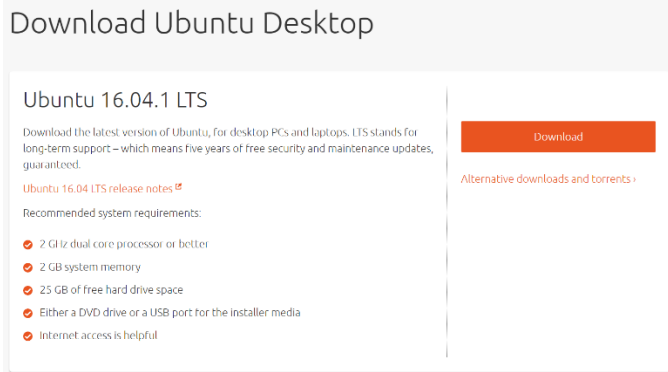


Computer Networks Assignment #1 Zhang Zhexian (0545080) zhangzhexian@outlook.com

1. [Chapter 1 Hands-on 2] Install a newest Linux distribution, and summarize: (1) its installation process and (2) things inside a Linux distribution.

The Linux distribution installation process:

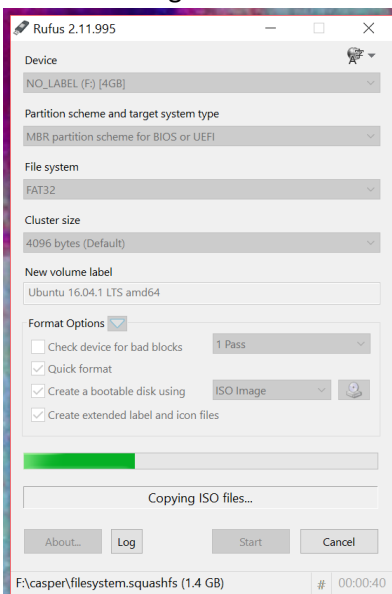
- a. Download the installation package



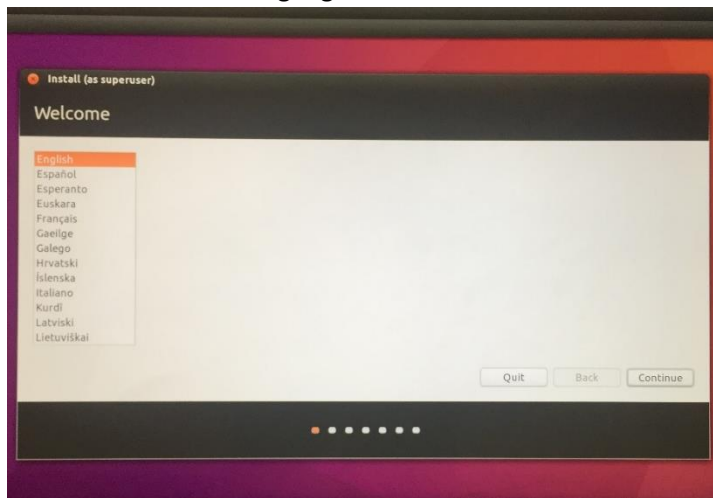
- b. Create bootable USB



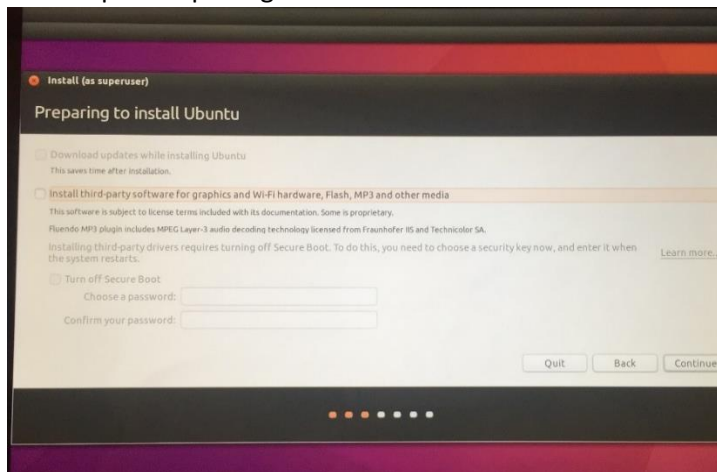
- c. Write ISO image into the USB



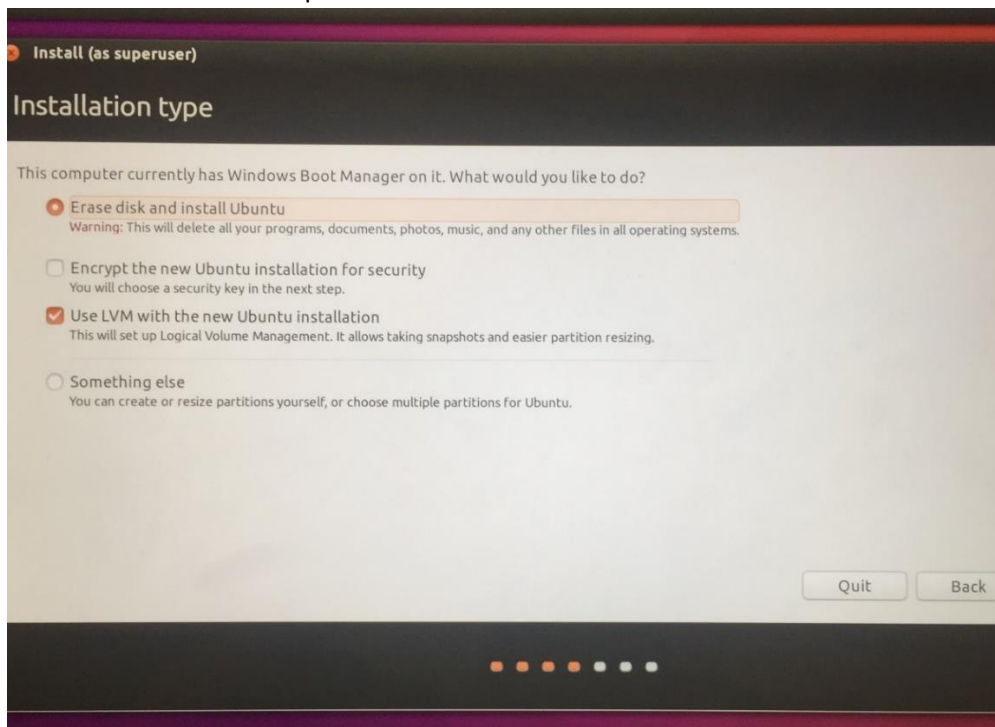
d. Choose installation language



e. Select optional packages to install



f. Choose installation option



As you may see from the above screenshot, since my computer currently has Windows Boot Manager on it, I do not have the option to keep my current Windows 10 operating system while installing Ubuntu. Since I do not wish to erase disk, the installation process is only carried until this step.

Things inside a Linux distribution:

As I have Linux Ubuntu installed in a virtual machine, I am still able to describe what's inside a Linux distribution. The distribution includes a Linux kernel, GNU tools and libraries. Also included are additional software, documentation, as well as a window, a window manager, and a desktop environment. Most of the included software is free and open-source software made available both as compiled binaries and in source code form.

2. [Chapter 1 Hands-on 3] First read Appendix B, then look up the programs under `/src`, `/usr/src`, or other directories where the source files reside, depending on the version of the Linux distribution being used; summarize and categorize what's inside that directory.

For this question, I am referring to the Linux directory in this GitHub repository:

<https://github.com/torvalds/linux>

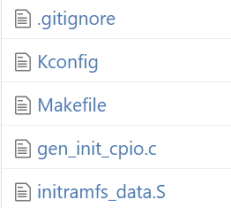
First level directory:






Documentation	Merge branch 'x86-asm-for-linus' of git://git.kernel.org/pub/scm/linu...	3 hours ago
arch	Merge branch 'x86-vdso-for-linus' of git://git.kernel.org/pub/scm/lin...	2 hours ago
block	blk-mq: skip unmapped queues in blk_mq_alloc_request_hctx	10 days ago
certs	certs: Add a secondary system keyring that can be added to dynamically	6 months ago
crypto	crypto: rsa-pkcs1pad - Handle leading zero for decryption	12 days ago
drivers	Merge branch 'x86-platform-for-linus' of git://git.kernel.org/pub/scm/linu...	2 hours ago
firmware	WHENCE: use https://linuxtv.org for LinuxTV URLs	10 months ago
fs	Merge branch 'x86-vdso-for-linus' of git://git.kernel.org/pub/scm/linu...	2 hours ago
include	Merge branch 'x86-vdso-for-linus' of git://git.kernel.org/pub/scm/linu...	2 hours ago
init	sched/core: Add try_get_task_stack() and put_task_stack()	18 days ago
ipc	ipc: delete "nr_ipc_ns"	2 months ago
kernel	Merge branch 'x86-vdso-for-linus' of git://git.kernel.org/pub/scm/linu...	2 hours ago
lib	Merge branch 'x86-asm-for-linus' of git://git.kernel.org/pub/scm/linu...	3 hours ago
mm	Merge branch 'x86-vdso-for-linus' of git://git.kernel.org/pub/scm/linu...	2 hours ago
net	sctp: fix the issue sctp_diag uses lock_sock in rcu_read_lock	4 days ago
samples	Merge git://git.kernel.org/pub/scm/linux/kernel/git/davem/net	2 months ago
scripts	scripts/recordmcount.c: account for .softirqentry.text	5 days ago
security	Merge branch 'linus' of git://git.kernel.org/pub/scm/linux/kernel/git/...	10 days ago
sound	ALSA: rawmidi: Fix possible deadlock with virmidi registration	26 days ago
tools	Merge branch 'x86-asm-for-linus' of git://git.kernel.org/pub/scm/linu...	3 hours ago
usr	usr/Kconfig: make initrd compression algorithm selection not expert	2 years ago
virt	Merge tag 'kvm-arm-for-v4.8-rc3' of git://git.kernel.org/pub/scm/linu...	2 months ago

Based on Appendix B, the directory can be categorized into following groups:

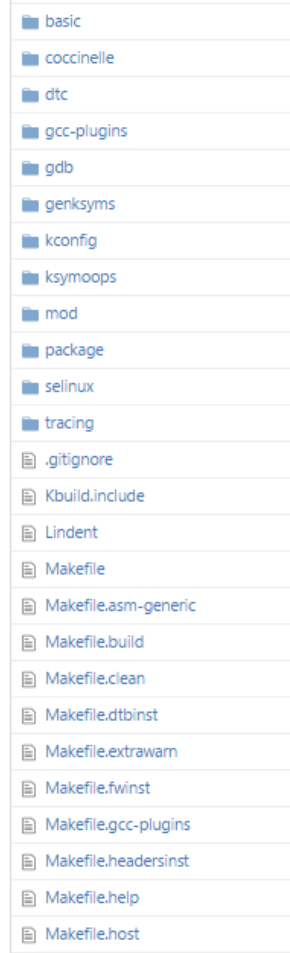
- Creation: help in the making of the kernel



























[usr/]



 .gitignore
 Kconfig
 Makefile
 gen_init_cpio.c
 initramfs_data.S

[scripts/] contains command-line scripts, and C source codes to build the kernel

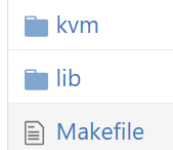


 basic
 coccinelle
 dtc
 gcc-plugins
 gdb
 genksyms
 kconfig
 ksymoops
 mod
 package
 selinux
 tracing
 .gitignore
 Kbuild.include
 Lindent
 Makefile
 Makefile.asm-generic
 Makefile.build
 Makefile.clean
 Makefile.dtbinst
 Makefile.extrawarn
 Makefile.fwinst
 Makefile.gcc-plugins
 Makefile.headersinst
 Makefile.help
 Makefile.host

- Architecture-specific: architecture-specific source and header files
[arch/] platform-dependent code is placed here

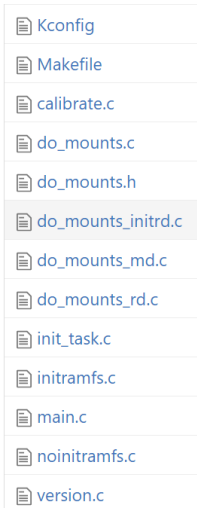


[virt/] code to support kernel-based, hardware-assisted virtualization

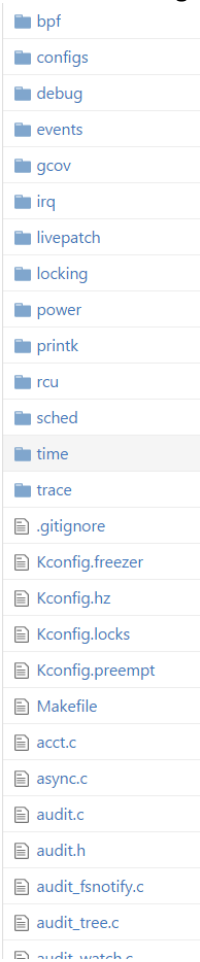


- Kernel core: core functions and frameworks used in kernel

[init/]

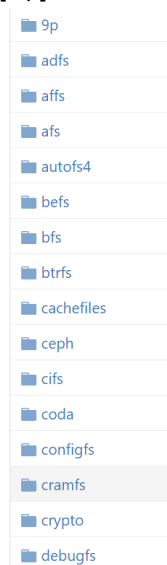


[kernel] the implementations of the initialization, scheduling, and synchronization and process management functions



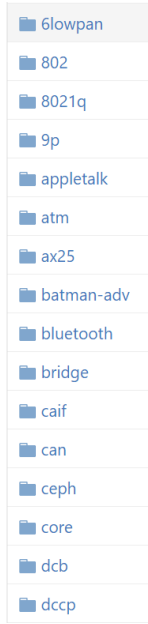
Other folders include [include/], [lib/], [block/], [ipc/] (inter-process communication (IPC) functions, e.g., shared memory management), [mm/] (the memory management routines), [security/], and [crypto] (The cryptographic application programming interface (API))

- File system: file system-related source code [fs/]



- Networking: Networking-related source code

[net/]



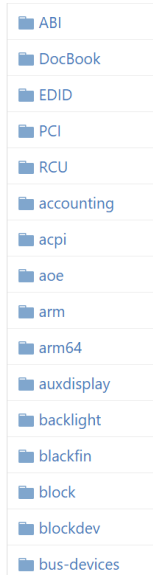
- Drivers: device drivers

[drivers/] all kinds of drivers except the sound card drivers are put under this



Other folders include [sound/] (sound card drivers) and [firmware/] (firmware image files extracted from device drivers)

- Helper: document and sample codes that help the reader get involved in the kernel development [documentation/]



The other folder under this category is [samples/]

3. [Chapter 1 Hands-on 6] Trace the Linux kernel code to find:
 - a. Which function calls `alloc_skb()` to allocate `sk_buff` for the request and the response, respectively, in Figure 1.19.
 For the request: `e1000_setup_desc_rings()`
 For the response: `tcp_send_ack()`
 - b. Which function calls `kfree_skb()` to release `sk_buff` for the request and the response, respectively, in Figure 1.19.
 For the request: `tcp_close()`
 For the response: `tcp_data_queue()`
 - c. Which function calls `alloc_skb()` to allocate `sk_buff` in Figure 1.21.
`inet_rtm_getroute()`
 - d. Which function calls `kfree_skb()` to release `sk_buff` in Figure 1.21.
`inet_rtm_getroute()`
 - e. How you trace these dynamically or statically.
 I statically traced these by downloading the Linux source code from GitHub, and searching for keywords in the directory. It is not a practical way as there are 1000+ matches across all files.
4. [Chapter 1 Written 1] Consider a transcontinental link of 5000 miles with a bandwidth of 40 Gbps. Assume the propagation speed is 2×10^8 m/sec.
 - a. What is the width of one bit in time and in length, respectively?

40 Gbps = 40 billion bits per second

5000 miles = 8046720 meters

1 bit time = $1/(40 \times 10^9) = 2.5 \times 10^{-11} \text{ s}$

1 bit length = $2.5 \times 10^{-11} \times 2 \times 10^8 = 5 \times 10^{-3} \text{ m}$

- b. How many bits can be contained in the link at most?

Number of bits = $8046720 / (5 \times 10^{-3}) = 1609344000 \approx 1.61 \times 10^9 \text{ bits}$

- c. What is the transmission time of a packet of 1500 bytes?

1500 bytes = 12000 bits

Transmission time = $12000 \times 2.5 \times 10^{-11} = 3 \times 10^{-7} \text{ s}$

- d. What is the propagation time through this link?

Propagation delay is how long it takes one bit to travel from one end of the link to the other.

Thus, propagation time = $8046720 / (2 \times 10^8) = 0.0402336 \approx 4.02 \times 10^{-2} \text{ s}$

5. [Chapter 1 Written 3] Suppose a 1 Gbps link has exponential packet inter-arrival time and service time. We like to apply the queueing theory and Little's result to calculate mean latency and occupancy.

- a. If the mean arrival rate is 500 Mbps, what are the mean latency, queuing time, and occupancy?

The link has exponential packet inter-arrival time and service time \rightarrow M/M/1 queue

1 Gbps = 1 billion bits per second

500 Mbps = 500 million bytes per second

Latency = $1 / (\text{bandwidth} - \text{offered load})$

Latency = $1 / (1 \times 10^9 - 500 \times 10^6) = 2 \times 10^{-9} \text{ s}$

Queuing time is negligible since mean arrival rate is much smaller than bandwidth

By Little's Result, occupancy = throughput * latency

At low arrival rate, throughput \approx arrival rate

Thus, occupancy = $500 \times 10^6 \times 2 \times 10^{-9} = 1$

- b. If the link bandwidth and mean arrival rate are increased by an order of magnitude to 10 Gbps and 5 Gbps, respectively, what are the mean latency, queuing time, and occupancy?

The link has exponential packet inter-arrival time and service time \rightarrow M/M/1 queue

Latency = $1 / (\text{bandwidth} - \text{offered load})$

Latency = $1 / (10 \cdot 10^9 - 5 \cdot 10^9) = 5 \cdot 10^{-9} \text{ s}$

Queuing time is negligible since mean arrival rate is much smaller than bandwidth

By Little's Result, occupancy = throughput * latency

At low arrival rate, throughput \approx arrival rate

Thus, occupancy = $5 \cdot 10^9 \cdot 5 \cdot 10^{-9} = 1$

6. [Chapter 1 Written 5] Suppose there are 3,000,000 new phone call arrivals per minute to the switched telephone system worldwide, with each call lasting for 5 minutes on average, and there are 6 hops (i.e. 6 links and 6 nodes) on average between the callers and callees. How many memory entries are occupied on average to support the switched connectivity worldwide?

Circuit switching: the number of entries in switching table corresponds to the number of hops/connections

Number of memory entries = $3000000 \cdot 5 \cdot 6 = 90000000$

7. [Chapter 1 Written 10] Here we compare the overhead of routing packets and switching packets. Why is the time complexity of routing higher than switching, while the space complexity of switching is higher than routing?

Time complexity: less time is required for switching, as reading off directly from the switching index table is much faster (only one memory access is required), compared to table entry matching with several memory accesses used in routing (with possible collisions during matching);

Space complexity: switching needs to establish a much larger entry table for indexing, thus occupying more memory space; routing only needs the number of destination, not the number of hops, as the table entries, so routing is more space efficient.

8. [Chapter 1 Written 13] ATM (asynchronous transfer mode) and MPLS (multi-protocol label switching) do not have stateless core networks. What states do they keep? What is the main difference in the way they keep these states?

ATM: hard-state switching, it provides bandwidth allocation to individual connections

MPLS: soft-state switching, which switches more packets and routes fewer packets

Main difference: soft-state is more flexible than hard-state, as the states only persist for a limited period of time, and they will be removed if they are not access for certain length of time. Soft-state switching allows MPLS to better comply with stateless routing, and thus to easily deploy on a small scale, say an ISP.

9. [Chapter 1 Written 17] In Figure 1.13, why do we put the routing task as a daemon in the user space while keeping the routing table lookup in the kernel? Why not put both in the user space or the kernel?

Routing table lookup provide services to user space processes, so it needs to be kept in the kernel.

As for the routing task, though it is application-independent, the tasks are complicated (sometimes need to run complex route computation algorithms). Thus, they are put into the user space programs, which run as daemon processes in background persistently. Another reason for not putting them into the kernel is because there are so many of them.

We do not put both in the user space or in the kernel, because routing table takers time, and we do not want to block other kernel applications or to slow down the kernel processes. Also, we do not want to put programs that may crash into kernel space, because when one application in kernel crashes, the entire kernel will crash, but in user space even if there are crashes the kernel will not be affected.

10. [Chapter 1 Written 20] We need to understand how the hardware works along with its driver.
- What is the interface between the driver of a network adaptor and the controller of the network adaptor?

The interface is the set of registers provided by the controller for the driver to read and write. By writing or reading these registers, the driver can issue commands to or read status from the device. The two methods to access the registers are I/O commands and memory-mapped I/O.

- How does the driver ask the controller to send a packet and how does the controller report it has completed the job?

The driver asks the controller to send a packet by writing a *transmit* command into its command register. It may also repeatedly try to retransmit should a collision occur.

The controller reports that it has completed the job by writing into the command register too.

- How does the controller report to the driver when a packet has arrived at the network adaptor?

The controller report to the driver by triggering an interrupt to ask the corresponding driver to move the packet into the host memory.