

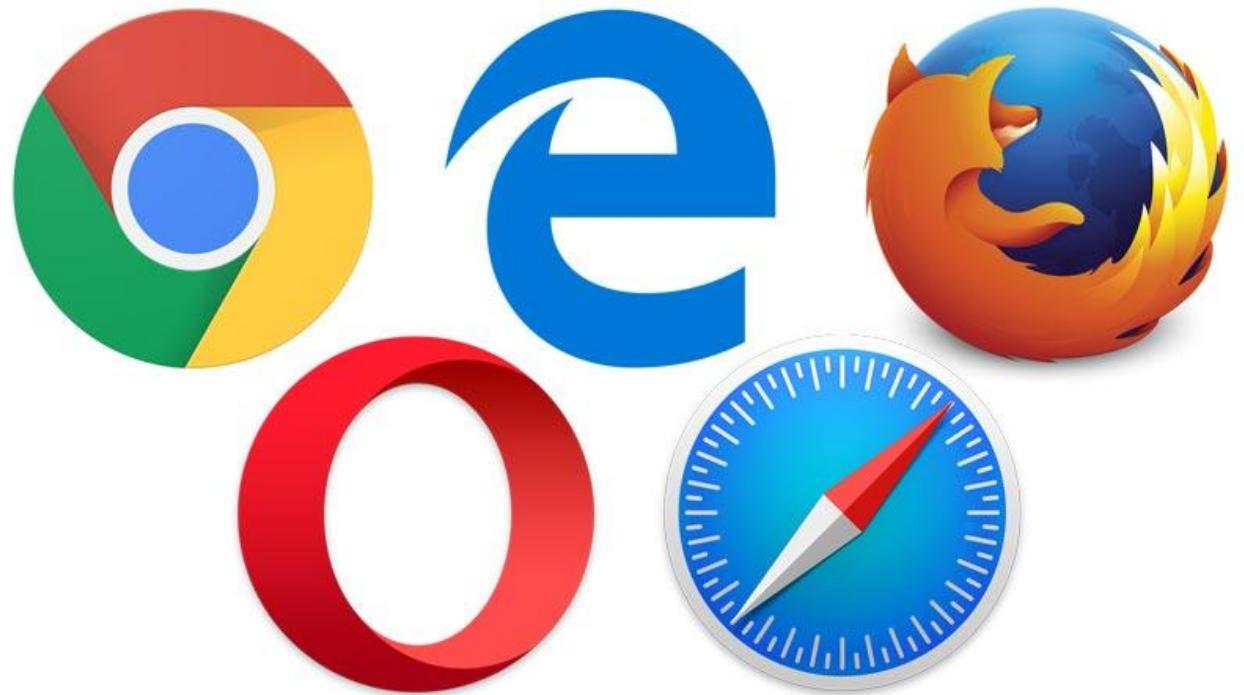


Operating System Concepts: Lecture 3

programs/processes and threads : from bios to application startup

Processes

Processes: Building blocks of what we do on a computer



Job vs Process

- Job: slightly older term (used in batch systems) but still used in literature (“job scheduling”)
- Process: Actually the same thing
- View processes :
 - Windows
 - Linux : ps command

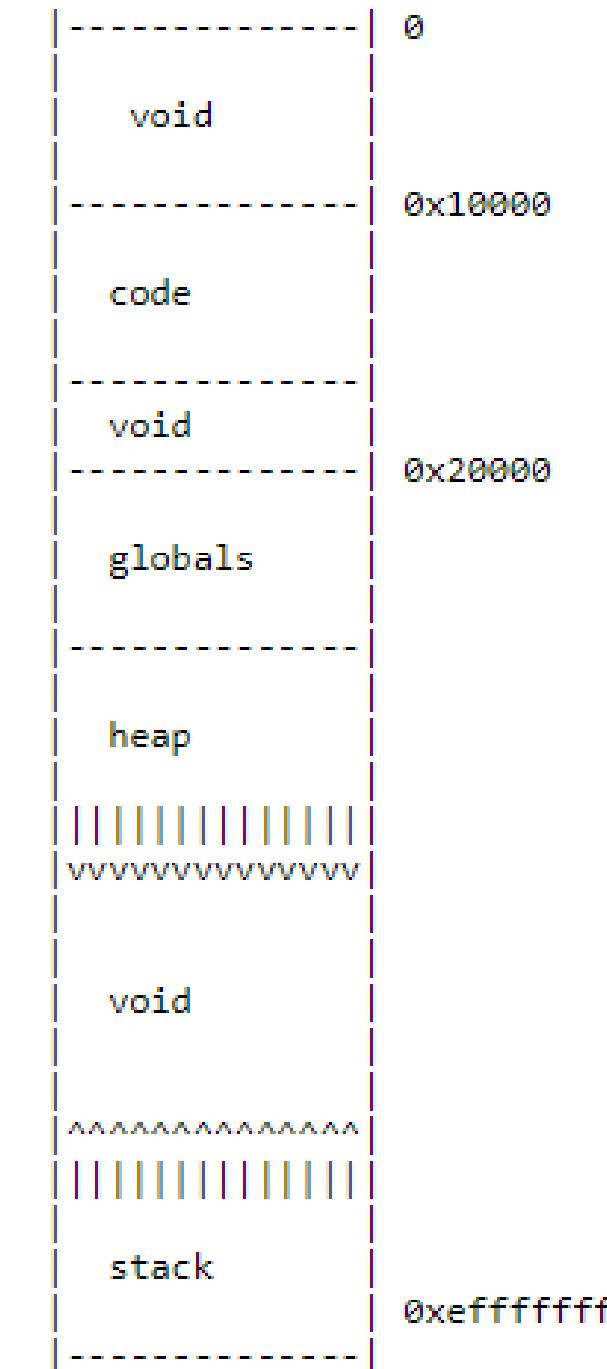
```
sagar@itsFOSS:~$ ps -aux
sagar@itsFOSS:~$
```

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Threads	VirusTotal
Registry		3.952 K	40.036 K	120			4	
System Idle Process	97.19	56 K	8 K	0			8	
System	0.18	196 K	1.248 K	4			232	
+ csrss.exe	< 0.01	1.984 K	3.508 K	636			13	
+ wininit.exe		1.736 K	4.552 K	720			2	
GoogleCrashHandler.exe		1.688 K	1.208 K	5020			4	
GoogleCrashHandler64.exe		1.696 K	888 K	5360			4	
csrss.exe	0.08	2.964 K	5.400 K	12644			15	
winlogon.exe		2.600 K	8.264 K	16044			5	
igfxEM.exe		4.220 K	12.688 K	240 igfxEM Module	Intel Corporation		8	
+ explorer.exe	0.03	149.908 K	166.220 K	6536 Windows Verkenner	Microsoft Corporation		98	
SecurityHealthSystray.exe		1.880 K	7.620 K	16172 Windows Security notification ...	Microsoft Corporation		3	
RtkNGUI64.exe		4.716 K	11.408 K	9952 Realtek HD Audio configurati...	Realtek Semiconductor		10	
RAVBg64.exe	< 0.01	4.756 K	9.880 K	2172 HD Audio Background Proce...	Realtek Semiconductor		7	
WavesSvc64.exe	< 0.01	22.420 K	13.956 K	4668 Waves MaxxAudio Service A...	Waves Audio Ltd.		12	
OneDrive.exe	0.15	338.120 K	307.448 K	4532 Microsoft OneDrive	Microsoft Corporation		37	
Zoomlt.exe		1.544 K	6.356 K	11888 Sysinternals Screen Magnifier	Sysinternals - www.sysinter...		3	
openvpn-gui.exe		2.808 K	7.624 K	5472			3	
CCXProcess.exe		776 K	2.588 K	13636 CCXProcess	Adobe Systems Incorpor...		2	
ONENOTEM.EXE		2.908 K	1.912 K	14308 Send to OneNote Tool	Microsoft Corporation		3	
notepad.exe		3.052 K	12.540 K	14744 Kladblok	Microsoft Corporation		3	
chrome.exe	0.04	115.444 K	158.872 K	15680 Google Chrome	Google LLC		32	
OUTLOOK.EXE	< 0.01	188.040 K	244.900 K	10748 Microsoft Outlook	Microsoft Corporation		64	
vmware.exe	0.01	73.332 K	73.376 K	10276 VMware Workstation	VMware, Inc.		16	
vmware-unity-helper.exe		7.052 K	15.440 K	428 VMware Unity Helper	VMware, Inc.		3	
POWERPNT.EXE	< 0.01	271.200 K	315.496 K	11072 Microsoft PowerPoint	Microsoft Corporation		40	
AcroRd32.exe		11.232 K	17.380 K	9504 Adobe Acrobat Reader DC	Adobe Systems Incorpor...		11	
AcroRd32.exe	< 0.01	84.636 K	80.424 K	10160 Adobe Acrobat Reader DC	Adobe Systems Incorpor...		20	
RdrCEF.exe		17.532 K	29.300 K	6520 Adobe RdrCEF	Adobe Systems Incorpor...		22	
RdrCEF.exe		50.252 K	32.764 K	6544 Adobe RdrCEF	Adobe Systems Incorpor...		13	
procexp64.exe	0.90	88.144 K	104.408 K	16736 Sysinternals Process Explorer	Sysinternals - www.sysinter...		7	
ApntEx.exe		1.996 K	6.764 K	15896 Alps Pointing-device Driver f...	Alps Electric Co., Ltd.		4	
vpnv.exe	0.04	6.572 K	19.692 K	12956 Cisco AnyConnect User Interf...	Cisco Systems, Inc.		7	

Process vs Program

- Program/Binary/Executable (/application/app):
 - passive
 - has no state
 - program code + initialized data
 - (= is not running!)
- Process:
 - active
 - has a state
 - program code + program counter + stack + data section + heap + ...

If we view memory as a big array, the regions (or "segments") look as follows:



Process states

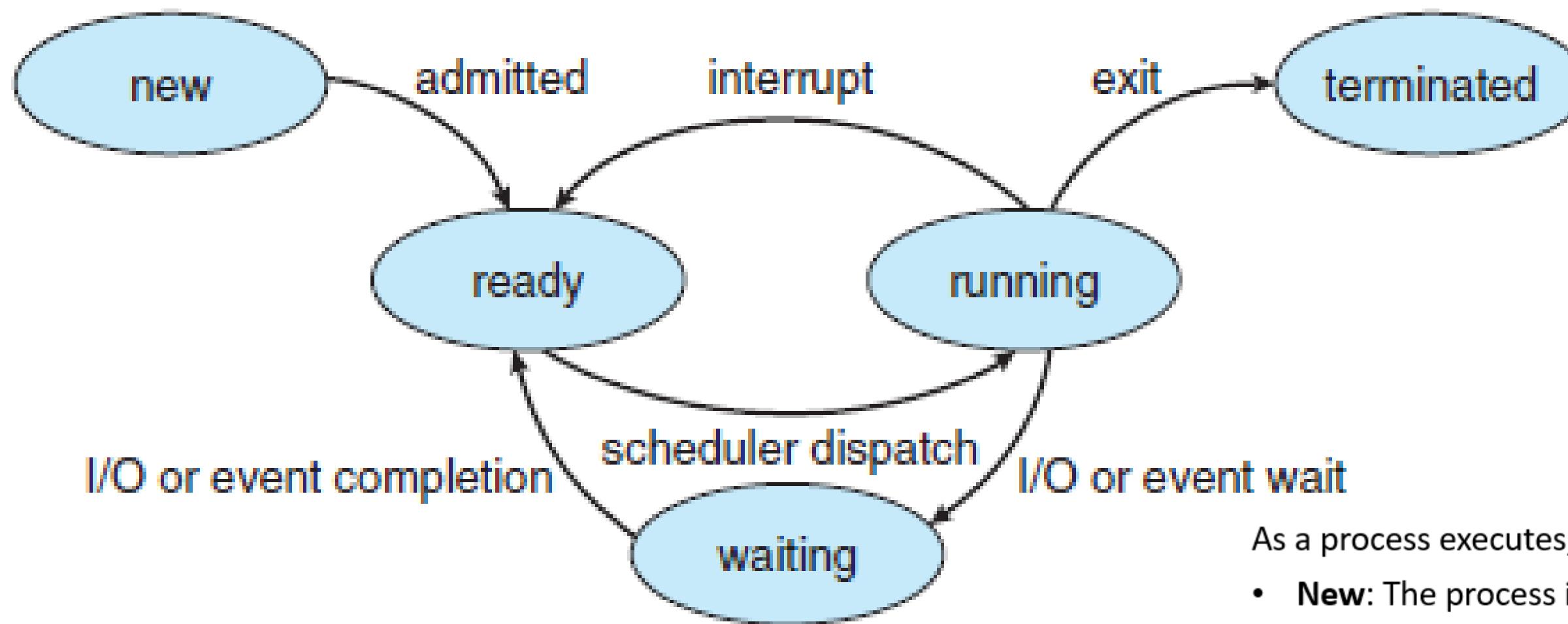


Figure 3.2 Diagram of process state.

As a process executes, the **state** changes:

- **New:** The process is being created
- **Running:** Instructions are being executed
- **Waiting:** The process is waiting for some event to occur
- **Ready:** The process is waiting to be assigned to a processor
- **Terminated:** The process has finished execution

Context Switch

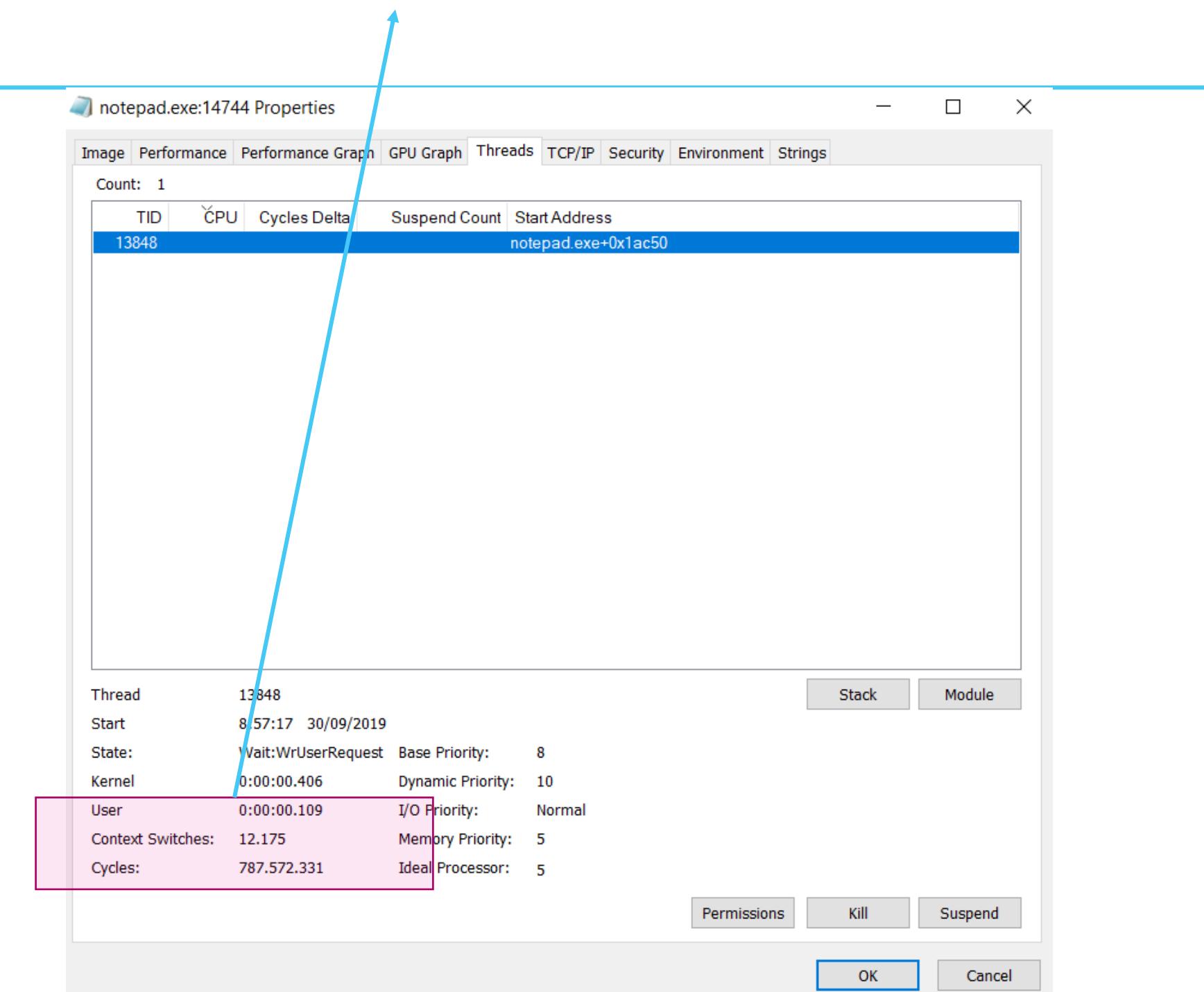
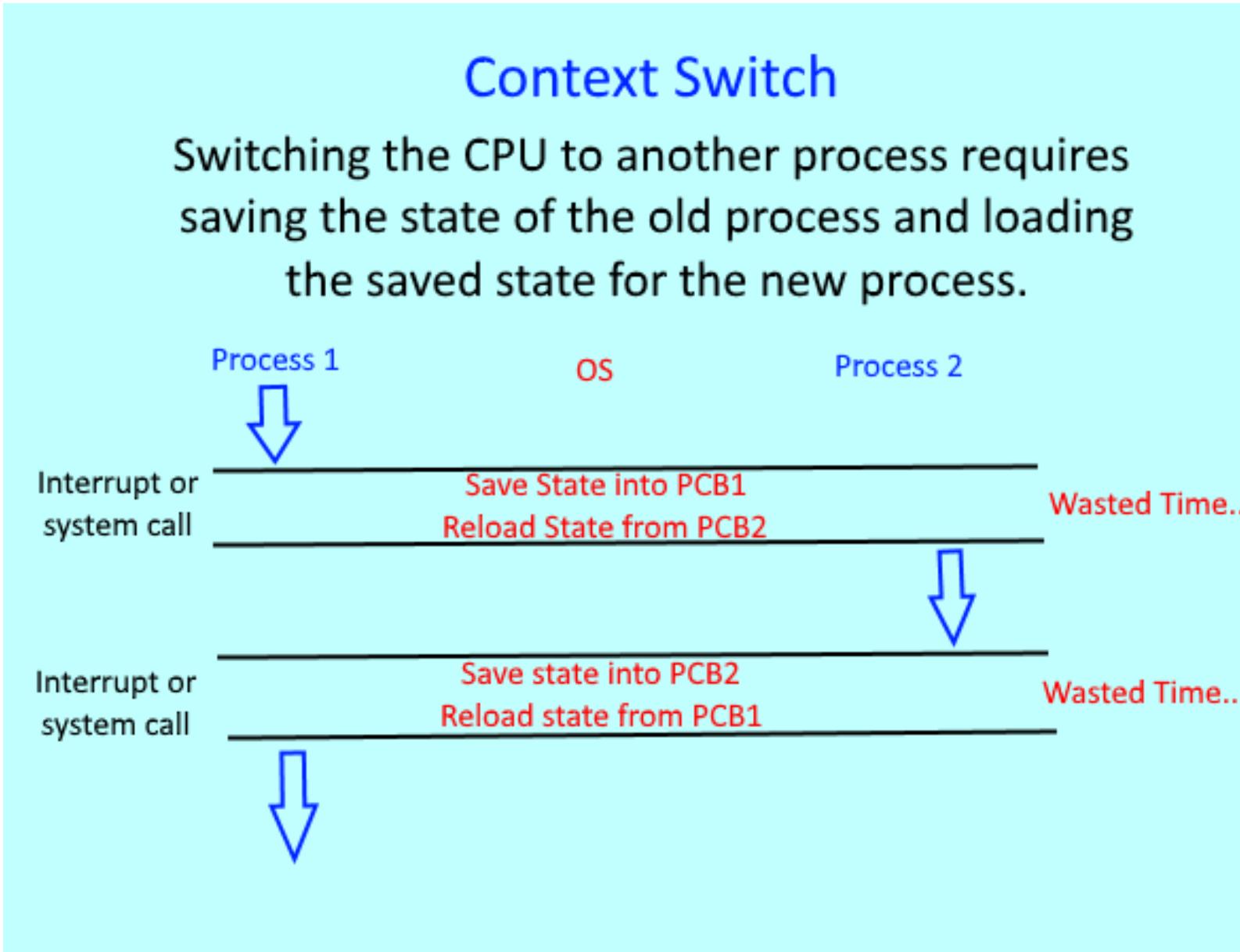
= Saving/storing the state of a process (or thread)

As a result a **single CPU** can **allow multiple processes** -> multitasking operating system!

What actions the context switch performs depends on multiple things:

- Type of OS
- One or more register set
- Sometimes it is needed to go from user mode to kernel mode
- Sometimes as a result of interrupts
- ...

Context Switch



<https://stackoverflow.com/questions/7439608/steps-in-context-switching>

(OS) Stack vs Heap

Stack vs. Heap

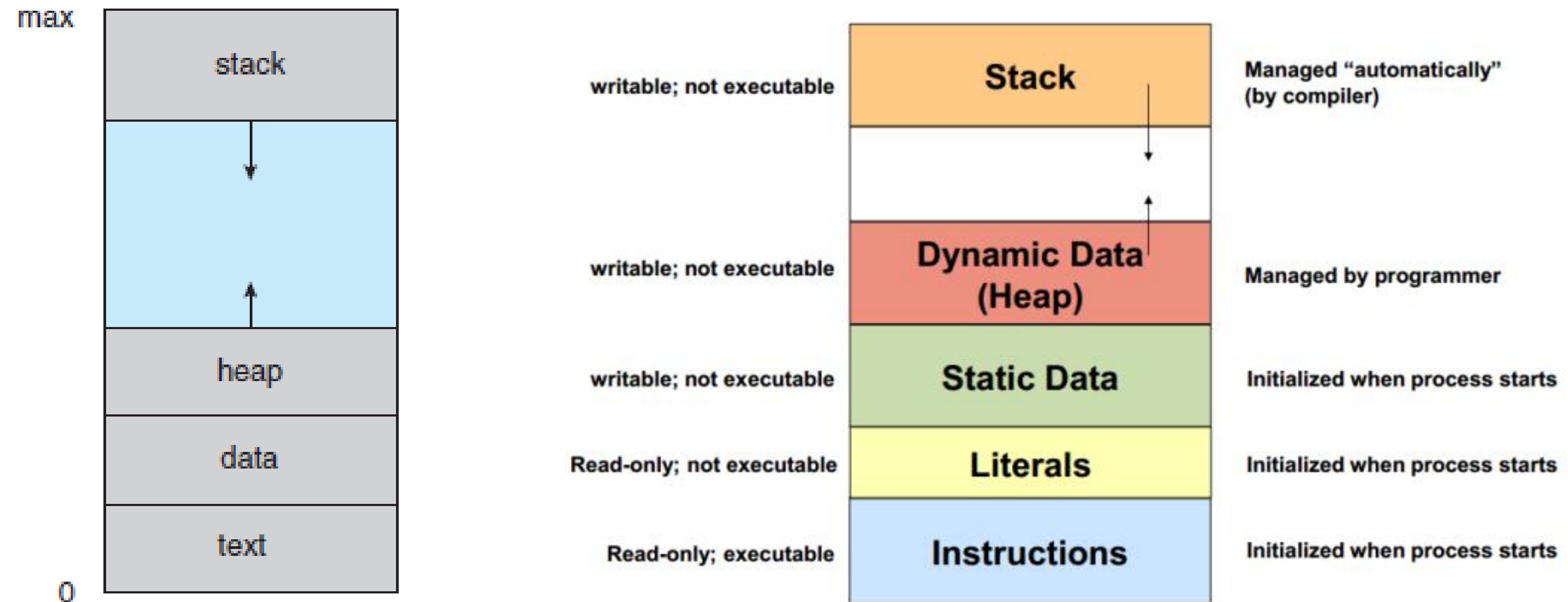
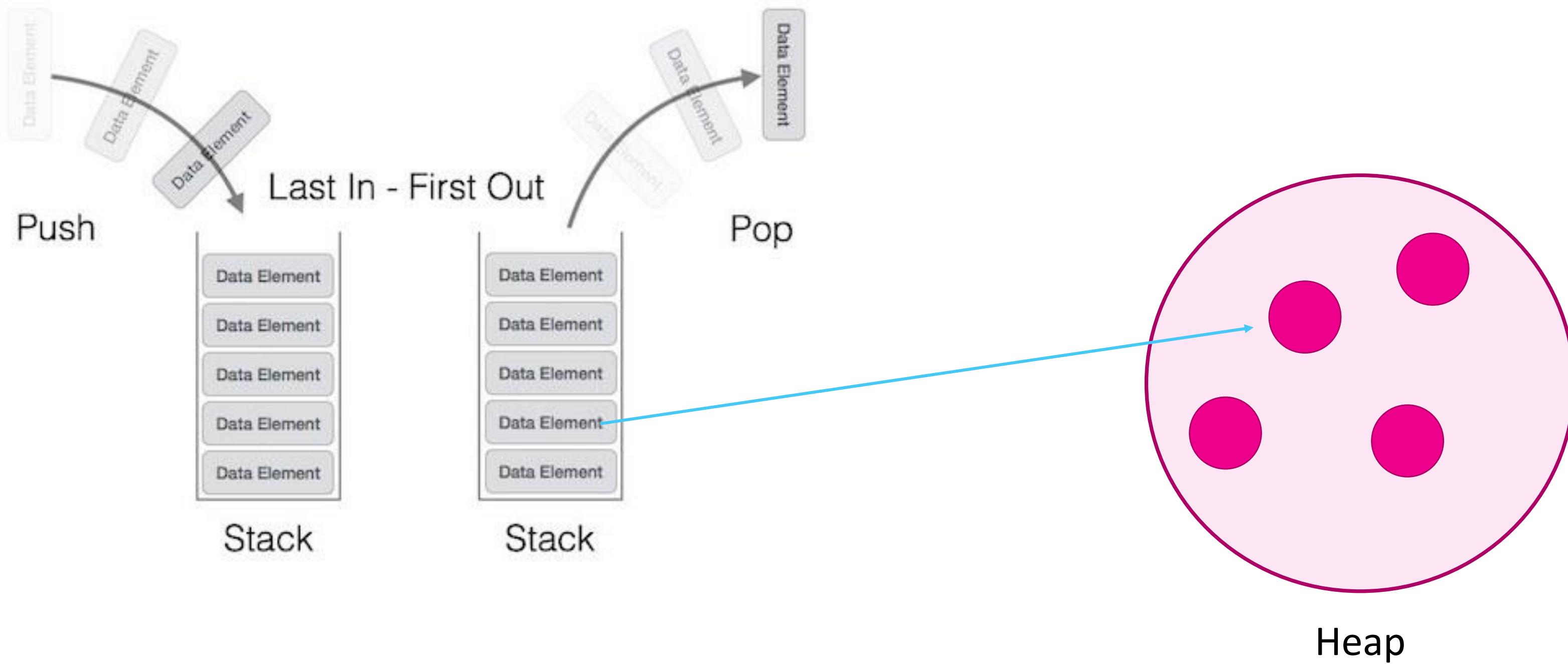
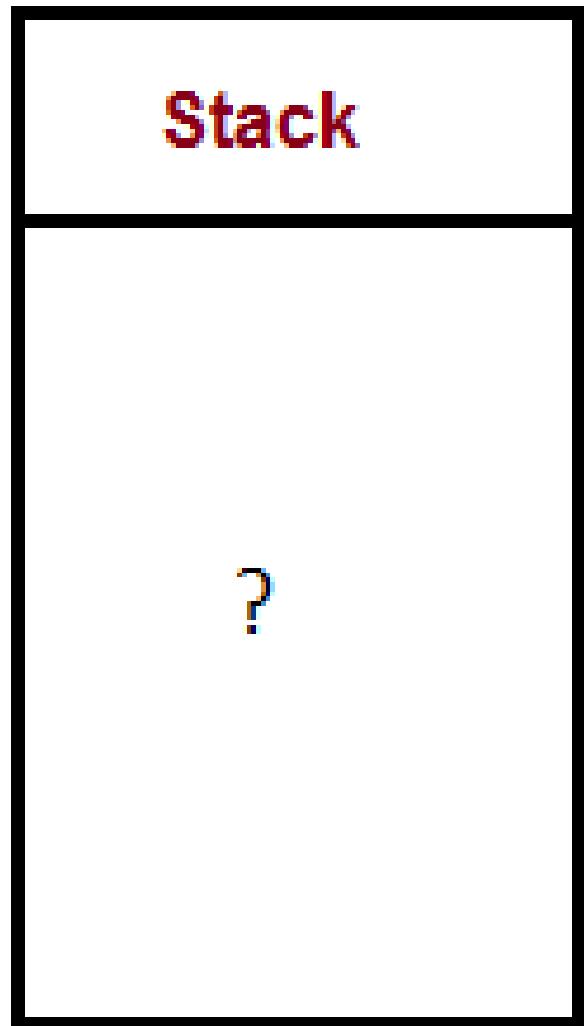


Figure 3.1 Process in memory.

Stack vs. Heap



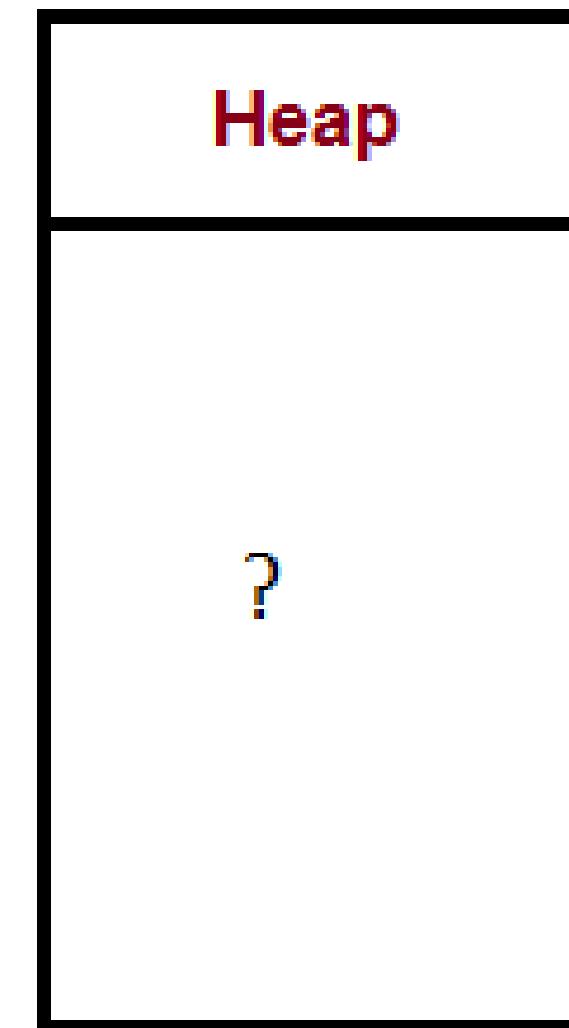
Stack vs. Heap



`int x=1`

`int y = 2`

`Form1 frm = new Form1()`



Stack or heap?

- X?
- Y?
- Frm-object?
- Pointer/reference to Frm-object?

Memzoom

<https://justine.lol/memzoom/index.html>

Linear Memory Order

Renders memory bytes using traditional left-to-right ordering.



The screenshot shows a memory dump from the /proc/32195/mem file. The data is displayed in a grid format with three columns: Address, Value, and ASCII. The memory is organized by address, starting at 10008002b400 and ending at 10008003ac00. The data is highly compressed and appears as a series of illegible characters due to the linear memory order rule. The bottom right corner of the interface displays the 'memzoom' logo and the number '16x'.

Address	Value	ASCII
10008002b400	0x00000000	00000000
10008002b401	0x00000000	00000000
10008002b402	0x00000000	00000000
10008002b403	0x00000000	00000000
10008002b404	0x00000000	00000000
10008002b405	0x00000000	00000000
10008002b406	0x00000000	00000000
10008002b407	0x00000000	00000000
10008002b408	0x00000000	00000000
10008002b409	0x00000000	00000000
10008002b40a	0x00000000	00000000
10008002b40b	0x00000000	00000000
10008002b40c	0x00000000	00000000
10008002b40d	0x00000000	00000000
10008002b40e	0x00000000	00000000
10008002b40f	0x00000000	00000000
10008002b410	0x00000000	00000000
10008002b411	0x00000000	00000000
10008002b412	0x00000000	00000000
10008002b413	0x00000000	00000000
10008002b414	0x00000000	00000000
10008002b415	0x00000000	00000000
10008002b416	0x00000000	00000000
10008002b417	0x00000000	00000000
10008002b418	0x00000000	00000000
10008002b419	0x00000000	00000000
10008002b41a	0x00000000	00000000
10008002b41b	0x00000000	00000000
10008002b41c	0x00000000	00000000
10008002b41d	0x00000000	00000000
10008002b41e	0x00000000	00000000
10008002b41f	0x00000000	00000000
10008002b420	0x00000000	00000000
10008002b421	0x00000000	00000000
10008002b422	0x00000000	00000000
10008002b423	0x00000000	00000000
10008002b424	0x00000000	00000000
10008002b425	0x00000000	00000000
10008002b426	0x00000000	00000000
10008002b427	0x00000000	00000000
10008002b428	0x00000000	00000000
10008002b429	0x00000000	00000000
10008002b42a	0x00000000	00000000
10008002b42b	0x00000000	00000000
10008002b42c	0x00000000	00000000
10008002b42d	0x00000000	00000000
10008002b42e	0x00000000	00000000
10008002b42f	0x00000000	00000000
10008002b430	0x00000000	00000000
10008002b431	0x00000000	00000000
10008002b432	0x00000000	00000000
10008002b433	0x00000000	00000000
10008002b434	0x00000000	00000000
10008002b435	0x00000000	00000000
10008002b436	0x00000000	00000000
10008002b437	0x00000000	00000000
10008002b438	0x00000000	00000000
10008002b439	0x00000000	00000000
10008002b43a	0x00000000	00000000
10008002b43b	0x00000000	00000000
10008002b43c	0x00000000	00000000
10008002b43d	0x00000000	00000000
10008002b43e	0x00000000	00000000
10008002b43f	0x00000000	00000000
10008002b440	0x00000000	00000000
10008002b441	0x00000000	00000000
10008002b442	0x00000000	00000000
10008002b443	0x00000000	00000000
10008002b444	0x00000000	00000000
10008002b445	0x00000000	00000000
10008002b446	0x00000000	00000000
10008002b447	0x00000000	00000000
10008002b448	0x00000000	00000000
10008002b449	0x00000000	00000000
10008002b44a	0x00000000	00000000
10008002b44b	0x00000000	00000000
10008002b44c	0x00000000	00000000
10008002b44d	0x00000000	00000000
10008002b44e	0x00000000	00000000
10008002b44f	0x00000000	00000000
10008002b450	0x00000000	00000000
10008002b451	0x00000000	00000000
10008002b452	0x00000000	00000000
10008002b453	0x00000000	00000000
10008002b454	0x00000000	00000000
10008002b455	0x00000000	00000000
10008002b456	0x00000000	00000000
10008002b457	0x00000000	00000000
10008002b458	0x00000000	00000000
10008002b459	0x00000000	00000000
10008002b45a	0x00000000	00000000
10008002b45b	0x00000000	00000000
10008002b45c	0x00000000	00000000
10008002b45d	0x00000000	00000000
10008002b45e	0x00000000	00000000
10008002b45f	0x00000000	00000000
10008002b460	0x00000000	00000000
10008002b461	0x00000000	00000000
10008002b462	0x00000000	00000000
10008002b463	0x00000000	00000000
10008002b464	0x00000000	00000000
10008002b465	0x00000000	00000000
10008002b466	0x00000000	00000000
10008002b467	0x00000000	00000000
10008002b468	0x00000000	00000000
10008002b469	0x00000000	00000000
10008002b46a	0x00000000	00000000
10008002b46b	0x00000000	00000000
10008002b46c	0x00000000	00000000
10008002b46d	0x00000000	00000000
10008002b46e	0x00000000	00000000
10008002b46f	0x00000000	00000000
10008002b470	0x00000000	00000000
10008002b471	0x00000000	00000000
10008002b472	0x00000000	00000000
10008002b473	0x00000000	00000000
10008002b474	0x00000000	00000000
10008002b475	0x00000000	00000000
10008002b476	0x00000000	00000000
10008002b477	0x00000000	00000000
10008002b478	0x00000000	00000000
10008002b479	0x00000000	00000000
10008002b47a	0x00000000	00000000
10008002b47b	0x00000000	00000000
10008002b47c	0x00000000	00000000
10008002b47d	0x00000000	00000000
10008002b47e	0x00000000	00000000
10008002b47f	0x00000000	00000000
10008002b480	0x00000000	00000000
10008002b481	0x00000000	00000000
10008002b482	0x00000000	00000000
10008002b483	0x00000000	00000000
10008002b484	0x00000000	00000000
10008002b485	0x00000000	00000000
10008002b486	0x00000000	00000000
10008002b487	0x00000000	00000000
10008002b488	0x00000000	00000000
10008002b489	0x00000000	00000000
10008002b48a	0x00000000	00000000
10008002b48b	0x00000000	00000000
10008002b48c	0x00000000	00000000
10008002b48d	0x00000000	00000000
10008002b48e	0x00000000	00000000
10008002b48f	0x00000000	00000000
10008002b490	0x00000000	00000000
10008002b491	0x00000000	00000000
10008002b492	0x00000000	00000000
10008002b493	0x00000000	00000000
10008002b494	0x00000000	00000000
10008002b495	0x	

“Low” level programming

Introduction to C

- Pretty old language (1972-1973)
- Was created to make UNIX !
- Interesting in this module because it is low-level
- Needs a Compiler (gcc for linux)
- Benefits from a make/builder (gmake)



Structure of C Program

<i>Header</i>	#include <stdio.h>
<i>main()</i>	int main()
<i>Variable declaration</i>	{ int a = 10;
<i>Body</i>	printf("%d ", a);
<i>Return</i>	}

DG

Introduction to C

To Build a c program from source you can either use:

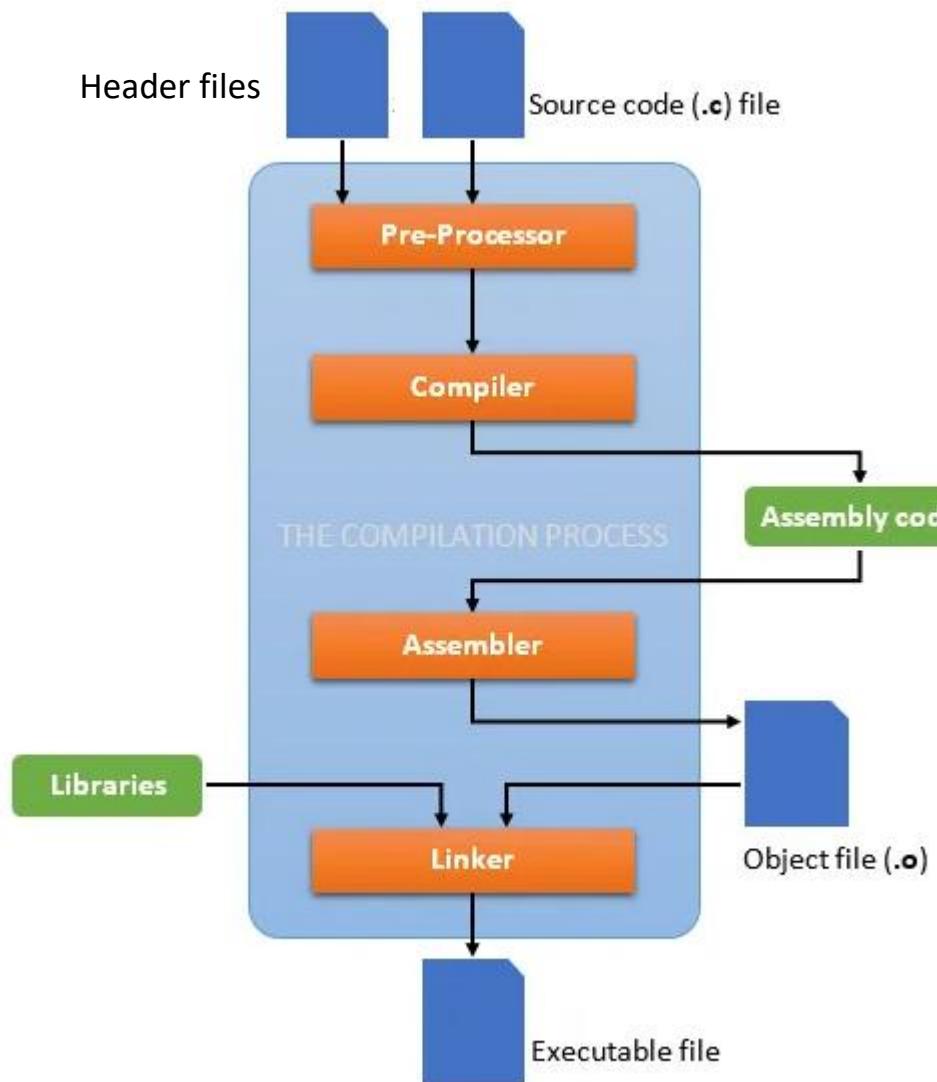
gcc hello.c -o hello

Or

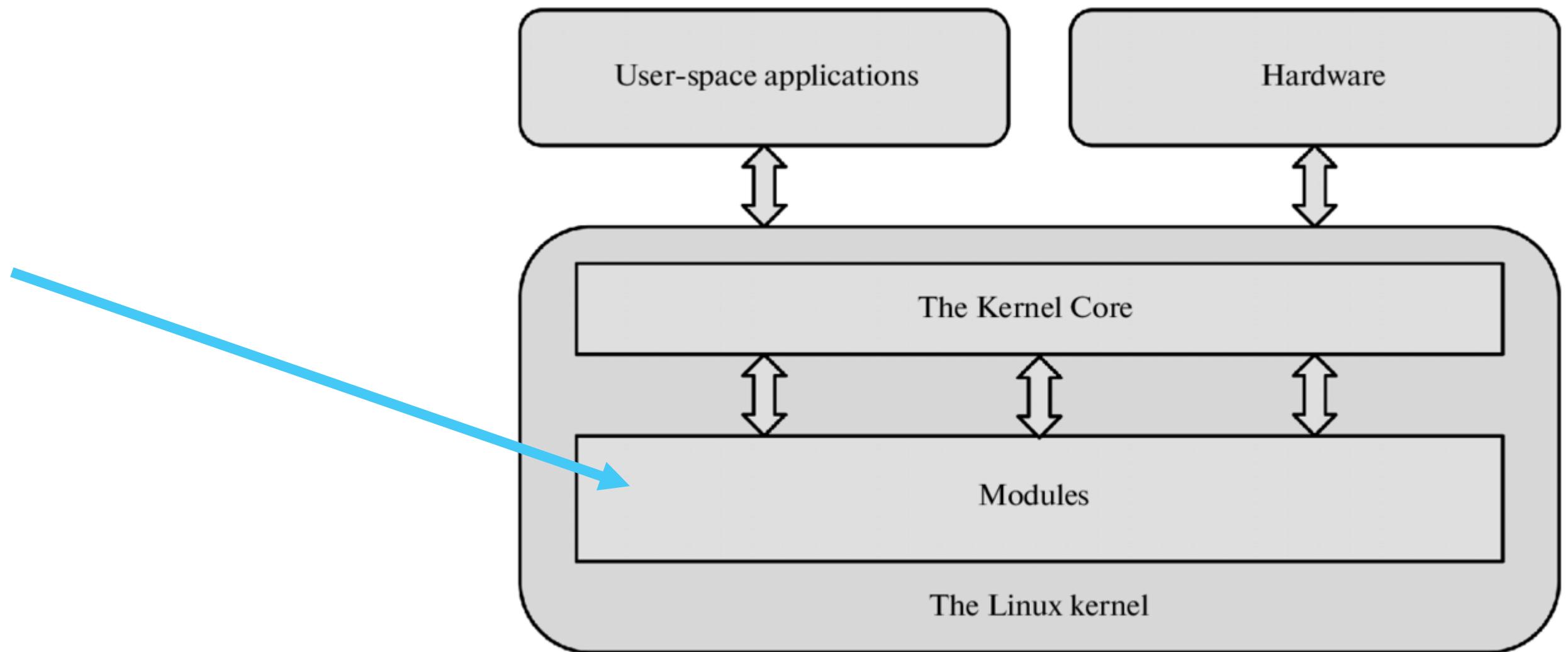
make hello

This will generate an executable.

Makefiles exist to make the life of a C programmer easier.



Kernel module



C can do low level things if run in kernel mode ! (i.e. wreck your system)

Pass by value vs. Pass by reference intermezzo

Nice visual representation: <https://www.mathwarehouse.com/programming/passing-by-value-vs-by-reference-visual-explanation.php>

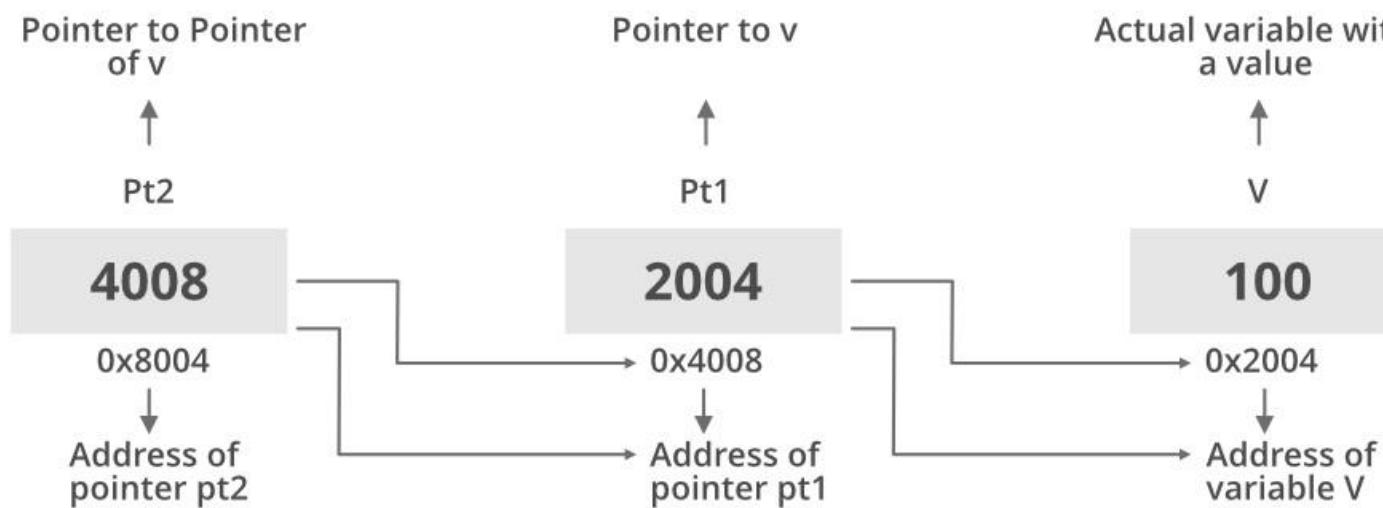
Authentic definition:

- When a parameter is **passed by reference**, the caller and the called **use the same variable** for the parameter. If the called modifies the parameter variable, the effect is visible to the caller's variable.
- When a parameter is **passed by value**, the caller and called have **two independent variables** with the same value. If the called modifies the parameter variable, the effect is not visible to the caller.

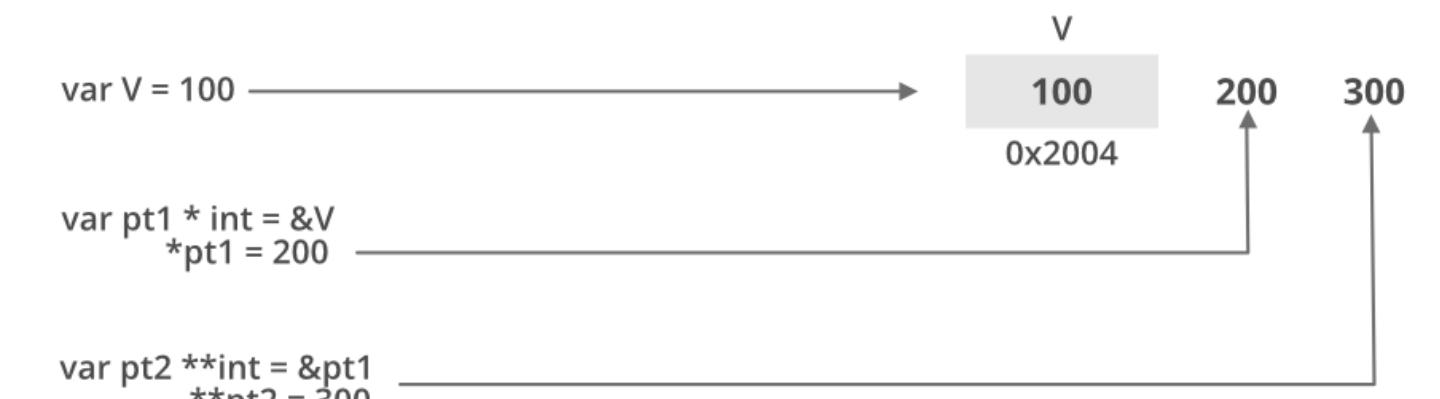
Pointers in low level programming languages

<https://www.geeksforgeeks.org/go-pointer-to-pointer-double-pointer/>

Pointer to Pointer



How Pointers works in Go



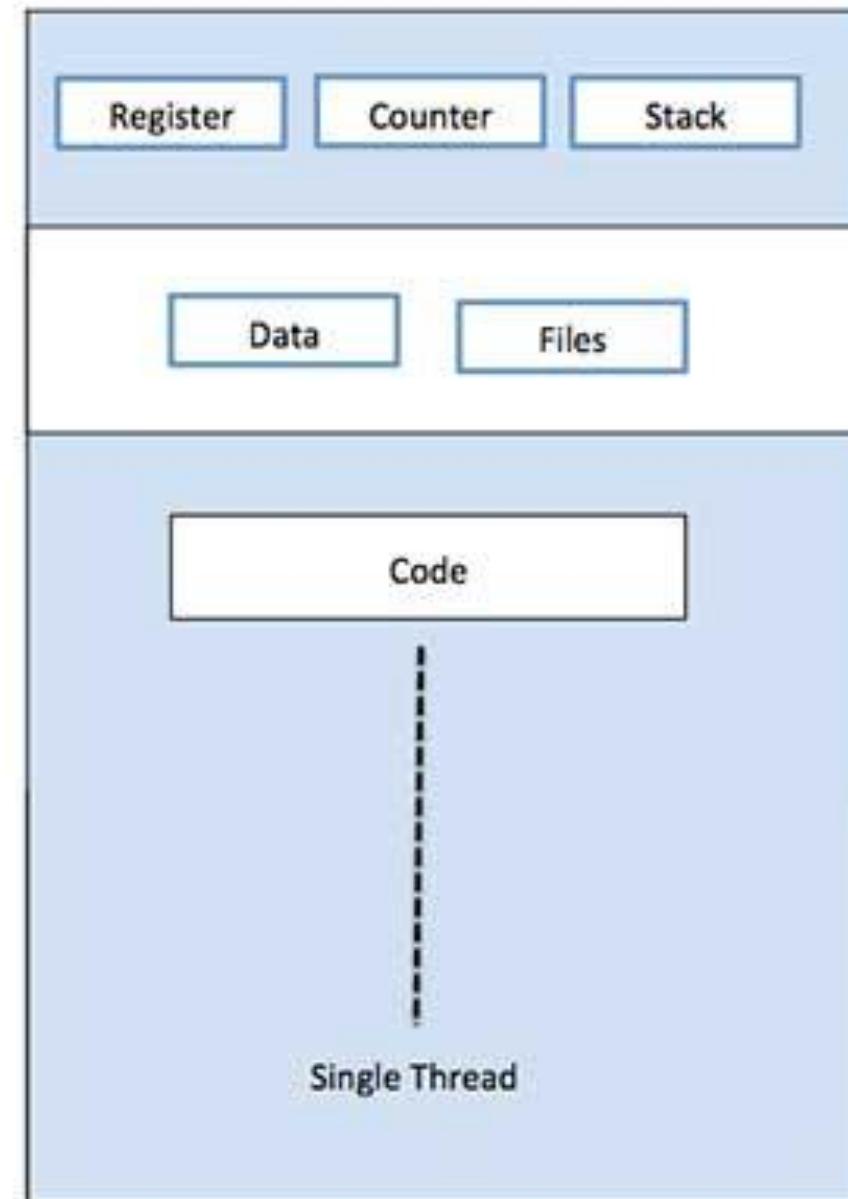
Threading

What is a “Thread”

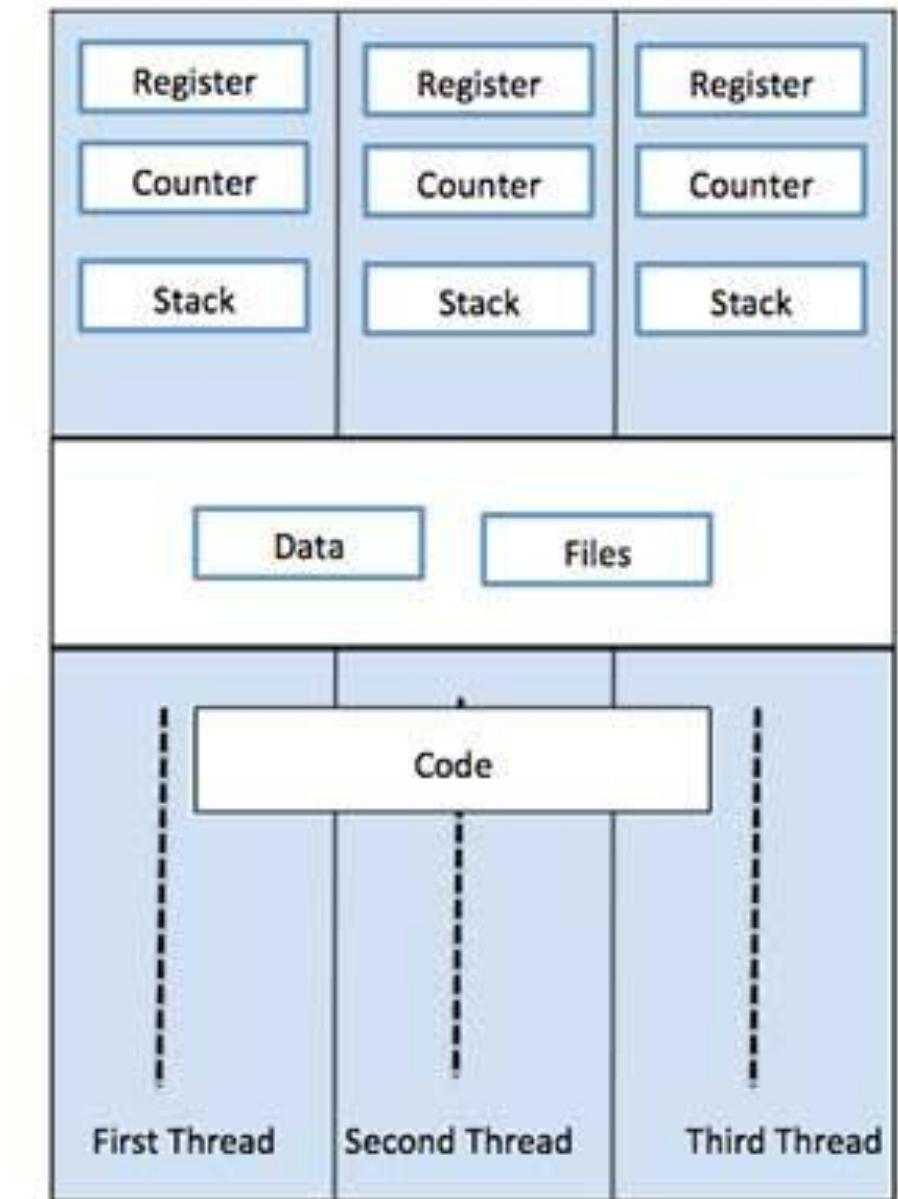
Often called a “**lightweight process**”

- Minimize **context switching** time
- A “blocking” thread does not block other threads

Question: We know that switching between processes requires interaction from the OS. Is this also the case for threading (thread switching)?



Single Process P with single thread



Single Process P with three threads

Img src: https://www.tutorialspoint.com/operating_system/os_multi_threading.htm

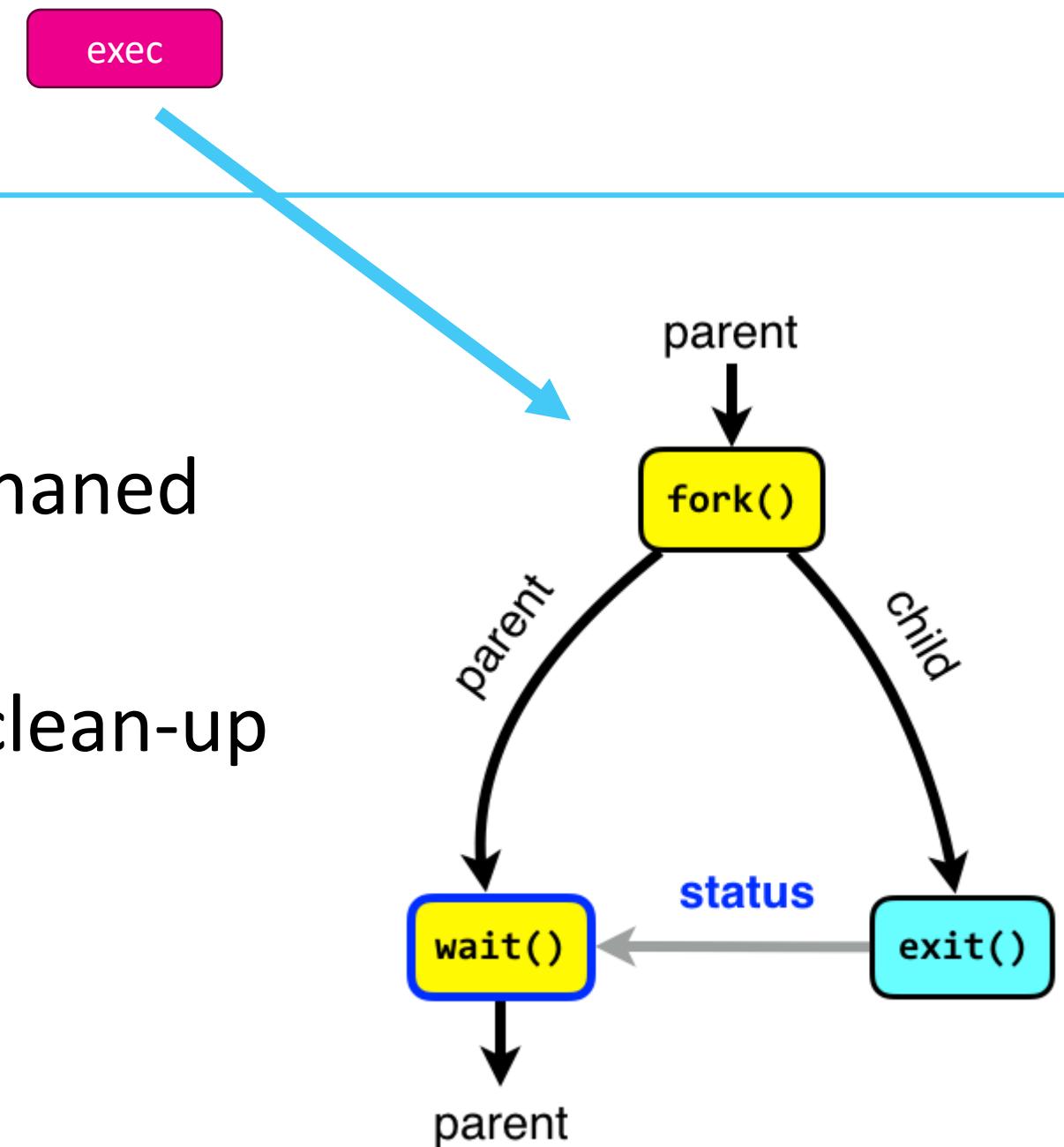
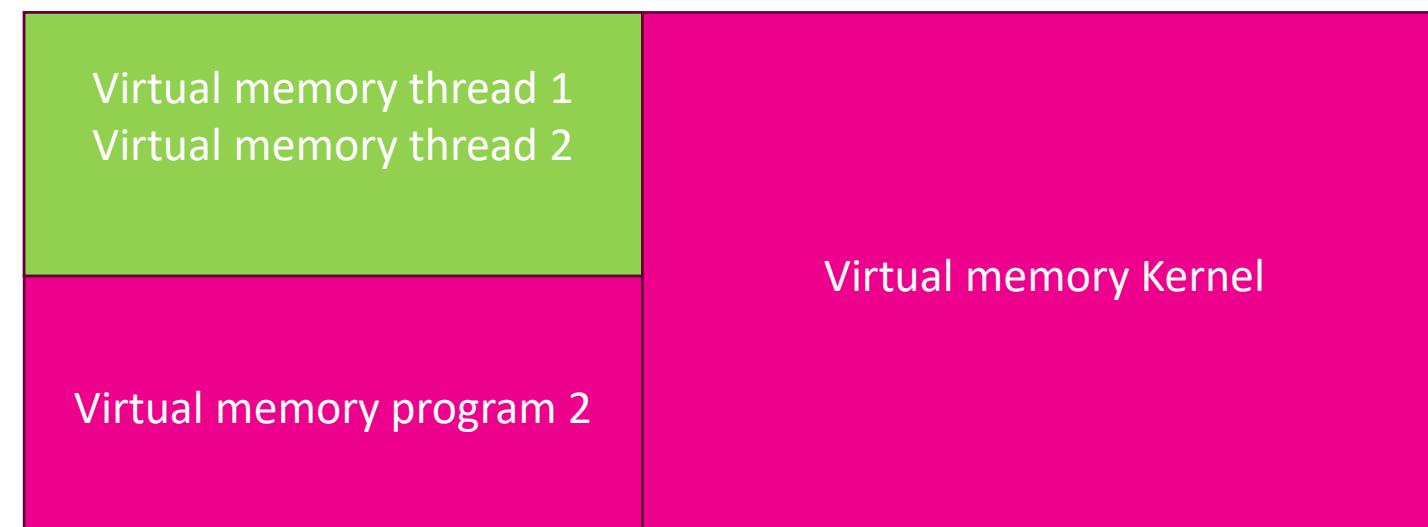
Threads vs Processes

- Processes & scheduling
 - Responsibility with the OS -> CPU scheduling
- Threads & scheduling
 - Responsibility with the application / program designer

```
0110011001111000101101100011000110110  
0110001101100110001101100110001101101  
1001100011011011001100011011011001100  
0110011001111000101101100011000110110  
011000First, solve the problem.101101  
011001100Then, write the code.00110110  
01100011011001John Johnson10001101101  
1001100011011011001100011011011001100  
0110011001111000101101100011000110110  
0110001101100110001101100110001101101
```

Threads vs forks (in c)

- Fork -> System call that creates a new child **process**
 - If parent ends before child, the child becomes orphaned
- Pthread_create -> All “part of” the same process
 - There is no exit as virtual memory does not need clean-up



Fork example

How many times will “hello” be printed?

	File: hello.c
1	#include <stdio.h>
2	#include <sys/types.h>
3	int main()
4	{
5	fork();
6	fork();
7	fork();
8	printf("hello\n");
9	return 0;
10	}

```
fork(); // Line 1
fork(); // Line 2
fork(); // Line 3
          L1      // There will be 1 child process
          / \      // created by line 1.
          L2      L2      // There will be 2 child processes
          / \      / \      // created by line 2
          L3      L3      L3      L3 // There will be 4 child processes
                                // created by line 3
```

Child processes vs Parent processes

Why is this important?

- What happens with the parent process if a child process gets killed?
- **What happens with the child process if a parent process gets killed?**
 - → Actually OS dependend

Multiple options are possible:

- Child process gets killed as soon as parent gets killed (behind the scenes “kill” signals are send to the child process)
- Child becomes an “orphan”
- Child gets to live on without any issues
- Etc.

Attention: Sometimes it gets tricky!

Example 1:

→ Typically (on Unix/Linux)

if a parent dies before the child process. The child becomes an orphan and get's a new parent process (init/systemd (= one of the first processes that starts with the OS)).

Example 2:

→ If the parent process receives proper exit signals – and it is able to send signals to its children – child behaviour depends on what the programmer implemented.

→ Example: SSH into a linux machine, issue a ping and in another session kill the SSH connection. Is your ping still going?

Thread example

	File: printthread.c
1	#include <stdio.h>
2	#include <stdlib.h>
3	#include <unistd.h> //Header file for sleep(). man 3 sleep for details.
4	#include <pthread.h>
5	
6	// A normal C function that is executed as a thread
7	// when its name is specified in pthread_create()
8	void *myThreadFun(void *vargp)
9	{
10	sleep(1);
11	printf("Printing GeeksQuiz from Thread \n");
12	sleep(200);
13	return NULL;
14	}
15	
16	int main()
17	{
18	pthread_t thread_id;
19	printf("Before Thread\n");
20	pthread_create(&thread_id, NULL, myThreadFun, NULL);
21	pthread_join(thread_id, NULL);
22	printf("After Thread\n");
23	936 943 1 943 debian 20 0 8836 5576 3820 S 0.0 0.1 0:00.14 -bash
24	exit(0);
25	943 1379 2 1379 debian 20 0 10648 540 464 S 0.0 0.0 0:00.00 ./printthread
	943 1379 2 1380 debian 20 0 10648 540 464 S 0.0 0.0 0:00.00 printthread

```
debian@debian:~/threads$ gcc printthread.c -lpthread -o printthread
```

Java example

```
File: HelloWorld.java

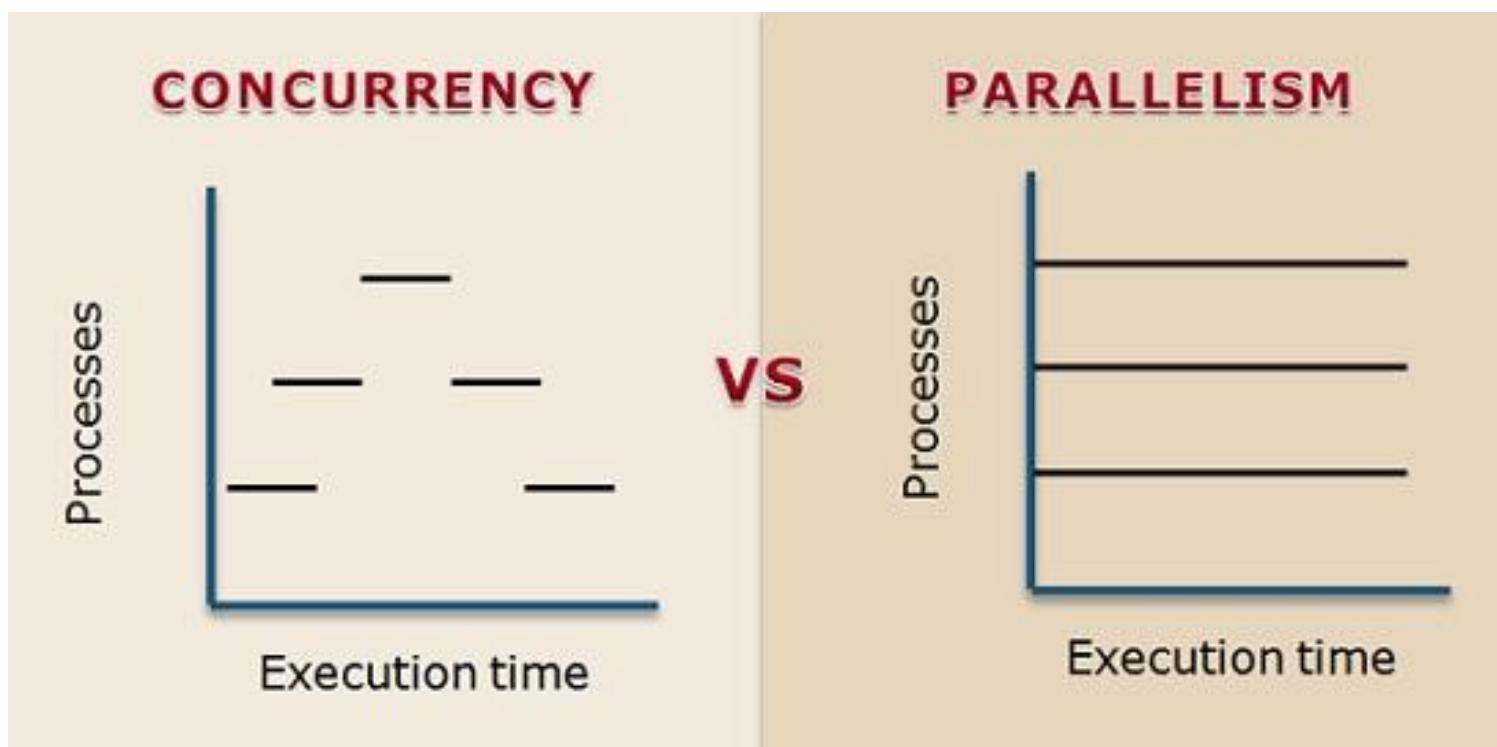
1 import java.util.*;
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         System.out.println("Hello World");
6         Scanner sc= new Scanner(System.in);
7         System.out.print("Enter a string: ");
8         String str= sc.nextLine();
9     }
10 }
```

How many threads ?

936	943	1	943	debian	20	0	8820	5800	3884	S	0.0	0.1	0:00.18	-bash
943	1486	12	1486	debian	20	0	2957M	40664	25396	S	0.7	1.0	0:00.13	└── java HelloWorld
943	1486	12	1487	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.04	└── java
943	1486	12	1488	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── VM Thread
943	1486	12	1489	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── Reference Handler
943	1486	12	1490	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── Finalizer
943	1486	12	1491	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── Signal Dispatch
943	1486	12	1492	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── Service Thread
943	1486	12	1493	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.01	└── C2 CompilerThre
943	1486	12	1494	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.02	└── C1 CompilerThre
943	1486	12	1495	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── Sweeper thread
943	1486	12	1496	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.03	└── VM Periodic Tas
943	1486	12	1497	debian	20	0	2957M	40664	25396	S	0.0	1.0	0:00.00	└── Common-Cleaner

Example : where do we use threads?

- “Everywhere”?!
- “We don’t want clients to wait!”
 - Parallelism vs concurrency?
 - Race conditions



Process vs. Thread

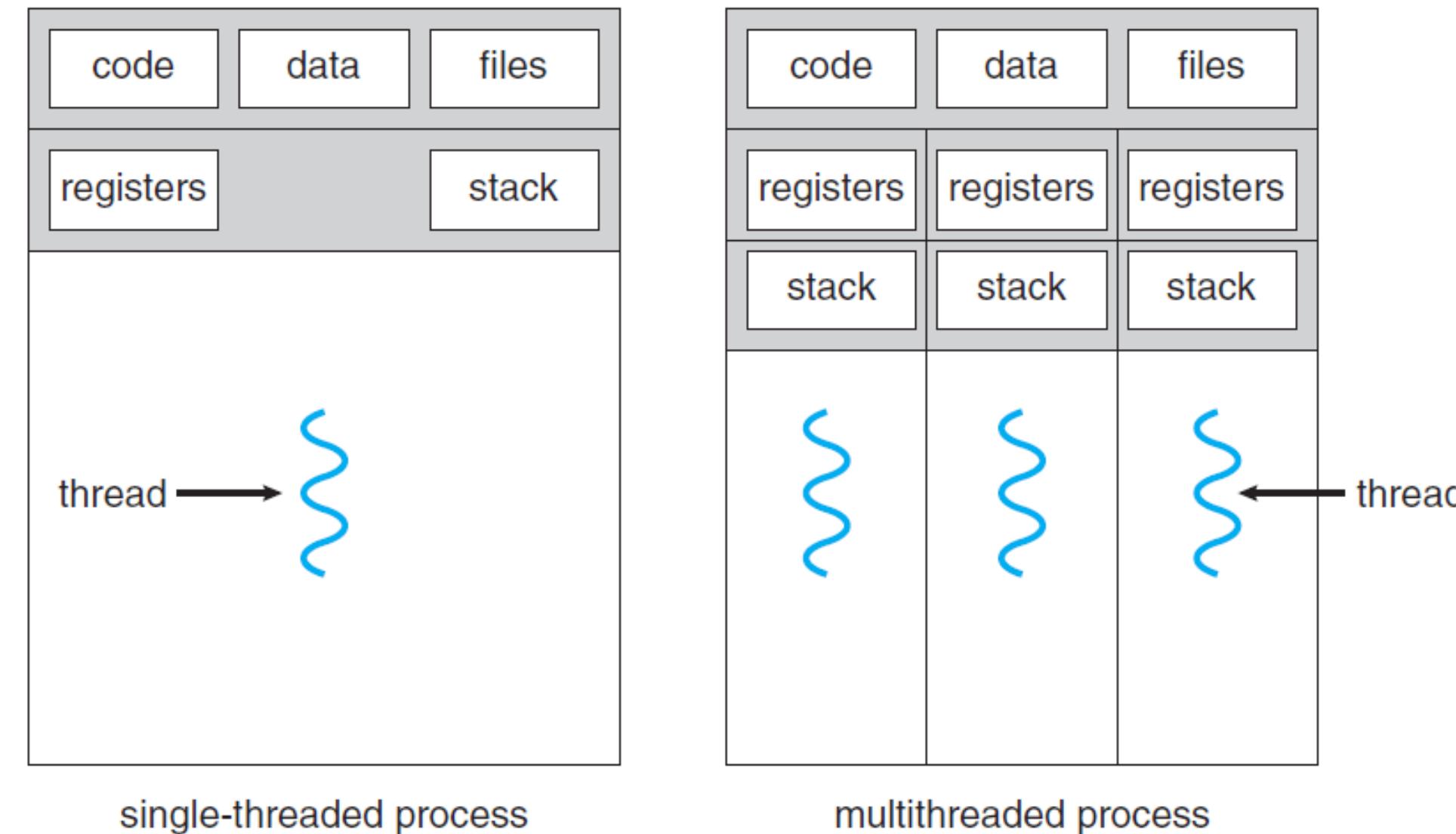


Figure 4.1 Single-threaded and multithreaded processes.

Kernel vs user-level threads

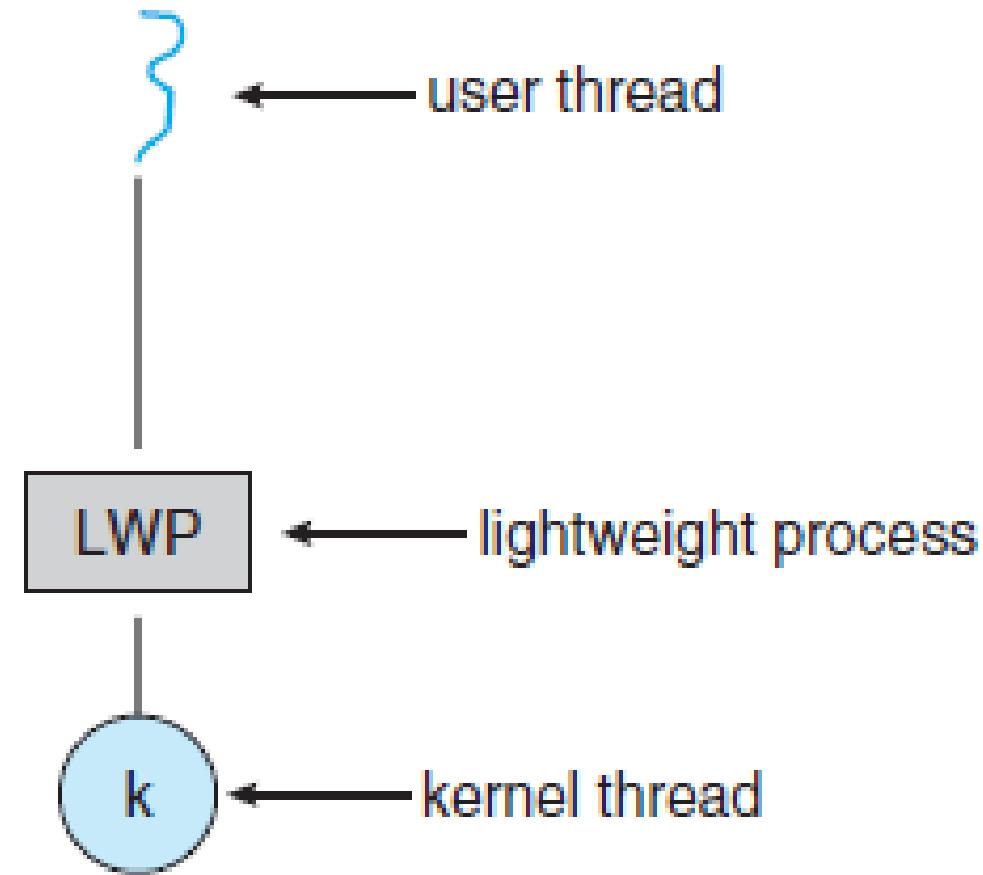


Figure 4.13 Lightweight process (LWP).

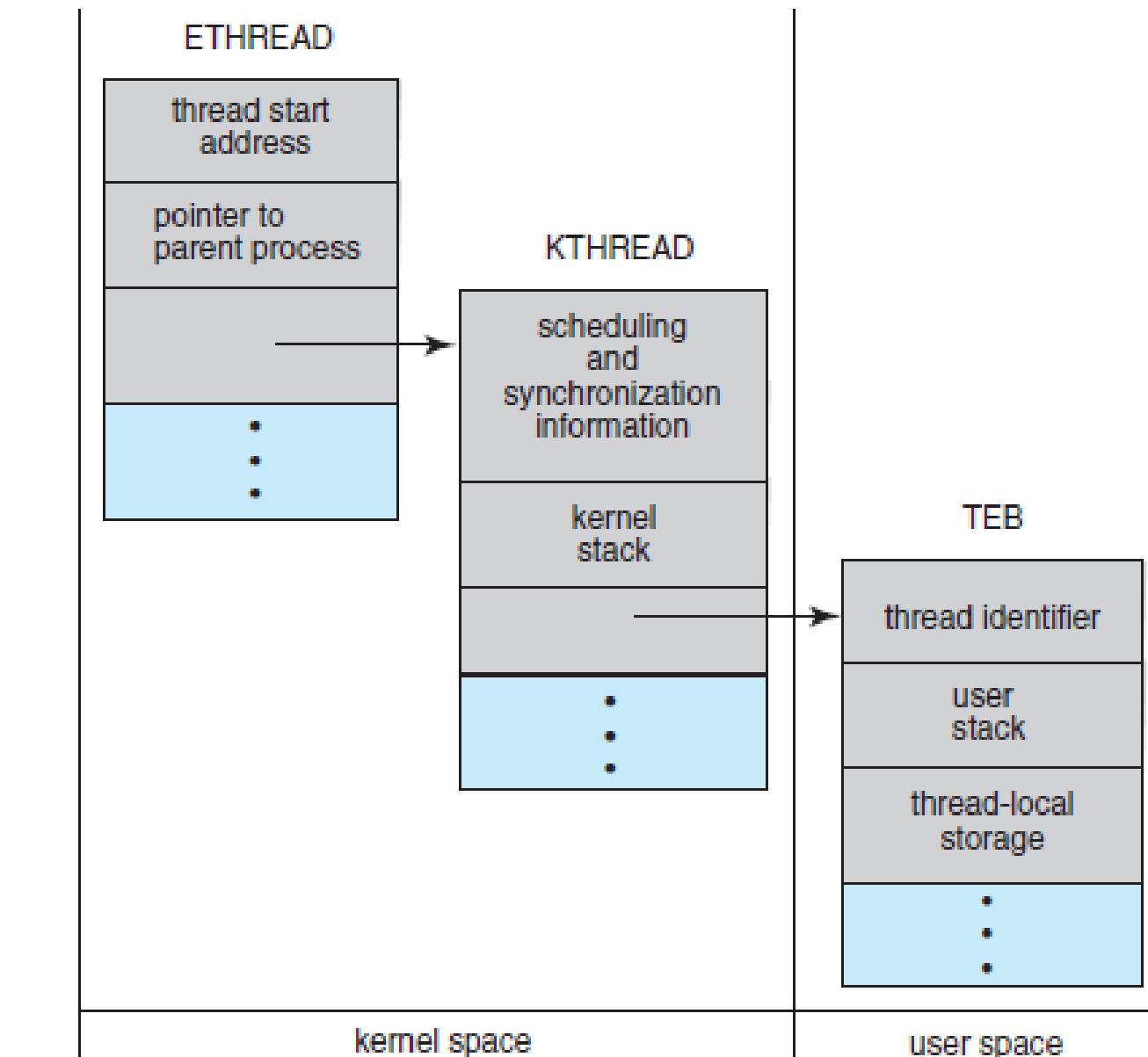
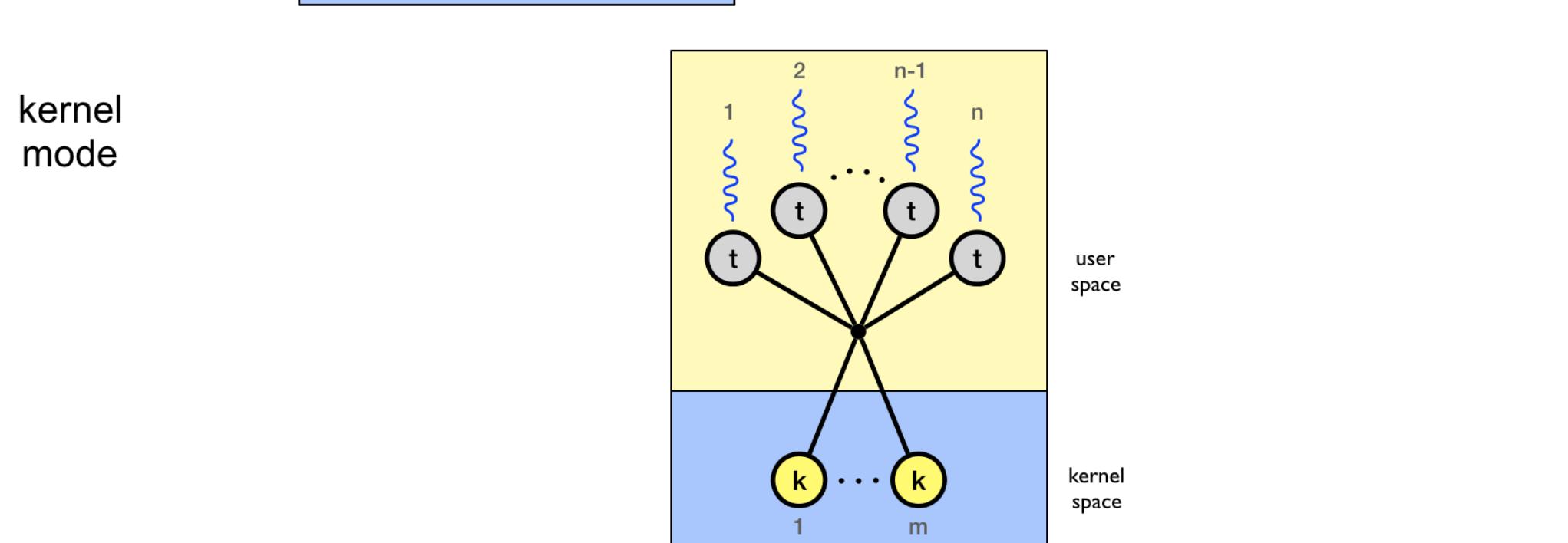
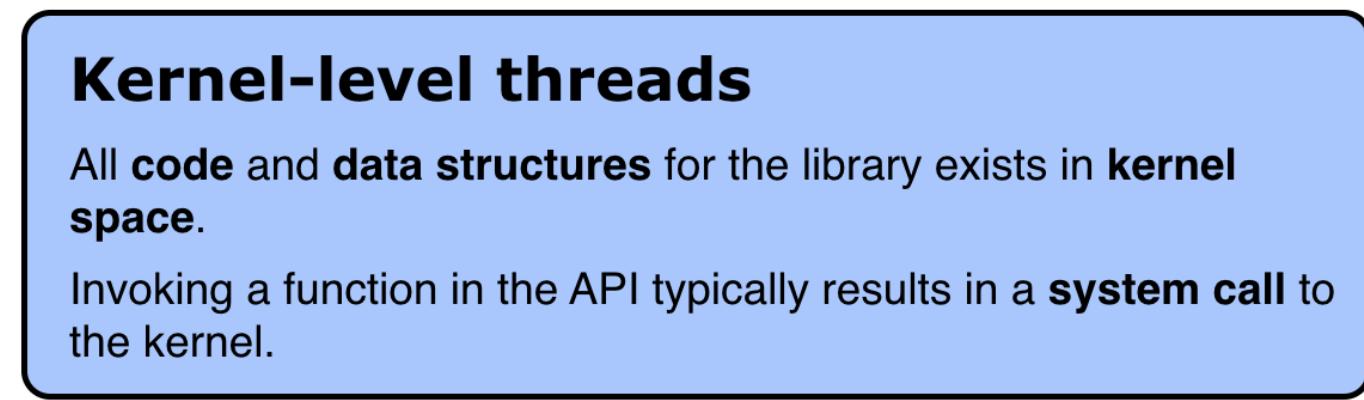
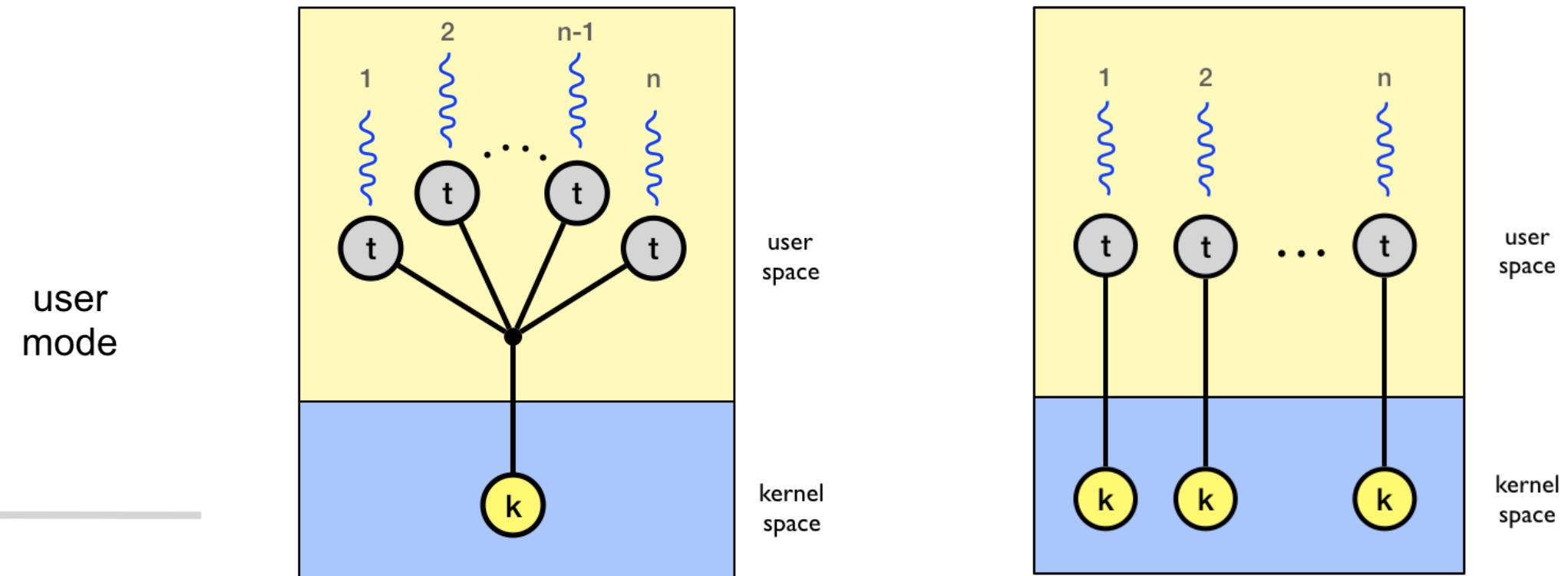
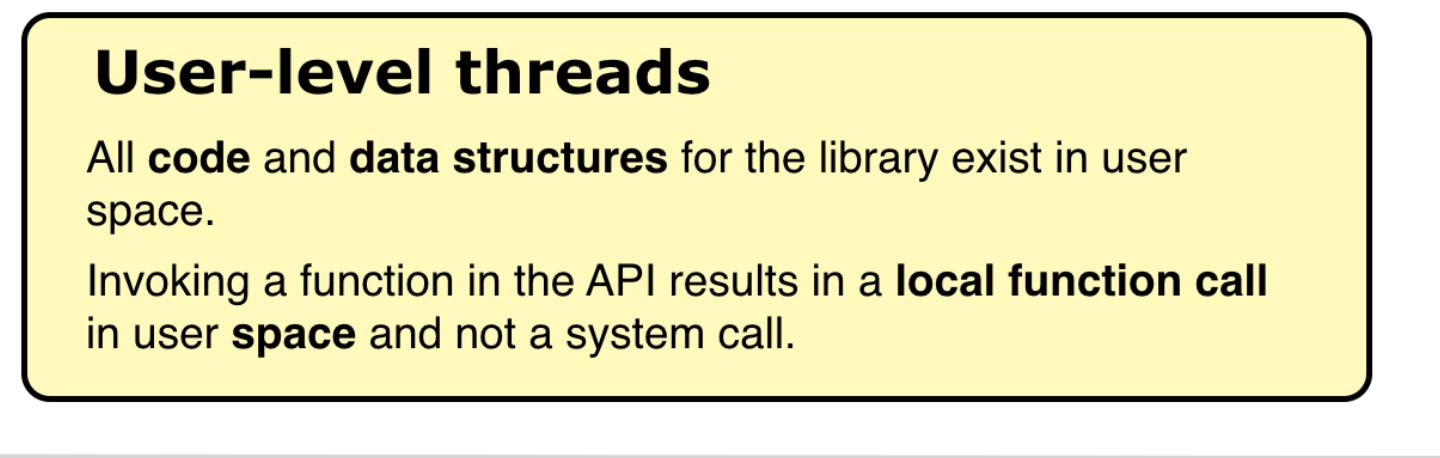


Figure 4.14 Data structures of a Windows thread.

Kernel vs user-level threads & multithreading models



Kernel vs user-level threads

The slide has a dark background with yellow and white text. It features two main sections: 'Threads' on the left and 'The benefits of multithreaded programming' on the right.

Threads

- A thread is a basic unit of CPU utilization.
- It comprises:
 - A thread ID
 - A program counter
 - A register set
 - and
 - A stack
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
- A traditional / heavyweight process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time.

The benefits of multithreaded programming can be broken down into four major categories:

- Responsiveness**: Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
- Resource sharing**: By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
- Economy**: Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
- Utilization of multiprocessor architectures**: The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases concurrency.

Src: <https://www.youtube.com/watch?v=LOfGJcVnvAk>



On Linux

Some interesting things:

- htop
 - nlwp via f2 setup, add tgid
- ps -e -T
- /proc/<nr>/status
- pstree -p

Main	I/O	TGID	PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
		1	1	root	20	0	163M	5624	3604	S	0.0	0.0	0:06.43	/sbin/init
		730	730	lightdm	20	0	19268	368	368	S	0.0	0.0	0:00.18	/lib/systemd/systemd --user
		731	731	lightdm	20	0	164M	128	0	S	0.0	0.0	0:00.00	(sd-pam)
		781	781	lightdm	20	0	9128	0	0	S	0.0	0.0	0:00.01	/usr/bin/dbus-daemon --session --addr
		773	773	lightdm	9	-11	313M	112	0	S	0.0	0.0	0:00.31	/usr/bin/pulseaudio --daemonize=no --
		773	783	lightdm	-6	0	313M	112	0	S	0.0	0.0	0:00.00	/usr/bin/pulseaudio --daemonize=no
		787	787	lightdm	20	0	303M	4	4	S	0.0	0.0	0:00.00	/usr/libexec/at-spi-bus-launcher
		793	793	lightdm	20	0	8996	0	0	S	0.0	0.0	0:00.00	/usr/bin/dbus-daemon --config-file
		787	788	lightdm	20	0	303M	4	4	S	0.0	0.0	0:00.00	/usr/libexec/at-spi-bus-launcher
		787	789	lightdm	20	0	303M	4	4	S	0.0	0.0	0:00.00	/usr/libexec/at-spi-bus-launcher
		787	791	lightdm	20	0	303M	4	4	S	0.0	0.0	0:00.00	/usr/libexec/at-spi-bus-launcher
		821	821	lightdm	20	0	160M	0	0	S	0.0	0.0	0:00.00	/usr/libexec/at-spi2-registrayd --use

Process vs. Thread (generally speaking)

- A **thread** is a basic unit of CPU utilization; each thread has a:
 - Thread ID
 - Program Counter
 - Register set
 - Stack
- It **shares** with other threads (belonging to the **same process**):
 - Code section
 - Data section
 - Other OS-resources such as “Open-files”

Why would we want to use threads? - Motivation

- Imagine a browser tab “being stuck” while downloading a file?
 - Other examples?
- **Advantages of multithreaded programming**
 - Responsiveness
 - Resource sharing
 - Economy
(it is less costly to manage threads than processes)
 - Scalability

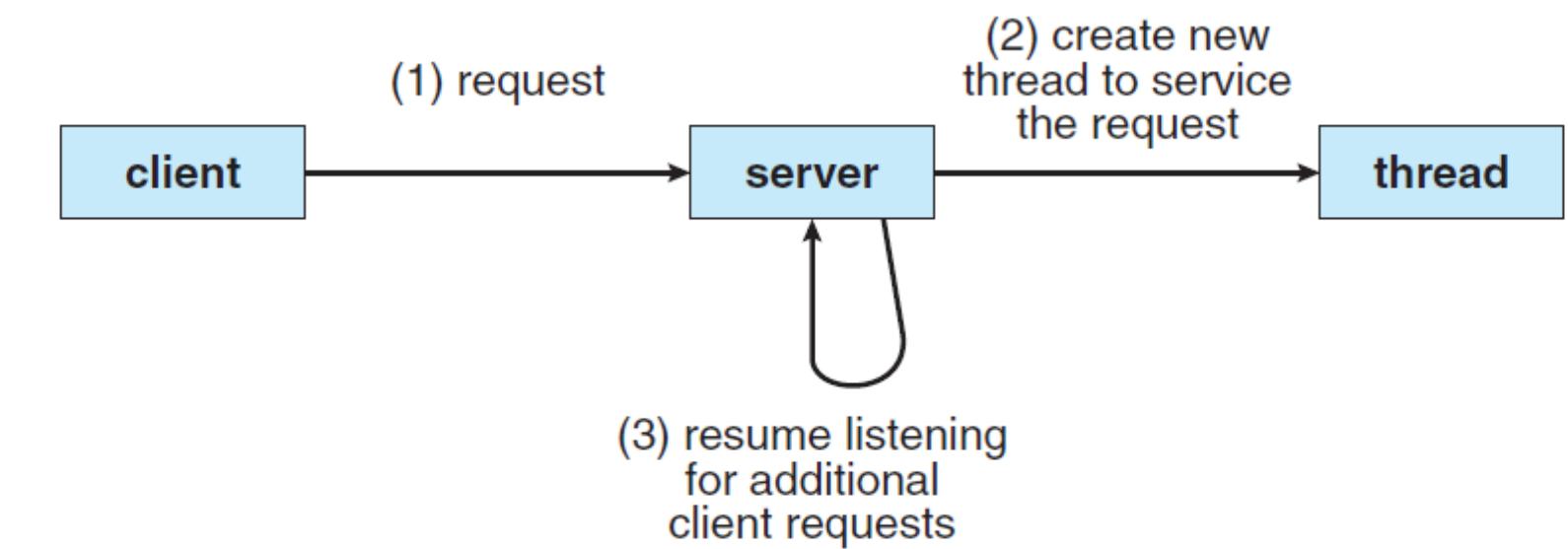
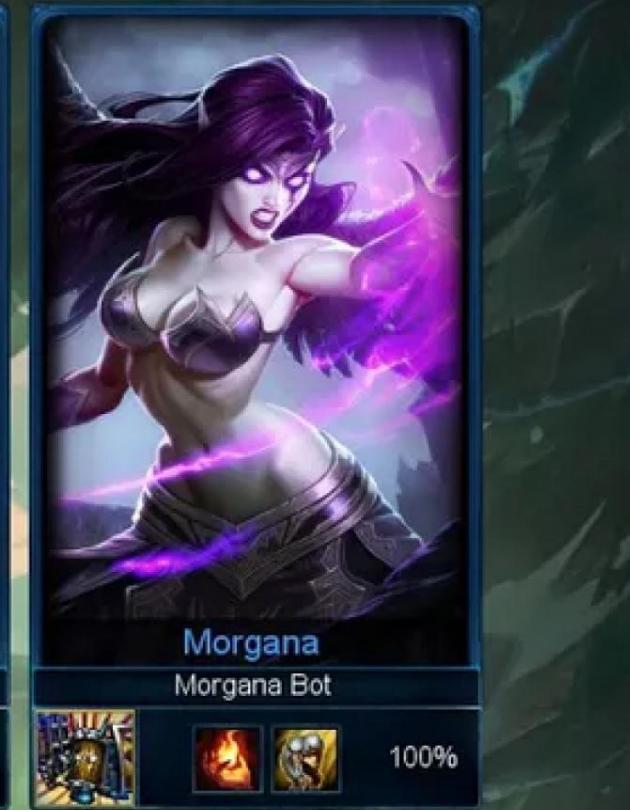


Figure 4.2 Multithreaded server architecture.



Ideal:
1 Server
10 Clients



Without threads

- leagueserver.exe
- league_accept_client.exe (for every client)
- leagueclient.exe

With threads

- leagueserver.exe
- leagueclient.exe

Server creates a thread for each client

Disadvantages?

- **Programming** (harder to write, harder to debug, harder to manage)
- **Concurrency issues with shared resources**
 - Similar problems with transactions in databases!
 - Dirty read
 - Loss update
 - Phantom read
- It is never set in stone that a “Java thread” is always mapped to an “OS thread”.
 - You have to trust the libraries/frameworks or write everything yourself if you want that.
 - Luckily we rarely have to worry about that as a programmer ☺
 - -> User and kernel threads

Disadvantages? – Race Conditions

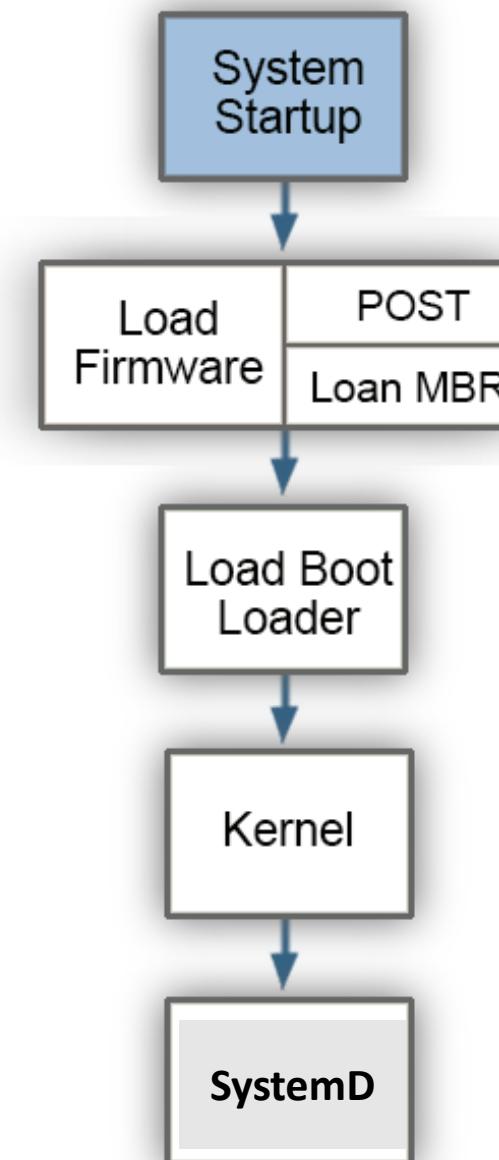
Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value		←	0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

The boot process

Boot Process Overview

- Four main stages
- Some stages can be modified by administrators



Firmware Stage

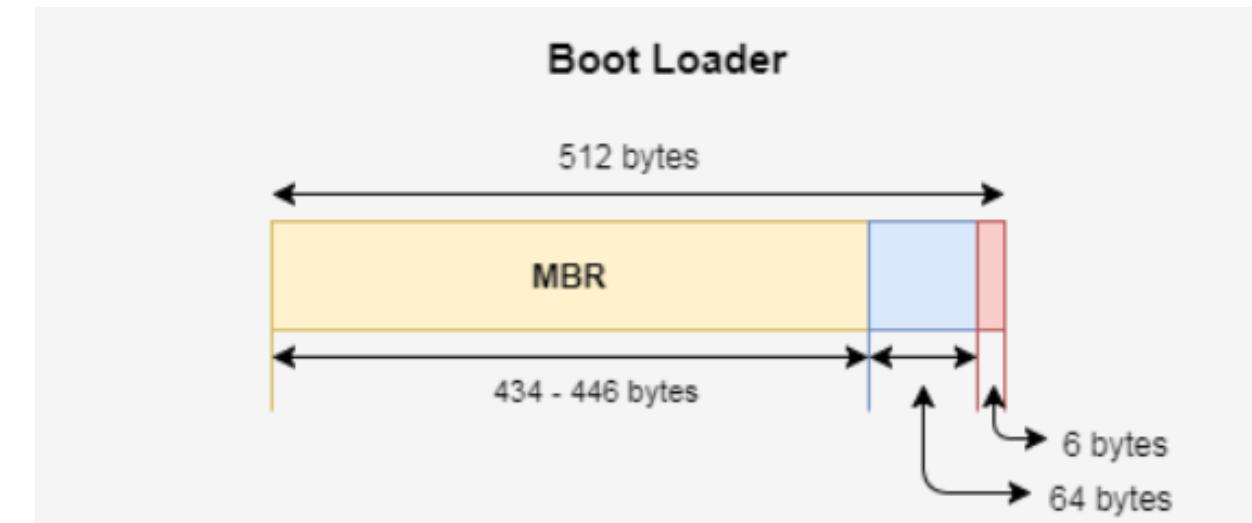
- Firmware is referred to as the BIOS (Basic Input Output System)
- UEFI (Unified Extensible Firmware Interface) has replaced the BIOS on most systems but typically still referred to as “BIOS”

Firmware Stage

- Two primary jobs in this stage:
 - 1) Power-On Self Test (POST) – ensures system hardware (CPU, RAM, peripherals, etc.) is functioning properly
 - 2) Load Master Boot Record (MBR) – contains drive partition table and loads *first stage bootloader* whose purpose is to load the *second stage bootloader* (next stage)

First Stage Boot Loader

BIOS looks for Master Boot Record (MBR) on first found hdd or equivalent – finds partition table and loads *first stage bootloader*



- *Max 446 bytes!*
 - *64 bytes Partition Table, 6 bytes crc*
 - *That bootloader will load the second stage bootloader, who lies elsewhere*
- * (exceptionally this First stage bootloader loads kernel directly)

Bootloaders

- Most common bootloaders
 - LILO - Linux Loader
 - *Supports systems with BIOS
 - *ELILO – Efi Linux Loader supports systems with UEFI
 - GRUB - Grand Unified Bootloader
 - *Supports systems with UEFI
 - *Supports Kernel Flavor choice

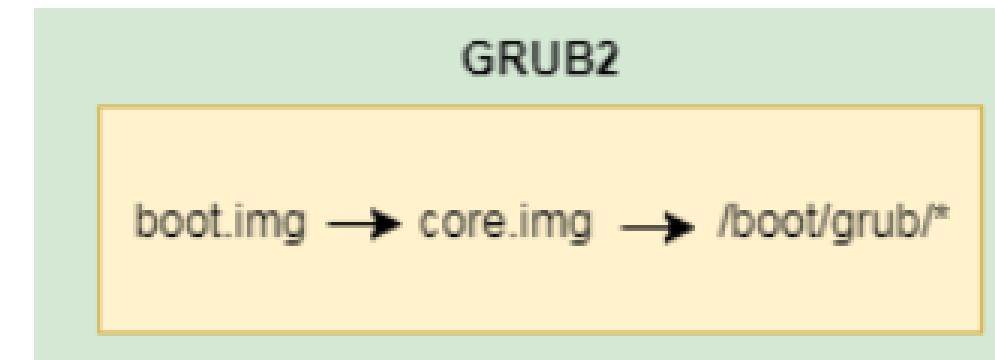
Bootloaders

- Other bootloaders
 - SILO – SPARC Improved bootLoader
 - *Supports Linux on Sun SPARC hardware
 - YABOOT – Yet Another Bootloader
 - Supports PowerPC hardware
- Network booting
 - PXE – Preboot Execution Environment
 - *For hardware that supports TFTP used to download the bootloader from a server

Bootloaders

- Windows
 - BOOTMGR (can be chainloaded from GRUB)
- MacOS
 - IBOOT for INTEL
 - LLB for ARM (Low-Level Bootloader)
 - Can be used alongside GRUB

GRUB2



- Boot.img (1st stage bootloader)
 - <512 bytes on MBR
 - Points to location of core.img
- Core.img (2nd stage | 1,5th stage bootloader)
 - Chosen Kernel image + generic modules to
 - Access files in /boot/grub (the REAL 2nd stage ?)
 - Loads other (kernel) modules at runtime

Kernel phase

- GRUB2 loads the chosen kernel in memory and passes control to it using the files in
- the /boot directory :
 - An initial RAM disk image (initrd / initramfs)
 - temporary / as tmpfs ←!!!
 - the kernel files (vmlinuz) with basic devices
 - Kernel start up : SystemD becomes active

SystemD phase

- Set kernel options using /etc/sysctl.conf
- Starts udevd daemon to detect all (other) devices
- Imports network configuration
- file system check (fsck) the root file system if necessary

SystemD phase

- Decrypts filesystems if encrypted.
- Mount filesystems according to /etc/fstab
- the REAL (root) / is mounted
- Enable swap devices
- Eventually
 - The system boots in a specific target mode
 - SystemD is managed by systemctl

dmesg Command

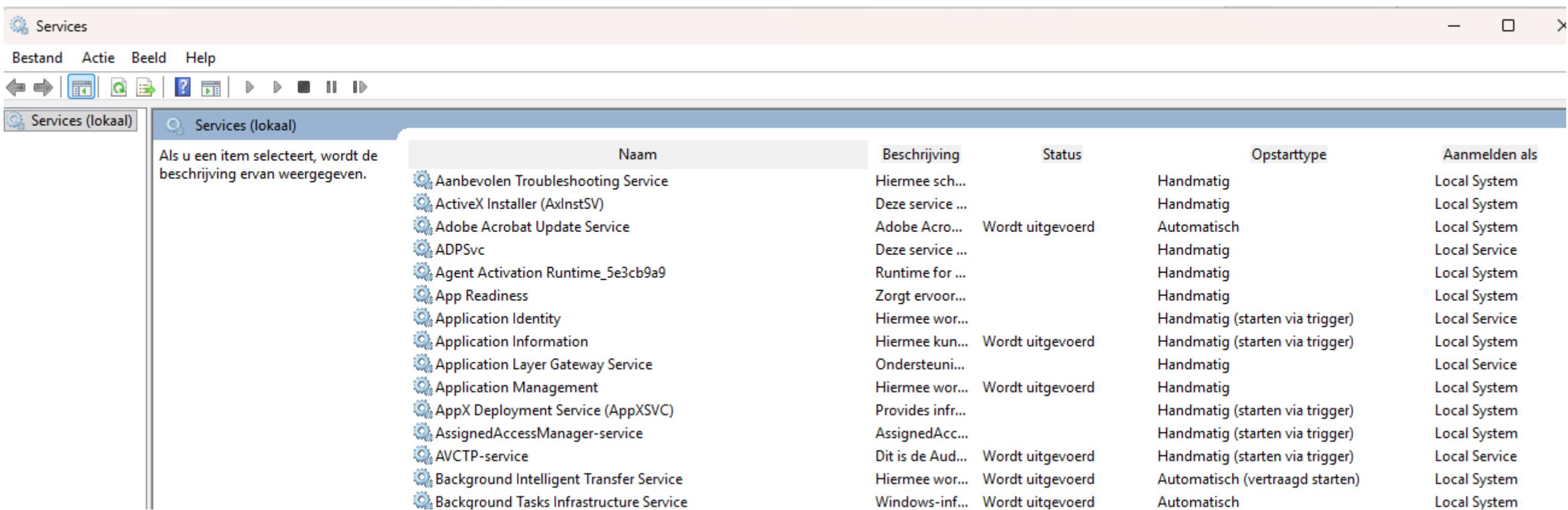
- Executed after boot to view messages generated by the kernel during the boot process
 - Useful for troubleshooting boot issues
- Also executed upon connecting a new device to see device pathname

Daemons

- The administrator can control which services will be provided by the various *daemons*
- A daemon is a running program that provides a service

No Daemons in Windows

- A daemon in windows is called a service



The screenshot shows the Windows Services console window titled "Services". The menu bar includes "Bestand", "Actie", "Beeld", and "Help". Below the menu is a toolbar with icons for back, forward, search, and other operations. The main area is titled "Services (lokaal)" and contains a table of service details. The columns are: Naam (Name), Beschrijving (Description), Status (Status), Opstarttype (Startup type), and Aanmelden als (Log on as). The table lists numerous services, many of which are currently running ("Wordt uitgevoerd").

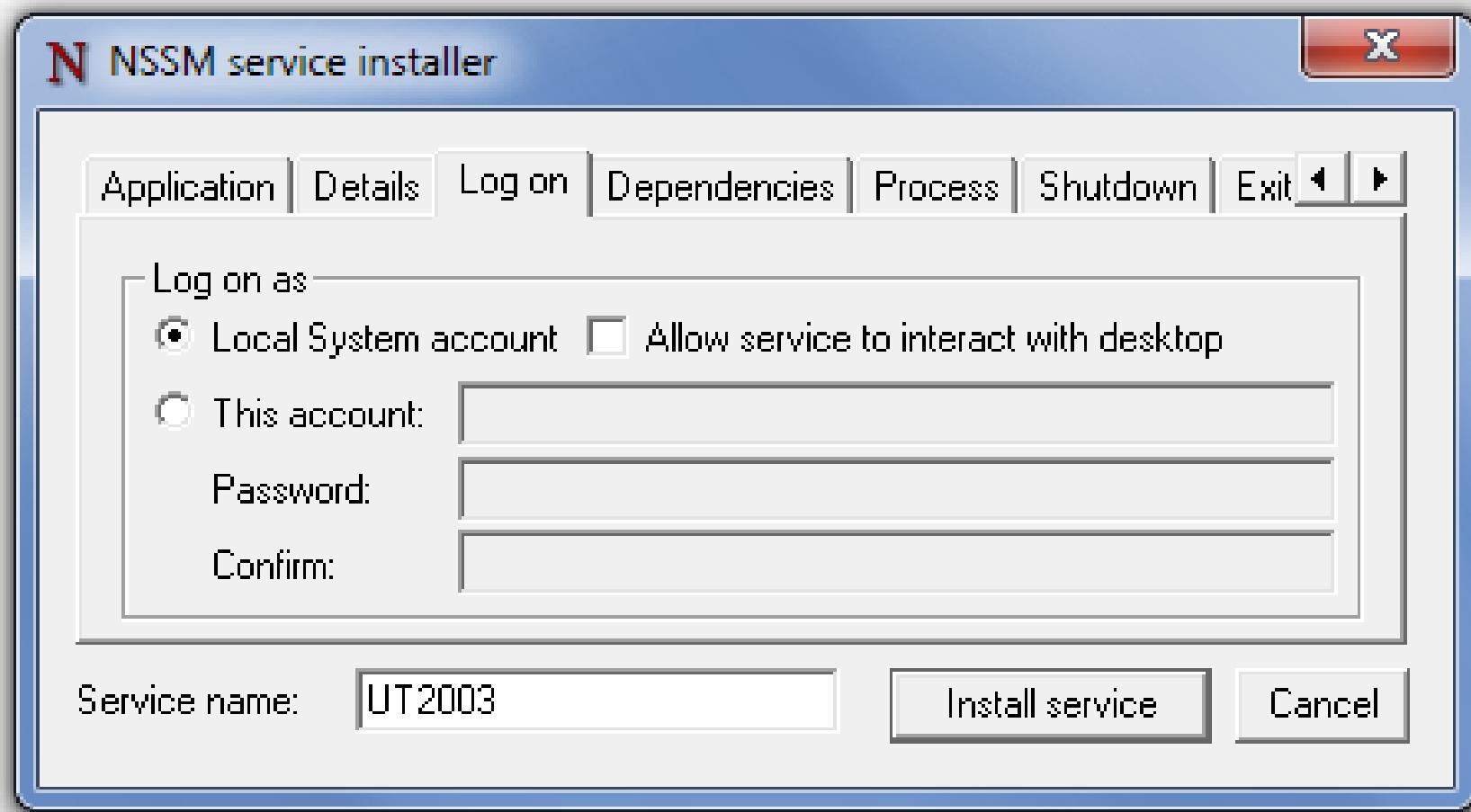
Naam	Beschrijving	Status	Opstarttype	Aanmelden als
Aanbevolen Troubleshooting Service	Hiermee sch...	Handmatig	Local System	
ActiveX Installer (AxInstSV)	Deze service ...	Handmatig	Local System	
Adobe Acrobat Update Service	Adobe Acro... Wordt uitgevoerd	Automatisch	Local System	
ADPSvc	Deze service ...	Handmatig	Local Service	
Agent Activation Runtime_5e3cb9a9	Runtime for ...	Handmatig	Local System	
App Readiness	Zorgt ervoor...	Handmatig	Local System	
Application Identity	Hiermee wor...	Handmatig (starten via trigger)	Local Service	
Application Information	Hiermee kun... Wordt uitgevoerd	Handmatig (starten via trigger)	Local System	
Application Layer Gateway Service	Ondersteuni...	Handmatig	Local Service	
Application Management	Hiermee wor... Wordt uitgevoerd	Handmatig	Local System	
AppX Deployment Service (AppXSVC)	Provides infr...	Handmatig (starten via trigger)	Local System	
AssignedAccessManager-service	AssignedAcc...	Handmatig (starten via trigger)	Local System	
AVCTP-service	Dit is de Aud... Wordt uitgevoerd	Handmatig (starten via trigger)	Local Service	
Background Intelligent Transfer Service	Hiermee wor... Wordt uitgevoerd	Automatisch (vertraagd starten)	Local System	
Background Tasks Infrastructure Service	Windows-inf... Wordt uitgevoerd	Automatisch	Local System	

No Daemons in Windows

- Limited wrapping of executables into services is possible
- Need Admin powers
- E.g.
 - `sc create ServiceName binPath= "C:\Path\To\YourExecutable.exe"`
 - `sc start ServiceName`
 - `New-Service -Name "ServiceName" -BinaryPathName "C:\Path\To\YourExecutable.exe" - DisplayName "My Service" -StartupType Automatic`
 - `Start-Service -Name "ServiceName"`

No Daemons in Windows

- NSSM (third party open source tool)



Systemctl controls the daemons

- The systemctl command is used in systems that have Systemd
- Services are found in
 - /etc/systemd/system
 - They are formal configuration files
- To start a service:
`#systemctl start httpd.service`
- To stop a service:
`#systemctl stop httpd.service`

The systemctl command

- To start check the state of a service:

```
#systemctl status httpd.service
```

- To view all running services:

```
#systemctl -a
```

- To configure a service to start automatically:

```
#systemctl enable httpd.service
```

Targets

A Target is a state where a consistent number of services are running

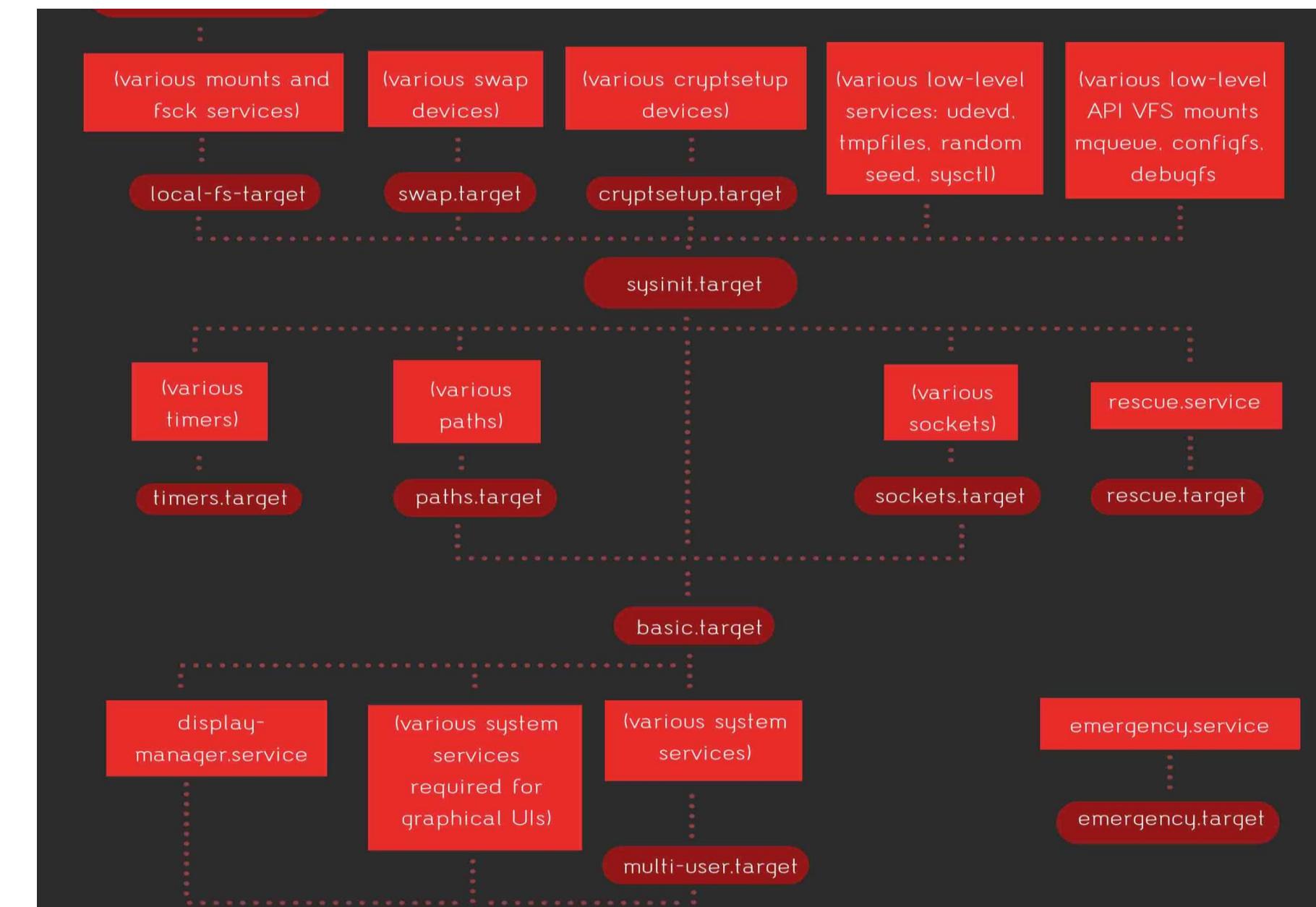
A target has

Required dependencies

Parallel dependencies

Conflicting dependencies

Sequential dependencies (Before / After)



The systemctl command

- To change the current target state

```
#systemctl isolate graphical-target
```

- To change the default target state

```
#systemctl set-default multi-user.target
```

SYSTEMD

In Linux, SYSTEMD is in control

SystemD is in Control

It determines which daemons are running

Consult it using systemctl command (no args)

```
vaneekhoutguy@debian-vaneekhout-guy-prime: ~
run-user-1000.mount          loaded active mounted   /run/user/1000
sys-fs-fuse-connections.mount loaded active mounted   FUSE Control File System
sys-kernel-config.mount      loaded active mounted   Kernel Configuration File System
sys-kernel-debug.mount       loaded active mounted   Kernel Debug File System
sys-kernel-tracing.mount    loaded active mounted   Kernel Trace File System
systemd-ask-password-plymouth.path loaded active waiting  Forward Password Requests to Plymouth Directory Watch
systemd-ask-password-wall.path loaded active waiting  Forward Password Requests to Wall Directory Watch
init.scope                   loaded active running  System and Service Manager
session-2.scope              loaded active running  Session 2 of user vaneekhoutguy
session-5.scope              loaded active running  Session 5 of user vaneekhoutguy
apache2.service              loaded active running  The Apache HTTP Server
apparmor.service              loaded active exited   Load AppArmor profiles
atd.service                  loaded active running  Deferred execution scheduler
bluetooth.service            loaded active running  Bluetooth service
connman-wait-online.service  loaded active exited   Wait for network to be configured by ConnMan
connman.service              loaded active running  Connection service
console-setup.service         loaded active exited   Set console font and keymap
cron.service                 loaded active running  Regular background program processing daemon
dbus.service                 loaded active running  D-Bus System Message Bus
dnsmasq.service              loaded active running  dnsmasq - A lightweight DHCP and caching DNS server
dundee.service               loaded active running  DUN service
exim4.service                loaded active running  LSB: exim Mail Transport Agent
getty@tty1.service            loaded active running  Getty on tty1
ifup@ens33.service           loaded active exited   ifup for ens33
ifupdown-pre.service          loaded active exited   Helper to synchronize boot up for ifupdown
keyboard-setup.service        loaded active exited   Set the console keyboard layout
```

.service ?

.service

Configured in a .service file !

Constructed so SystemD knows what (you want it) to do

```
cli Selecteren vaneeckhoutguy@debian-vaneckhout-guy-prime: ~  
vaneckhoutguy@debian-vaneckhout-guy-prime:~$ cat apache.service  
[Unit]  
Description=Apache web server  
After=network.target  
Before=nextcloud-web.service  
[Service]  
ExecStart=/usr/local/apache2/bin/httpd -D FOREGROUND -k start  
ExecReload=/usr/local/apache2/bin/httpd -k graceful  
Type=notify  
Restart=always  
[Install]  
WantedBy=default.target  
RequiredBy=network.target  
  
vaneckhoutguy@debian-vaneckhout-guy-prime:~$
```

The Unit Section

DESCRIPTION :- Human-readable title of the systemd service.

AFTER :- Set dependency on a service. (I want the server to start after the network is online. This typically includes targets or other services.)

BEFORE :- Start current service before specified service. (I want Apache web server running before the service for Nextcloud is started).

```
cat Selecteren vaneekhoutguy@debian-vaneekhout-guy-prime: ~  
vaneekhoutguy@debian-vaneekhout-guy-prime:~$ cat apache.service  
[Unit]  
Description=Apache web server  
After=network.target  
Before=nextcloud-web.service  
[Service]  
ExecStart=/usr/local/apache2/bin/httpd -D FOREGROUND -k start  
ExecReload=/usr/local/apache2/bin/httpd -k graceful  
Type=notify  
Restart=always  
[Install]  
WantedBy=default.target  
RequiredBy=network.target  
vaneekhoutguy@debian-vaneekhout-guy-prime:~$
```

The Service Section

ExecStart:- The command that needs to be executed when the service starts

ExecReload:- (optional).

how a service is restarted. Use this field in case you wish to have a specific restart mechanism.

Type:- start-up type of a process for a given systemd service. Options are simple, exec, forking, oneshot, dbus, notify and idle.

Restart:- (optional)

specifies if/when a service should be restarted or not.

options are no, on-success, on-failure, on-abnormal, on-watchdog, on-abort and always.

The Install Section

is used when you run either systemctl enable and systemctl disable command for enabling or disabling a service.

WantedBy:

similar to the After and Before fields

used to specify systemd-equivalent "runlevels".

The default.target is when all the system initialization is complete (when the user is asked to log in. user-facing services (like Apache, cron, GNOME-stuff, etc.) use this target.)

RequiredBy:

similar to WantedBy

specifies hard dependencies.

(if a dependency, this service will fail).

Take control

Create/start/Stop/Manage your own service

Creating a root service

Start with a (root) script **update-on-boot.sh**

```
#!/usr/bin/env bash

if [ ${EUID} -ne 0 ]
then
    exit 1 # this is meant to be run as root
fi

apt-get update 1>/dev/null 2>>/root/sys-update.log
```

Put it in /root/ directory

Make sure root can execute this script

Test it !

./update-on-boot.sh

Turning your script into a service

cd to /etc/systemd/system

*create a file **update-on-boot.service***

[Unit]

Description=Keeping my sources minty fresh

After=multi-user.target

[Service]

ExecStart=/usr/bin/bash /root/update-on-boot.sh

Type=simple

[Install]

WantedBy=multi-user.target

Enabling the service

sudo systemctl daemon-reload

makes the systemD daemon aware of the new service

sudo systemctl enable update-on-boot.service

```
root@debian-vaneeckhout-guy-prime:/etc/systemd/system# sudo systemctl enable update-on-boot.service
Created symlink /etc/systemd/system/multi-user.target.wants/update-on-boot.service → /etc/systemd/system/update-on-boot.service.
root@debian-vaneeckhout-guy-prime:/etc/systemd/system#
```

This creates a symlink in the multi-user.target.wants (remember the targets ?)

How to check if it is enabled ?

```
root@debian-vaneeckhout-guy-prime:/etc/systemd/system# sudo systemctl is-enabled update-on-boot.service
enabled
```

So every time a system enters this target, the script will be executed !

Creating a USER script

Create a script keep-uptime.sh to keep the time your system is up

```
#!/usr/bin/env bash  
uptime | tee -a ${HOME}/uptime.log
```

Make it executable

Test it !

Creating a service for a regular user

Make another .service file ☺

cd to /etc/systemd/system ?

~/.config/systemd/user/ !

Directory not present by default

mkdir -p

cd to ~/.config/systemd/user/

Creating a service for a regular user

Create a file `keep-uptime.service` in `~/.config/systemd/user/`

Content :

[Unit]

Description=Log uptime in scoreboard

DefaultDependencies=no

Before=shutdown.target

[Service]

Type=oneshot

ExecStart=/usr/bin/bash %h/keep-uptime.sh

TimeoutStartSec=0

[Install]

WantedBy=shutdown.target

Enabling your user service

makes the systemD daemon aware of the new service

No sudo necessary !

systemctl --user daemon-reload

systemctl --user enable keep-upptime.service

```
vaneekhoutguy@debian-vaneekhout-guy-prime:~$ systemctl --user enable keep-upptime.service
Created symlink /home/vaneekhoutguy/.config/systemd/user/shutdown.target.wants/keep-upptime.service → /home/vaneekhoutguy/.config/systemd/user/keep-upptime.service.
vaneekhoutguy@debian-vaneekhout-guy-prime:~$
```

Start & Check your user service

Start it !

systemctl --user start keep-uptime

Check what happened (log)

systemctl --user status keep-uptime

```
vaneekhoutguy@debian-vaneekhout-guy-prime:~$ systemctl --user status keep-uptime
● keep-uptime.service - Log uptime in scoreboard
  Loaded: loaded (/home/vaneekhoutguy/.config/systemd/user/keep-uptime.service; enabled; vendor preset: enabled)
  Active: inactive (dead) since Wed 2022-11-16 13:41:28 CET; 4s ago
    Process: 3108 ExecStart=/usr/bin/bash /home/vaneekhoutguy/keep-uptime.sh (code=exited, status=0/SUCCESS)
   Main PID: 3108 (code=exited, status=0/SUCCESS)
     CPU: 9ms

Nov 16 13:41:28 debian-vaneekhout-guy-prime systemd[1206]: Starting Log uptime in scoreboard...
Nov 16 13:41:28 debian-vaneekhout-guy-prime bash[3110]: 13:41:28 up 3:18, 1 user, load average: 0.13, 0.15, 0.12
Nov 16 13:41:28 debian-vaneekhout-guy-prime systemd[1206]: keep-uptime.service: Succeeded.
Nov 16 13:41:28 debian-vaneekhout-guy-prime systemd[1206]: Finished Log uptime in scoreboard.
vaneekhoutguy@debian-vaneekhout-guy-prime:~$
```

Want more

Let's build a true server service

Chatbot 1.0

Dumb chatbot.php that answers anything you type in gibberish

```
<?php
$sock = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
socket_bind($sock, '0.0.0.0', 10000);
for (;;) {
    socket_recvfrom($sock, $message, 124, 0, $ip, $port);
    $reply = str_rot13($message);
    socket_sendto($sock, $reply, strlen($reply), 0, $ip, $port);
}
```

Run it !

Php chatbot.php

Test it in another terminal

nc -u 127.0.0.1 10000



Turning Chatbot 1.0 into a service

Let's create a chatbot.service file for our service

[Unit]

Description=Dumb chat service

After=network.target

StartLimitIntervalSec=0

[Service]

Type=simple

Restart=always



RestartSec=1



ExecStart=/usr/bin/env php %h/chatbot.php

[Install]

WantedBy=default.target

What steps do you need to get it running ?

Examine your service process

systemctl --user status chatbot

```
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ systemctl --user status chatbot
● chatbot.service - Dumb chat service
  Loaded: loaded (/home/vaneekhoutguy/.config/systemd/user/chatbot.service; enabled; vendor preset: enabled)
  Active: active (running) since Wed 2022-11-16 14:27:27 CET; 3s ago
    Main PID: 3494 (php)
       Tasks: 1 (limit: 2284)
      Memory: 5.2M
        CPU: 20ms
      CGroup: /user.slice/user-1000.slice/user@1000.service/app.slice/chatbot.service
              └─3494 php /home/vaneekhoutguy/chatbot.php
```

```
Nov 16 14:27:27 debian-vaneekhout-guy-prime systemd[1206]: Started Dumb chat service.
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ nc -u 127.0.0.1 10000
Hi Daemon
Jv Qnrzba
^C
```

Examine your service process

ps -ef | grep php

```
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep php
vaneeck+ 3494 1206 0 14:27 ? 00:00:00 php /home/vaneekhoutguy/chatbot.php
vaneeck+ 3505 3222 0 14:28 pts/1 00:00:00 grep php
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ systemctl --user stop chatbot
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep php
vaneeck+ 3508 3222 0 14:28 pts/1 00:00:00 grep php
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ systemctl --user start chatbot
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep php
vaneeck+ 3510 1206 0 14:28 ? 00:00:00 php /home/vaneekhoutguy/chatbot.php
vaneeck+ 3512 3222 0 14:28 pts/1 00:00:00 grep php
```

Examine your service process

Try and kill your server process !

Kill pid_of_chatbot

```
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep php
vaneek+  3494  1206  0 14:27 ?        00:00:00 php /home/vaneekhoutguy/chatbot.php
vaneek+  3505  3222  0 14:28 pts/1    00:00:00 grep php
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ systemctl --user stop chatbot
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep php
vaneek+  3508  3222  0 14:28 pts/1    00:00:00 grep php
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ systemctl --user start chatbot
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep php
vaneek+  3510  1206  0 14:28 ?        00:00:00 php /home/vaneekhoutguy/chatbot.php
vaneek+  3512  3222  0 14:28 pts/1    00:00:00 grep php
```



Examine your service process

Your chatbot gets respawned automatically !

```
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep chat
vaneek+ 3530 1206 0 14:30 ? 00:00:00 php /home/vaneekhoutguy/chatbot.php
vaneek+ 3532 3222 0 14:30 pts/1 00:00:00 grep chat
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ kill 3530
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ ps -ef | grep chat
vaneek+ 3533 1206 0 14:30 ? 00:00:00 php /home/vaneekhoutguy/chatbot.php
vaneek+ 3535 3222 0 14:30 pts/1 00:00:00 grep chat
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$ systemctl --user status chatbot
● chatbot.service - Dumb chat service
  Loaded: loaded (/home/vaneekhoutguy/.config/systemd/user/chatbot.service; enabled; vendor preset: enabled)
    Active: active (running) since Wed 2022-11-16 14:30:39 CET; 20s ago
      Main PID: 3533 (php)
        Tasks: 1 (limit: 2284)
       Memory: 5.2M
         CPU: 22ms
        CGroup: /user.slice/user-1000.slice/user@1000.service/app.slice/chatbot.service
                  └─3533 php /home/vaneekhoutguy/chatbot.php

Nov 16 14:30:39 debian-vaneekhout-guy-prime systemd[1206]: chatbot.service: Scheduled restart job, restart counter is at 1.
Nov 16 14:30:39 debian-vaneekhout-guy-prime systemd[1206]: Stopped Dumb chat service.
Nov 16 14:30:39 debian-vaneekhout-guy-prime systemd[1206]: Started Dumb chat service.
vaneekhoutguy@debian-vaneekhout-guy-prime:~/.config/systemd/user$
```