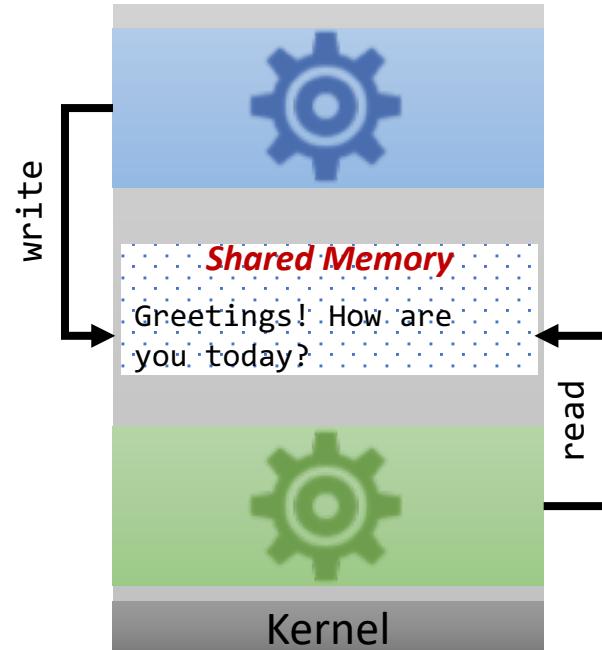
Process Creation

Each process has its own copy of address space



Cooperating processes need interprocess communication (IPC)

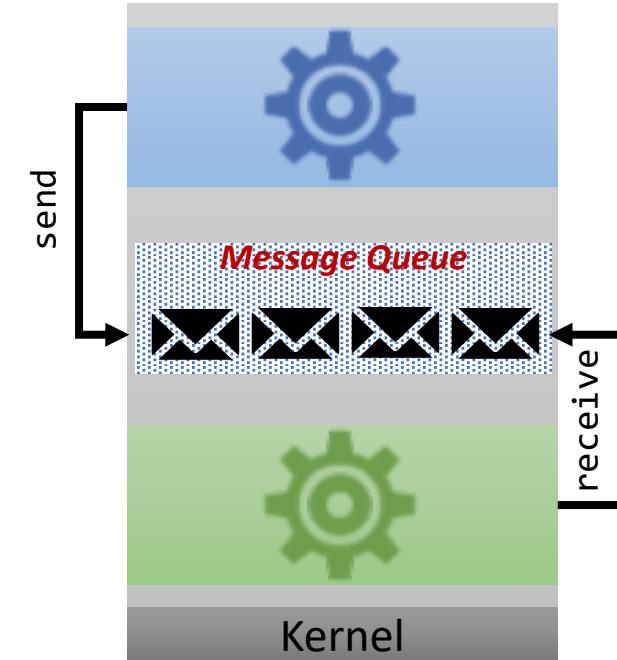
The operating system provides multiple mechanisms that allow processes to exchange data and information



Shared Memory

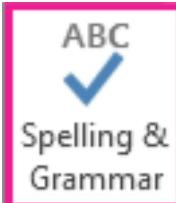
A region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region

Speed
Many Implementations



Message Passing

Communication takes place by means of messages exchanged between the cooperating processes

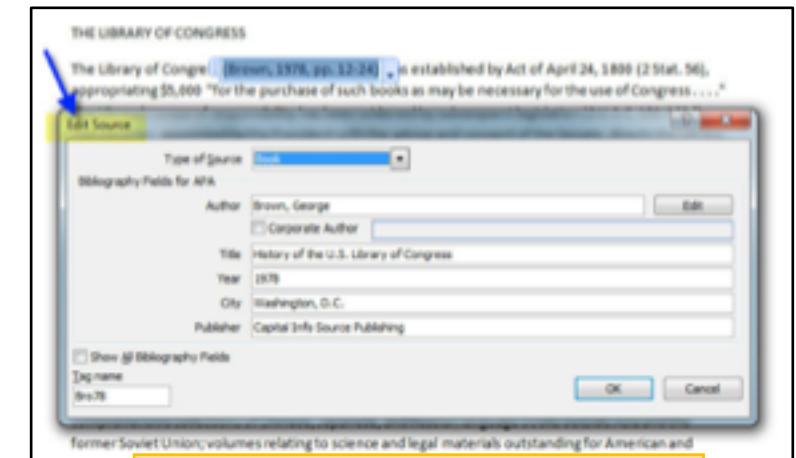


Spell and Grammar Check the 600 pages

If I am bad in spelling, there will be tons of errors and I need to review them

I am writing a book using Microsoft Word

The are currently 600 pages worth of 1GB of data

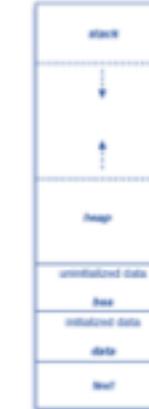
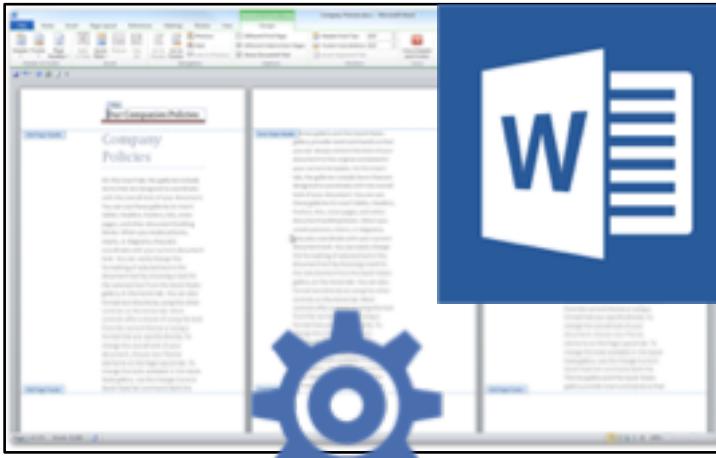


Add a citation to first page

Adding citation will to first page will affect the formatting of all pages

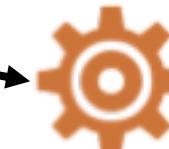
The image consists of a solid blue rectangular background. Overlaid on the right side is large, stylized white Arabic calligraphy. The text is arranged in two rows: the top row contains the letters 'ش', 'و', 'ل', 'ه', 'ب', 'ي', and the bottom row contains 'ال', 'د', 'ل', 'ل', 'خ', 'ا', 'ل'. To the left of this text, there is a white silhouette of a human head facing right. Inside the head, there is a black brain-like shape containing three white question marks. The overall composition is minimalist and graphic.

The are currently 600 pages worth of 1GB of data



1.2GB for the address space

`fork()`

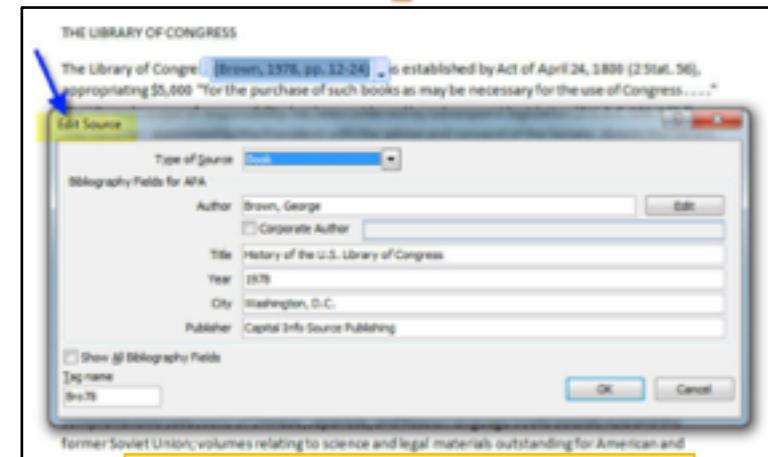


1.2GB for the address space



Spell and Grammar Check the 600 pages

If I am bad in spelling, there will tons of errors and I need to review them

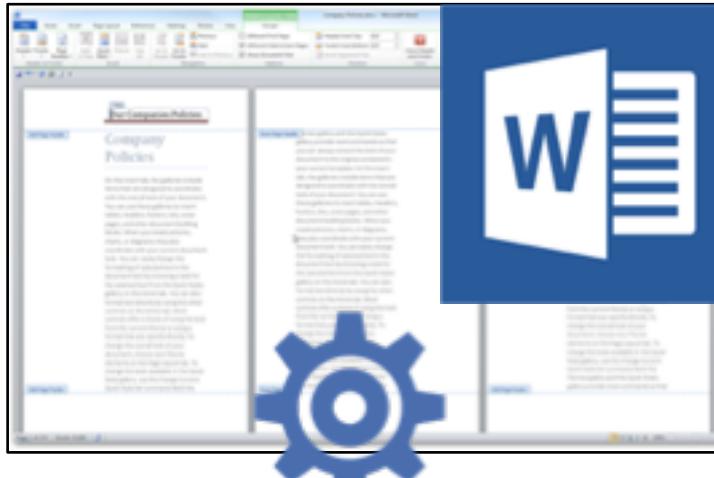


1.2GB for the address space

Add a citation to first page

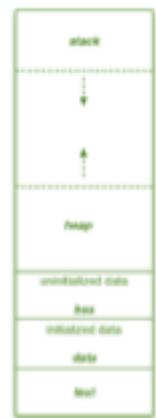
Adding citation will to first page will affect the formatting of all pages

The are currently 600 pages worth of 1GB of data



1.2GB for the address space

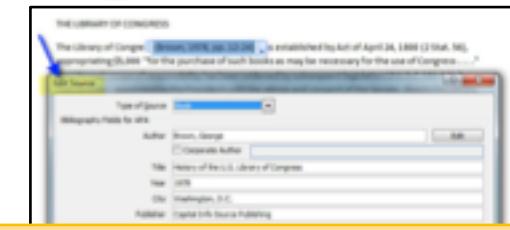
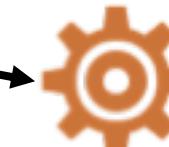
1.2GB for the address space



Spell and Grammar Check the 600 pages

If I am bad in spelling, there will tons of errors and I need to review them

`fork()`

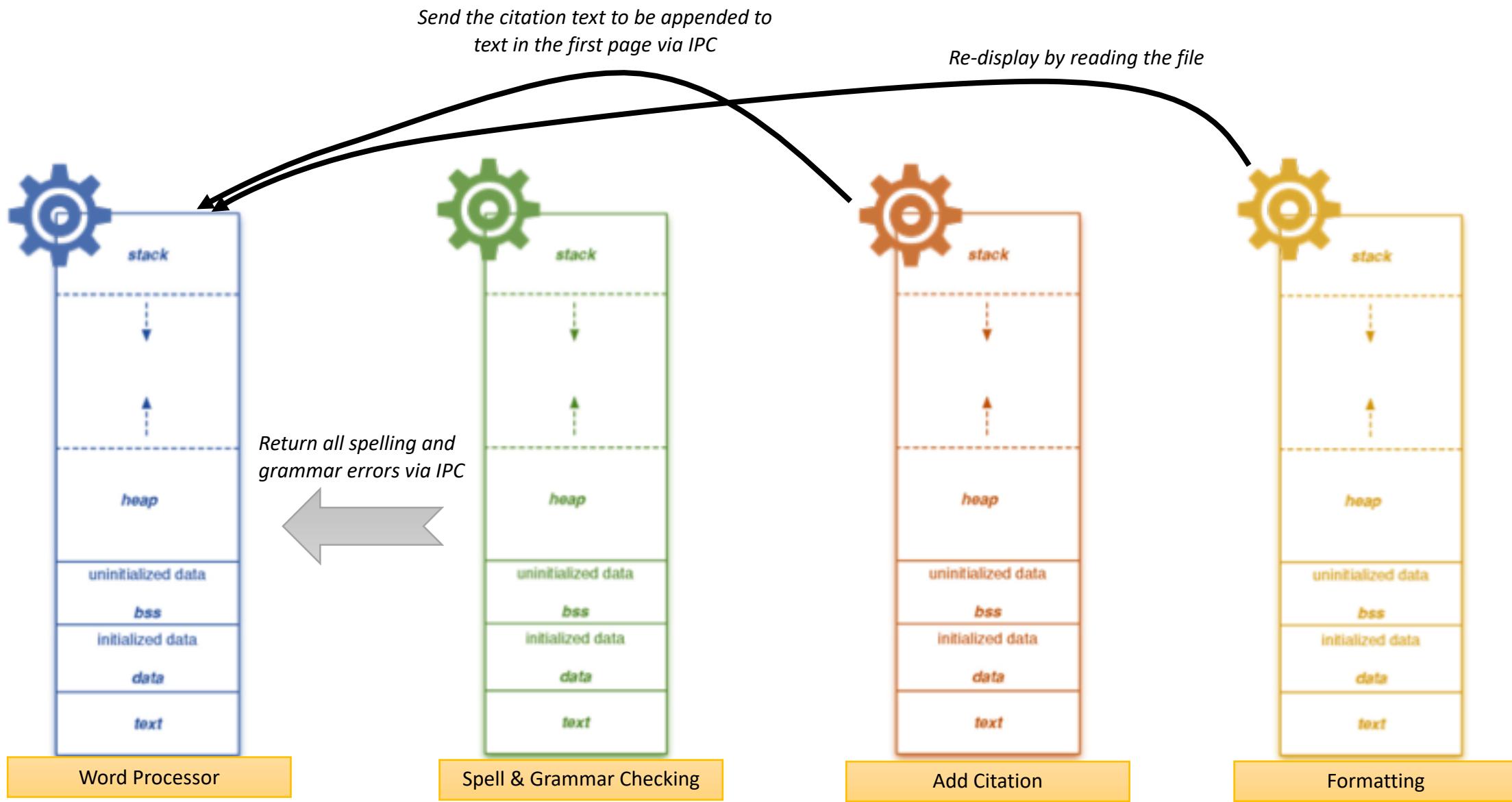


Add a citation to first page



1.2GB for the address space

Format the pages from 1 to end of document



This is not a feasible solution

It leads to substantial delays leading to unhappy user



Word Processor



Spell & Grammar Checking



Add Citation



Formatting

Recall that there is a limit on the amount of data that can be communicated via IPC

IPC is, in general, expensive due to the need for system calls

Context switching among large processes is costly and time consuming

Forking and copying address spaces is time and space consuming



We need a way to share the same address space between all cooperating processes so we can work on parallel



We need a way to share the same address space between all cooperating processes so we can work on parallel



Word Processor



Spell & Grammar Checking



Add Citation



Formatting

What is similar in these cooperating processes?

Share the same privileges

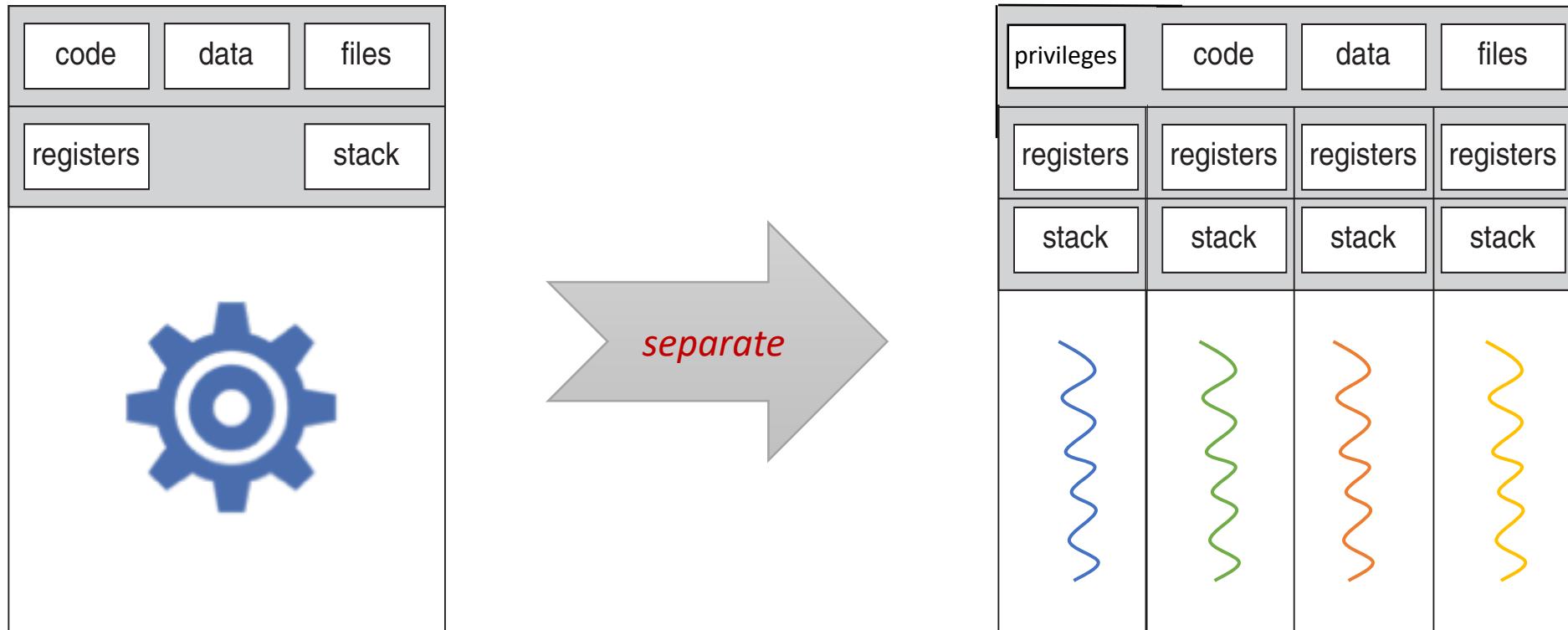
Share the same code and data
(address space)

Share the same resources

What is not shared among them?

Each has its own execution state: PC, SP, and registers

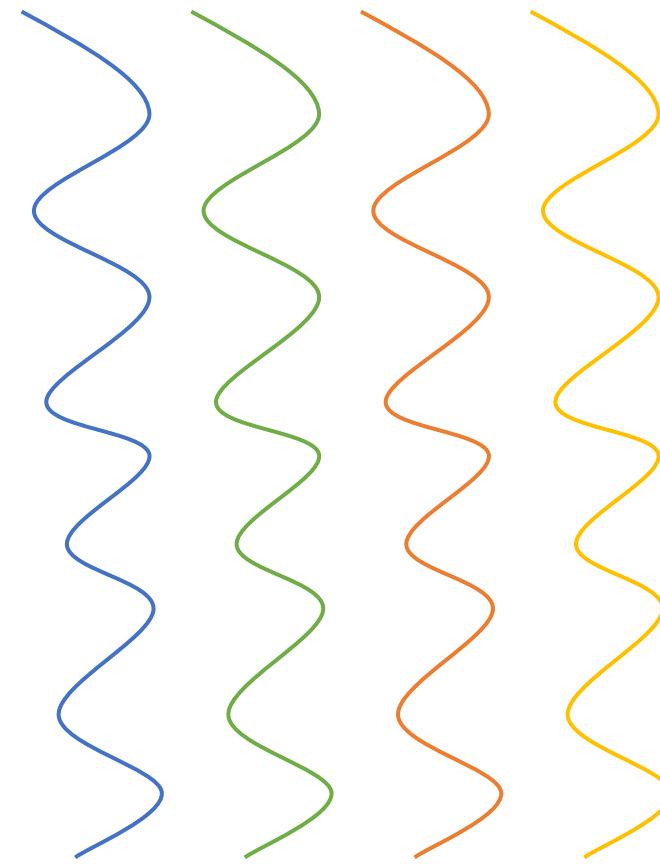
Why don't we *separate* the concept of a **process** from its **execution state**?



هذا Thread!



هذه Threads!





Search Sheet

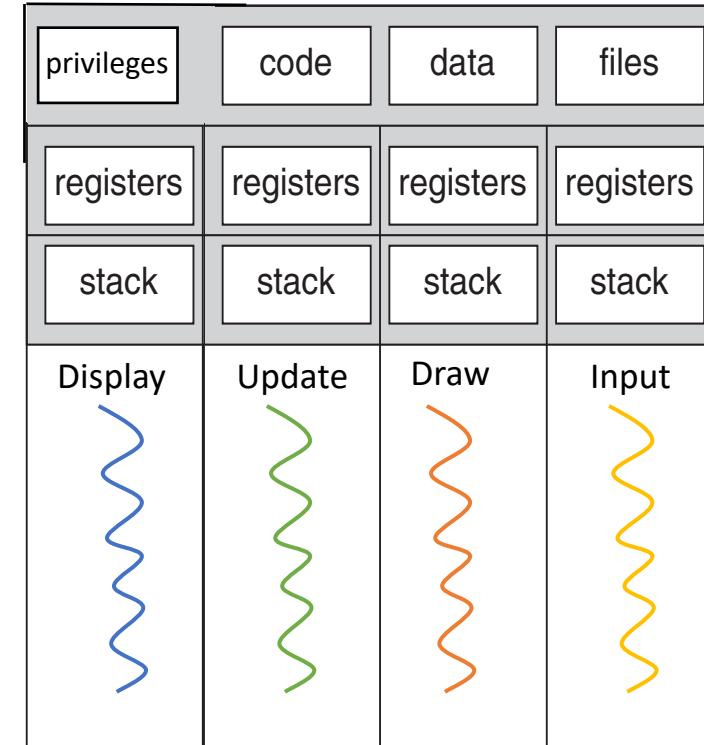
Home Insert Page Layout Formulas Data Review View

Cells Editing

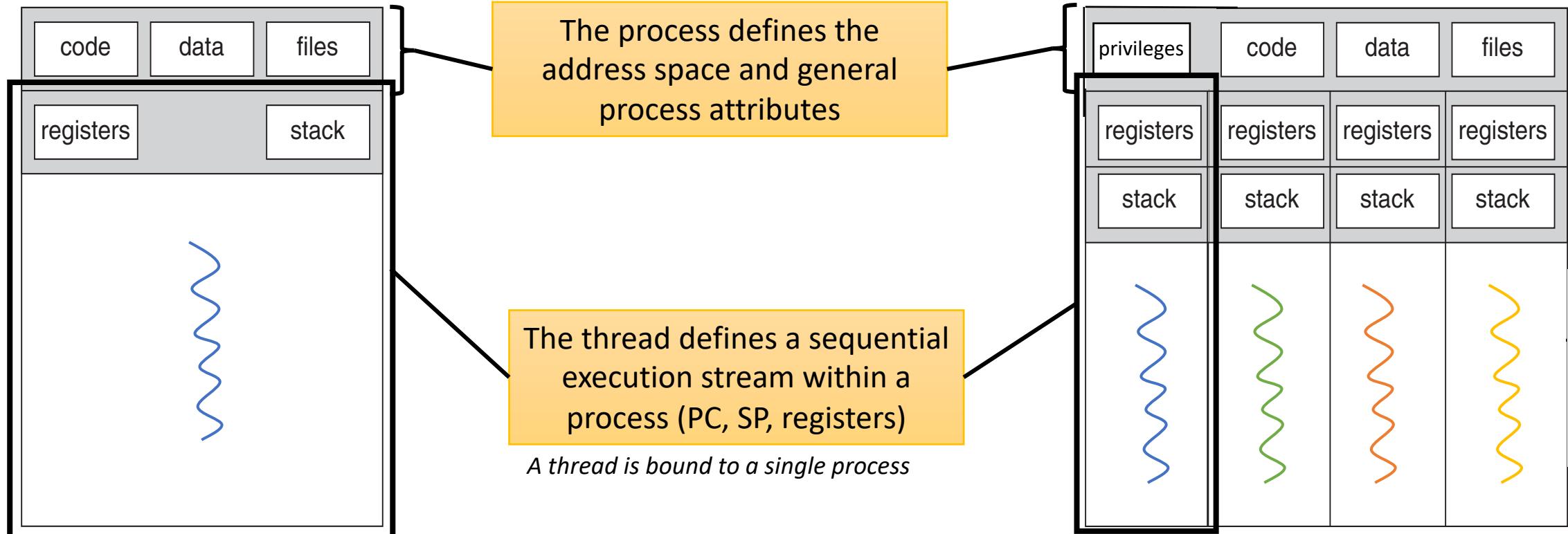
068

A B C D E F G H I J K L M N O P

	Code	SKU	Part Number	Title	Note	Minimum Q	Minimum C	Cost	Retail Price	Sale Price	Weight	Val.	Weight	Unit	Reorder Pq	Brand	Supplier	Classification	Cry
1	54454	1u6h9t22	8000100067014	Gorilla		1000	MQQ; note	\$13.26	\$38.40	\$38.14	1.6 lbs				900	Gorilla Auto	Auto Accs	Lug Nuts	
2	3213186	1uK1208	8000100070714	Gorilla		1500		\$5.00	\$9.00	\$8.14	5.6 lbs				900	Gorilla Auto	Auto Accs	Lug Nut Key	
3	321556	BPK214	A21502013	Stainless Steel Brakes Sh		200		\$100.00	\$100.00	\$403.99	15 lbs				400	SSBC	Auto Parts	Brake Pads and L	
4	467-170105		Bill's Turbo Kits		175			\$11,000.00	\$15,000.00	\$14,492.00	20 lbs				200	Bill's Brand	Bill's Comp	Turbo	
5	355-30627		Call-Back Aluminized Sh		150			\$350.00	\$300.00	\$403.00	78 lbs				250	Bill's Brand	Bill's Comp	Exhaust	
6	023605098U		Pre-Series Driving Shoe		200			\$40.00	\$100.00	\$89.00	2 lbs				150	G-Fence	Auto Parts	Shoes	
7	124-315005		Double Layer Driving S		300			\$27.00	\$49.95	\$35.00	0.3 cu				450	RaceQuip	Auto Racing	Gloves	
8	80115WV7025		Night Off		200			\$10.00	\$16.99	\$14.99	8 lbs				300	Night	ATV Parts	S Light Bar Kits	
9	800046175H		Night Off 2x		350			\$12.99	\$17.99	\$14.99	1.7 lbs				450	Night	ATV Parts	S LCD Free Lights	
10	8007KN010		CARBURET		210			\$11.00	\$13.00	\$14.99	13.6 lbs				300	Bill's Brand	Bill's Comp	Carburetors	
11	800912U07M		PT Dirt		300			\$13.00	\$24.99	\$19.99	8.8 lbs				350	Leadline*	ATV Parts	S Grip Cables	
12	558037		Dirt		275			\$10.00	\$162.95	\$154.85	1.8 lbs				350	SWAT	ATV Parts	S Tire Kits	
13	800MDU771G		Starter		175			\$8.99	\$13.00	\$12.95	0.8 lbs				200	Bill's Brand	Bill's Comp	Start Relays	
14	8005PWP10		Fuel		300			\$2.00	\$3.00	\$4.58	0.3 cu				300	Bill's Brand	Bill's Comp	Fuel Filters	
15	41029		HHP Power		75			\$180.00	\$300.00	\$287.99	3.42 lbs				200	HHP	Muffler	Sug Mufflers	
16	-4000230		Carb		175			\$15.00	\$23.00	\$19.99	8 lbs				250	Bill's Brand	Bill's Comp	Carburetor Kits	
17	5006440185Z		Intake		350			\$99.99	\$107.00	\$100.00	2 lbs				300	S&S	Motorcycle	Intake Manifold	
18	7069533		Bell Block/		350			\$1,000.00	\$3,000.00	\$1,299.95	3 lbs				500	Bell	Motorcycle	Helmets	
19	11-0087		Black		350			\$20.00	\$30.00	\$29.69	0.63 lbs				500	Moto Pro	Motorcycle	Helmets Lock	
20	80050202N		Harley		210			\$100.00	\$160.00	\$129.95	5 lbs				200	Harley Dav	Motorcycle	Boots	
21	16118		Harley		300			\$210.00	\$600.00	\$399.00	10 lbs				210	Harley Dav	Motorcycle	Jackets	
22	2879600		Victory Motorcycle Oil		250			\$50.00	\$70.00	\$66.91	10 lbs				300	Victory	Motorcycle	Oil Change Kit	
23	18793		Spill Saver		350			\$2.10	\$3.00	\$4.72	1.3 lbs				400	Hopkins	Motorcycle	Funnels	
24	80020203N		Covercraft		250			\$17.00	\$21.44	\$68.35	10 lbs				300	Covercraft	Cover Suppl	Car Covers	
25	8000CP18M		DR		300			\$11.00	\$19.00	\$16.90	1.7 lbs				450	Red Line-O	Auto Maint	Transmission Fl	
26	805108 WH40-300-Y		Exhaust Y		300			\$80.77	\$149.99	\$109.99	3.3 lbs				600	Bill's Brand	Bill's Comp	Exhaust Pipes	
27	805108 WH40-300-Y		Exhaust Y		300			\$80.77	\$149.99	\$109.99	3.3 lbs				300	Dev Bo	Boat Accss	Marine Chocks	
28	805108 WH40-300-Y		Dev Bo		210			\$44.57	\$114.00	\$69.99	16.4 lbs				270	Polyform	Boat Accss	Boat Fenders	
29	805108 WH4130		Polyform		430			\$8.99	\$11.00	\$12.44	1 lbs				250	Shurhold	Boat Accss	Boat Hooks	
30	3005108 WH4130		Shurhold		300			\$6.48	\$19.00	\$114.47	4.8 lbs				300	Bill's Brand	Bill's Comp	Boat Ladders	
31	680870 IMP1JAM		Under		215			\$114.00	\$199.00	\$141.00	14.8 lbs				250	Kwick Tek	Boat	Boat Bag In Boot Buffer Bags	
32	431128 DP-01		Dry Pak		300			\$18.89	\$50.00	\$45.00	5.1 lbs				250	Denco	Boat Accss	Flashing	
33	648483 41-2510		Donty		200			\$4.01	\$10.99	\$6.98	7.2 lbs				270	OMM/FAD	Boat Parts	Fuel Pump Assy	
34	2332054 356-040050-E		DEM		300			\$43.00	\$65.00	\$15.10	4.8 lbs				350	K&N	Engine	Motorcycle Oil Filters	
35	83482 KN204		K&N KN		275			\$5.23	\$13.11	\$12.45	7 lbs				200	Atlantic	Boat Accss	Flush Kits	
36	239951 AHS42		800210H-TKE		150			\$8.19	\$21.95	\$14.36	3.6 lbs				250	Gulf Perfalt	Auto Parts	Engines	
37	11648 EG7002		Genome		100	MQQ; note	\$1,444.99	\$5,490.00	\$4,045.01	\$ -	5 lbs				200	Reduced Tax	Auto Parts	Services	
38	399500 350087		Reduced		200			\$4.00	\$6.00	\$3.73	0.8 lbs				2000	Reduced Tax	Auto Parts	Services	



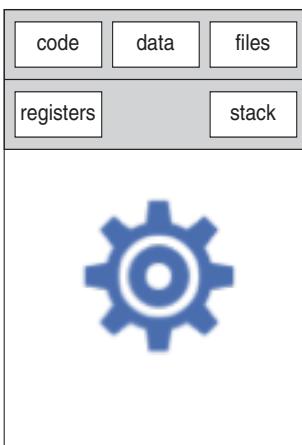
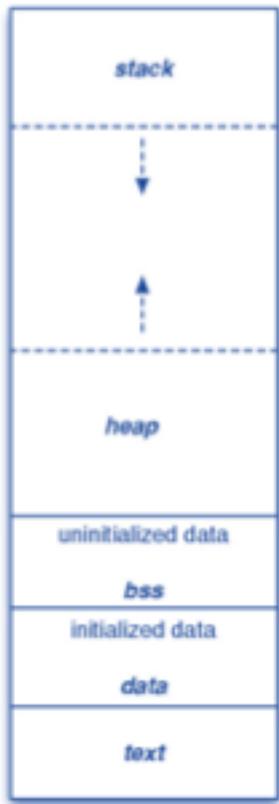
Modern OSes separate the concepts of processes and threads



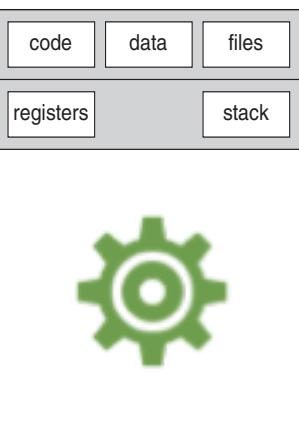
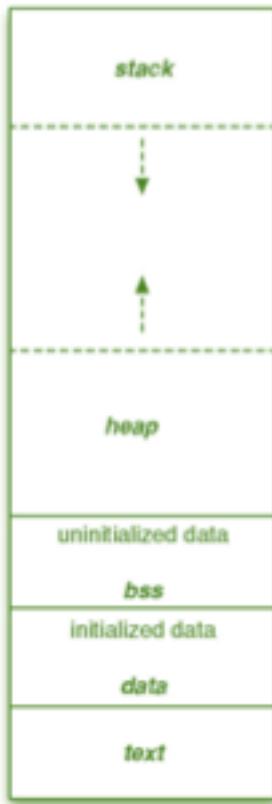
Each process has at least one thread

Processes, however, can have multiple threads

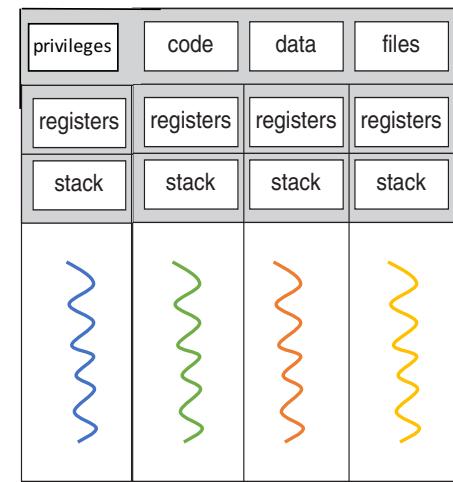




Process



Child Process



Multithreaded Process



Processes



Threads

Process is called *heavyweight* process

Process switching needs interface with the OS

Multiple processes can execute the same code
but each one has its own **address space**

If the process is blocked, nothing is executed
until the process is unblocked

Multiple redundant processes uses more
resources than multiple threads

In multiple process, each process separates
independently of the others

Threads are called *lightweight* processes

Threads switching is handled by the **threading library** and does not need to notify the OS or
cause any interrupts

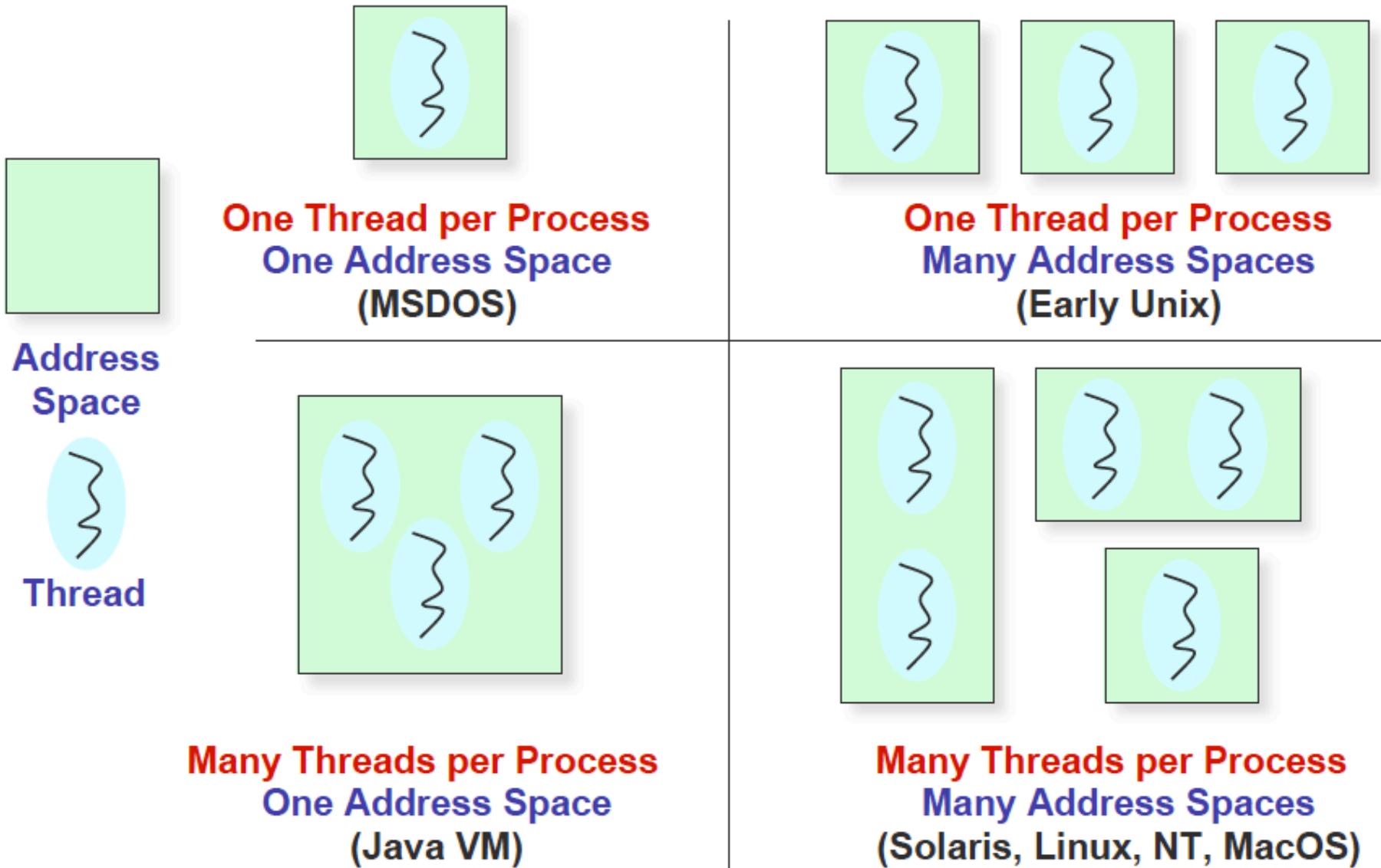
All threads **share the same address space** and see
the same code

While one thread within a process is blocked and
waiting, other threads in the same task can run

Multiple threaded processes uses fewer resources
than multiple redundant process

One thread can read, write or even completely
wipe out another threads stack

Thread Design Space



Why use thread?

Threads are economical

cheaper than process creation, thread switching
lower overhead than context switching

Per process items
Address space
Global variables
Open files
Child processes
Pending alarms
Signals and signal handlers
Accounting information

Per thread items
Program counter
Registers
Stack
State

Threads share Resources

Threads share resources of process, easier than shared memory or message passing



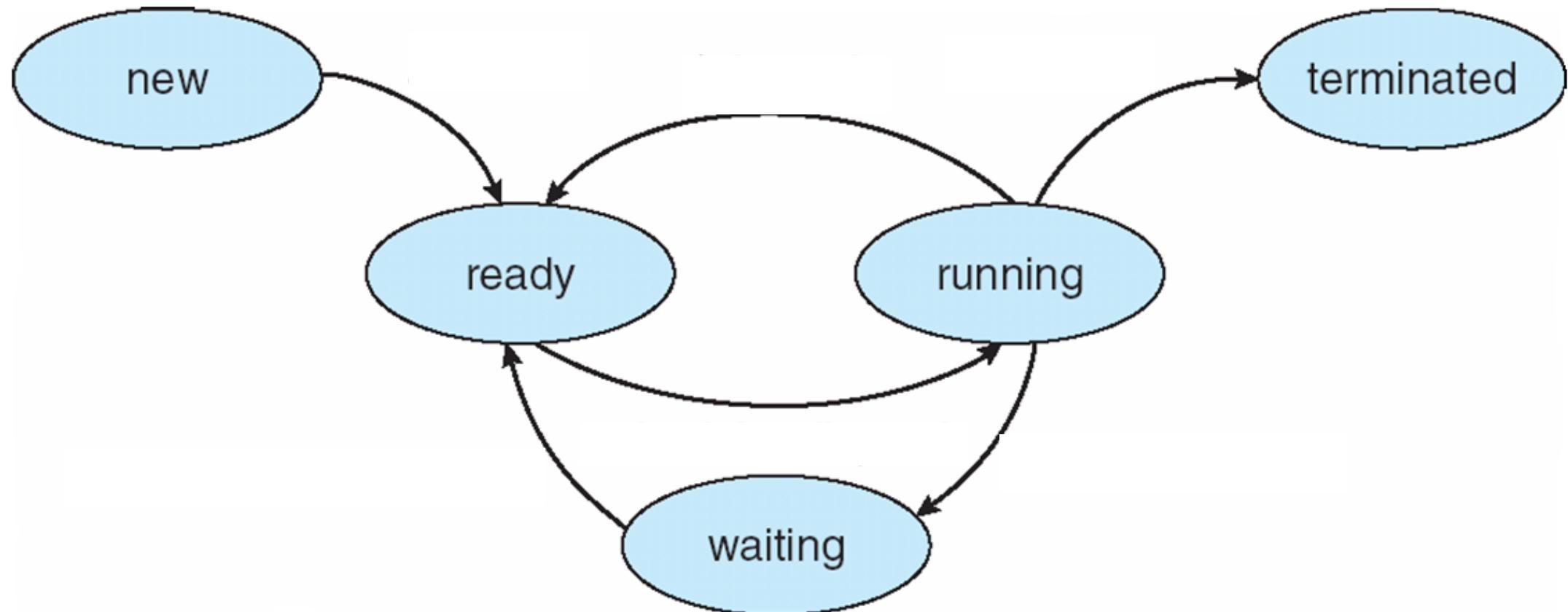
Threads enable Scalability

Process can take advantage of multiprocessor architectures

Threads enables Responsiveness

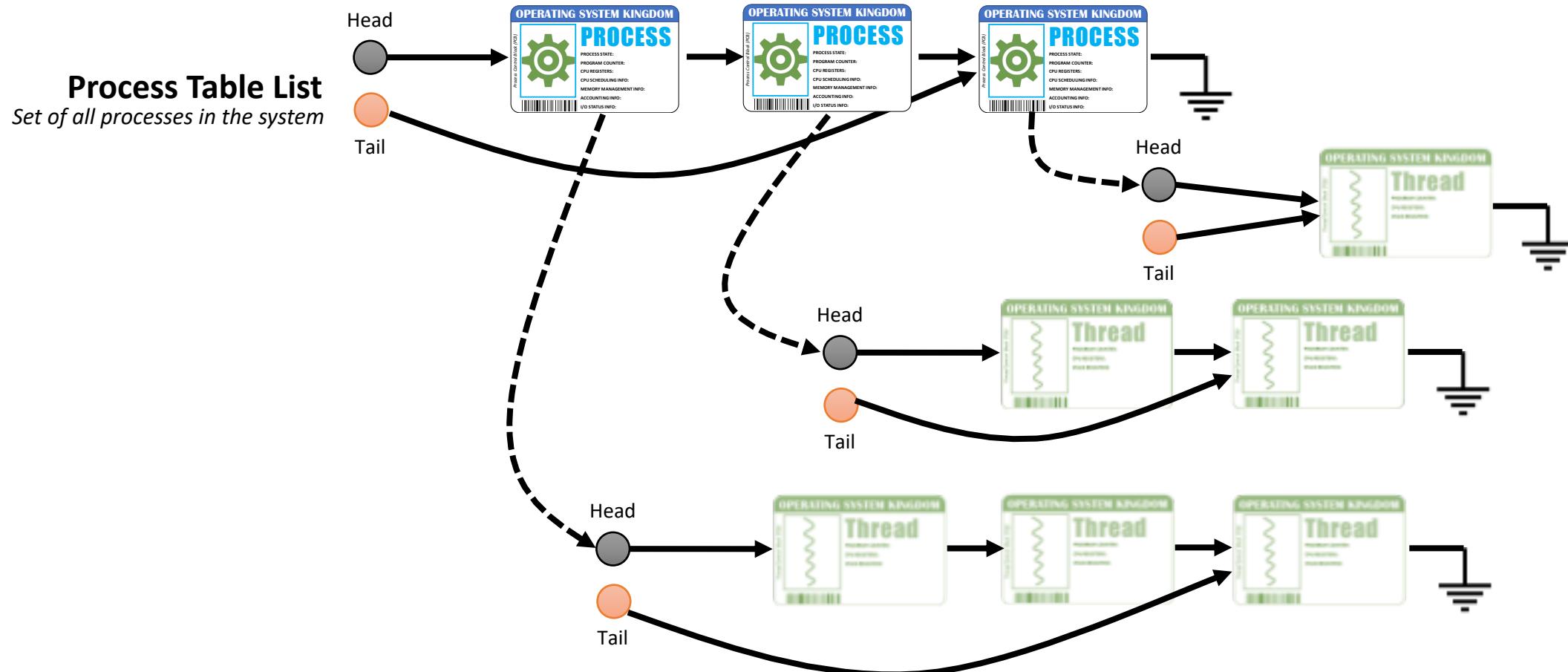
Threads may allow continued execution if part of process is blocked, especially important for user interfaces

Threads have the same as Process states



How the operating system manages threads?

Each PCB will point to a list of Thread Control Blocks



OPERATING SYSTEM KINGDOM

Thread Control Block (TCB)



Thread

PROGRAM COUNTER:

CPU REGISTERS:

STACK REGISTERS:

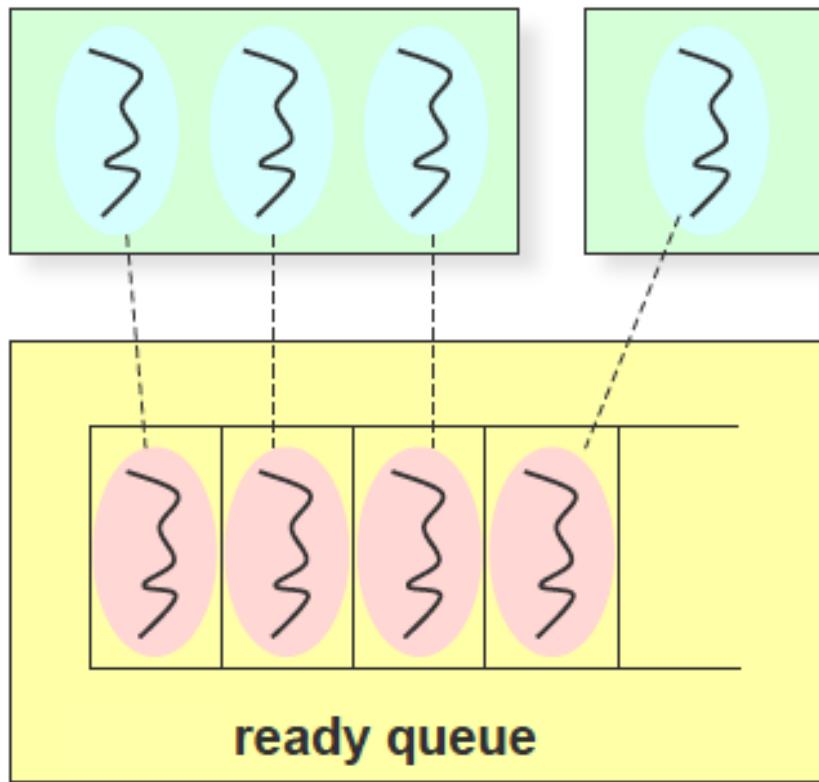
THREAD STATE:





Kernel-Level Threads

Managed directly by the operating system.

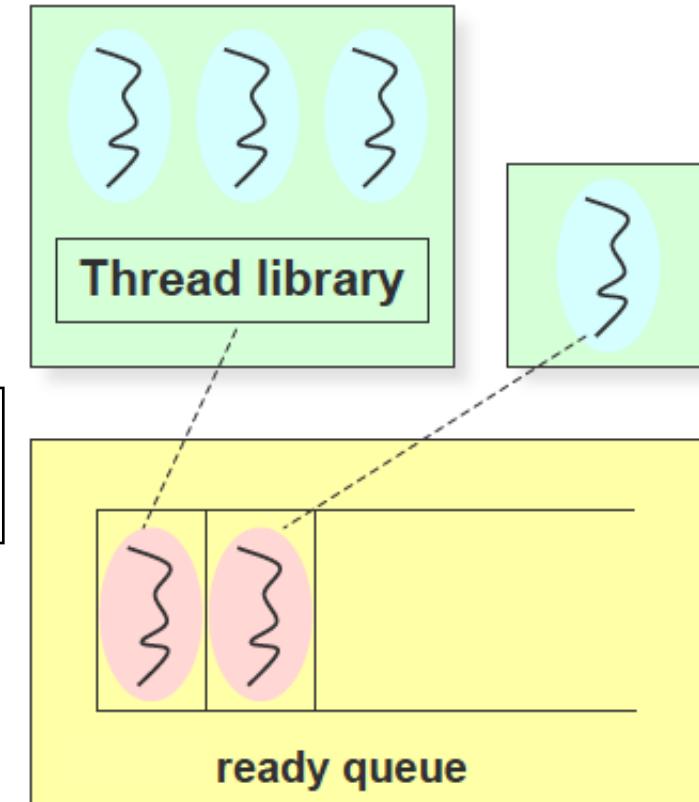


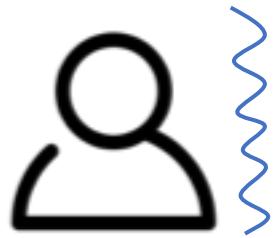
User-Level Threads

Management done by user-level threads library

Thread Libraries

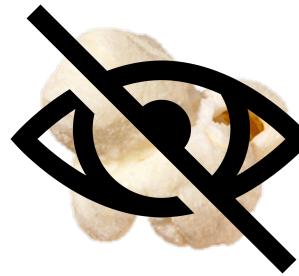
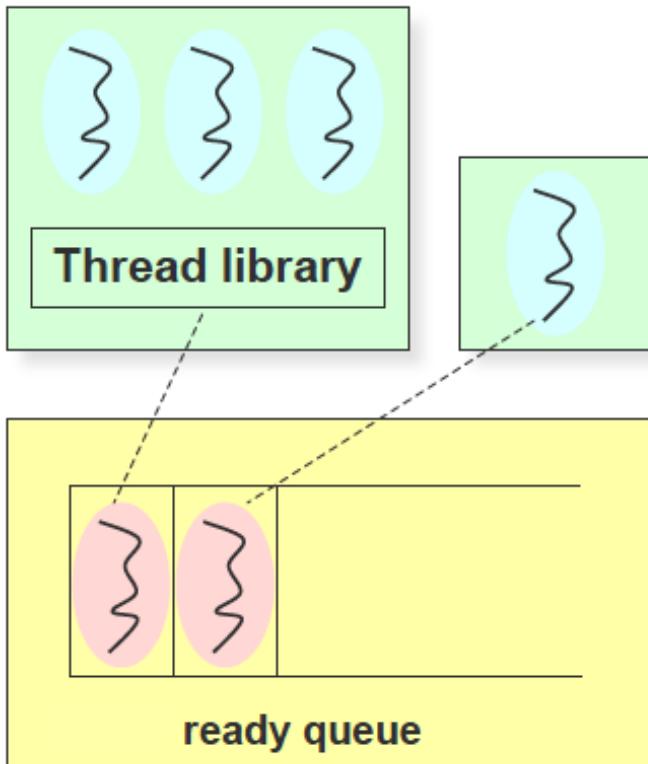
(POSIX Pthreads,
Windows threads,
Java threads)





User-Level Threads

Management done by user-level threads library



Invisible to Kernel

All thread management is done by the thread library and the kernel is not aware of the existence of the user-level threads

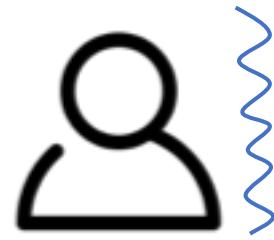
A thread represented inside process by a PC, registers, stack, and small thread control block (TCB)

The thread library contains code for creating and destroying, for passing messages and data between threads.

The thread library contains code for scheduling execution and for saving and restoring thread contexts “**Context Switching**”

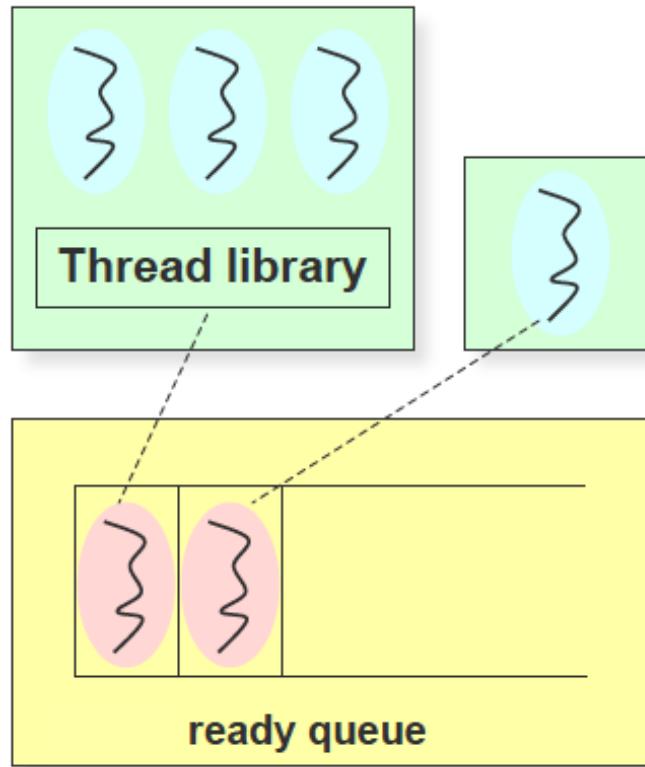
The process begins with one threads and begins running on that threads. Then, it starts spawning new threads as needed

User-level threads are fast to create and manage as there is no kernel involvement

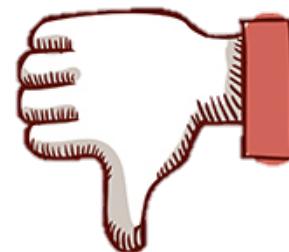


User-Level Threads

Management done by user-level threads library



- Thread switching does not require kernel mode privileges
- User-level thread can run on any operating system
- Scheduling can be application specific
- User-level threads are fast to create and manage
- Creating a new thread, switching, and synchronizing threads are done via **user-level procedure call**



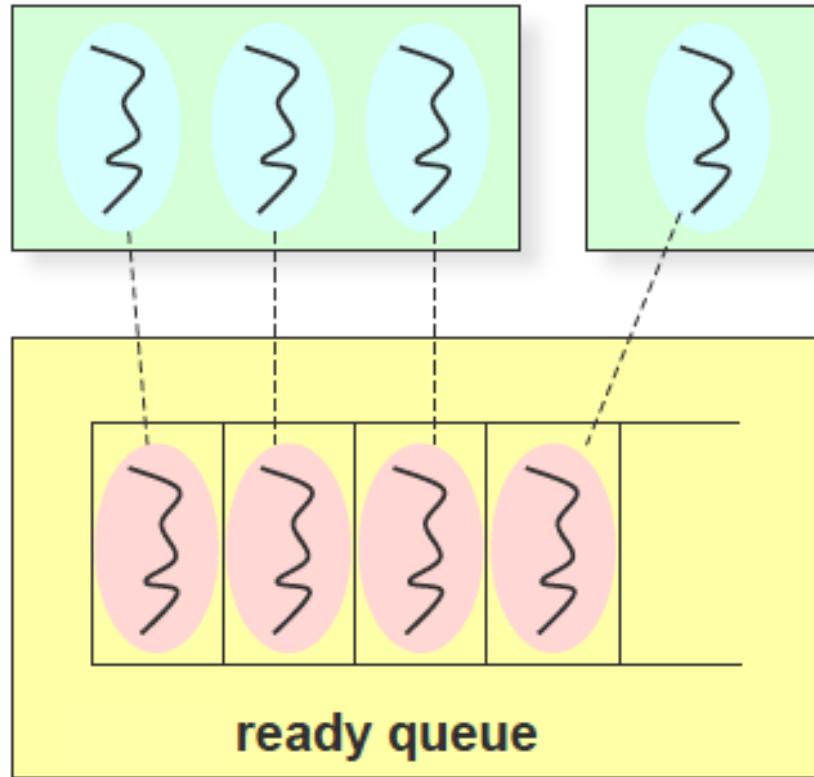
- Most system calls are blocking
- Blocking a process whose thread initiated an I/O, even though the process has other threads that can execute
- Multithreaded application cannot take advantage of multiprocessing as these threads are invisible to kernel

There should be communication between the kernel and the user-level thread manager



Kernel-Level Threads

Managed directly by the operating system.



OS-managed threads are called kernel-level threads or lightweight processes (LWP)

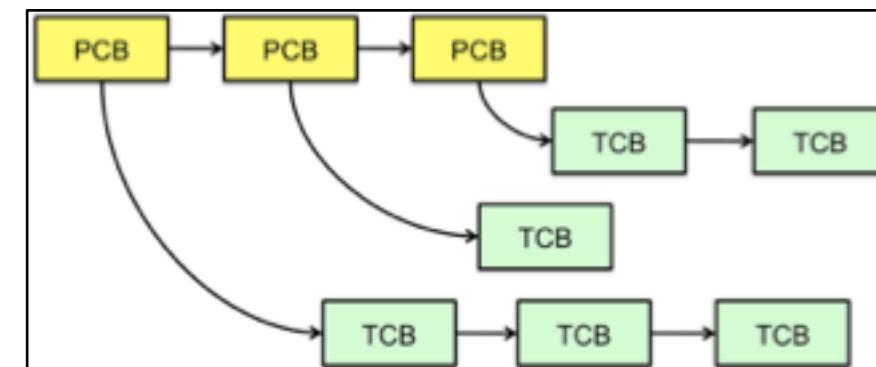
Thread management is done by the kernel and are supported directly by the operating system

The kernel maintains context information for the process as a whole and for the individual threads within the process

No longer scheduling processes, but scheduling in kernel is now on thread basis. (**Scheduler deals in threads**)

The kernel performs thread creation, scheduling, and management in kernel space

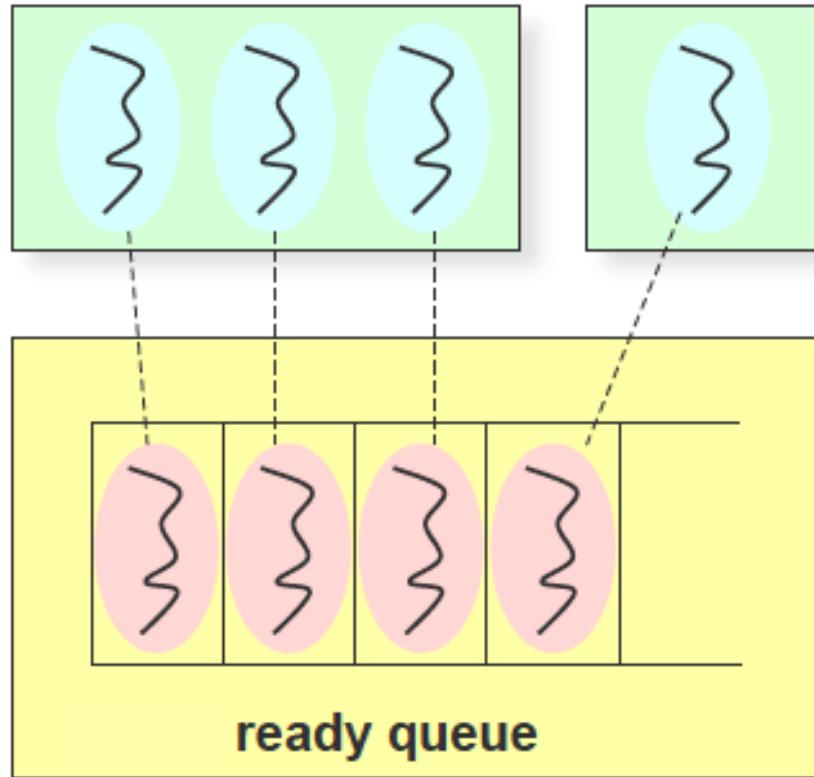
Kernel-level threads are generally slower to create and manage than the user-level threads; **Full Context Switching**





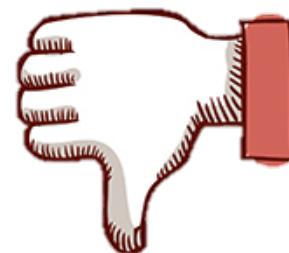
Kernel-Level Threads

Managed directly by the operating system.



Kernel can simultaneously schedule multiple threads from the same process on multiple processors

If one thread is blocked, the Kernel can schedule another thread in the same process



Kernel threads are generally slower to create and manage than user-level threads

Transfer of control from one thread to another thread within the same process requires a mode switch to the kernel



Kernel-Level Threads

Managed directly by the operating system.

Slower to create and manage

Directly supported by operating system

Specific to the operating system

Kernel routines can be multithreaded



User-Level Threads

Management done by user-level threads library

100x Faster to create and manage

Implemented by a thread library at the user level

Can run on any operating system

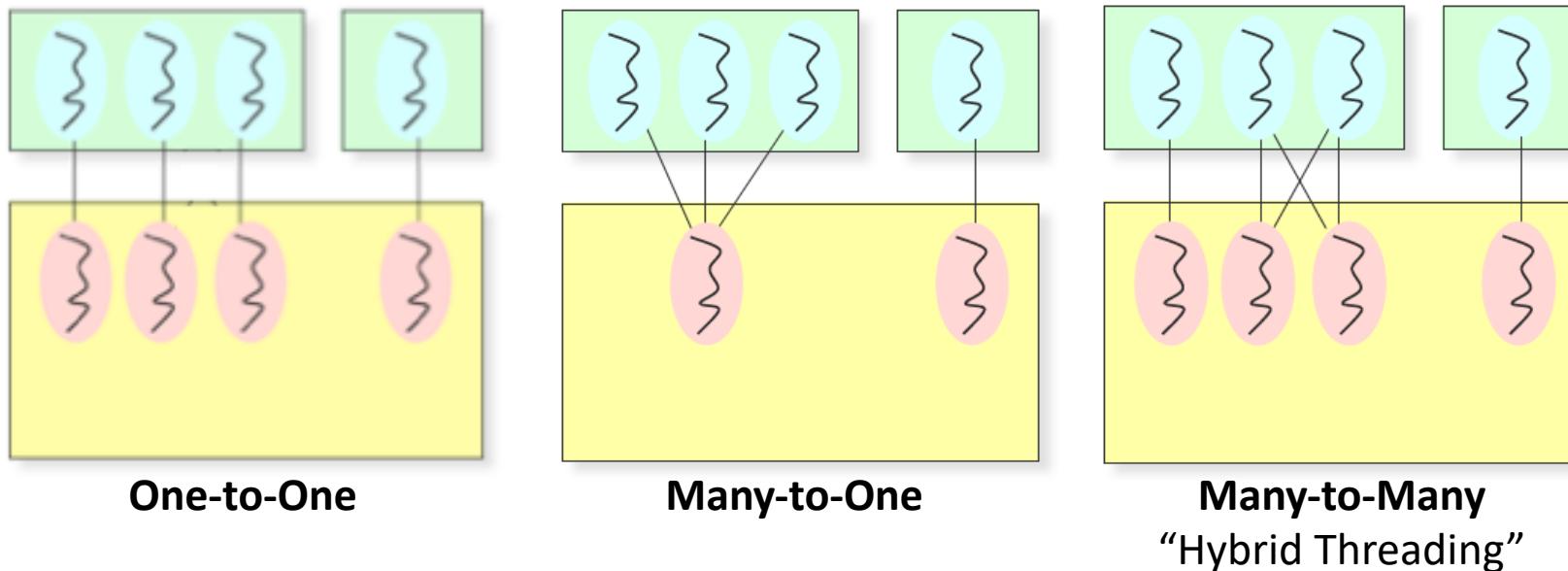
Multithreaded applications cannot take advantage
of multiprocessing

Combining user and kernel-level threads

it's possible to have a program use both user-level and kernel-level threads

An example of why this might be desirable is to have the thread library create several kernel threads to ensure that the operating system can take advantage of **hyperthreading** or **multiprocessing** while using more efficient user-level threads when a very large number of threads is needed.

Several user level threads can be run over a single kernel-level thread



Threads Libraries

provides programmer with API for creating and managing threads



User-Level Thread Library

All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.



Kernel-Level Thread Library

All code and data structures for the library exist in kernel space. Invoking a function in the API for the library typically results in a system call to the kernel.



POSIX Pthreads

Pthreads are IEEE Unix standard library calls “Specification not Implementation”

Linux Implementation: <https://www.gnu.org/software/hurd/libpthread.html>

Windows Implementation: <https://sourceforge.net/projects/pthreads4w/>

```
$ locate libpthread.so
```

There are around 100 Pthreads procedures, all prefixed `pthread_`

Thread call	Description
<code>Pthread_create</code>	Create a new thread
<code>Pthread_exit</code>	Terminate the calling thread
<code>Pthread_join</code>	Wait for a specific thread to exit
<code>Pthread_yield</code>	Release the CPU to let another thread run
<code>Pthread_attr_init</code>	Create and initialize a thread's attribute structure
<code>Pthread_attr_destroy</code>	Remove a thread's attribute structure

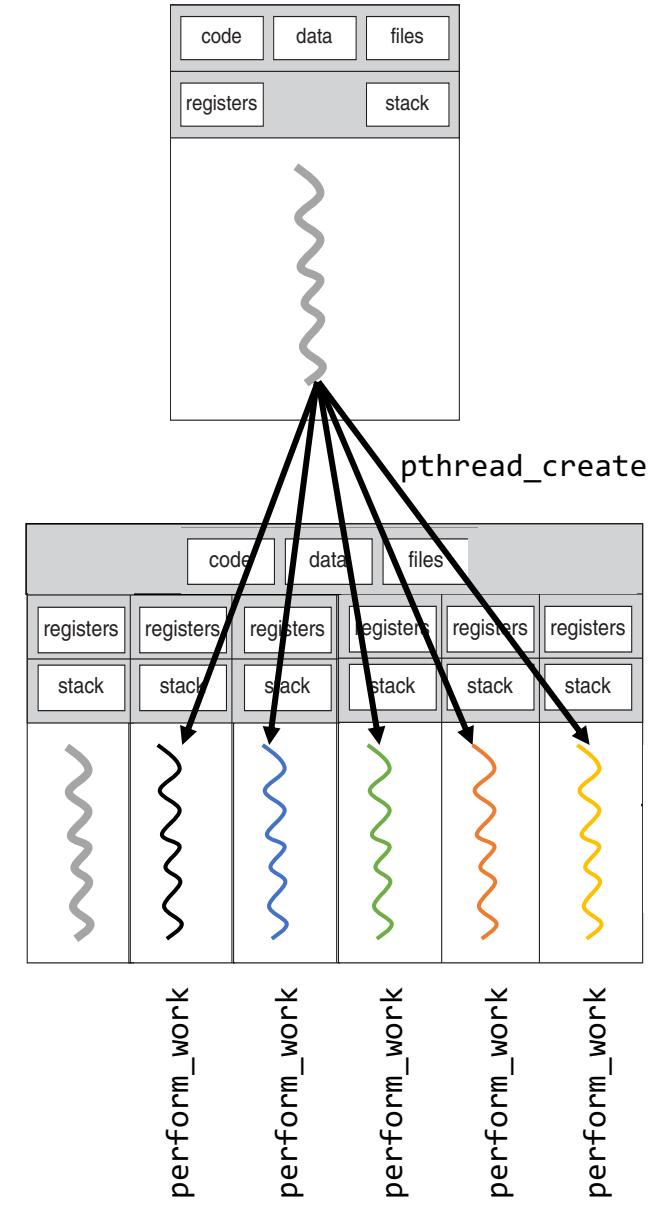
```

1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <assert.h>
5
6 #define NUM_THREADS      5
7
8 void* perform_work(void *argument){
9     int passed_in_value;
10    passed_in_value = *( ( int* )argument );
11    printf( "Hello World! It's me, thread with argument %d!\n", passed_in_value );
12    return NULL;
13 }
14
15 int main( int argc, char** argv ){
16    pthread_t threads[ NUM_THREADS ];
17    int thread_args[ NUM_THREADS ];
18    int result_code;
19    unsigned index;
20
21    // create all threads one by one
22    for( index = 0; index < NUM_THREADS; ++index ){
23        thread_args[ index ] = index;
24        printf("In main: creating thread %d\n", index);
25        result_code = pthread_create( threads + index, NULL, perform_work, thread_args + index );
26        assert( !result_code );
27    }
28
29    // wait for each thread to complete
30    for( index = 0; index < NUM_THREADS; ++index ){
31        // block until thread 'index' completes
32        result_code = pthread_join( threads[ index ], NULL );
33        assert( !result_code );
34        printf( "In main: thread %d has completed\n", index );
35    }
36
37    printf( "In main: All threads completed successfully\n" );
38    exit( EXIT_SUCCESS );
39 }

```

Synchronous Threading

https://en.wikipedia.org/wiki/POSIX_Threads#Example



Linux Kernel Threads

<https://github.com/torvalds/linux/blob/master/include/linux/sched.h>

<https://github.com/torvalds/linux/blob/master/kernel/kthread.c>

Further Reading: http://www.crashcourse.ca/wiki/index.php/Kernel_threads



つづく