1. Popište obecné schéma startu systému s důrazem na Hardwarovou kontrolu systému

- Spouští se firmware, který kontroluje
  - paměť, základní součásti systému, periferie
  - zda je možnost vstoupit do setup a nastavit další parametry, provádět další kontroly a detekce
  - předání řízení BIOSu nebo UEFI

2. Popište obecné schéma startu systému s důrazem na Načtení a start zavaděče

- BIOS
  - obsahuje základní funkce pro práci s HW
  - zavaděč se získá načtením startovacího záznamu
  - dnes jen hlavní část – natažení celého zavaděče odjinud
  - úkol zavaděče – načíst jádro systému
    - moderní zavaděče jako GRUB nepotřebují znát konkrétní uložení jádra
- UEFI
  - disk je formátovaný jako GPT
  - na začátku disku je seznam všech oddílů
  - UEFI hledá EFI oddíl, kde je zavaděč

3. Popište obecné schéma startu systému s důrazem na Načtení a rozbalení jádra

- najde-li zavaděč jádro, musí se načíst a rozbalit do paměti
- rozbalení zajišťuje rutina na začátku jádra
- komprese LZ4
- při rozbalování se kontroluje integrita jádra
- příprava na inicializaci – příprava tabulky paměťových stránek a detekce procesoru

4. Popište obecné schéma startu systému s důrazem na Inicializace jádra

- složitý proces
- nastavení přerušení, vytvoření datových struktur, inicializace ovladačů a detekce zařízení
  - ovladače v modulech nejsou inicializovány
  - některé ovladače musí být zakompilovány v jádře, aby systém běžel
- vytvoření speciálního vlákna, z něhož se stává první proces -init

5. Popište obecné schéma Spouštění procesů

- připojení speciálních souborových systémů
- aktivace bezpečnostních technologií
- nastavení parametrů pro konzoly
- inicializace síťových rozhraní
- kontrola souborových systémů

- úklid v adresářích pro dočasné soubory
- zapnutí paketového filtru
- start základních systémových démonů
- start konzol, případně grafického rozhraní

6. Start systému Ubuntu 12.04 ver. Ubuntu 16.04

- v dřívějším systému nastartoval program init prostředí systému podle zvoleného runlevelu
    - ten představuje množinu služeb, které mají být spuštěny / zastaveny
    - runlevel 0 znamená vypnutí systému a ž spouštění grafického režimu
- myšlenka je nastavení systému podle předem definovaných úrovní
    - docíleno pomocí symlinků v /etc/init.d/rcX.d (X reprezentuje level)
    - služby spouštěny sériově
- V 15.4 došlo ke změně, init nahrazen za systemd
    - náhrada levelů na systemd targets
    - runlevel lze stále využít
    - pro ovládání systemd se využívá systemctl
        - restart – runlevel 6 – init 6 nebo systemctl isolate reboot.target
    - systemd targets je jeden z systemd units
        - units slouží jako konfigurační soubory, ve kterém je definováno jaký režim má na starosti a jaké další služby aktivovat
            - služba .service
            - zařízení .device
            - mount .mount

7. Démon Systemd

- démon určen pro správu systému linux
- nahrazuje starší init
    - vyžaduje jiné metody a přístupy při používání
- rodič všech dalších procesů
- je spuštěn hned po inicializaci jádra
    - jeho PID je 1
    - pokud dojde k problému při startu tohoto démona, další procesy také nenaběhnou – to vyvolá Kernel panic
    - když skončí rodič procesu, jeho nový rodič je Systemd
- funkcionality:
    - journal --> journalctl command
    - ukládání systémových zpráv
    - /run/log/journal nebo /var/log
        - stačí vtvořit /var/log a restartovat systemd-journal
- možné vytvoření závislostí pomocí After
- start více služeb najednou
- kompatibilita s SysV skripty

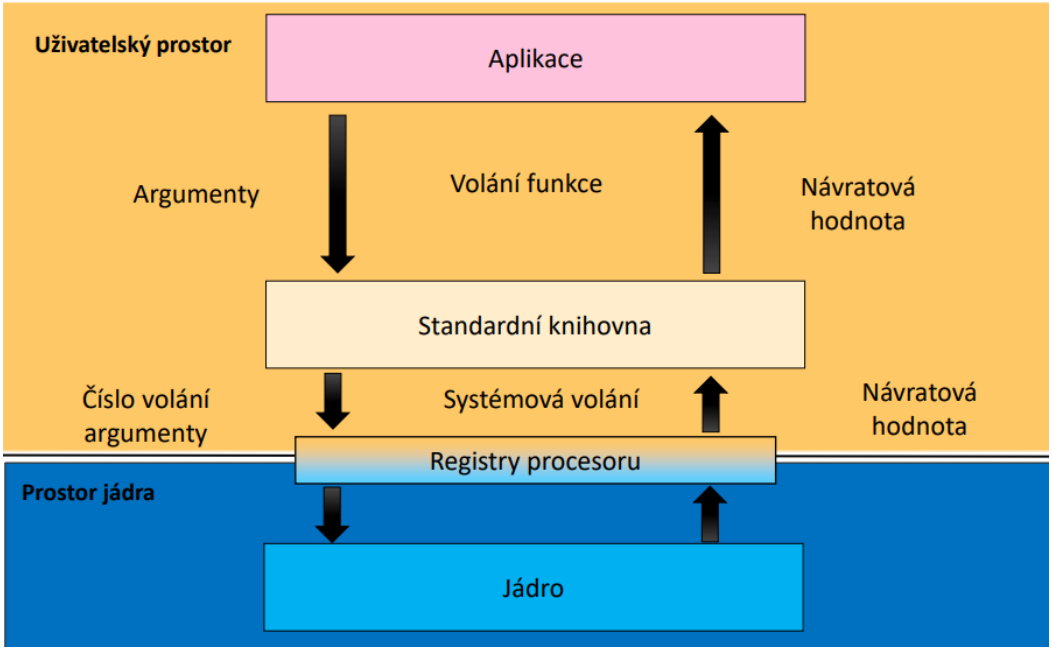
8. Démon Crond

- démon, který slouží k automatizování opakovaných událostí
- lze nastavit čas, kdy se má provést

- /etc/cron.daily nebo /etc/cron.weekly
- individuální nastavení pak v /etc/cron.d
- * * * * * = minuta, hodina, den, měsíc, den v měsíci
- příkazy crontab –e a crontab –l

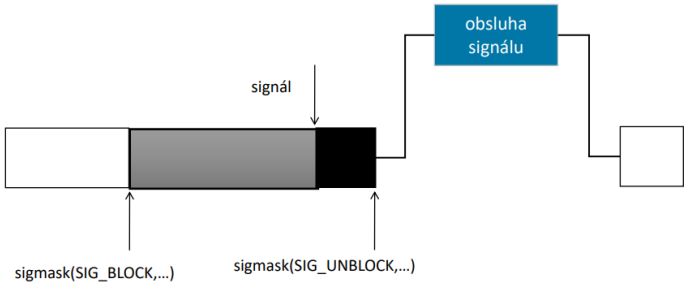
9. Pojednejte o problematice Systémových volání - Základní princip, Jak to funguje - například ioctl() a schéma systémového volání

- systémové volání = volání jádra
  - základní komunikace programů s jádrem
- tvůrce programu oddělen vrstvou standardní knihovny
  - nepotřebuje znát přesné chování jádra
  - je dobré je pochopit
- základní principy:
  - jak vypadá vztah proces – jádro
  - proces běží ve svém adresném prostoru a vykonává kód
  - někdy potřebuje službu, kterou pro něj vykoná jen jádro --> systémové volání
- IOCTL
  - uloží se aktuální obsah registrů
  - do registrů se uloží číslo systémového volání a parametry
  - předá se řízení jádru (přepnutí do režimu jádra a začne vykonávat jeho kód)
  - podle čísla volání se vyhodnotí, jaká funkce v jádře se zavolá
  - protože jde o volání spojené s otevřeným souborovým deskriptorem, najde se odpovídající objekt souboru
  - podle objektu souboru se zavolá příslušná výkonná funkce v ovladači
  - ve výkonné funkci se provede vše potřebné a na konci se vrátí výsledek
  - do registru se uloží výsledek operace
  - řízení se předá zpět programu resp. standardní knihovně (uživatelský režim)
  - výsledek se přenese z registru do paměti
  - obnoví se uložený stav registrů



10. Pojednejte o problematice Signálů - Reakce na signály, Blokace signálu

- nejstarší metoda komunikace mezi jádrem a procesem
- proces vykonává činnost – přijde signál – přeruší původní činnost – obslouží signál – pokračuje dál ve své činnosti
- rozdíl oproti přerušení a speciálním instrukcím
    - přerušení
        - legacy způsob přechodu User space procesu – Kernel space
        - HW přerušuje kernel
    - Speciální instrukce
        - doporučený způsob přechodu User space – Kernel space
- existují případy, kdy nechceme, aby vyl signál doručen okamžitě
    - to vyřeší blokace signálu



- využívá se bitová maska a sada funkcí
- funkce sigemptyset() masku signálu vynuluje – nebude obsahovat žádné signály
- funkce sigfillset() naplní, sigddset() přidá signál do masky, sigdelset() signál z masky odstraní, sigismember() zkontroluje, zda je signál v masce

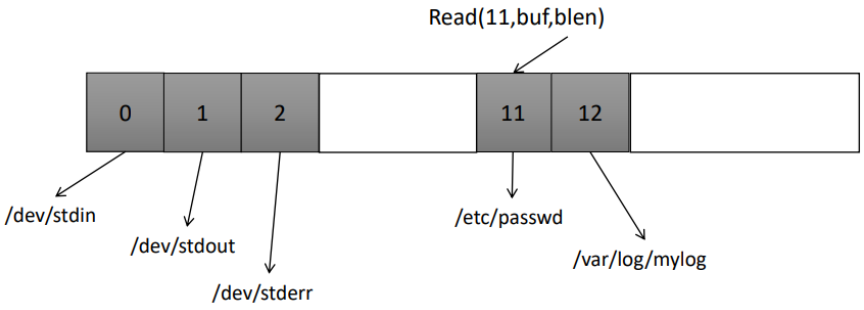
11. Pojednejte o problematice Signálů - Druhy signálů, Posílání a doručování signálů

- podle implementace
    - obyčejné
        - bity v masce signálu
        - v příchozím signálu se bit nastaví na 1
        - u zpracovávaného signálu se bit vynuluje
        - neprojeví se jeho vícenásobné doručení před obsloužením
    - real-time
        - používají frontu – zaručení, že se signál neztratí
        - doručen tolikrát, kolikrát byl poslán
        - čísla od 32 výše
        - pro komunikaci mezi vlákny
- dělení podle reakce na daný signál
    - SIGKILL – okamžité ukončení procesu
    - SIGSTOP – zastavení procesu
- synchronizace
    - synchronní
        - přesně víme, kdy ho proces obdrží

    - asynchronní
        - může přijít kdykoliv za běhu procesu a reakce by tomu měla být přizpůsobena
- 2 skupiny
    - posílané zásadně jádrem
    - posílané uživatelskými procesy
- základní metoda pro posílání signálu je funkce kill()
    - umožňuje poslat signál jednomu procesu nebo skupině
- funkce raise()
    - posílá signál stejnému procesu, který ji zavolal
- sigqueue()
    - využita pro real-time signály
    - informuje, zda byl signál vložen do fronty
- pthread_kill()
    - posílá signál určitému vláknu – lze použít jen v rámci jednoho procesu
- signál se dříve nebo později doručí, pokud je komu
- blokované signály se berou jako doručené až v okamžiku jejich odblokování
- signál poslaný procesu jako celku, je považován za doručení, až je přijatý celým procesem
- je-li signál poslán vláknu, pak může být doručen jen jemu

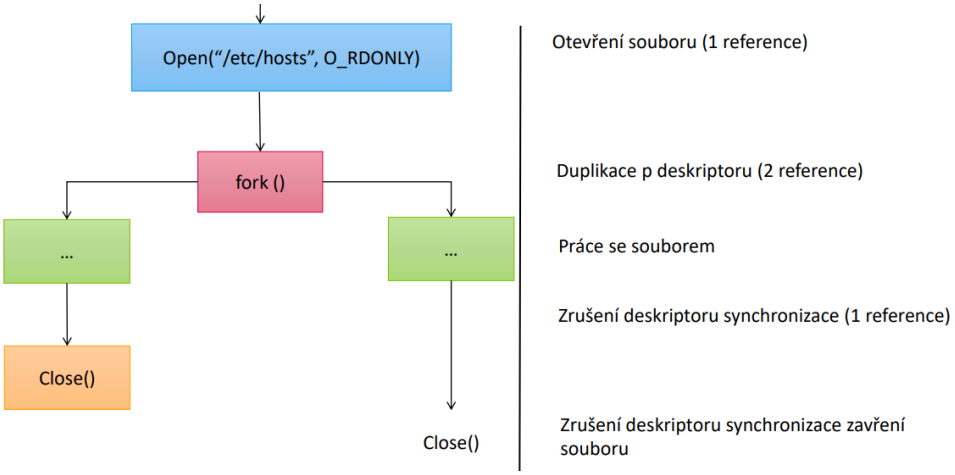
12. Pojednejte o Souborových operací - Deskriptor souborů, Otevírání a zavírání souborů

- identifikován číslem
- používá se v rámci procesu pro přístup k otevřenému souboru
- pouze nezáporné hodnoty
- první 3 patří standardním komunikačním kanálům
    - 0 – standardní vstup
    - 1 – standardní výstup
    - 2 – chybový výstup



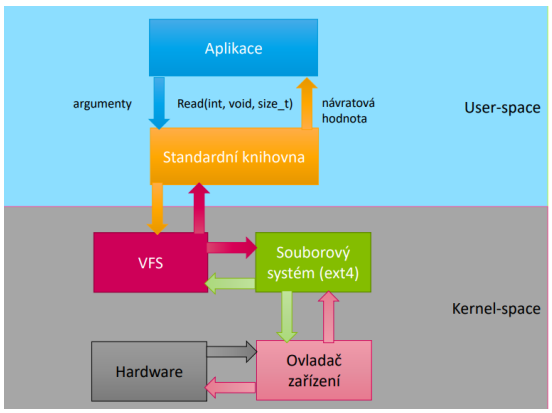
- otevření souboru
    - jádro vytvoří potřebné datové struktury pro přístup do souboru
    - označí soubor jako otevřený procesem
    - proces získá deskriptor pro odkazování na soubor
    - po použití soubor zavře
- způsob otevírání závisí na typu souboru
- základní funkce open()
- otevíraný soubor určujeme cestou

- každý otevřený soubor se musí zavřít
    - každý soubor vyžaduje alokované prostředky v jádře
    - nebezpečí nechtěného zápisu do nezavřeného souboru
    - počet procesem otevřených souborů není omezen
- zavření pomocí funkce close()
    - parametrem je deskriptor souboru
    - může selhat
        - přerušení signálem
        - problém na zařízení
        - problém na souborovém systému
- soubory v podprocesech
    - close() – neznamená skutečné zavření, jen sníží počet referencí na něj
    - ovladač v jádře pak neprovede následné činnosti – ty provede, až se zavře poslední proces



13. Pojednejte o Souborových systémech - Virtuální souborový systém

- VFS (Virtual File Systém)
    - abstraktní rozhraní
    - podoba zobecněného stromu
    - má jeden kořenový systém
    - lze připojovat různé konkrétní systémy
    - pevné odkazy i symbolické odkazy
    - speciální soubory
    - pravidla pro tvorbu názvů jsou velice volná



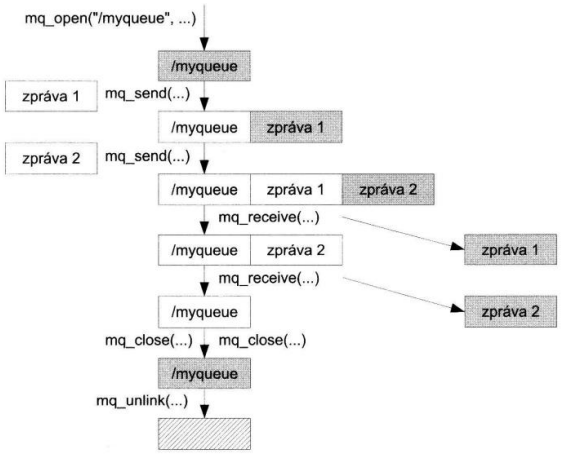
14. Komunikace pomocí pipe - Anonymní roury, Pojmenované roury, Jednoduchá roura, Roura s oběma konci

- místní komunikace mezi procesy
- typy
    - anonymní roury (pipe)
        - nejjednodušší mechanismus pro komunikaci mezi procesy
        - proud bajtů – do jednoho konce zapisuje, z druhého čte
        - na obou koncích stejný, nebo jiné procesy
        - konec roury může sdílet více procesů
        - z pohledu implementace jde o rozhraní paměťového bufferu
        - starší jádra 4096B, novější 65536B
        - pokud zapisuje více procesů do jedné roury, nelze garantovat pořadí, ve kterém se data objeví
    - pojmenované roury
        - fronta FIFO
        - používaný podobně jako anonymní roura
        - přebývá ve svém adresáři i v době, kdy s ní proces nepracuje
        - může být uložena do archivu a následně obnovena
        - před použitím se musí vytvořit pomocí mkfifo()
        - jako argument využívá cestu a masku oprávnění
        - pokud ji chceme zrušit, příkaz unlink()
    - komunikace pomocí zpráv
        - roury jsou objekty orientované na proud bajtů – nestrukturované
        - pokud potřebujeme pevný formát, využijeme
            - POSIX Message Queues
            - MQTT
            - REST API
            - WebSockets
            - SOAP
            - D-Bus
    - roura s oběma konci
        - flexibilnější řešení
        - náročnější implementace
        - roura, která má k dispozici oba konce
        - plnohodnotná komunikace s podprocesy
        - komunikace mezi vlákny procesu
        - řízení procesu událostí
        - relativně bezpečné ošetření signálu

15. Komunikace pomocí zpráv – rozdíly mezi pipe a zprávami, POSIX PMQ

- roury jsou objekty orientované na proud bajtů – nestrukturované
- pokud potřebujeme pevný formát, využijeme
    - POSIX Message Queues
    - MQTT
    - REST API

    - WebSockets
    - SOAP
    - D-Bus
- POSIX
    - vyžaduje podporu jádra, příslušnou knihovnou pro programy
    - umožňuje vytvářet pojmenované fronty
    - do téže fronty může proces zapisovat i odebírat
    - notifikace při příchodu zprávy může být asynchronní nebo synchronní
    - každá fronta má svoji prioritu – 32768 úrovní
    - fronta funguje do restartu systému nebo do svého zrušení
    - nepřečtená zpráva zůstává ve frontě
    - limity
        - měkké
        - tvrdé



16. Komunikace pomocí socketů - Druhy přenosu socketové komunikace, Spojová komunikace, Datagramová komunikce, Porovnání komunikace pomocí rour a socketů

- socket
    - jedná se o komunikační metodu v síti, tak místní
    - výhoda v transparentnosti na transportní vrstvě
    - rozhraní Berkeley Sockets
    - socket – datový objekt obsahující informace o stavu komunikace s druhou stranou
    - pro vlastní komunikaci používáme systémová volání
    - druhy
        - spojový
            - nejdříve sestaven komunikační kanál, na který se zasílají data v podobě proudu bajtů
        - datagramový
            - sestaven balík dat – datagram – který je odeslán jako celek
- spojová komunikace
    - nemusí řešit délku paketu, ani spolehlivost přenosu
    - klient
        - vytvoří socket

            - případné přiřazení názvu
            - připojení na server
            - vlastní komunikace
            - odpojení od serveru
            - zrušení socketu
        - server
            - vytvoření socketu
            - přiřazení názvu
            - nastavení socketu na naslouchací režim
            - čekání na připojení klienta
            - komunikace s klientem
            - ukončení spojení
            - zavření socketu
        - obsluha klientů na stejném vlákně jako čekání na další klienty
        - obsluha klientů v samostatných procesech a samostatných vláknech
        - lokální doména slouží pro komunikaci uvnitř systému
        - vytvořen speciální soubor na disku
        - můžeme nastavit přístupová oprávnění
        - eliminujeme nebezpečí obsazení portu jinou aplikací
        - nespecifikujeme protokol
- datagramová komunikace
    - využijeme ve chvíli, kdy komunikace nevyžaduje 100% spolehlivost
    - upřednostňuje se propustnost
    - komunikace
        - vytvoření socketu
        - přiřazení názvu
        - sestavení datagramu
        - odeslání datagramu
        - příjem datagramu
        - zpracování datagramu
        - uzavření socketu
- porovnání
    - sockety – síťově orientované
    - roury – ryze místní
    - sockety volíme tam, kde se počítá se síťovou transparencí
    - bezpečnost
        - roury – klasická práva pro přístup do souborů
        - sockety – pro místní sockety ano
        - pro síťové sockety řešeno na úrovní aplikace nebo paketového filtru v jádře

17. Procesy a vlákna - Plánování úloh, Plánovače

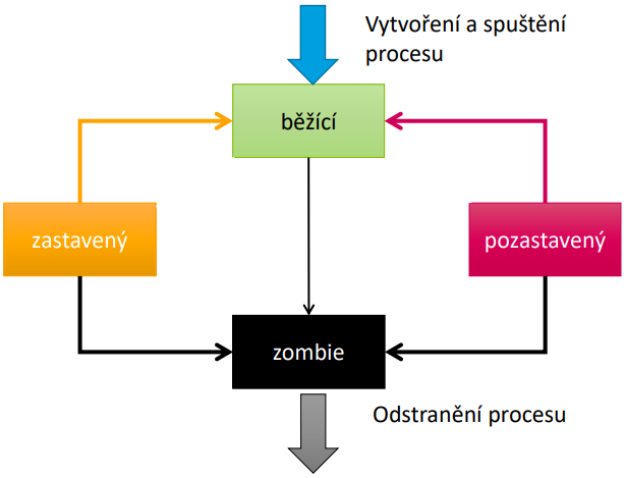
- základní úlohou jádra je správa procesů a vláken
    - vytvoření
    - plánování
    - nastavení
    - ukončení
- proces – běžící program

- vlákno – objekt pracující podle kódu programu
- plánování úloh
    - úlohy běží na jednotlivých procesorech v rámci přiděleného času
    - úloha dostane přidělený čas, a dokud ho nevyčerpe, nebo není odebrána z procesoru, vykonává své instrukce
    - proces plánování
        - použitá plánovač
        - priorita úlohy
        - vlastnosti jádra (preemtivita)
        - každé vlákno procesu je chápáno jako samostatná úloha, je tedy plánováno nezávisle na jiných vláknech téhož procesu
- plánovač
    - 4 typy
        - normální – pro většinu použití
            - využívá jedinou úroveň statické priority
            - používá hodnoty nice a časová kvanta přiděluje na základě dynamických priorit, které jsou určovány podle předchozího využití
            - čas získá první úloha ve frontě, množství času podle dynamické priority
        - dávkový – jako normální plánovač, ale s úpravou pro neinteraktivní úlohy
            - pro úlohy, které mají velké požadavky na čas a nepotřebují být interaktivní
            - liší se výpočtem dynamických priorit
            - úloha se posuzuje, jako kdyby neustále požadovala běh na procesoru, je tedy permanentně penalizována
        - FIFO – pro real-time operace
            - úloze přidělí procesor, a dokud ho sama neodevzdá, žádná jiná úloha na tomto procesoru nedostane příležitost
            - 99 úrovní statických priorit
        - round-robin
            - podobný jako FIFO, ale liší se přidělováním časových kvant, které přiděluje dokola na určité úrovni priority (až 99)
            - úloha běží nejdéle po dobu přiděleného času, pak jí je procesor odebrán a je předán další úloze se stejnou prioritou

18. Procesy a vlákna - Preemtivita jádra, Proces a jeho vlastnosti, Problematika paměťového prostoru, Souborové deskriptory, Oprávnění procesů, Životní cyklus procesu++

- preemtivita
    - 3 způsoby
        - nepreemtivní – úloha se v jádře musí sama vzdát procesoru
        - dobrovolně preemtivní – jsou přidány body, kde se úloha vzdá procesoru
        - plně preemtivní – plánovač odebere procesor po vyčerpání časového kvanta
- proces a jeho vlastnosti
    - každý proces má jednoznačný číselný identifikátor – PID

    - každý proces má svého rodiče – proces, kterým byl vytvořen – jediná výjimka je proces init
- problém paměťového prostoru
    - stránky namapované procesem se z hlediska procesu zkopírují
    - dokud se ze stránek čte, zůstávají společné pro starý i nový proces
    - zkopírování nastává až v okamžiku, kdy se některý z procesů pokusí o zápis nebo si zkopírování explicitně vynutí
- souborové deskriptory
    - až na výjimky se dědí – jsou tedy platné beze změny i v novém procesu
    - problém hrozí v nechtěném přístupu k datům
    - proti nechtěnému úniku deskriptorů se používá uzavření všech možných deskriptorů hned po jeho zkopírování, krom těch, které potomek zdědit má
- oprávnění procesů
    - reálná – odpovídají oprávnění uživatele, který spustil program
    - efektivní – mohou být ovlivněna tím, že má program nastaven SUID, neboli proces běží s právy vlastníka souboru s programem
- životní cyklus
    - běžící – procesory vykonávají programový kód procesu nebo některá vlákna čekají na splnění nějaké podmínky
    - zastavený – proces obdržel signál SIGSTOP, SIGTTIN, SIGTSTP a byl zastaven, žádný z procesorů nevykonává kód tohoto procesu
    - pozastavený – provádění kódu bylo přerušeno signálem SIGTRAP kvůli splnění nějaké podmínky
    - zombie – provádění kódu bylo dokončeno, již neběží na žádném procesoru. Proces zůstává v tomto stavu, dokud není jádro požádáno o jeho odstranění
    - proces tedy běží, dokud má k dispozici instrukce, které lze vykonat
    - po dokončení procesu může mít rodič nastaveno SIG_IGN – pak proces tiše zanikne
    - pokud má nastavenou explicitní obsluhu, proces nezanikne a zůstává ve stavu zombie



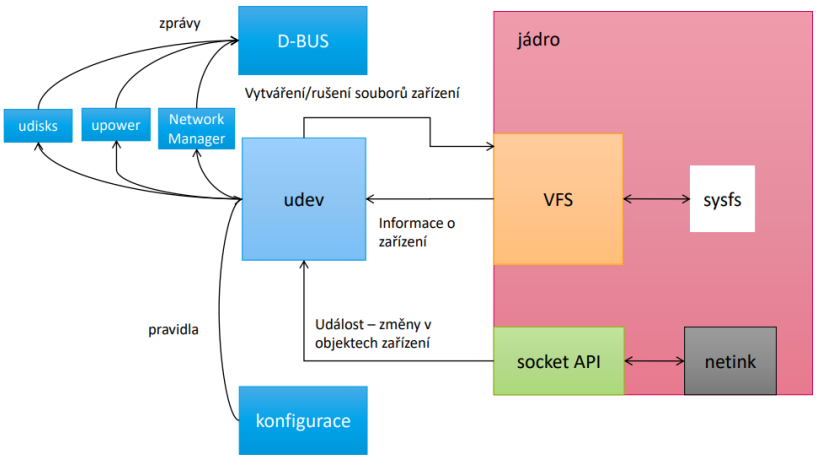
19. Problematika Spouštění programů - Scénář jak jádro spouští program, Handler, Spouštěcí domény

- po vytvoření procesu obvykle následuje spouštění programu

- k tomu slouží systémové volání execve()
- scénář
    - otevření souboru s programem
    - výběr procesoru pro start
    - kontrola a nastavení oprávnění
    - načtení hlavičky programu
    - zkopírování parametrů
    - vyhledání odpovídajícího handleru podle hlavičky
    - zavolání handleru k vlastnímu spuštění
    - pokud něco selhalo, úklid a vrácení výsledku programu
- handler binfmt_misc
    - jde o univerzální handler, umí detekovat různé formáty a nakládat s nimi
    - spuštění např.:
        - javovské třídy
        - programy pro jiný OS
        - různé dokumenty, obrázky, audio soubory
    - celá funkce je podobná jako u handleru pro skripty, rozdíl v tom, jak se sekundárně spuštěný program vybírá
    - handler je implementován tak, že se navenek tváří jako speciální souborový systém – ten se musí před použitím připojit – obvykle
- spouštěcí domény
    - linux umožňuje spouštět i programy, které byly zkompilovány pro jiný operační systém splňující standard POSIX
    - jádro obsahuje mechanismus, který umisťuje programy pro OS do spouštěcí domény
    - každá doména má v jádře svoji strukturu exec_domain
        - obsahuje informace potřebné pro úspěšné spuštění z této doményjako je
            - handler pro systémová volání
            - mapování signálů
            - chyb, socketů atd…

20. Démon Udev - Princip systému udev, výhody udev, udev a sysfs, rozdíl oproti HAL, technologie D-Bus

- pro flexibilní práci v systému je v uživatelském prostoru implementován démon, který úzce spolupracuje s jádrem
- nyní Udev, dřive HAL
- Udev
    - jde o virtuální zařízení představující rozhraní mezi uživatelskými procesy a ovladači v jádře – ty se používají prostřednictvím speciálních souborů
    - tyto soubory obvykle v adresáři /dev a jeho podradresářích
    - procesy očekávají, že pokud mají využít určité zařízení, jeho soubor najdou na svém místě
    - od roku 2021 součást systemd
- princip Udev
    - funguje jako proces, který běží v uživatelském prostoru
    - nepotřebuje zvláštní oprávnění – jen zápis do adresáře /dev
    - přijímat informace o událostech hlášených do uživatelského prostoru



- výhody Udev
  - udržuje aktuální stav souborů podle toho, co je aktuálně podporováno v jádře
  - umožňuje dodržovat názvy souborů podle LSB nebo podle přání správce systému
  - umožňuje používat libovolné číslování zařízení
  - názvy zařízení mohou být persistentní
  - v kombinaci s D-Bus poskytuje abstrakci pro High Level aplikace
  - je schopen informovat jiné procesy o vytvořených a rušených souborech
  - pokud se aktuálně nepoužívá, jeho paměť lze odložit na disk
  - pro daná konkrétní zařízení lze automaticky vytvářet symlinky
- Udev a sysf
  - pro svoji činnost ho Udev nezbytně potřebuje
  - čerpá odtud informace o virtuálních zařízení
    - při změnách v sysfs se musí upravit implementace Udev
- Udev oproti hal
  - hal má cílem zajistit co nejabstraktnější podobu fyzických zařízení
  - předchozí implementace – zastaralá, dnes již výhradně udev
  - dříve Udev využíval část hal, dnes je samostatný
- D-BUS
  - sběrnicová technologie
  - původně jako projekt KDE
  - dneska IPC standard
  - knihovny pro čisté C, C++, Java, Python, Ruby, Mono, ...

21. Architektura jádra - Monolitické jádro a mikro jádro, základní subsystémy, modularita jádra

- mono jádro
  - všechny služby OS běží spolu s hlavním vláknem jádra, a tedy i ve stejné oblasti paměti
- mikro jádro
  - poskytuje jen základní funkčnost nezbytnou pro vykonávání služeb
  - většina služeb je realizována speciálními ovladači v uživatelském prostoru
- základní subsystémy
  - základní funkcionalita
  - knihovna jádra

    - správce paměti
    - souborový subsystém
    - bloková vrstva
    - síťová vrstva
    - šifrovací subsystém
    - bezpečnostní subsystém
    - multimediální subsystém
    - platformě závislá část
    - ovladače sběrnice a dalších systémových zařízení
    - ovladače zařízení
    - ovladače souborových systémů
    - ostatní ovladače
- modularita jádra
    - kód jádra rozdělen na základní (jádro) a moduly (ovladače)
    - vlastnosti modularity
        - jednotné rozhraní
        - rozhraní modulů je rozhraním jádra
        - modul může být pevnou součástí jádra
        - filosofií je platformní nezávislost
        - žádná ztráta výkonu
        - možnost automatického zavádění modulů
    - build-in moduly
        - zabudované přímo v binárním souboru jádra
    - LKM – Loadable kernel modules
        - moduly jsou soubory v uživatelském prostoru
        - zavádějí se dle potřeby

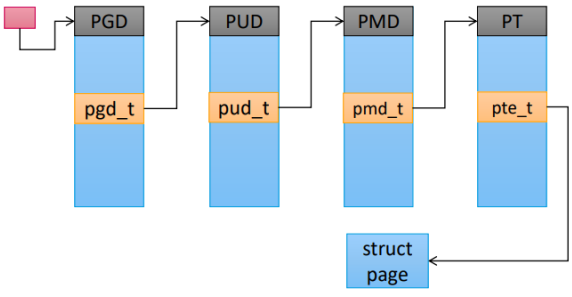
22. Architektura jádra - Zavádění a uvolnění modulů, init_module(), delete_module()

- proces je založen na systémových voláních init_module() a delete_module()
- typické načítání modulu, které implementuje program insmod vypadá tak, že se:
    - otevře soubor s modulem
    - překopíruje se do paměti
    - zavolá se init_module
    - paměť se uvolní/odmapuje
    - soubor se uzavře
    - zkontroluje oprávnění
    - alokuje dočasnou paměť pro modul
    - zkontroluje formát, zjistí umístění sekcí modulu, zkontroluje verzi modulu
    - alokuje paměť pro modul a pro inicializaci
    - vypočítá definitivní adresy pro jednotlivé sekce modulu a překopíruje sekce na nové místo
    - inicializuje objekty pro sysfs, nastaví licence a další informace
    - sestaví tabulky symbolů podle informací v hlavičce a provede realokace
    - vytvoří atributy v sysfs pro argumenty modulu, jeho sekce atd.
    - uvolní dočasnou paměť
    - přidá modul do seznamu

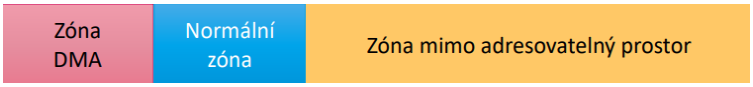
    - zavolá inicializační funkci modulu
    - zamění stav modulu na live (MODULE_STATE_LIVE)
    - uvolní paměť pro inicializace a provede další úklidové činnosti
- delete_module():
    - kontrola oprávnění
    - zjištění názvu a vyhledání modulu v seznamu
    - kontrola případných závislých modulů
    - kontrola možnosti modul odebrat a nastavení stavu going – pokud je modul používán a volání je neblokující – konec odpojení
    - čekání na uvolnění modulu
    - zavolání úklidové funkce modulu
    - odebrání modulu ze seznamu, uvolnění paměti a dalších použitých prostředků

23. Správa paměti - hlavní úkoly, Stránkování paměti, Zóny paměti, Alokace stránek

- jeden z hlavních úkolů jádra systému – zjistit dostatek prostoru pro jádro a všechny procesy při omezené velikosti paměti a co nejmenším zdržením režijními operacemi
- úkoly:
    - převody mezi různými druhy adres
    - zpřístupnění paměti, která leží mimo adresní prostor
    - nízko-úrovňová alokace pro různé účely
    - správa paměti a alokace pro jádro
    - správa paměti procesů, přidělování, mapování, sdílení
    - různé cache, přednačítání, zpožděný zápis
    - odkládání na disk, zpětné načítání
    - řešení nedostatku paměti
- stránkování
    - PGD – page global directory – globální adresář stránek
    - PUD – pake upper directory – horní adresář stránek
    - PMD – page middle direktory – střední adresář stánek
    - PT – page table – tabulka stránek



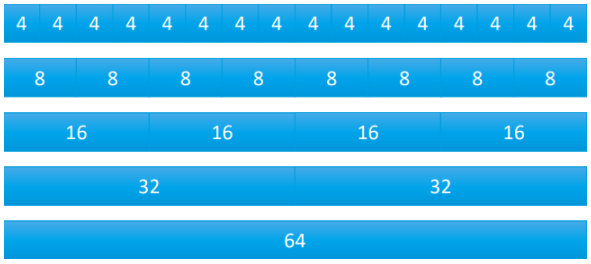
- zóny
    - z důvodů HW limitů je paměť rozdělena na zóny, kde každá má své vlastnosti
    - dělení je dáno architekturou
    - obecně
        - zóna pro DMA
        - zóna pro normální použití
        - zóna mimo adresní prostor



- alokace stránek
  - ?

24. Správa paměti - Alokace stránek, Algoritmus buddy, Rezervované stránky, Cache pro stránky, Proces alokace

- alokace stránek
  - ?
- Algoritmus buddy
  - skupiny bloků od 1 do 2^n stránek, n obvykle 10
  - vždy zdvojnásobuje další úrovně velikosti
  - celkem 10 úrovní velikosti ve výchozím stavu
  - volné bloky do seznamů
  - blok typicky 4096B

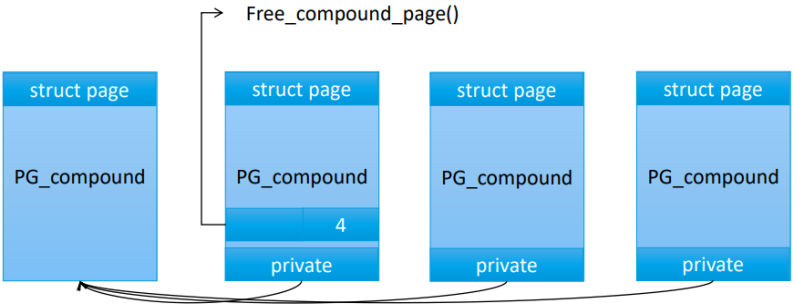


  - pravidla pro alokaci
    - zkus najít blok odpovídající velikosti – pokud existuje – přiděl ho
    - zkus najít blok o úroveň větší velikosti – neexistuje-li, pokračuj na další úroveň
    - nalezený blok rozděl tak, aby se přidělila potřebná část a zbytek byl použitelný pro další zpracování
  - při uvolňování se postupuje opačně
    - lze-li dva sousední bloky sloučit, učiní se tak
    - postupuje se rekurzivně tak dlouho, dokud lze slučovat
- rezervované stránky
  - normální alokace – není-li k dispozici volný blok paměti – volání se uspí – spouštěn mechanismus uvolnění paměti – uvolnění dostatečné velikosti – alokace probuzena
  - ne vždy možné – jádro udržuje některé stránky vyhrazené pro použití, kdy se alokace nesmí blokovat
  - tato paměť se dělí mezi ZONE_NORMAL a ZONE_DMA resp. ZONE_DMA32
- cache
  - vylepšení výkonnosti při alokaci
  - pro každý procesor v jádře vytvořeny cache – horká a studená
- proces alokace
  - _alloc_page() základní alokační funkce volaná z alloc_pages()
  - na začátku zkusí získat volné stránky pomocí get_page_from_freelist() – pokud ne – volán démon kswapd, aby uvolnil dostatečný počet stránek
  - paralelně s démonem proveden druhý pokus o získání stránek

    - pokud ne – proveden třetí pokus
    - pokud dojde k selhání i nyní – zacyklení a pokus neustále opakován dokud se nepovede
    - automatická alokace – zde končí neúspěchem
    - normální alokace
    - dá prostor k běhu jiným procesům
    - pak synchronní uvolňování paměti (try_to_free_pages())
    - nový pokus získat volné stránky
    - je-li neúspěšné a současně neuspělo ani synchronní uvolňování = out of memory

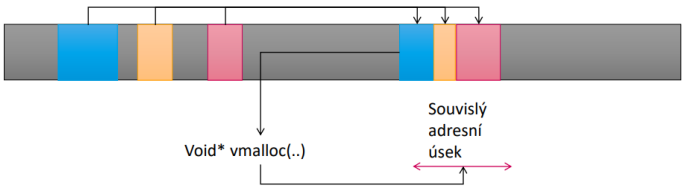
25. Správa paměti - Proces alokace, Sloučené stránky, Uvolnění stránek, Alokace nesouvislých úseků, Uvolnění alokované paměti

- proces alokace
    - _alloc_page() základní alokační funkce volaná z alloc_pages()
    - na začátku zkusí získat volné stránky pomocí get_page_from_freelist() – pokud ne – volán démon kswapd, aby uvolnil dostatečný počet stránek
    - paralelně s démonem proveden druhý pokus o získání stránek
    - pokud ne – proveden třetí pokus
    - pokud dojde k selhání i nyní – zacyklení a pokus neustále opakován dokud se nepovede
    - automatická alokace – zde končí neúspěchem
    - normální alokace
    - dá prostor k běhu jiným procesům
    - pak synchronní uvolňování paměti (try_to_free_pages())
    - nový pokus získat volné stránky
    - je-li neúspěšné a současně neuspělo ani synchronní uvolňování = out of memory
- sloučené stránky



- uvolnění stránek
    - jednoduchý proces na vyšší úrovni – složitější na úrovní algoritmu buddy
    - funkce _free_pages() dekrementuje počítadlo referencí
        - testuje uvolnění stránky
    - volá free_hot_page(), která stránku vloží do hot cache aktuálního procesoru
    - pokud dosažena horní mez cache
        - funkce free_pages_bulk() nadbytečné stránky vrací do buddy seznamů
- alokace nesouvislých úseků
    - není jednoduché nalézt souvislý úsek paměťových stránek
        - alokace nesouvislých úseků a jejich mapování

    - alokuje se sada úseků a ty se namapují do virtuálního adresného prostoru jádra – získáme souvisle adresovatelný úsek – pak lze normálně pracovat
    - funkce get_vm_area_node() hledá volnou oblast, kam bude paměť po alokaci namapována, v rozsahu adresného prostoru vyhrazeného pro tento účel
    - oblast musí být dostatečně velká
        - velikost + 1 stránka navíc – ochrana
    - vlastní alokace
        - funkce _vmalloc_area_node()
        - alokuje jednotlivé stránky
    - ukazatele alokovaných stránek ukládány do pole – použito při mapování



- uvolnění alokované paměti
    - alokovanou paměť je potřebo po použití uvolnit
    - funkce vfree() – volá funkci _cumap(), která se stará o odmapování a uvolnění
    - nejdříve dojde k odmapování a je-li potřeba, postupně prochází všechny stránky a nastává dealokace

26. Správa paměti - Řešení nedostatku paměti, Reverzní mapování, Základní mechanismus uvolňování

- řešení nedostatku paměti
    - metody řešení nedostatku fyzické paměti
        - uvolnění aktuálně nepoužívaných stránek
        - aktivace mechanismu OOM killer – postupně ukončí tolik procesů, aby vyřešil situaci
    - kategorie uvolňovaných stránek
        - odložitelné
        - synchronizovatelné
        - přímo uvolnitelné
- reverzní mapování
    - používá se pro najití všech položek odkazujících na určitou stránku
    - v linuxovém jádře se používá objektové reverzní mapování a to buď anonymní, nebo souborové
- základní mechanismus uvolňování
    - nejlépe a nejrychleji řešit nedostatek paměti, zároveň nejméně omezovat jádro a procesy
    - pravidla
        - přednost mají nejdéle nepoužívané stránky
        - přednost mají stránky, které nikdo nepoužívá
        - co nejvíce stránek procesů musí být uvolnitelných – všechny kromě aktivních
        - pro uvolnění sdílených stránek se musí odmapovat všechny odkazující položky

27. Správa procesů a vláken - Vlastnosti a stav úlohy, Příznaky úlohy, Vztahy mezi úlohami

- vlastnosti a stav úlohy
  - task_struct – datová struktura popisující úlohu
    - obsahuje obrovské množství položek, které se dynamicky mění
  - důležité položky
    - stav úlohy
    - příznaky úlohy
    - datové struktury
  - stav úlohy
    - TASK_RUNNING
    - TASK_INTERRUPTIBLE
    - TASK_UNITERRUPTIBLE
    - TASK_STOPPED
    - TASK_TRACED
    - TASK_DEAD
- příznaky úlohy
  - dalším důležitým parametrem je ukazatel na strukturu thread_info
  - každá architektura má vlastní definici – jedná se o data silně spjatá s HW
  - obsahuje informace o aktuálním procesoru úlohy, o návratu ze systémového volání, o adresném prostoru atd..
  - PID
    - datová struktura sloužící k identifikaci vlákna, procesu, skupiny procesů a session
    - v porovnání s TID, TGID/PID, PGID, SID má výhodu, že je vždy jednoznačná a netrpí problémy s recyklací číselných hodnot
    - ukládá se do hashové tabulky, kde je lze velmi rychle najít podle číselné hodnoty
- vzaty mezi úlohami
  - mezi jednotlivými úlohami panují určité vztahy
  - struktura task_struck obsahuje položky, které tyto vztahy popisují
  - parent, real_parent, children, sibling, group_leader, ptrace_children, ptrace_list, tgid, signal -> pgrp, signal -> session

28. Správa procesů a vláken - Kopírování úlohy, Ukončení běhu úlohy

- kopírování úlohy
  - ve starších unixových systémech vytvoření procesu = kompletní zkopírování procesu
  - v linuxu NE
    - paměť se kopíruje až při zápisu do stránky
    - lze určit co kopírovat
    - lze určit co sdílet
  - proces kopírování zajišťuje funkce copy_process(), ten provede:
    - kontrolu příznaku
    - ověří oprávnění
    - zkopíruje základní strukturu úlohy
    - kontrolu limitu
    - zvýšení reference

        - aktualizace dalších stavových hodnot
        - do struktury úlohy se nastaví příznak a struktura pid
        - dále se kopíruje vše co má nastaveno příznak pro kopírování
            - deskriptory, signály a jejich obsluha, paměťová struktura
        - vždy se volají všechny funkce, teprve uvnitř každé se rozhoduje, jak a zda se bude kopírovat
        - kopírování se dokončí funkcí copy_thread()
        - poté pokračuje inicializace datových položek
        - pak dojde k přepočtení signálů, aby byly doručeny určité relaci
        - následuje řada nastavení, které definují vztahy mezi rodiči, skupinou procesů
- ukončení běhu úlohy
    - implementace ukončení běhu úlohy je složitá
    - běh končí vždy stejným mechanismem
    - na nejvyšší úrovni funkce do_exit_group(), která je volána systémovým voláním exit_group()
        - zajistí ukončení všech vláken ve skupině daného procesu – posílá všem signál SIGKILL a pak zavolá do_exit()
            - do_exit() začíná kontrolami – posílá notifikaci a aktualizuje paměť systému
        - po posledním vlákně ve skupině ruší časovače a odsraní jejich infrastrukturu
        - postupně uvolňuje jednotlivé komponenty
    - dalším krokem jsou notifikace
        - nejdříve se posílá zpráva přes connector a volá se exit_notify(), která mimo jiné zajistí přesměrování čekajících signálu do jiných vláken
        - změní ukončovací stav úlohy na EXIT_ZOMBIE a pokud nikdo nečeká pak přímo na EXIT_DEAD
        - uvolňovací funkce release_task() se ukončuje daná úloha a také se zasílá signálová notifikace vláknu, které odstartovalo ukončení skupiny a je ve stavu zombie
        - sched_exit() zajistí dostatečné časové kvantum na zbytek ukončovacích prací
    - na samém závěru ukončování se úloha ještě jednou naplánuje – volá se funkce finish_task_switch() s voláním put_task_struct(), které dekrementuje počítadlo referencí – pokud na úlohu již nikdo nečeká, klesne na nulu a úloha se odstraní
    - nedostane již časové kvantum a zůstane uzavřena
    - pokud na úlohu čeká rodič, provede se definitivní uvolnění ve funkci wait_task_zombie()
        - poté se ve funkci změní stav úlohy na EXIT_DEAD a přečtou se potřebné údaje o úloze