CRITICAL TECHWORKS (CTW)

QEMU and KVM

Assignements:

First steps

1st task

On the cover use a dive image. You go from Application level tweaks to kernel level tweaks

1. Install QEMU – DONE!
2. Mount a W10 x64 inside QEMU – DONE!
3. Don't forget to remove the iso file from the VM details (i);
4. Benchmark its performance;
5. When building binary, W10_x86-64_KVM.sh, try to understand what the commands do;
6. What sort of configurations QEMU have, and what do they do to the machine;

0.1 Field notes:

- To check if a machine can run KVM, and the output gives us the number of processors that can run KVM:
  egrep -c '(vmx—svm)' /proc/cpuinfo
  - 'cpuinfo' is a file that has the description of all flags and things that the processor can run.
    we can view its content by running the following command:
    cat /proc/cpuinfo/
  - '*vmx*' and '*svm*' are the flags that need to be present in the cpu to run hardware emulation and virualization, allowing us to install kvm.
- dont forget to adduser to kvm and libvirt (this one to allow creating vm) groups;
- bridge-utils command when installing allows us to create a bridge connection network, so this way we will be able to utilize for example a router to get an IP address and access it from there. Router (....1) -¿ host PC (....2) — virtual machine (....3);

- Host pc -≤ virtual machine -≤ host based networking ≥ - different from bridge networking =¿ NAT to have a local ip address it implies two networks on the host PC.
- Some Qemu commands to add to our qemu-system-x86_64:
  - *'–enable kvm'* – allows virtualization machines (kernel linux faster);
  - *'-m 4G or 4096'* – for ram;
  - *'-smp 4'* – for allocating numbre of cores (cpu threads);
  - *'-name name'* – to give a name to the virtual machine, 'name';
  - *'-boot d'* – boot system from iso (disk);
  - *'-cdrom path'* – to select iso path;
  - *'qemu-img'* – to create virtual storage; We should add -hda disk location in the qemu-system command
    - create - f qcow2 'debianbuster.qcow2' ¡-this is the name, 30G ¡- This is the size
    - qemu -image info debianbuster.qcow2
- to exit full screen ctl+alt+f

CREATE A NEW VM:
qemu-system-x86_64 -cpu max -enable-kvm -smp cores=1,threads=20 -
cdrom /home/franciscosantos/Desktop/QEMU/Iso_Images/Windows/Win10_21H2_E
-drive file=/home/franciscosantos/Desktop/QEMU/Virtual_Disks/disk2.qcow2,cache=
-m 4G -vga virtio -display gtk,gl=on

GITHUB – Talk about this and the script and ofofofofofof

1 Investigar estes topicos

Cross compiling Usar a gpu?

Open source, é 5 estrelas, o problema é que está sempre a mudar!

1.1 KVM

https://www.youtube.com/watch?v=G21c-sbggk4
3:44 to install the program that checks if the machine can run KVM.
Kernel based Machines, for hardware acceleration. It gives the opportunitie to run coomands that affect directly the hardware.

Resource over-committing – wich allows you to allocate more virtualizes CPU's and memory than available resources on the host. The VM's the only use what they need, allowing other VM'S to use unused resources.

To disable KVM you just remove *'-enable kvm'* and on the *'-cpu'* we remove the argument and insert a new one that is a different cpu that the host.

KVM uses qemu as his device simulator, any device operation is simulated by user space QEMU program. When you write to 0xB8000, the graphic display is operated which involves guest's doing a CPU 'vmexit' from guest mode and returning to KVM module, who in turn sends device simulation requests to user space QEMU backend.

In contrast, QEMU wo KVM does all the jobs in unified process except for usual system calls, there's fewer CPU context switches. Meanwhile, your benchmark code is a simple loop which only requires code block translation for just one time. That cost nothing, compared to vmexit and kernel-user communication of every iteration in KVM case.

KVM provides near native performance
KVM can over-commit (if we have 20gb kvm can alocatte more) is it good or bad?
It is not supported by every processors linux-kvm.org/page/Processor_support

What is an hypervisor? https://www.youtube.com/watch?v=6k6gtzseq2A&list=PI

2 Qemu

QEMU stands for Quick Emulator, and it is a program that creates a virtual machine and runs the code that we feed into it. It has a built in **bootloader** – where it copys a file to virtual memory and jumps to the start information that it gets out of the header of it –

qemu-system-x86_64 - -help

qemu-img create -f qcow2 nome.qcow2 tamanho em G
Para criar o disco virtual

Reasons to use QEMU?
Performance, near native performance by executing guest code directly on the host CPU. Basicly run sthe guest code on the real CPU.
Where does this helps us?
KVM – helps the QEMU performance expecially n the x64 CPU. We can get even closer to near native performance.

Qemu is an application that tries to emulate a particular architecture.

the -cpu option in QEMU?

Here accelerator refers to KVM. You can use other accelerators by using the -accel option. The -cpu help flag gives you a pretty good explanation.

-cpu max means emulate a cpu that has all the features supported by KVM (limited by the set of features supported by your physical machines, of course)

-cpu host means emulate a cpu that is the same as your host cpu (limited by the set of features supported by the accelerator)

There isn't a whole lot of difference between the two, unless the

accelerator you are using has very limited supported feature set.

It is recommended to just stick with -cpu host, which is also the default. If you have used QEMU on Linux, you have probably enjoyed the performance boost brought by KVM: the same VM runs a lot faster when you launch QEMU with the -accel kvm (or -enable-kvm) option, thanks to hardware-assisted virtualization. On Windows, you can achieve a similar speed-up with -accel hax (or -enable-hax), after completing a one-time setup process.

qemu-system-x86_64 -cpu host -enable-kvm -smp 4 -cdrom /home/franciscosantos/Desktop/QEMU/Iso_Images/Windows /Win10_21H2_English_x64.iso -boot menu=on -drive file=/home/ franciscosantos/Desktop/QEMU/Virtual_imgs/Image.img -m 2G -vga virtio -display gtk,gl=on
O GTK dá, o dld não. Ver se vale a pena investigar. Isto enables opengl. It continuos emulate de display. Emulates VGA com isto

Para posteriores usos remover o -cdrom pois já nao é necessario.

qemu-system-x86 -cpu help
...
x86 host KVM processor with all supported host features
x86 max Enables all features supported by the accelerator in the current host
So it is better to use max?
QEMU also integrates a Linux specific user mode em- ulator. It is a subset of the machine emulator which runs Linux processes for one target CPU on another CPU. It is **mainly used to test the result of cross compilers or to test the CPU emulator without having to start a complete virtual machine**.

www.winehq.org
What is Wine?
Wine (originally an acronym for "Wine Is Not an Emulator") is a compatibility layer capable of running Windows applications on several

POSIX-compliant operating systems, such as Linux, macOS, & BSD. Instead of simulating internal Windows logic like a virtual machine or emulator, Wine translates Windows API calls into POSIX calls on-the-fly, eliminating the performance and memory penalties of other methods and allowing you to cleanly integrate Windows applications into your desktop.
QEMU has two operating modes: Full system emulation. In this mode, QEMU emulates a full system (for example a PC), including one or several processors and various peripherals. It can be used to launch different Operating Systems without rebooting the PC or to debug system code.
User mode emulation. In this mode, QEMU can launch processes compiled for one CPU on another CPU. It can be used to launch the Wine Windows API emulator (http://www.winehq.org) or to ease cross-compilation and cross-debugging.

An "irqchip" is KVM's name for what is more usually called an "interrupt controller". This is a piece of hardware which takes lots of interrupt signals (from devices like a USB controller, disk controller, PCI cards, serial port, etc) and presents them to the CPU in a way that lets the CPU control which interrupts are enabled, be notified when a new interrupt arrives, dismiss a handled interrupt, and so on.
An emulated system (VM) needs an emulated interrupt controller, in the same way that real hardware has a real hardware interrupt controller. In a KVM VM, it is possible to have this emulated device be in userspace (ie QEMU) like all the other emulated devices. But because the interrupt controller is so closely involved with the handling of simulated interrupts, having to go back and forth between the kernel and userspace frequently as the guest manipulates the interrupt controller is bad for performance. So KVM provides an emulation of an interrupt controller in the kernel (the "in-kernel irqchip") which QEMU can use instead of providing its own version in userspace. (On at least some architectures the in-kernel irqchip is also able to use hardware assists for virtualization of interrupt handling which the userspace version cannot, which further improves VM performance.)
The default QEMU setting is to use the in-kernel irqchip, and this gives the best performance. So you don't need to do anything with this command line option unless you know you have a specific reason why the in-kernel irqchip will not work for you.

Em vez de usar -smp 4 -> usar cores=..., threads=..., (691 vs 654)

in the qemu "-smp" argument we can pass the number of: cores, threads and sockets. Does any of you know how do they correlate? I know the concepts os each on in singular, but mixed i am not certain of its functionality. The cores and threads i think i understand, but the socket part i don't know.
i am asking to answer the question: **How can i fine tune this relation to improve performance, and if it can be done to achieve that purpose?**
https://blogs.vmware.com/customer-experience-and-success/2021/06/sockets-cpus-cores-and-threads-the-history-of-x86-processor-terms.html
10 cores and 20 threads. I still can only execute 10 OS instructions per cycle. But I can queue up 20 OS instructions.

Disk perspective https://blogs.igalia.com/berto/2015/12/17/improving-disk-io-performance-in-qemu-2-5-with-the-qcow2-l2-cache/
L1 and L2 tables
What level cache is, and why we use it
Explain how to use it. Use the excel doc.

QCOW2 ARGUMENTS tests and results:

Conclusion, from cinebench and geekbench outputs, so far on the 3 arguments for the .qcow2 format:

preallocation – behaves similar to normal. But my image didn't change, so this behavior was expected;

lazy_refcounts – the best on both so far. (it benefits from the cache=writethrough argument that is passed to be default on all tests)

cluster_size – not optimized for the disk size, it under-performs versus all. I will try to understand how to fine tune this, make new tests and see if the result changes.

Note: The cache is not persistent, you have to specify its size every time you open a new disk image.

[*] Updated 10 March 2021: since QEMU 3.1 the default L2 cache size has been increased from 1MB to 32MB.

QCOW3

Improving performance is the key motivation for a QCOW3 image format.

L2-cache-size vs Cluster-size

https://www.ibm.com/cloud/blog/how-to-tune-qemu-l2-cache-size-and-qcow2-cluster-size

So L2-cache-size is fine tuned in QEMU and Cluster-size is fine tuned in QCOW2 image file.
QCOW2 format stores 2 headers with levels of reference: L1 – that saves l2 references; L2 – that saves data cluster references.
L1 – can is stored alway sin the hypervisor cache memory because it is small in size.
L2 – can be pretty large, so fine tune of the hypervisor L2 cache sise if needed for improoved performance.

What is an hypervisor – https://www.vmware.com/topics/glossary/content/hyperv
Is QEMU an hypervisor? https://sumit-ghosh.com/articles/virtualization-hypervisors-explaining-qemu-kvm-libvirt/
Why do we need to fine tune the hypervisor?
To access a data block in the virtual disk, without any caching, the hypervisor needs to make three IO operations to the QCOW2 image file. This is quite expensive. To improve performance, the hypervisor caches L1 and L2 tables in memory, significantly reducing actual IOs. Since the

L1 table is small, it can be easily cached in memory all the time. The L2 tables can be pretty large. Therefore, the L2 cache size needs to be tuned properly to obtain the best performance.

qemu command accepts a simple -D switch which can create a log file. So for example including -D ./log.txt will create "log.txt" in your working directory.

You can access more logging/debugging options via QEMU Monitor (e.g. qemu -monitor stdio).

3 Fine tunning

See linux kernel version: uname -r
Desativar cortana e essas merdas

QCOW file formats can have: (pagina 78 do qemu.doc)
cluster_size
Changes the qcow2 cluster size (must be between 512 and 2M).
Smaller cluster sizes can improve the image file size whereas larger
cluster sizes generally provide better performance.

A virtual disk, much like how operating systems treat physical disks, are split up into clusters; each cluster being a predefined size and holding a single unit of data. A cluster is the smallest amount of data that can be read or written to in a single operation. There is then an index lookup that's often kept in memory that knows what information is stored in each cluster and where that cluster is located.

preallocation
Preallocation mode (allowed values: off, metadata, falloc, full).
An image with preallocated metadata is initially larger but can improve performance when the image needs to grow. falloc and full preallocations are like the same options of raw format, but sets up metadata also.

lazy_refcounts
If this option is set to on, reference count updates are postponed with the goal of avoiding metadata I/O and improving perfor- mance. This is particularly interesting with cache=writethrough which doesn't batch metadata updates. The tradeoff is that after a host crash, the reference count tables must be rebuilt, i.e. on the next open an (automatic) qemu-img check -r all is required, which may take some time.

A socket is the physical socket where the physical CPU capsules are placed. A normal PC only have one socket.
Cores are the number of CPU-cores per CPU capsule. A modern standard

CPU for a standard PC usually have two or four cores.
And some CPUs can run more than one parallel thread per CPU-core. Intel (the most common CPU manufacturer for standard PCs) have either one or two threads per core depending on CPU model.
If you multiply the number of socket, cores and threads, i.e. 2*6*2, then you get the number of "CPUs": 24. These aren't real CPUs, but the number of possible parallel threads of execution your system can do.
Just the fact that you have 6 cores is a sign you have a high-end workstation or server computer. The fact that you have two sockets makes it a very high-end computer. Usually not even high-end workstations have that these days, only servers.

CACHE https://documentation.suse.com/sles/11-SP4/html/SLES-all/cha-qemu-cachemodes.html

By default, the cache=writeback mode is used. It will report data writes as completed as soon as the data is present in the host page cache. This is safe as long as your guest OS makes sure to correctly flush disk caches where needed. If your guest OS does not handle volatile disk write caches correctly and your host crashes or loses power, then the guest may experience data corruption. For such guests, you should consider using cache=writethrough. This means that the host page cache will be used to read and write data, but write notifi- cation will be sent to the guest only after QEMU has made sure to flush each write to the disk. Be aware that this has a major impact on performance. The host page cache can be avoided entirely with cache=none. This will at- tempt to do disk IO directly to the guest's memory. QEMU may still perform an internal copy of the data. Note that this is considered a writeback mode and the guest OS must handle the disk write cache correctly in order to avoid data corruption on host crashes. citado do qemu-doc.pdf

cpu isolation block app to access this cpu

savevm

savevm name

Save the virtual machine as the tag 'name'. Not all filesystems support this. raw does not, but qcow2 does.
loadvm

loadvm name

Load the virtual machine tagged 'name'. This can also be done on the command line: -loadvm name

With the info snapshots command, you can request a list of available machines.

Example: Create of current state of VM with a name ubuntu_fresh

qemu-img snapshot -c ubuntu_fresh ubuntu.qcow2

16.1 Accessing Monitor Console

To access the monitor console from QEMU, press Ctrl–Alt–2. To return back to QEMU from the monitor console, press Ctrl–Alt–1.
Test scenario of 4 options to improove qcow disk image performance
Install commands
franciscosantos@T14Lenovo: /Desktop/QEMU/Virtual_Disks qemu-img create -f qcow2 -o cluster_size=2M 'cluster_size_2M.qcow2' 40G
Formatting 'cluster_size_2M.qcow2', fmt=qcow2 size=42949672960 cluster_size=2097152 lazy_refcounts=off refcount_bits=16
franciscosantos@T14Lenovo: /Desktop/QEMU/Virtual_Disks ls cluster_size_2M.qcow2 disk2.qcow2 snapshot.qcow2 franciscosantos@T14Lenovo: /Desktop/QEMU/Virtual_Disks qemu-img create -f qcow2 -o preallocation=full 'preallocation_full.qcow2' 40G
Formatting 'preallocation_full.qcow2', fmt=qcow2 size=42949672960 cluster_size=65536 preallocation=full lazy_refcounts=off refcount_bits=16

franciscosantos@T14Lenovo: /Desktop/QEMU/Virtual_Disks qemu-img create -f qcow2 -o lazy_refcounts=on 'lazy_refcounts_on.qcow2' 40G
Formatting 'lazy_refcounts_on.qcow2', fmt=qcow2 size=42949672960 cluster_size=65536 lazy_refcounts=on refcount_bits=16

All arguments direct to a corrupt image ,Don't USE!
franciscosantos@T14Lenovo: /Desktop/QEMU/Virtual_Disks qemu-img create -f qcow2 -o cluster_size=2M,preallocation=full,lazy_refcounts=on '4_options_enabled.qcow2' 40G
Formatting '4_options_enabled.qcow2', fmt=qcow2 size=42949672960 cluster_size=2097152 preallocation=full lazy_refcounts=on refcount_bits=16

cluster size define-se na imagem
l2 cache size define-se no comando.
tb se pode definir outro parametro ao mesmo tempo.

...The table below shows the effectiveness of using the default L2 cache size. For example, if the cluster size is set to 2MiB, each L2 table of 2MiB can contain 262,144 references of data ... – Cada tabela L2 de 2Mb referencia 262K de data clusters de 2Mb cada um.
There are several possible default configurations for the L2 cache and all of them have drawbacks:

a) Have a small(ish) size, like now (1 MB per disk image).
+ Pros: low memory footprint, good enough for many common scenarios.
+ Cons: bad (or very bad) performance with images larger than 8GB and lots of I/O.

b) Have the maximum possible cache size (1MB per 8GB of disk image if using the default cluster size).
+ Pros: good performance in all cases, no need to worry about it.
+ Cons: it can be very wasteful of RAM. A 1TB disk image takes 128MB of RAM for the L2 cache alone. If there are more disk drives (or backing images) the problem gets worse. In most cases you're not going to peform

random I/O on the whole disk, so you don't need such a big cache.
c) Have a large cache size (like in b), and remove unused entries periodically (cache-clean-interval setting).
+ Pros: it provides the best of both worlds, you'll get good performance and the unused memory will be returned to the system.
+ Cons: you can still have peaks of RAM usage. We'd still need to decide what's the best length for the cache cleaning interval. The memory footprint of the VM becomes more volatile and difficult to control.
https://bugzilla.redhat.com/show_bug.cgi?id=1377735

For each 8gb of disk space we need 1Mb of l2-cache-size ram. So if we use default cluster size (64kB) we need to L2 tables

If we use CS of 128GB and 1Mb l2 cahce we can alocate 16GB of disk size.

Subcluster https://blogs.igalia.com/berto/2020/12/03/subcluster-allocation-for-qcow2-images/
Muito interessante isto
It needs qemu 5.2 or higher

previews without cluster size: 1sec ~ 25secs
after cluster size: 1sec ~

Process and interrupt priorities matter a lot when alocatting resources.

3.1 BEST SO FAR

Image creation with cluster size and lazy_refcounts on

img launch, l2 cache fine tuned, cache writethrough

3.2 BASH SCRIPS

Está feito um para criar as VM's.

echo $(nproc) – this says run echo command and turn nproc into an argument

3.3 BCCBPF Tools

They are used for high speed custom packet filtering

https://opensource.com/article/17/11/bccbpf-performance
https://github.com/iovisor/bcc

A tracer is down to 200nanoseconds per trace. so it is very fast

execsnoop

One of the benefits that Ftrace brings to Linux is the ability to see what is happening inside the kernel
The ftrace has a front end that is trace-cmd (better for human reading)
sudo apt-get install bpfcc-tools linux-headers-$(uname -r) to install
sudo apt-get install bcc

PAra dar run as tools:
sudo nome da tool-bpfcc

verificar o nome da tool na pasta usr sbin bpfcc.

fix python dependencies

echo "deb https://repo.iovisor.org/apt/bionic bionic main" — sudo tee /etc/apt/sources.list.d/iovisor.list

sudo apt-get install -y libbpfcc

sudo apt-get install bpfcc-tools linux-headers-$(uname -r)

cannot import name 'MutableMapping' from 'collections' (/us-

r/lib/python3.10/collections __init__.py) -¿ go to the script and write import MutableMapping and save

Install it from store is easier for now.

instalar ferramentas de analise como mpstat:
sudo apt install sysstat

https://www.youtube.com/watch?v=zUYCXBlUcYI
19:57 talks about how insecure this is.
there is eBPF and BPF.

3.4 Tracing

compilando um ficheiro c. Ver o output de objdump -d hello
it creates an elf – executable and linkeble format file – it is written in assembly . It is like an archive or rar to store executable data

Kernel is the system provider for appications, it play this via system calls.

we can see system calls with strace ./hello a lot of info.
Strace we can see the communication between the application and the kernel

ftrace, built and mainteined by steven rostedt, is best. The oficial linux tracing
How? first see if sudo mount -t tracefs nodev /sys/kernel/tracing
go to that directory cd /sys/....
ls and see everything that we can do with ftrace
it need to be su sudo su
cd /sys/kernel/tracing
ls

It is very heavy, and it works like by dinamicly modify the code that calls a tracer.
Patchin gruning kernel uses this.

To enable tracer:
echo function > current_tracer
cat trace to see

to disable tracer:
echo nop > current_tracer

NOTE: you have to always be root to do anything usefull.
we can use trace-cmd, it is easier to run (less instructions) and what it does is created by the same guy, it is a command line utility. this mount from us withou we have to

Instalar o make e sudo apt-get -y install asciidoc
then go sudo su
to start:
trace-cmd start -p function
trace-cmd show
to stop:
trace-cmd start -p nop

trace-cmd record -e syscalls -F ./hello (guarda isto num ficheiro)
trace-cmd report para ver o report
trace-cmd report — less para ver pagina a pagina

Example: AS we know trace-cmd is the front end for ftrace, it needs to be root to output any result. Don't forget!
trace-cmd -e syscalls -F ./hello
→ -e syscalls – we will record all system calls;
→ -F – we will follow the program;
→ ./hello – the program that is going to be followed and recorded;
So we can sum that trace-cmd records everthing that happens inside the

kernel, regarding, in this case, system calls for ./hello.

More complex analysis:

trace-cmd record -p function_graph –max-graph-depth 1 -e syscalls -F ./hello
trace-cmd report -IS does not record HArd interrupts and Soft interrupts (see this)

What does the it mean?
see 24:34 https://www.youtube.com/watch?v=68osT1soAPM
It shows all functions that are called, and shows them on a "C" format. It show the enter and the exit of a function. Max depth of 1 only traces the first function that is going into the kernel and back, nothing else. It shows pages faults, a thing that is not shown in strace. (do page fault) – whe we execute a function it ddoes not load the elf file to memory. It just represents it in memory. 33:50 https://www.youtube.com/watch?v=JRyrhsx-L5Y
It only loads on demand, it is more efficient.
This means that it tryed to read memory that wasnt there.
we can do max depth 2 and this shows not only the first function but the functions that function calls.
we can look at a specific function by doing this:
trace-cmd record -p function_graph -g nome da funcao que queremos ver -F ./hello
to see the report, trace-cmd report
the -g means graph this function only. we can ignore functions inside our serch by doing this:
trace-cmd record -p function_graph -g nome da funcao que queremos ver -n nome da funcao a ignorar -F ./hello
trace-cmd report

we can -o interrupts (confirm what it does better):
trace-cmd record -p function_graph -g nome da funcao que queremos ver -n nome da funcao a ignorar -o funcao interrupt \-F ./hello
trace-cmd report

VFS means virtual file system, wich handles all filesystem, all files pass in this layer.
ttwu means try to wake up

https://www.youtube.com/watch?v=68osT1soAPM
49' **Digg deeper:**
\-F it means only to trace this program (./hello) and nothing else. we can remove it to trace all.
we can trace just certain events with -e

Note that -f it stands for filter!
Note that -R stands for trigger!

man trace-cmd is a bless!

trace-cmd record -e sched_wakeup -e sched_switch -e sys_enter_write ./hello
with this we can really see the shcedular switching, it is not at the same time, but it is so fast that it seams.

but this e too much information! How can we visualize it better?
With KernelShark, it can read trace.dat files (this file is produced by trace-cmd record)
it gives visualization of what is happening
much easier to parse large amounts of data
sudo apt-get install -y kernelshark
open the app and load the .dat file created from the trace-cmd
It shows the thing that really worked up.

Now with visualization we can record all and see it.
trace-cmd record -e all ./hello
trace-cmd report

we can scroll wheel to zoom in
we can search based on columns!

we can rigth click on the entry to see more advanced stuff and filters.
we can drag an area on the plot to zoom.
Note: The reports are rewritten so be carefull if you want to save any.

Search for a systemcall process id:
trace-cmd record -e syscalls -P 1779 ./hello
(enable all systemcalls and see what that process is doing)
trace-cmd report

we can then run the:
trace-cmd record -p function_graph -g function ./hello
trace-cmd report -F '.:common_pid == 1779' (fazer so o report deste pid)

All timestamps are in seconds, with a resolution of 1/10 of a microsecond.
The data on the plots are either CPU specific or task specific.
If they are CPU specific, then the data holds the timeline of events that happened on a given CPU (which CPU is shown in the plot title area).
If the plot is task specific, then the timeline of events are for the given task regardless of what CPU it was on at the time. The task name is also shown in the plot title area.
The CPU plots change colors as different tasks run on the CPU, and the task plots change color depending on what CPU the task is running on. This makes it easy to see how much a task bounces around the CPUs. Zooming in on a task plot also shows some more characteristics of the task.
The hollow green bar that is shown in front of some events in the task plot represents when the task was woken up from a sleeping state to when it actually ran. The hollow red bar between some events shows that the task was preempted by another task even though that task was still runnable.

The columns of the list are:

- the position of the event in the full data set.
CPU - the CPU that the event occurred on.
Time Stamp - The timestamp of the event. This is in seconds with 1/10th

microsecond resolution.
Task - The name of the process that was running when the event occurred. PID - The process ID of the task that was running when the event occurred. Latency - The latency is broken into 4 fields:
Interrupts disabled - 'd' if interrupts are disabled, otherwise '.' Need reschedule - 'N' if the kernel was notified that a schedule is needed, otherwise '.'
In IRQ - 'h' if in a hard IRQ (hardware triggered), 's' if in a soft IRQ (context where the kernel initiated a the IRQ handler) or if soft IRQs are disabled, 'H' if in a hard IRQ and soft IRQs are disabled or the hard IRQ triggered while processing a soft IRQ, otherwise '.'
Preemption counter - The index of the preemption counter. If it is other than zero, then the kernel will not preempt the running tasks, even if a schedule has been requested by some event. If the counter is zero, then '.' is shown.

Note: These may be different depending on the kernel the trace.dat file came from.
Event - The name of the event.
Info - The data output of a particular event.

The list search can find an event based on the contents in a row. Select a column, a match criteria and the content to match to find the next row that matches the search. The match criterion is one of the following:

contains - the row cell of the selected column contains the match data. This works with numbers as well.
full match - the row cell of the selected column matches exactly the match data.
does not have - the row cell of the selected column does not contain the match data.

NOTE: we can Export Filter: save the filter to a file, that can be imported at a later time
in filters in the "show..." we can just show what we want.

Filtering on events may not be enough. The ability to filter on the content of individual events may be needed. In order to accomplish this, the advanced event filtering is used. Selecting Advance Filtering from the Filter menu will pop up the advanced event filtering dialog.

see 49 minuts. very hard.

To disable all tracing, which will ensure that no overhead is left from using the function tracers or events, the reset command can be used. It will disable all of Ftrace and bring the system back to full performance.

trace-cmd reset

https://lwn.net/Articles/410200/

You can do extensive filtering on events and what CPUs you want to focus on:
trace-cmd report –cpu 0 -F 'sched_wakeup: success == 1'
version = 6
cpus=2
..
The –cpu 0 limits the output to only show the events that occurred on CPU 0. The -F option limits the output further to only show sched_wakeup events that have its success field equal to 1. For more information about the filtering, consult the trace-cmd-report(1) man page.

To track a specific cpu:
trace-cmd report –cpu 0

3.4.1 Tracing QEMU:

It is good to formulate questions xD
Enable tracing while a specific command is being executed:
trace-cmd record -p function ls -¿¿ Test this with qemu
we need to start the trace.cmd, launch this (see upper line) cmd and then

run the ls. Lastly run the report, and stop the tracing by either set nop or reset.
https://lwn.net/Articles/425583/

1. as sudo su type: trace-cmd start -p function (no need with record)!
2. trace-cmd stat (to see if it is enabled)
3. trace-cmd record -p function ls (run it)
4. In other terminal window run your script
5. in the initial terminal generate the .dat file with: trace-cmd report
6. stop tracing by trace-cmd stop or reset.

NOTE: DOnt forget that the report saves in the current directory!

Don't forget to go to the filter tab to filter the information on the list, and go to plot tab to show more info in the plot area.

to trace just a function we use: trace-cmd record -p function_graph -g 'switch_fpu_return' ls

To search for a function inside the kernel that has a specifi word, we will use write as the word, on it we can run:
trace-cmd list -f write
We can then filter more by grepping it to syscals and see which syscall functions have write on it:
trace-cmd list -f write — grep syscals

We can combine that all in the record command like this:
trace-cmd record -p function_graph -g '*sys*write' -F ./hello
The -g stans for graph this function.
We can now trace any sys write functions that are called when running the hello program.
We can the create a trace-cmd report — less and see what the write syscall does and follow it through.

we can then open up the souce code related to any function and learn how a OS really works. That is the beauty of open source.

Something so simple as Hello Word, can be extremly complex at kernel level.

funcgraph_entry quando entra
funcgraph_exit quando sai
O tempo mostrado é o que demorou la dentro.

Start just enables tracing, record puts it into a file.

3.5 NUMA

Non Uniform Memory Access
See if it is worth explore.

https://www.youtube.com/watch?v=w3yT8zJe0Uw
13'33"

See RT-Tests. They are anotehr set of tools.

Basic Definition

A node refers to the physical box, i.e. cpu sockets with north/south switches connecting memory systems and extension cards, e.g. disks, nics, and accelerators

A cpu socket is the connector to these systems and the cpu cores, you plug in chips with multiple cpu cores. This creates a split in the cache memory space, hence the need for NUMA aware code.

A cpu core is an independent computing with its own computing

pipeline, logical units, and memory controller. Each cpu core will be able to service a number of cpu threads, each having an independent instruction stream but sharing the cores memory controller and other logical units.

CPUs (chips) and each chip may have multiple cores. Each core can execute one (or sometimes multiple) stream of instructions.

3.6 No HZ

Good for power managment. but worst for performance

3.7 mmiotrace

Trace interactions between drivers and hardware
It puts the system in uni mode only one CPU.

https://www.youtube.com/watch?v=68osT1soAPM
9:27
nao foi o gajo que inventou mas talvez seja de interesse.

3.8 Latency

What is it? https://www.youtube.com/watch?v=Tkra8g0gXAU
The time from when an event is suppose to happen to the time it actually does happen.

trace-cmd record says we want to record it to a file.
19:39
trace-cm list -e will list all available events.

The reports we can let them run for a while or time it or just apend it to ls

31' for hystogram latency.

4 HOST preparation

1. run check.sh to see if IOMMU and VT-D is enabled and suported in your chip and Ensuring that the groups are valid, and confirm that iommu is on: dmesg — grep -i -e DMAR -e IOMMU;
2.

4.1 CPU PINNING

if we want to have a really good performance this is it.
We allocate a CPU to work on a specific task, it prevents unpredicted process to disturb the normal behaviour.
System managment threads will still run on this CPU. And it is ok, because some are for the linux kernel, and are needed.

See cpu topology:
lscpu -e

teste=$(echo $ola — awk 'print $5')
arr=($(echo $teste — tr ":" "\n"))

4.2 GPU ISOLATION

Warning: Once you reboot after this procedure, whatever GPU you have configured will no longer be usable on the host until you reverse the manipulation. Make sure the GPU you intend to use on the host is properly configured before doing this - your motherboard should be set to display using the host GPU.

Most use cases for PCI passthroughs relate to performance-intensive domains such as video games and GPU-accelerated tasks. While a PCI passthrough on its own is a step towards reaching native performance, there are still a few ajustments on the host and guest to get the most out of your virtual machine.

5 KernelShark Before and After CPU pinning

In linux every process has a unique PID (process identification value). that's how we can track it. We can check the process using ps in the terminal.
Threads are the smallest unit of processing that can be performed in an operating system. In the modern days multiple threads can be done in a single process.
each thread is considered a single porcess in the kernel.

6 Questions that need a response?

1. What is a good way to benchmark a machine?
2. Compare the same app or math computation with and withouth KVM enabled.
3. is time and memory a good benchmark?
4. running for the second time is way more fast. why?
5. why use W10? (thats not may goal)
6. **How to avoid loading boo everytime QEMU runs?**
7. **How to increase the performance when compiling QEMU**
8. try to increase clock on a CPU for highr time
9. gpu passthroug?
10. looking glass????
11. does adding sudo to the qemu run helps? Nope
12. What is a cluster vs l2 cache?
13. Cluster size and Clean time?
14. **How it works?**
15. **Why it works?**

Note. Multiplciar floats, its a good test? Comando htop para visualizar alguns parametros

7 Benchmark tools

Cinebench -> install it on the virtual machine to test effect of arguments.

https://en.wikipedia.org/wiki/Benchmark_(computing)

Who Should Use Cinebench? ... hardware manufacturers can utilize the feedback in optimizing their latest products. Any computer owner can evaluate their individual system. Unlike abstract benchmarks, which only test specific functions of CPUs, Cinebench offers a real-world benchmark that incorporates a user's common tasks within Cinema 4D to measure a system's performance.

Cinebench R23 now supports Apple's M1-powered computing systems
Cinebench is now based on the latest Release 23 code using updated compilers, and has a minimum runtime activated by default (previously hidden in preferences) Cinebench R23 provides improved benchmark accuracy for current and next generation CPUs to test if a machine runs stable on a high CPU load, if the cooling solution of a desktop or notebook is sufficient for longer running tasks to deliver the full potential of the CPU, and if a machine is able to handle demanding real-life 3D tasks.
Users now have the option to directly test the single core performance without manually enabling the "Advanced benchmark" option. The "Advanced benchmark" allows users to set arbitrary minimum runtimes to stress test the hardware for even longer periods of time.
Because of the code and compiler changes, Cinebench score values are readjusted to a new range so they should not be compared to scores from previous versions of Cinebench.
Cinebench R23 does not test GPU performance.
Cinebench R23 will not launch on unsupported processors. On systems lacking sufficient RAM to load the test scene, a warning will be displayed and the CPU benchmark will not be executed.
Background tasks can significantly influence measurement and create diverse results. It's always a good idea to shut down all running programs and disable any virus checking or disk indexing but it's impossible to eliminate all background processes. Modern operating systems perform various background tasks that cannot or should not be disabled, even though they could have a minor influence on the results.

Test results can vary slightly because it's impossible to disable every background task of the operating system. These tasks are one factor that may have a slight influence on measurements. Also, modern computers and graphics cards dynamically adjust clock speeds based on environmental conditions like power and temperature. For instance, processors will reduce clock speed when running too hot to allow for cooling and prevent damage. With many modern processors, the reverse is also true. They are able to overclock themselves when the temperature is low enough.
Therefore, a system freshly started in a relatively cool environment will typically run faster than the same system that has been performing benchmarks for several hours in a heated office.

Cinebench is a CPU-based rendering benchmark which tells you how fast your CPU can render a scene using all CPU threads. The score can be compared to other CPUs to determine the relative performance between them. The numbers won't tell you anything in regards to video production or gaming as those workloads are different.

Cinebench without vga see excel file lazy_refcounts was the best

Geekbench
Takes too much time. The bse single core score is 815
But...

Uset -rtc!

qemu-system-x86_64 -cpu max -enable-kvm -smp cores=1,threads=20 -name 'cluster' -rtc base=localtime,clock=host -drive file=/home/franciscosantos/Desktop/QEMU/Virtual_Disks/disk2.qcow2,cache=write -m 4G

8 Create snapshot

This is for internal snapshot
https://www.youtube.com/watch?v=EaTPEy2Tw_c
At 6:47 we can see how to enter via terminal to the VM.

1st boot the VM then create snapshot in VMManager
virsh snapshot-create-as –domain nome da VM –name ”nome entre aspas do snap”
Example:
virsh snapshot-create-as –domain dummy1 –name teste
click enter
”Domain snapshot ....” should print in the terminal

See if it created:
virsh snapshot-list dummy1
Name Creation Time State
————————————————-
teste 2022-03-18 14:49:23 +0000 running

virsh shutdown dummy1
Domain dummy1 is being shutdown

virsh list –all
Id Name State
————————————-
- dummy1 shut off

Revert to the snapshot:
virsh snapshot-revert –domain dummy1 –snapshotname teste –running
Click on open to enter VM
State == running it means that the VM was running when the snapshot was created.
virsh start dummy1 par aexecutar? Testar isto

IF BIOS all ok
IF UEFI we have to change the xml file. See bookmark https://www.youtube.com/watch?v=1SDvth66i-4

NOT YET TESTED!
This is for external snapshots
https://www.youtube.com/watch?v=1SDvth66i-4 8 minuts Virt-manager doesnt support external snapshots, so we do it from the terminal.
first turn the VM off

then virtual machine – details – overview – xml tab
(enable xml editing)
fazer cenas
remove metadata associated to external machine
delete external snapshot file itself
Snapshot is then restored
start the virutal machine.

Qemu monitor 1st STOP 2nd savevm name run the load via terminal

BETTER YET snapshot repository:
1 – Create external .qcow2 file to store the snapshot:
qemu-img create -f qcow2 -b disk2.qcow2 snapshot.qcow2
2 – Run qemu with this new file as the virtual disk:
qemu-system-x86_64 -name 'dummy3' -cpu max -enable-kvm -smp cores=1,threads=20 -drive file=/home/franciscosantos/Desktop/QEMU/Virtual_Disk -m 4G

3 – We put the machine in the state that we want and enter the qemu monitor to save the snapshot:
stop
savevm name
4 – To load this snapshot we run:
qemu-system-x86_64 -name 'dummy3' -cpu max -enable-kvm -smp cores=1,threads=20 -drive file=/home/franciscosantos/Desktop/QEMU/Virtual_Disks/snapshot.qcow2,cache=writethrough -m 4G -loadvm snap1
Note: The new file will be smaller than the original, and all the changes will be on this one. Be carefull with this for future changes.

List snapshots inside qcow2 image:
qemu-img snapshot -l image name
delete snapshot:
emu-img snapshot -d snap_dummy4 disk2.qcow2

Save changes to the original disk image
https://techpiezo.com/linux/use-and-implementation-of-backing-file-and-snapshot-in-qemu-kvm/

Trying to benchmark guest software under QEMU emulation is at best extremely difficult. QEMU's emulation does not have performance characteristics that are anything like a real hardware CPU's: some operations that are fast on hardware, like floating point, are very slow on QEMU; we don't model caches and you won't see anything like the performance curves you would see as data sets reach cache line or L1/L2/etc cache size limits; and so on.

Important factors in performance on a modern CPU include (at least):

raw instruction counts executed
TLB misses
branch predictor misses
cache misses

QEMU doesn't track any of the last three and only makes a vague attempt at the first one if you use the -icount option. (In particular, without -icount the RDTSC value we provide to the guest under emulation is more-or-less just the host CPU RDTSC value, so times measured with it will include all sorts of QEMU overhead including time spent translating guest code.)

Assuming you're on an x86 host, you could try the -enable-kvm option to run this under a KVM virtual machine. Then at least you'll be looking at the real performance of a hardware CPU, though you will still see some noise from the overhead as other host processes contend for CPU with the VM.

Five sections, in the Libvirt virtual machine configuration, are crucial in order to optimize the virtual machines performance:
CPU pinning
CPU model information
Hyper-V enlightments
Clock settings

Hugepages
https://mathiashueber.com/performance-tweaks-gaming-on-virtual-machines/
List CPU info on the terminal: lscpu

Inside the kernel thr priorities are reversed, so the higher the priority the lowest priority it has. So 98 priority in kernel is a pryority 2, and a 0 priority is a 100 so its first.

circular dependencies?

Kernel space vs User space

We cannot push something into the kernel, it must be pulled in - Steven Rosted
https://www.youtube.com/watch?v=4UY7hQjEW34

mutex is a lock (wait) and release mechanism.
Semaphores are signalling mechanisms that signal to processes the state of the Critical section in OS and grant access to the critical section accordingly.

Mutex works in user-space and Semaphore for kernel
Mutex is for Threads, while Semaphores are for processes.
A thread may acquire more than one mutex

When I am having a big heated discussion at work, I use a rubber chicken which I keep in my desk for just such occasions. The person holding the chicken is the only person who is allowed to talk. If you don't hold the chicken you cannot speak. You can only indicate that you want the chicken and wait until you get it before you speak. Once you have finished speaking, you can hand the chicken back to the moderator who will hand it to the next person to speak. This ensures that people do not speak over each other, and also have their own space to talk.

Replace Chicken with Mutex and person with thread and you basically have the concept of a mutex.

Of course, there is no such thing as a rubber mutex. Only rubber chicken. My cats once had a rubber mouse, but they ate it.

Of course, before you use the rubber chicken, you need to ask yourself whether you actually need 5 people in one room and would it not just be easier with one person in the room on their own doing all the work. Actually, this is just extending the analogy, but you get the idea.

Interrupts are threads, and they can be traced by their priority.

Latency always happens.

Both processes and threads are independent sequences of execution. The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

I'm not sure what "hardware" vs "software" threads you might be referring to. Threads are an operating environment feature, rather than a CPU feature (though the CPU typically has operations that make threads efficient).

Erlang uses the term "process" because it does not expose a shared-memory multiprogramming model. Calling them "threads" would imply that they have shared memory.

In kernel we dont do recursive calls because it can blow up a stack with ease.

10 Cross compiling

canadian cross compiling
Cross compiling is ver hard.
https://landley.net/writing/
53mints is where he talks about this.

In sum if we can avoid cross compiling we should try!

Native compiling under emulation is so much easier. (see if native compiling is the same as normal compiling. i think yes).

the guy Rob Landley made mkroot.

see muscl-cross-make

the difference between normal compiling and cross compiling is

11 FLUTTER APP

End goal

12 Uteis

How Do I Make A Script Executable?
Create a script by going to the directory where it will be created.
Your file should be made as simple as. sh.
In order to create the script, you will have to work with an editor.
Use command chmod +x *fileName¿ to create the script executable.
You can script by using "/*fileName" instead of [.]"file".

qemu-img info nomeimg.qcow2
qemu-img map nomeimg.qcow2

We can link a qemu imge to a qemu machine so we don't have to pass it always.

-p option shows the percentual progress of the task

find folders:
sudo find . -name QEMU -¿ procurar merdas

nproc no terminal dá-nos o numero de processadores que o host tem.

Find inside files:
grep -r 'switch_fpu_return*'