Operating system components

- ellipse at 3.740,7.547 wid 1.181 ht 0.394 ellipse at 3.740,7.547 wid 1.969 ht 1.181 ellipse at 3.740,7.547 wid 3.150 ht 1.969 ellipse at 3.740,7.547 wid 4.724 ht 3.150 line from 3.740,9.122 to 3.740,8.531 line from 1.378,7.547 to 2.165,7.547 line from 5.315,7.547 to 6.102,7.547 line from 3.740,6.563 to 3.740,5.972 line from 2.756,6.760 to 2.257,6.340 “HW” at 3.648,7.528 ljust “Shell” at 3.543,8.329 ljust “Device drivers” at 3.228,7.869 ljust “daemons” at 4.331,8.565 ljust “applications” at 2.165,8.565 ljust “utilities” at 1.772,6.990 ljust “commands” at 2.953,6.203 ljust “compiler” at 4.724,6.596 ljust “Kernel” at 3.386,7.148 ljust

Operating system components

- compiler
  - `gcc`, `g++`, `gm2`, `gjc` and `gpc`.
- utilities
  - `emacs`, `vi`, and `fsck`.
- commands
  - `cp`, `mv`, and `tar`.
- applications
  - gnome, X windows, kde.
- shells
  - `bash`, `sh`, `csh`, `ksh` and `tsch`.

Further reading

- Mike Gancarz, “Linux and the UNIX Philosophy”, ISBN 1-55558-273-7, 2003
- Karl Fogel, “How to Run a Successful Free Software Project”, ISBN 0-596-10759-0, 2005

Modern Operating system requirements

- must exploit parallelism
- needs to be extremely portable

Very Brief Implications UNIX History

- Ken Thompson of AT&T invented UNIX in 1969 and it ran on a PDP-7
  - UNIX was based on Multics
  - one motivation for writing UNIX was to run a program called "space travel"
- Thompson borrowed ideas from Multics
- *Good programmers write great software; great programmers "steal"*
- notice how Thompson avoided the "not invented here" syndrome

Very Brief Implications GNU/Linux History

- split into two sections
  - the Linux kernel (which bridges the hardware/software interface)
  - and GNU utilities, compilers, shells, daemons, applications

Very Brief Implications Linux History

- Linus Torvalds a Finnish student at the University of Helsinki wrote Linux
- famous posting to `comp.os.minix` on 25 August 1991 read:
- ```
  Hello everybody out there... I'm doing a (free) operating system
  ```
- sealed his fate..
- initially Linux was not designed for portability
  - but only to run on the 386 architecture
  ```

Very Brief Implications Linux History

- started out as “for fun” to run `bash` and `gcc` on his “toy” operating system
  - Linus started by utilising the file system code of Minix in Linux
- later on he found that good design principles led the way to portability for purity’s sake alone
  - other people helped porting Linux to other architectures
- Linux initially used components of Minix (which was later totally replaced)
  - it also heavily used the `gcc` compiler extensions (such as assembly language in macros and header files)

Very Brief Implications Linux History

- again avoided the NIH syndrome, certainly “stole” the ideas of `UNIX`
  - pipes, sockets, files, IO etc.

GNU History

- the GNU project started in 1984 to develop a complete UNIX like operating system which is free software
  - “free” as in speech and but not as in beer
- Richard Stallman started the GNU project and began work upon `gcc`, `emacs` and `gdb`.
- all GNU software is licenced through the GPL

GNU History

- in the early days the each software component had to be complete written from scratch
  - so that it could be completely owned by the GNU foundation

- more recently, almost nothing is written from scratch anymore
  - as each new software component borrows heavily from other GNU software components

GPL designed to uphold Free Software

- it provides
- the freedom to run the program, for any purpose (freedom 0).
- the freedom to study how the program works, and adapt it to your needs (freedom 1)
  - access to the source code is a precondition for this.
- the freedom to redistribute copies so you can help your neighbour (freedom 2)
- the freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3)
  - access to the source code is a precondition for this.

Software engineering implications of GPL

- liberally encouraged to “borrow” code from other projects
  - encouraged to re-factor borrowed code if you extend a project (rather than to duplicate code with minor variants)
- libraries are constructed at both the source level and also at the object level
  - tremendous productivity gain compared to closed source development
  - todays new GPL project can become tomorrows library of source code
- software engineers advocate code reuse - but often miss the most important component
  - source code reuse

UNIX Philosophy in a nutshell

- small is beautiful
  - small software programs are generally fast
  - they are often combined with other software programs in useful ways (often unseen by original author)
  - any advance in computer speed will have a noticeable effect on a small program
- make each program do one thing well
  - removes extraneous code, removes complexity and improves flexibility
- build a prototype as soon as possible
- choose portability over efficiency

UNIX Philosophy in a nutshell

- store data in flat text files
  - all configuration files are plain text
- use software leverage to your advantage
  - source code reuse
- use shell scripts where possible
  - try to avoid writing an equivalent C program
- avoid captive interfaces
- make every program a filter

UNIX and GNU/Linux culture/goals

- allow the user to tailor the environment
- make operating system kernels small and lightweight
  - not always achieved but certainly a goal
  - see GNU Hurd
- use lowercase and keep it short
- save trees
- silence is golden
- think parallel

UNIX and GNU/Linux culture/goals

- sum of the parts is greater than the whole
  - large applications are built from smaller programs (eg shell scripts)
- look for 90% solution
  - doing 90% of the solution is much easier and will satisfy 90% of the user base
- worse is better
  - inclusive and least common denominator is likely to survive
  - worse is likely to be cheaper and thus more popular
- think hierarchically

Software projects

- Fredrick Brookes in his land mark book “Mythical Man-Month, The: Essays on Software Engineering”, Anniversary Edition, 2/E, ISBN: 0-201-83595-9, 1995
  - defines three systems of Man

- man builds the first system with his back to the wall
  - no time to do it right
  - built by a small number of people
  - fuelled by excitement
  - first system is a “lean, mean computing machine”

Software projects

- second system of Man
  - built by so called experts
  - built by much design and attention
  - might capture the hearts and minds of millions of users

- experts sometimes exact revenge on the basic idea and “improve” some of the basic algorithms of the first system
  - often fall into the NIH syndrome

- second system designed by a committee
  - overweight software which is slow
  - does not yield huge benefits from increases in compute power

Software projects

- second system is a success and failure
  - extra features which users *might* use
  - big, slow and bug ridden software

Third system of Man

- built by people who have been burned by the second system
  - it usually involves a name change from the second system
  - original concept intact and is regarded as obvious
- third system combines the best characteristics of the first and second system
- the designers of the third system are usually given time to do it right

GNU and Linux is both a third and second system

- GNU and Linux came along during 1984..1991 when UNIX went through a tumultuous second system period
- UNIX was of the 1980s was definitely a second system
  - slow (applications rarely ran faster than their 1970s counterparts..)
- bickering over UNIX standards
  - AT&T
  - Sun
  - Open Software Foundation (no relation to open source - or GNU ..)

GNU and Linux is both a third and second system

- people tired of bloat of UNIX are happy with GNU/Linux
- best ideas of UNIX are found in GNU/Linux
- much of the implementation is written correctly
  - developers have had the time to write code correctly
- it also uses a name change
  - UNIX to GNU/Linux

GNU/Linux as a second system

- arguably it is using a second system methodology
  - namely OpenSource or Free Software
- remember these generate much excitement in some areas..
- but it is now becoming formalised by many books
  - many conferences, symposiums, journals etc

Producing a successful free software project

- choose a good name
- have a clear mission statement
- state that licence terms unambiguously
- list the features and requirements
- clearly state the development status
- provide easy access to source code
  - download a tar.gz file using http or ftp
  - allow the git repository to be browsable

Release often/release early

- do not be afraid to release early
  - do not be afraid to release often

- Eric Raymond in his landmark essay The Cathedral and the Bazaar ⟨`http://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar`⟩ cites Linus Torvalds as sometimes releasing two kernels a day

Use software version control

- popular choices are git, svn
  - concurrent version system
- allows you to keep a project of many source files
  - each modification can be `push`ed to the `git` repository
- co-developers can obtain copies of the project at any time in its life
- developers may `clone` a complete git repository
  - examine changes between software revisions