DUBLIN CITY UNIVERSITY

ELECTRONIC AND COMPUTER ENGINEERING

EE562 - Network Stack Implementation

Assignment 1

Author

Michael Lenehan michael.lenehan4@mail.dcu.ie

Student Number: 15410402

23/03/2020

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at `https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf` and IEEE referencing guidelines found at `https://loop.dcu.ie/mod/url/view.php?id=448779`.

Signed: _____ Date: 23/03/2020

Michael Lenehan

Contents

1 Completely Fair Scheduler

1.1 Scheduling Algorithms

Scheduling algorithms are a way for the kernel to provide CPU resources to processes in a balanced way. The original scheduling algorithm used by the Linux kernel based it's scheduling on the time a process was running for. Processes which require user input require more time on the CPU, as there cannot be a perceptible delay in execution of user commands. This original scheduling algorithm was of constant time complexity, and as such was named the Linux O(1) scheduler[1].

The Completely Fair Scheduler was introduced as an improvement upon the O(1) scheduler, improving the time complexity to O(logn), and reducing the scheduling codebase by approximately 700 lines[2].

1.2 Completely Fair Scheduler

The completely fair scheduler is the currently used scheduling algorithm in the Linux kernel. The algorithm was merged into the codebase for the release of version 2.6.23 (current release is 5.5.4) This scheduling algorithm is based on CPU power sharing, rather than CPU time sharing, as used in O(1) scheduling.

CFS utilises "Red-Black" trees in order to map the execution order of queued processes. The process at the leftmost node in the tree represents the process which is to be allocated resources, with new processes being added to the right of this node. The tree is traversed, with each process being given a predetermined amount of time. This use of red-black trees allows for the removal of the arrays used for tracking processes in O(1) scheduling[3].

1.3 Group Scheduling

Group scheduling was added to the CFS algorithm in kernel version 2.6.24. This update accounts for the "multi-user" aspect of Linux, dividing the CPU power evenly across the users, regardless of the number of processes each user is executing[1].

1.4 Recent Updates

The CFS mechanism, as with the entire Linux kernel, is being updated for maintenance or improvements. The most recent updates have included reducing the "maximum time per-request". This modification, which is expected to be implemented in kernel version 5.7.x, reduces the average per-request performance to 10ms, and the worst-case timings to 21ms[4].

2 Wi-Fi Support in Linux

Wireless networking in Linux is essentially broken into three sections, these sections, as discussed in detail below, are the userspace, kernel space, and device driver level. Each processes data at different stages of transmission. The userspace deals with the user applications, and also with some management processes. The kernelspace deals with network sockets, and network protocols. On the management side it deals with MAC implementation. Finally, at the device driver level, the information of higher levels is combined and placed in a queue for transmission[5].

2.1 Userspce

As previously mentioned, in userspace, the user application data is found. This is also where management services are found. These services include the WPA Supplicant, nl80211 configuration utilities, and access point authenticators[5].

WPA Supplicant supports both WPA and WPA2, and is run as a background process, controlling wireless connections. wpa_supplicant, the implemented Linux kernel process, can be configured using a text configuration file on the users machine. This process controls aspects of the wlan driver, such as the roaming and 802.11 authentication and association[6].

“iw” is a configuration utility for nl80211, which is a netlink interface, providing management between the kernel and user spaces. iw can be used to connect to an access point, modify transmission configuration such as transmission bitrates or power, and get the status of the current network link[7].

“hostapd” is an authenticator for 802.11AP and 802.1X/WPA/WPA2/EAP/RADIUS. It is used to configure the wireless infrastructure, including authenticating clients, and setting encryption keys. hostapd provides authentication and encryption settings for MAC filtering, ignoring broadcast packets, enabling WPA/WPA2, and key manage-

ment algorithms for WPA/WPA2[8].

2.2 Kernel Space

In the kernel space, there are three main services which must be explored. These services are nl80211, cfg80211, and mac80211[5].

"nl80211" is a netlink interface, which, when used in conjunction with with cfg80211 replaces the previously used "Wireless-Extensions"[9]. It is used as an interface between the user and kernel space, and is used to configure cfg80211.

"cfg80211" is an API used for configuring 802.11 on Linux. cfg80211 is used when writing wireless drivers for "fullmac" devices[10]. The device must be "registered" with cfg80211 in the driver before use. Both cfg80211 and nl80211 are required in Linux wireless drivers[11].

For "softmac" devices, "mac80211" is used. mac80211 is used for frame management, and supports the IEEE 802.11a/b/g/n/d/s/r standards[12]. mac80211 is implemented in many of the commonly used device drivers for a number of vendors, including Broadcom and Intel[13].

2.3 Device Driver

As previously mentioned, a device must be registered with the "cfg80211_ops" (nl80211 and cfg80211) system within the driver before use. The device driver is responsible for the lower layer operations, including the queueing and transmission of the data[5]. The device driver receives information from the mac80211, and is converted for transmission. Device drivers must be written in a way that is specific to the device hardware, and must contain an implementation of the mac80211, cfg80211 and nl80211 systems.

3 IPv6 vs. IPv4

There are a number of differences which have been made with the implementation of IPv6. One of the most notable is the change from 32 bit IP addresses to 128 bit IP addresses. This change supports and increase in the total number of addresses from 2^{32} addresses to 2^{128} addresses[14]. This increase allows for the limiting of routing table sizes, and helps to deal with the exhaustion of available IPv4 addresses[15]. The Internet of Things will benefit greatly from this increase in available addresses, as all connected devices can be given unique addresses.

Categorization of addresses has changed from the IPv4 model of unicast, multicast, and broadcast, to the IPv6 model of unicast, multicast, and anycast. With broadcast, all nodes on a network receive the sent packets, even those who do not require the message. This creates load on the network. Anycast sends a packet to the closest/least expensive node of a group of nodes with the same destination address in the routing table[16].

3.1 IPv4 vs IPv6 Functions

There are a number of functions used in the Linux kernel for the IPv4 protocol. In receiving a packet, the ip_rcv function checks the addressing of the incoming packet, and performs a check on the IP header, length, version, and checksums. The ip_rcv_finish function then uses the ip_route_input function to perform routing of the packet. From here, one of the following functions is called: a) ip_local_deliver: The packet is destined for the local host and must be passed to the transport layer for further processing. b) ip_forward: The packet is for another destination and must be forwarded. c) ip_mr_input: The packet is a multicast packet. Any included IP options are processed using the ip_rcv_options function[17].

For sending data, the ip_queue_xmit function sets the destination of the packet. The routing cache can be queried using the ip_route_output_keys function to find any

cached routes to the destination. This function is called via the ip_route_output_ports and ip_route_output_flow functions, which check the security of the packet. The ip_que_xmit function then creates the header for the packet and the ip_finish_output function sends the packet. Fragmentation can be performed if required using the ip_fragment function[17].

The ip_fragment function has no parallel in IPv6, as there is no fragmentation in IPv6 at intermediate nodes. Fragmentation can only happen at the original source of the data[18].

There are, however, a number of functions which do have parallels between IPv4 and IPv6. For receiving packets, IPv6 has ipv6_rcv function. This code performs many of the same functions as in IPv4, including error checking, packet length checking, and version checking[19].

4 6LoWPAN and 802.15.4

4.1 802.15.4

The IEEE 802.15.4 standard gives specification for Low-Rate Wireless Personal Area Networks (LR-WPAN). It defines specifications for Layer 1 and 2 protocols (Physical and Data Link layers respectively)[20]. This standard aims to give a platform on which higher level protocols can be based. The specification allows for data rates as low as 20kbps, and is intended for use on low-power devices, allowing for extended battery life, due to the low power consumption for data transfers[21].

The initial release of this standard occurred in 2003, giving specifications for two physical layer implementations, one lower frequency band, and one upper, with the upper frequency band being 2.4GHz. Updates have included improvements in the available data rates, modulation schemes, the addition of ultra-wide band frequency ranges, MAC support, and, with the latest release (802.15.4g), physical layer implementations for "smart neighbourhood networks"[21].

There are a number of higher level protocols which make use of the 802.15.4 standard, including MiWi, 6LoWPAN, WirelessHART, and ISA100.11a. Each of these protocols uses the standard for different purposes, such as mesh networking or manufacturing automation. One of the most well known of these protocols, Zigbee, uses the original 802.15.4 specification, however, due to it's non-open source implementation, it is assumed that there will not be an implementation added to the Linux kernel[20].

IEEE 802.15.4 offers data rates up to 250kbps, with power consumption as low as 17mA during transmission, and 19mA during receiving. These specifications offer advantages for wireless sensors, and other devices, making it a good choice for the establishment of Personal Area Networks[20].

4.2 6LoWPAN

6LoWPAN is an implementation of IPv6 over Low-Power Wireless Personal Area Networks. The ability to implement IPv6 on low power devices allows for the inclusion of these low power devices in Internet of Things networks. 6LoWPAN is a layer 3 protocol, and specifies fragmentation, compression, and meshing implementations required for IPv6 using the 802.15.4 standard[20][22].

Implementing 6LoWPAN on small connected devices increases the maintainability of these devices, and their security implementations. Standard network administration tools such as ssh can be utilised by engineers in order to connect to these devices remotely[20].

A number of IoT operating systems have support for 6LoWPAN, including TinyOS, Contiki, and RIOT[22]. These operating systems support the standard to various degrees, with many implementations still being under developement.

References

[1] C. S. Pabla, *Completely fair scheduler*, Aug. 2009. [Online]. Available: https://www.linuxjournal.com/node/10267.

[2] I. Molnar, *[patch] modular scheduler core and completely fair scheduler [cfs]*, Apr. 2007. [Online]. Available: https://lwn.net/Articles/230501/.

[3] *Cfs scheduler*. [Online]. Available: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt.

[4] M. Larabel, *Linux cfs improvement forthcoming to help with faster spreading of cpu utilization*, Mar. 2020. [Online]. Available: https://www.phoronix.com/scan.php?page=news_item&px=Linux-CFS-Spread-Util-Better.

[5] F. Chou, *Linux wireless networking: A short walk*, Mar. 2015. [Online]. Available: https://www.linux.com/training-tutorials/linux-wireless-networking-short-walk/.

[6] *Linux wpa/wpa2/ieee 802.1 supplicant*. [Online]. Available: http://w1.fi/wpa_supplicant/.

[7] *Linux wireless - about iw*. [Online]. Available: https://wireless.wiki.kernel.org/en/users/documentation/iw.

[8] *Linux wireless - about hostapd*. [Online]. Available: https://wireless.wiki.kernel.org/en/users/documentation/hostapd.

[9] *Linux wireless - about nl80211*. [Online]. Available: https://wireless.wiki.kernel.org/en/developers/documentation/nl80211#dokuwiki__top.

[10] *Linux wireless - about cfg80211*. [Online]. Available: https://wireless.wiki.kernel.org/en/developers/documentation/cfg80211.

[11] *Cfg80211 subsystem¶*. [Online]. Available: https://www.kernel.org/doc/html/latest/driver-api/80211/cfg80211.html.

[12] *Linux wireless - about mac80211*. [Online]. Available: https://wireless.wiki.kernel.org/en/developers/documentation/mac80211.

[13] *Linux wireless - existing linux wireless drivers*. [Online]. Available: https://wireless.wiki.kernel.org/en/users/drivers.

[14] *Comparison of ipv4 and ipv6*. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rzai2/rzai2compipv4ipv6.htm.

[15] *The recommendation for the ip next generation protocol*. [Online]. Available: https://tools.ietf.org/html/rfc1752#page-7.

[16] *Unicast, broadcast, multicast, anycast ipcisco*. [Online]. Available: https://ipcisco.com/lesson/unicast-broadcast-multicast-anycast/.

[17] F. U. Khattak, "Ip layer implementation of linux kernel stack", [Online]. Available: https://wiki.aalto.fi/download/attachments/70789059/linux-kernel-ip.pdf.

[18] *Ipv4 vs ipv6: What's the difference?* [Online]. Available: https://www.guru99.com/difference-ipv4-vs-ipv6.html.

[19] M. Collier, "Ee562 - network stack implementation", 2020.

[20] A. Ott, *Wireless networking with ieee 802.15.4 and 6lowpan*, Nov. 2012. [Online]. Available: https://elinux.org/images/7/71/Wireless_Networking_with_IEEE_802.15.4_and_6LoWPAN.pdf.

[21] *Ieee 802.15.4 standard: A tutorial / primer*. [Online]. Available: https://www.electronics-notes.com/articles/connectivity/ieee-802-15-4-wireless/basics-tutorial-primer.php.

[22] S. Schmidt, *Linux-wpan: Ieee 802.15.4 and 6lowpan in linux*, Mar. 2017. [Online]. Available: http://connect.linaro.org.s3.amazonaws.com/bud17/Presentations/BUD17-120%20-%20Linux-wpan-%20IEEE%20802.15.4%20and%206LoWPAN%20in%20the%20Linux%20Kernel.pdf.