(Rozšířená) Anotace 11GNU (2024)

Spousta lidí sice dnes třeba umí psát něco např. v Pythonu, MatLabu nebo jiných jazykách s vyšší abstrakcí, které jsou často úplně odtržené od základní abstrakce, kterou implementuje počítačový hardware jako takový. Tito lidé bohužel často vůbec netuší jak takový hardware, který pak tak jako tak musí všechno vykonávat, vlastně funguje (a kolikrát nevědí ani jak vypadá, protože dnes už je zvykem kupovat počítač jako black box, namísto stavění si ho po komponentách). Přitom ale aspoň základní znalost principů fungování počítače (od hardwaru přes operační systém až po aplikace) by měl znát každý, kdo se snaží trochu programovat a není mu úplně jedno jak neefektivně a pomalu jeho výtvor funguje a když už mu to jedno je, tak minimálně, aby si byl vědom proč tomu tak je.

Předmět 11GNU se snaží posluchači dát přehled o tom, jak vlastně takový počítač fyzicky i logicky vypadá a funguje, zároveň je i takovým rámcovým úvodem do Linuxu (a to z uživatelské, ale především i programátorské stránky), naučí posluchače pracovat s BASHem jako primárním Linuxovým shellem a zároveň exemplářem interpretovaného jazyka, který je ale podstatný pro práci v Linuxu jako takovém a také poskytne posluchačům úvod do kompilovaného jazyka C, jakožto univerzálního jazyka, jehož abstrakce se nejvíce podobá abstrakci samotného hardwaru a tudíž umožňuje psát nejefektivnější kód. Vše s primárním důrazem na to, aby posluchači byli případně schopni tyto (a další) nástroje používat pro zpracování především fyzikálních dat ze svých experimentů a simulací, například na našich fakultních clusterech (ale samozřejmě nejen tam, a nejen tato data).

L1: (12.2.2024)

- **Úvod**, fyzická ukázka **HW** (PC, MB, CPU, RAM, karty, ...) *[jak to vypadá uvnitř počítače, fyzicky i logicky*], komponenty, **sběrnice** (fyzická podoba, sériové vs paralelní, elektrická podoba, logická podoba, princip fungování, obecný příklad PCIE a okrajově i USB, hlavně co do rozdílů) [*propojuje komponenty, tudíž odtud začínáme, spousta vlastností dalších komponent se od toho odvíjí, a navíc se podle toho dá optimalizovat i kvantování dat v komunikaci našeho aplikačního kódu*]

L2: (14.2.2024)

- **Procesor**, jádro, vlákno (princip, komponenty, struktura, architektura), instrukce (příklady, jak to vypadá, minimální kód, co vlastně procesor dokáže), RISC/CISC, instrukční pipeline (principy fungování a organizace, scheduling), registr (obecně), bistabilní klopný obvod, binární a logická aritmetika: **Booleova algebra**, NAND hradlo, D klopný obvod [*základní stavební prvek registrů a rychlých statických pamětí a vlastně všech chipů, jak to funguje a jak to fyzikálně vypadá v reálu*], **registry** architektury x86-64.
- DÚ na procvičení Booleovy algebry s pomocí hradel [*bez pochopení, jak to je s těmi nulami a jedničkami se opravdu pořádně programovat prostě nedá a zároveň se dá nahlédnout, že s těmi fyzickými hradly se dá čarovat trochu efektivněji než s logickými operátory na papíře či v kódu ... není divu, že FPGA jsou často o tolik efektivnější než SW kód*]

L3: (19.2.2024)

- Pokračování registrů x86-64 ((E)Flags registr, principy používání flagů v aritmetice a branchingu), IEEE754 [*aneb jak vlastně vypadají ty floatové registry a jak vlastně počítač vidí desetinná čísla, je to potřeba vědět, abychom se vyvarovali chyb při počítání ve svém kódu*], **cache** (hierarchie, typy, principy, strategie, fungování na různých úrovních od L1 v procesoru až po diskové cache pásek na síťových úložištích [*rychlost vs kapacita vs cena - věčný boj, ale*

když to pochopíme, umožní nám to zefektivnit svůj kód, především, pokud pracuje s daty]), cache-line, FIFO (queue) / LIFO (stack)

L4: (21.2.2024)

- **RAM** (topologie/organizace, columns, rows, banks, ranks, CL, multi-channel, princip uchování informace, vyčítání a refresh buněk) [*je celkem podstatné pochopit, jak principiálně funguje hlavní operační paměť, abychom dokázali pochopit jednak jak k ní lépe a efektivněji přistupovat z našeho softwaru při práci s větším množstvím dat a také proč práce s ní je jiná než práce se statickou pamětí, která se používá na registry a vyšší cache*]

L5: (26.2.2024)

- **virtualizace paměti** (physical space, virtual space, page-fault, page-miss), **MMU**, (multi-level) page-table, memory mapping, TLB, adresové prostory, swap, shared memory [*princip virtualizace paměti je jedna z nejzásadnějších a nejpodstatnějších vlastností procesoru, která vůbec umožňuje provozovat multi-táskový operační systém a oddělit od sebe jednotlivé procesy, umožňuje také efektivně sdílet paměť a komunikovat mezi procesy, pracovat se soubory a zařízeními a spoustu dalšího*]

L6: (28.2.2024)

- I/O, in/out instructions, device space, MMIO, **IOMMU** [*virtualizace paměti umožňuje i efektivnější komunikaci se zařízeními připojenými na interní a potažmo externí sběrnice*], příklad přístupu ke zvukové kartě, grafická karta, **GPU**, APU [*princip přístupu, sdílení paměti a přístup k ní, transfer dat, nastavování hardwaru, princip fungování grafické karty v zobrazovacím režimu a ve výpočetním režimu*], **interrupts and exceptions** [*způsob signalizace a reakce na vnější a vnitřní podněty procesoru, příklady praktických použití*], **character devices** vs **block devices**, čtení/zápis blokových zařízení, file-mapping, file cache, file handle [aneb *jak operační systém pracuje se soubory*]

L7: (4.3.2024)

- **protection mode**, protection rings [*copak nám vlastně zajišťuje bezpečnost a opravdovou separaci jednotlivých procesů spolu s virtualizací paměti a co k tomu stačí, aby bylo v Ringu 3 zakázané, aby to fungovalo*], proces, vlákno, kontexty, **user-space** vs **kernel-space**, **context switching** [*v čem se liší proces a vlákno na úrovni OS, jak probíhá context switching a proč je tak časově náročné přepínání mezi user-space a kernel-space, při čem všem je potřeba to podstupovat a proč a jak je tudíž lepší se mu v aplikacích vyhnout, kdykoliv to jde*]

L8: (11.3.2024)

- **Bloková zařízení**, **disky**, FDD, HDD, SSD, NVMe, princip strukturování dat na mechanickém disku, sektor, C/H/S, LBA, Advanced Format, 4kn/512e [*přeci jen, když už si kupujeme mechanický disk (které pořád ještě dávají smysl pro jejich kapacitu a spolehlivost), tak je třeba se v těchto pojmech vyznat, nehledě na to, že do značné míry ovlivňují způsob, jak se s blokovými zařízeními obecně pracuje*], **RAID** (JBOD, 0, 1, 2, 3, 4, 5, 6 a kombinace) [*je dobré vědět k čemu se to používá a proč*]
- **Filesystemy**, soubor, adresář, FAT filesystem [*je sice zastaralý, ale pro určité účely stále používaný, a hlavně dostatečně jednoduchý příklad na to, aby se dala v krátké době vysvětlit jeho struktura a princip fungování, a tudíž demonstrovat příklad, jak přibližně fungují i jiné filesystemy a navíc na něm lze i ukázat co a proč je tam špatně a jak to ostatní novější*

filesystemy vylepšují a co mají oproti němu navíc za funkce a featury], journal, symlink, hardlink, b-tree, inodes, snapshots.

L9: (13.3.2024)

- **OS booting** sequence, MBR/GPT format, partition table, (U)EFI/Legacy, BIOS, [*hodí se vědět jak vlastně takové startování počítače a zavádění systému funguje, zvlášť, když se snažíme instalovat víc systémů najednou*]
- **Kernel architecture** (microkernel vs monoblock + modules) [*filosofický, bezpečnostní a výkonnostní rozdíl, každý systém volí trochu jiný přístup a všechno má své výhody a nevýhody*]
- Distribuce Linuxu (RHEL vs CentOS vs Fedora, Ubuntu, ...), packaging (RPM, DEB, ...), package managers: dnf, yum, apt, ..., Desktop environment (GNOME/GTK, KDE/QT, ...), [*Existuje mnoho různých distribucí Linuxu, tak je potřeba se v tom trošku vyznat, vědět v čem se liší a že v základu je to vlastně všechno podobné.*]
- **Virtualizace** (platformy, paravirtualizace, virtualizace na úrovni OS, aplikační virtualizace), container, ukázka virtualizace [*v dnešní době velmi používaná věc a abychom to uměli správně použít, je potřeba vědět, jak to asi funguje a jaké jsou výhody a nevýhody jednotlivých variant*]

L10: (18.3.2024)

- **Úvod do Linuxu**, Linux vs Windows user environment, Unix time, Windows time, EPOCH, právový model Unixu, file attributes, UID, GID, PID [*prostě základní věci, které by asi měl vědět každý uživatel, který začíná s Linuxem*]
- **Filesystem Hierarchy Standard** (**FHS**) [*tohle je sice tak trochu nuda, ale většina Unixových systémů to používá a každý uživatel, natož programátor by měl alespoň rámcově vědět proč je to uspořádáno tak, jak je to uspořádáno, jaké to potenciálně přináší výhody a smysl, abychom to případně dokázali náležitě využít, prostě základní znalost*]

L11: (20.3.2024)

- **Rozdělení jazyků** (kompilované vs interpretované, imperativní vs deklarativní, dynamicky typované vs staticky typované, silně typované vs slabě typované) + ukázky a příklady, překladač (compiler), interpreter, JIT kompilace, zdrojový kód, skript, ...
- **Princip překladače** (s ukázkou konkrétní aplikace na GCC, fáze překladu) [*Je především podstatné pochopit, jak funguje překladač a jaký je rozdíl mezi překladačem a interpreterem*]

L12: (25.3.2024)

- Pokračování principu překladače (Gimple, RTL (Register Transfer Language), optimalizace, ukázky a příklady, compiler frontend and backend) [*Jak to vlastně principiálně funguje (s konkrétními skutečnými příklady), jak se tam dělají optimalizace na různých úrovních a proč může někdy překlad trvat i poměrně dlouho*], ukázky některých konkrétních parametrů GCC [*které nám ukáží jednotlivé fáze překladu a vnitřní struktury na kterých se pak dělají optimalizace*].

L13: (27.3.2024)

- Dokončení principu překladače (LTO (Link Time Optimization), PGO (Profile Guided Optimization))
- **Princip interpreteru** [*hodně důležitá věc, zvlášť s rostoucí oblibou využívání interpretovaných jazyků, je potřeba pochopit, jak to vlastně funguje, v čem se to liší od kompilovaných jazyků, v*

čem je výhoda a nevýhoda, proč interpretované jazyky nikdy nemohou výkonem ani z daleka konkurovat těm kompilovaným]

L14: (3.4.2024)

- přihlášení k Linuxu [*Když chceme s Linuxem začít pracovat, je dobré vědět, jak se k němu vzdáleně přihlásit a případně jak vzdáleně kopírovat soubory, ať už z jiného Linuxu anebo z Windows*], základní pomocné nástroje a příkazy pro práci v textovém terminálu, mc, mcedit
- Vim: základní princip, základní používání v rámci psaní programů a skriptů [*jasně, v dnešní době hodně lidí rádo používá různá IDE (typu Microsoft Code Studio), která dokáží dělat spoustu věcí za vás a různě vám pomáhat, ale přeci jen ne vždy a všude musí být tyto nástroje k dispozici a vůbec neuškodí, když člověk sám převezme kontrolu nad svým kódem a obejde se i bez těchto nástrojů, protože jinak obyčejný textový editor se syntax-highlightningem by měl úplně stačit*]
- **Shell** (Bourne shell (sh), Bourne Again Shell (bash), Korn shell (ksh), C Shell (csh), TENEX C Shell (tcsh), COMMAND.COM, Cmd.com, PowerShell, ...) [*Co to je, k čemu to je, proč se to používá*]
- **BASH**: Úvod [*jednak je důležité pro práci s příkazovou řádkou v Linuxu a pro možnost vytvářet dávkové soubory dělající různé věci a také je to ukázka interpretovaného jazyka*], hello world, **man**, **info** [*primární unixová nápověda každého programátora a uživatele používajícího příkazový řádek, jak se s tím pracuje, ukázky*], základní příkazy pro práci se soubory (cd, ls, cat, cp, mv, mkdir, rmdir, rm, ...)

L15: (8.4.2024)

- BASH: proměnné (globální, lokální), **shell expansion** (brace ex., tilde ex., param. and var. ex., cmd. subst., arith. exp., process subst., word splitting, pathname ex., quote removal), command **stdin**/**stdout**/**stderr**, redirekce, subshell, pole (indexované, asociativní)

L16: (10.4.2024)

- BASH: commands: test, list, [...], [[expr]], lexikografické porovnání, { list; }, ((expr)), if, for (všchny), while, for, break, continue, case, declare, exit, source, unnamed pipeline, Ctrl+C, Ctrl+D, history, !<num>, export, read, local, let, některé spec. výrazy a symboly, ...
- BASH: Aritmetické výrazy a operátory
- Psaní scriptů v BASHi, shebang #!

L17: (15.4.2024)

- BASH: pushd, popd, head, tail, find, wc, less, tr, xargs, ... [*praktické příkazy a nástroje mimo jiné pro práci s daty*]
- QuickSort pro čísla v BASHi (rekurzivní) [*Takové celkem jednoduché a praktické cvičení, které využije různé příkazy a vlastnosti BASHe*]
- DÚ: QuickSort v BASHi (lexikografický pro řetězce a nerekurzivní)

L18: (17.4.2024)

- **grep**: PCRE, základní a extended **regulární výrazy** [*Velmi důležitá věc pro práci s textem a vyhledávání v textu, což se při zpracování mnoha experimentálních dat hodí, škoda, že není čas probrat i Perl regulární výrazy, ale to je už kapitola na trochu delší povídání a pro práci s experimentálními daty základní regulární výrazy většinou postačují*]
- **sed**: princip fungování, většina příkazů, kromě ‘s’ a adresování

L19: (22.4.2024)

- sed: příkaz ‘s’ a adresování [*Velmi podstatný, a přitom jednoduchý nástroj na práci s textem, především na extrakci dat z textu, který dokonce dokáže nahradit i grep a jiné nástroje na práci s textem (na ještě o trochu vyšší nástroj awk už bohužel zde není čas, ale se sed-em se dá pro většinu aplikací vystačit)*]
- DÚ: rozvoj multi-řádků pomocí sed, náhrada head a grep pomocí sed [*jednoduché příklady na procvičení práce s nástrojem sed*]
- **C**: Úvod [*Příklad kompilovaného jazyka (pozor, bavíme se o čistém C (nikoliv C++ ani jiné mutace a hrůzy)), proč vznikl, co všechno umí, proč se mu nikdy nic nevyrovná, za jakou cenu a na co je třeba si dávat pozor*], standardy (C89/ANSI, C99, C11, C17), **libc**, minimální program, verze main(), Hello World, základní typy, ukazatele, referencing a dereferencing, void pointer, pole [*proč je vlastně jedno kolik má rozměrů, když stačí jeden*]

L20: (29.4.2024)

- C: struct, union, enum, typedef, operátory pro práci s typy, #define, #include, #if... #endif a jiná zajímavá makra, pointer arithmetic [*zdaleka nejmocnější zbraň spolu s type-castingem a type-punningem, kterou vám žádný jiný univerzální jazyk v takovéto míře nenabídne, má to svá úskalí, na která je třeba si dát pozor, ale když člověk ví, co dělá, tak dokáže naprosté zázraky*], jak psát C kód přehledně a čitelně
- C: typové kvalifikátory (const, volatile, restrict, _Atomic), atributy funkcí a proměnných (static, inline, ...)
- C: labels a základní příkazy: if, switch, while, for, break, continue, goto
- C: **Dynamicky alokovaná paměť**: malloc(3), free(3), realloc(3), heap, jak to funguje, příklad
- Garbage Collector princip v C – reference counting [*často se používá pro práci s dynamickou pamětí, je dobré vědět k čemu to je a že se to dá normálně používat i v obyčejném C*]
- DÚ: reference counting implementace (volitelná varianta pro pokročilejší: s použitím __atomic_*() funkcí)

L21: (6.5.2024)

- C: Předvedení řešení domácího úkolu s __atomic_*() funkcemi
- C: operátory, priority operátorů
- DÚ: extrakce méně-bitového integeru
- Vysvětlení principu kruhového lineárního spojového seznamu s jednou zarážkou.
- DÚ: Napsání funkcí pro práci s kruhovým lineárním spojovým seznamem s jednou zarážkou dle zadaných deklarací, včetně reference countingu.

L22: (13.5.2024)

- C: Předvedení funkcí pro práci s kruhovým lineárním spojovým seznamem s jednou zarážkou včetně reference countingu a lockingu (pomocí pthreads) pro možnost použití i v paralelním prostředí [*v rámci přednášky není čas detailně rozebírat úskalí a principy paralelního programování, takže alespoň zevrubně vysvětleno pro pochopení daného řešení, proč je to případně potřeba a proč právě takto, zároveň i pěkná úloha na ukázku podmíněné kompilace, jíž budeme moci případně dokončit v poslední přednášce*], základní pravidla pro reference counting v paralelním prostředí.
- C: Příklad použití výše vybudovaných funkcí pro práci se spojovým seznamem.

- C: práce s textovými soubory, práce s binárními soubory a práce se soubory v C jako taková. [*Pro práci s daty z experimentů je potřeba je umět přečíst a také zapsat*]

L23: (15.5.2024)

- **GNU Make**: Úvod [*k čemu je to použitelné, jak to funguje, mimochodem také pěkný příklad deklarativního jazyka, kdy má smysl to použít ručně a kdy se naopak za každou cenu vyhnout ručnímu používání (což bohužel bývá často zvykem u fyziků, kteří píší fyzikální výpočetní/simulační kódy), apel na vyvarování se toho a raději použít pokud možno co nejstandardizovanější podobu buildovacího systému*] Makefile, rules, normal prerequisities, order-only prerequisities, statické a dynamické knihovny, ruční linkování statických a dynamických knihoven, ukázka na knihovně pro lineární seznam.

L24: (16.5.2024)

- Psaní SW projektu v C (ale nejen v C), **buildovací systémy** [*k čemu je to dobré, proč je důležité to používat, proč je důležité je používat správně a ideálně ještě i ty správné, protože ne všechny tyto systémy jsou dostatečně univerzální a flexibilní, aby umožňovaly uživateli potřebnou volnost a flexibilitu kompilace*], **Autotools** (autoconf, automake, m4, libtool, autoconf-archive), příklad na knihovně pro lineární seznam [*snaha ukázat, že to lze použít velmi snadno, s mnohem menší námahou než přímé použití Make nebo, ještě hůř, BASH či Python scriptů, protože to jsou pak vždy jedna větší tragédie než druhá*], ukázka použití makroprocesoru m4 definováním vlastního jednoduchého testu, navázání výsledků testu na podmíněnou kompilaci.

L25 a L26: (bohužel odpadlo v průběhu semestru 😕)