Operating Systems

Gökçe Aydos

Lecture

Goals

- know the tasks of OSes
- know file-, user-, and process-management
- know how to work with files, paths, and file permissions
- know the tasks of device drivers

Prep

- which OSes are popular?
- what do you expect from your OS?
- imagine that you use your smartphone without an OS? How would it work?

OS Level

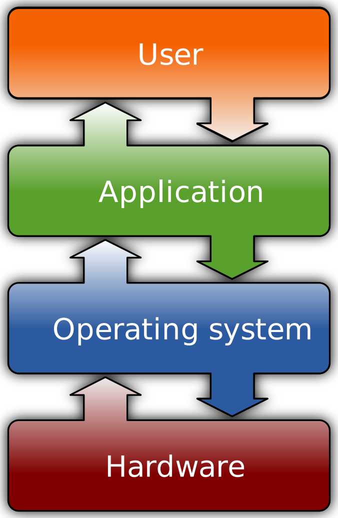


Figure 1: Golftheman [CC BY-SA 3.0]

Loving & hating OSes

OS is like a bureaucratic organization

- it is not productive

Loving & hating OSes

OS is like a bureaucratic organization

- it is not productive
- it makes us angry

Loving & hating OSes

OS is like a bureaucratic organization

- it is not productive
- it makes us angry
- but w/o it nothing works.

Definition

- *operation system* abbreviated as *OS*
- a platform for running application software
- applications mostly run on operating systems
    - some low-resource computers (e.g., embedded systems) work without operating systems

Examples

Android, iOS for smartphones

Windows, macOS, Linux for PCs, laptops, servers

Most popular OS

what is the most popular OS?

- in Germany
- in India
- worldwide

OS Tasks

mainly resource management, e.g., management of:

- files
- users
- processes

Discussion - OS tasks

do you have other other OS tasks in mind?

OS Tasks II

hardware abstraction

- *copy these two files to this directory*
- *I do not care how you do it*
- you copy the file the same way from a USB-drive and digital camera

OS classification

functional perspective:

- OS for embedded devices, e.g., bike computer, smartwatch
- for PCs, e.g., Windows and Linux
- OS for a mainframe

OS classification II

origins:

- from Unix: Linux, MacOS, Solaris
- from MS-DOS: MS-DOS, DR-DOS, Windows
- standalone: PalmOS, BeOS, IBM OS/2

System software

- software which is *not application software*
  - provides a platform for running application software
  - e.g., OS, device drivers, BIOS, game engine
- system software cannot be uninstalled without affecting application software, but application software can

Windows

- most popular OS on PCs
- most office programs run on Windows
- nowadays Windows 10 on most PCs

Windows 11 or 12?

- probably there will be no Windows 11 or 12 soon
  - Windows releases an incremental update every six months
  - e.g., 1903, 1909,...

Linux (OS)

- a family of open source *Unix-like* OS based on the *Linux kernel*, e.g.
  - free: Ubuntu, Debian, CentOS, openSUSE
  - commercial: Red Hat Enterprise Linux, SUSE Linux Enterprise
- Unix-like, but not Unix
  - Linux-is-not-Unix

Discussion - Linux popularity

- did you try Linux before?
- is it convenient compared to Windows or MacOS?
- why is Linux so popular among scientists and computer guys?

File management problem

- we have different kinds of files, e.g., text, picture, movies, programs
- with different sizes

How can we organize these files and access them fast?

File management solution

- store them as byte sequences
- give them names
- create a directory tree for organization

Files

- a *file* is a sequence of bytes
- smallest file can be 0 Bytes and max. file size many GBs
- typical naming *name.suffix*
- when working with shell, users typically avoid spaces in filenames

Directories

- directories can contain:
    - directories
    - files (including links or shortcuts)
- the beginning of the directory tree is called *root*

Directory tree Linux

/

- home
    - goekce
        - diary.txt
    - charu
        - 20200912_scan.pdf
- etc
    - hostname

Directory tree Windows

C:\

- Users
- Program Files
- Windows

File path

the address of a file on your computer, e.g.:

```
/home/goekce/diary.txt
```

Absolute vs relative paths

- absolute: `/home/goekce/diary.txt`
- relative: `../../elephant.jpg`
- `..` means *jump one directory higher*

Exercise - file paths

which path is described by
`/usr/../home/tantau/../../dev/null`?

Relative paths

- relative paths help us to conveniently access files which are nearby
- `..` means one level higher
- `.` means this directory

Relative paths example

- you want to open slides which are on:

  ```
  /home/charu/uni/WS2021/iti/slides/
  ```

- instead of:

  ```
  pdfviewer /home/charu/uni/WS2021/iti/slides/1.pdf
  pdfviewer /home/charu/uni/WS2021/iti/slides/2.pdf
  ```

- change the directory and open them w/o the long path:

  ```
  cd /home/charu/uni/WS2021/iti
  pdfviewer slides/1.pdf
  ```

Exercise - rel. paths

you are on `/home/charu/slides/text`.

- which file is addressed by `../img/paneer.jpg`
- which file is addressed by `../../../goekce/exams/../../charu/plan.txt`

Discussion - abs./rel. paths

when would you prefer an absolute/relative path?

User management - goals

- many users should be able to store their data on a single system
- public files should be readable by users, private files not

User management - solution

- every user has a private directory
- access rights for every file
- OS identifies the users by their username and password (authentification)

File rights in Unix

every file belongs to:

- a `user`. Generally `user` is the creator of the file.
- a `group`. `users` can belong to various groups.

Example:

```
$ ls -l diary.txt
-rw-r--r-- 1 charu students 498660 Sep 12 08:51 diary.txt
```

User & group

- only the `user` can change its rights who the file belongs to
- `group` has the permissions in file file attributes listed with `ls -l`

File permissions in Unix

three permissions:

1. read *r*
2. write *w*
3. execute *x*

File perm. - example

```
-rwxr--r-- 1 u u 498660 Sep 12 08:51 search.sh
```

	read	write	execute
user	yes	yes	yes
group	yes	no	no
other	yes	no	no

Exercise - file perm.

file	user	group	others
apple.txt	eva	eden	rwxr-----
snake.txt	adam	eden	rw-rw-r--
eat	adam	men	rwxrwxr-x

which files can Adam and Eva write/read?

Exercise - file perm. II

go to your home folder in Linux and look at the permissions of the files. Who can read, write to your files?

Process management - goals

- users should be able to start multiple processes in parallel
- if a process does evil things, the system must be able to stop it
- a user should only be able to stop their process
- the system must be able to prioritize crucial processes

Process management - solution

- the OS implements a *process list*
- every process gets an *id* and belongs to a user
- (only) the user can stop/kill their process
- the OS can give different priorities to processes

OS layers - motivation

example: print job management

1. management of print jobs
2. communicating with different models of printers

How do we achieve this versatility?

OS layers - solution

```
.
   applications
        |
HW independent mgmt. <
        |            < OS
    HW driver        <
        |
       HW
```

Below - driver

a *driver* is a program for controlling specific hardware

- e.g., a desktop printer needs different instructions than a big office printer.
- a printer driver is generally not involved in print queue management, e.g., cancel a file in print queue

Above - shell, system calls

- users interact with OS using:
  - *shell*
  - GUI, e.g., Start -> File Manager
- programs interact with OS using:

- shell

Above - shell, system calls

- users interact with OS using:
  - *shell*
  - GUI, e.g., Start -> File Manager
- programs interact with OS using:

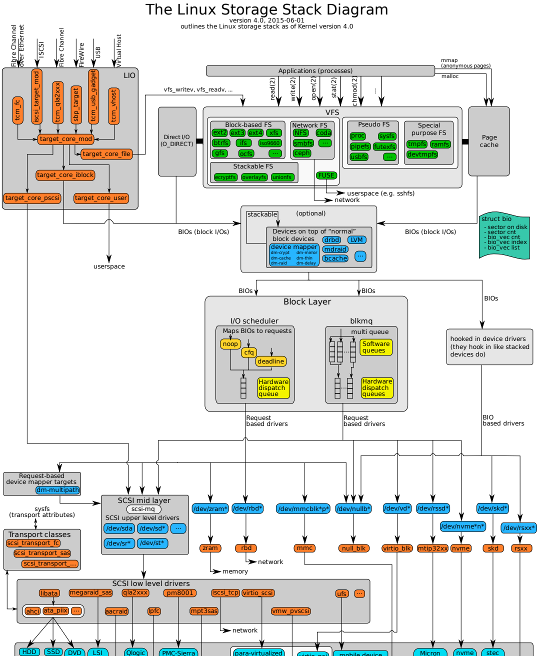
- shell
- *system calls*

Summary

- OS manages resources, e.g., hardware, processes, files, users
- files have a name, permissions, a user, a group, size
- directory tree
- access permissions `rwx`
- permissions for `user` `group` `others`

Appendix

Abstraction example — Storage



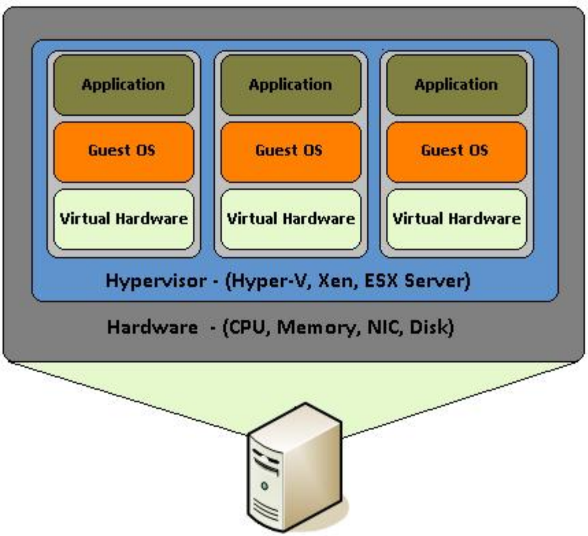
The Linux Storage Stack Diagram
version 4.0, 2015-06-01
outlines the Linux storage stack as of Kernel version 4.0

Virtualization

- *creating a virtual, rather than actual version of something*

Hardware Virtualization

- e.g., Virtualbox
- host vs guest system



Desktop Virtualization

- working directly on a remote server
- e.g.,
  - *remote desktop*
  - JupyterHub
  - SSH connection (remote command-shell)

Ex. Thin client

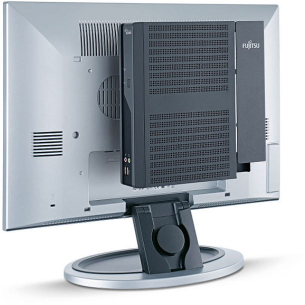


Figure 4: VIA Gallery from Hsintien, Taiwan [CC BY 2.0]

Thin vs Thick client

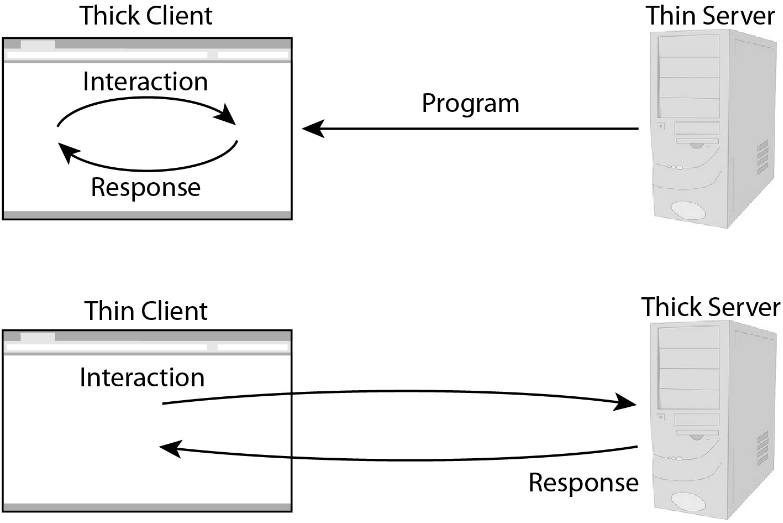


Figure 5

Thin vs Thick client II

- thin clients rely on a remote server
  - e.g., ChromeOS, web browser
- easy to administer
- less hardware resources
  - cheaper than a usual PC
- depends on a fast network connection
- data mostly stored on servers

UNIX & Unix-like OS

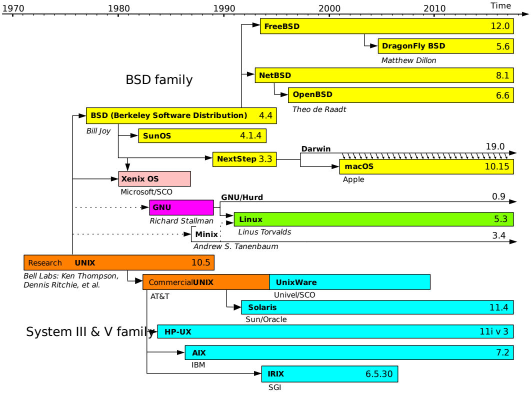


Figure 6: Guillem, Wereon, Hotmocha [Public domain]

Unix

- a *family* of OS that evolved from the works in the 1970s at the Bell Labs
- *UNIX*
  - is a specification (standard) for an OS, is a trademark
    - e.g., an Unix OS *must include* awk, cd, ls
  - e.g., macOS, z/OS
  - requires certification by a consortium

Unix-like OS

- OS is based on the Unix
- e.g., Linux, Android (Android is also based on Linux)
- an *Unix-like OS* behaves like a Unix system, but is not certified

Unix characteristics

- *Unix philosophy*
- modular design
- a unified filesystem, i.e., /a/b/c
- portable (written in C)

GNU Project

- 1983, free software project
- alternative to proprietary Unix
- software should be freely
  - run
  - copied
  - studied
  - modified

Figure 7: Aurelio A. Heckert [CC BY-SA 2.0]

- *GNU is not Unix!*

GNU Project II

- goal was to build a free OS
  - kernel
  - software tools, e.g., awk, sed
- GNU kernel was not successful
  - instead *Linux kernel*

GNU Project III

- GNU tools are used in most Linux OS
  - *gawk* → GNU-awk
  - generally it does not matter if you run `awk` or `gawk`
- meaning of *Linux* nowadays
  - a *Linux distribution*, e.g., Ubuntu

Linux distribution

- short: *distro*
- a software collection **based on the Linux kernel**
  - cf. Anaconda — a Python distribution
- typically comprises
  - Linux kernel
  - GNU tools, libraries
  - additional software
  - documentation
  - graphical interface

Graphical interface

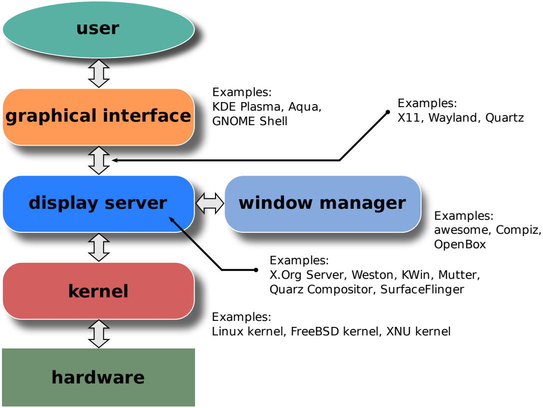


Figure 8: Shmuel Csaba Otto Traian [CC BY-SA 3.0]

Desktop environments

- implements the *desktop metaphor*
  - graphical shell for the OS on PCs
  - easily access and edit files
- e.g., Fluent (Win10), Aqua (macOS), Unity (Ubuntu), KDE, GNOME, LXDE
- command shell is used for advanced operations

Discussion - desktop env. ----------------------

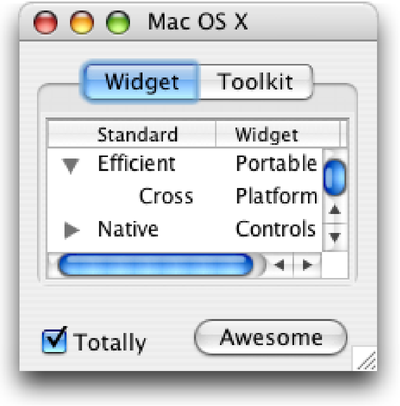
what should a desktop environment provide?

Desktop env. features

- a desktop + window system
- interaction using mouse and keyboard
- status bar, file manager, start menu, text editor
- a toolkit to program your own GUIs

Graphical widget

- an element of interaction with OS
  - *window gadget*
  - e.g., *OK button* analogy to push-buttons on physical devices



Standard	Widget
▼ Efficient	Portable
Cross	Platform
▶ Native	Controls

☑ Totally

Figure 9

Example

```
zenity --question --text="Cancel the class?"
if [ $? -eq '0' ]; then exit; fi
```

Different OS versions

- Ubuntu server has four different versions
  - e.g., x64, ARM, PowerPC, IBM Z
- these versions resemble different processor architectures
  - each processor architecture has an instruction set
    - e.g., x64: *add two 64 bit numbers*
    - x86 does not have the former instruction

32- vs 64-bit OS

- 64-bit OS is designed for 64-bit processors
    - nowadays most processors on PCs and smartphones have 64-bit processors
        - so the operating systems are also 64-bit
    - an 64-bit processor can handle 64-bit in each clock cycle

32- vs 64-bit applications

- even most OS nowadays are 64-bit, some older applications do not have 64-bit versions
  - e.g., look at C:Program Files (x86)
  - fortunately, an 64-bit OS can also run 32-bit applications

Long term support vs latest features

- *long term support release*
  - does not get updated regularly
    - removes the need for frequent software migration
  - mostly security updates and crash fixes
  - useful for long term projects
  - e.g., Firefox ESR, Ubuntu LTS
  - you do not get the latest features

OS Components

- kernel
- networking
- security
- user interface (shell)

Kernel

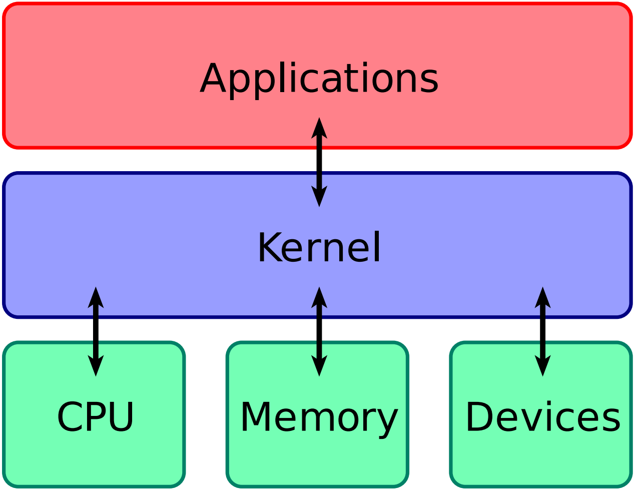


Figure 10: Bobbo [CC BY-SA 3.0]

Kernel functions

- orchestrates access to hardware
  - e.g., CPU, memory, devices
  - which program is allowed to use the processor right now?
- controls communications between different running programs (processes)
  - e.g., program a wants to send data to program b
  - kernel can also prohibit this access
  - kernel is like the housekeeper for programs

Kernel functions II

- manages memory
    - programs do not have to know how much RAM exists
- abstraction
    - programs see a file system but not directly your hard-disk or SSD
- has a consistent API for application software
    - rarely changes that existing programs run for a long time

Kernel functions III

- manages *device drivers*
  - *device driver* is a software for controlling a specific hardware
  - the Linux kernel typically contains modern hardware drivers
  - Windows 10 can automatically install them, or you have to install them manually

Linux (kernel)

- *a kernel* written by Linus Torvalds in 1991
    - free alternative to the kernel in Unix-based systems
    - a good example for open source collaboration at the beginning of internet
    - *Linux-based* OS—if the OS uses Linux kernel
- used in
    - PC, servers, smartphone
    - WLAN routers, TVs

Other kernels

- there is not only Linux, e.g.,
  - FreeBSD kernel
  - NetBSD kernel
  - Solaris kernel
  - Windows NT kernel

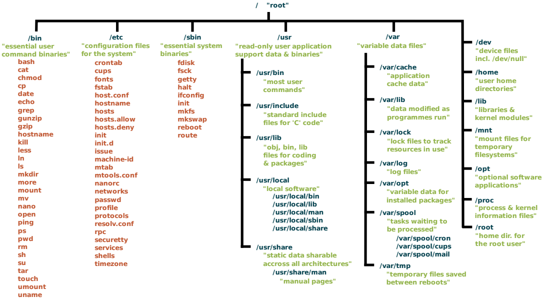
Question

- open a command-line on your Linux, and go to the root directory:

  ```
  ls /
  ```

- what could be these directories for?

Unix filesystem



/ "root"

/bin
"essential user command binaries"
bash
cat
chmod
cp
date
echo
grep
gunzip
gzip
hostname
kill
less
ln
ls
mkdir
more
mount
mv
nano
open
ping
ps
pwd
rm
sh
su
tar
touch
umount
uname

/etc
"configuration files for the system"
crontab
cups
fonts
fstab
host.conf
hostname
hosts
hosts.allow
hosts.deny
init
init.d
issue
machine-id
mtab
mtools.conf
nanorc
networks
passwd
profile
protocols
resolv.conf
rpc
securetty
services
shells
timezone

/sbin
"essential system binaries"
fdisk
fsck
getty
halt
ifconfig
init
mkfs
mkswap
reboot
route

/usr
"read-only user application support data & binaries"
/usr/bin
"most user commands"
/usr/include
"standard include files for 'C' code"
/usr/lib
"obj, bin, lib files for coding & packages"
/usr/local
"local software"
/usr/local/bin
/usr/local/lib
/usr/local/man
/usr/local/sbin
/usr/local/share
/usr/share
"static data sharable accross all architectures"
/usr/share/man
"manual pages"

/var
"variable data files"
/var/cache
"application cache data"
/var/lib
"data modified as programmes run"
/var/lock
"lock files to track resources in use"
/var/log
"log files"
/var/opt
"variable data for installed packages"
/var/spool
"tasks waiting to be processed"
/var/spool/cron
/var/spool/cups
/var/spool/mail
/var/tmp
"temporary files saved between reboots"

/dev
"device files incl. /dev/null"
/home
"user home directories"
/lib
"libraries & kernel modules"
/mnt
"mount files for temporary filesystems"
/opt
"optional software applications"
/proc
"process & kernel information files"
/root
"home dir. for the root user"

Figure 11: Ppgardne [CC BY-SA 4.0]

Shell kernel metaphor

Figure 12: Potkettle [CC BY 3.0]

Shell

- required if human-interaction needed
  - command-line interface
  - graphical user interface

Security

- authentication
  - who is the user?
  - e.g., a normal user or administrator
- authorization
  - what is the user allowed to do?
  - e.g., the right to install programs

Networking

- OS implements networking functions
- open networking protocols
  - e.g., Windows can communicate with Linux OS through the *Internet Protocol*
- vendor-specific protocols
  - e.g., *Server Message Block (SMB)* from Microsoft for shared access to files, printers

Firmware

- device-specific software for an embedded device
  - does not get updated very often
  - e.g., your computer's BIOS
  - the software that you flash to your Arduino board
  - the software for your car's brake controller

Figure 13: Public Domain

- cf. OS, which gets updated regularly

Files

- computer resource for recording data discretely
    - e.g., text, photo, computer program
    - on Unix-like systems can be also a device or virtual resource
        - e.g., /dev/sdb, /dev/null
- *paper* analogy
- file format
    - based on *file extension* on Windows
    - based on the file signature on Linux

File corruption

- if a file cannot be properly read
- when can it happen?
  - an image editing program crashes while saving the image file
  - removing a USB stick before unmounting
  - physical damage
  - aging of the disk

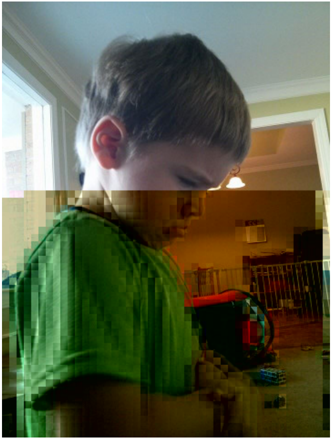


Figure 14: Jim Salter [CC BY-SA 4.0]

Task

- a unit of work for a computer
  - also called: process, thread
- multitasking, multiprocessing, multithreading

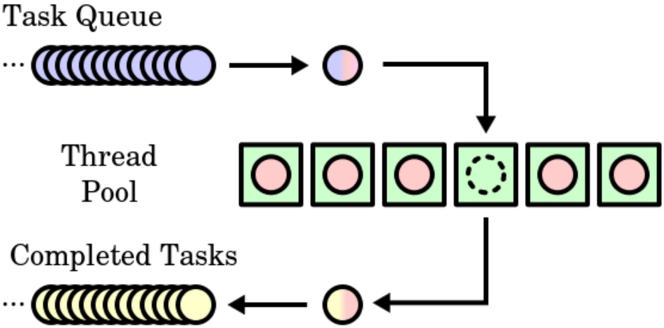


Figure 15: I, Cburnett [CC BY-SA 3.0]