Základy operačního systému LINUX – část II

Expansní znaky

Expansní znaky umožňují vytvářet skupiny jmen souborů na základě jejich podobnosti. Tyto skupiny se vytvářejí pomocí znaků se speciálním významem, tak zvaných ***expansních znaků*** . Pomocí obyčejných a expansních znaků se vytvoří výraz (prototyp), který splňují jen jména souborů určitého tvaru.

Expansní znaky jsou:

***** *Zastupuje libovolný řetězec, který nezačíná znakem* **.** *. Pokud expansní výraz začíná* *******, nesplňují jej řetězce, které začínají znakem* **.** *.*

? *Zastupuje libovolný znak. Pokud expansní výraz začíná* `?` *, nesplňují jej řetězce, které začínají znakem* **.** *.*

[ ] - ^ *Slouží pro zadání skupiny alternativních znaků. Znak* `^` *má význam negace a mají jej jen novější verse shellů. Například následující výrazy zastupují*

`[abc]`	*jeden ze znaků* `a`, `b` *nebo* `c`
`[0-9]`	*jeden ze znaků, které jsou v základním znakovém kódu počítače (obvykle ASCII tabulce) mezi znaky* `0` *a* `9`
`[^xy]`	*jeden znak, který je jiný než* `x` *nebo* `y`

Příklad 1: Následující výrazy s expansními znaky označují:

`ls` *adresář* `x*[0-9]`	*soubory, jejichž název začíná na* `x` *a končí číslicí*
`ls` *adresář* `???`	*soubory s tříznakovým názvem*
`ls` *adresář* `[^abc]?`	*soubory, které mají dvouznakový název a začínají jiným znakem než* `a`, `b` *nebo* `c`

Pokud shell při analýze příkazu narazí na výraz s expansními znaky, výraz expanduje, to jest nahradí jej všemi názvy v systému existujících souborů, které splňují expansní výraz.

Scripty

Příkazy shellu je možné vložit (včetně parametrů) jako řádky do souboru. Takový soubor, obsahující „dávku“ příkazů, které se postupně vykonají po spuštění tohoto souboru, se nazývá **script.**

Je-li spouštěný soubor script, pak shell spustí nový shell *(podřízený shell*, *subshell*), který bude příkazy scriptu provádět. V systému je ale více shellů. Proto lze v první řádce scriptu specifikovat úplnou cestou, který shell se má spustit. Je-li spouštěný shell v první řádce specifikován, musí první řádka začínat znaky `#!` a po nich následovat úplná cesta k souboru, který spouštěný shell obsahuje. Například první řádka může být

```
#!/bin/csh
```

http://ki.pef.czu.cz/ELVIS

Pokud první řádka znaky `#!` nezačíná, záleží na konkrétní implementaci shellu, jaký shell je spuštěn.

Příklad 2:

Vytvořte script s názvem *davka*, který bude vykonávat postupně tyto příkazy:

`cd ~`	*- přepnutí do domovského adresáře*
`clear`	*- smazání obrazovky*
`date`	*- vypsání aktuálního datumu a času*
`ls -l /etc`	*- vypsání obsahu adresáře /etc*

Řešení:

1) Vytvoříme nový soubor *davka*:

```
cat > davka
```

2) Nyní je možné zapsat jednotlivé řádky souboru

```
cd ~
clear
date
ls -l /etc
```

3) Ukončení editace souboru se provede stiskem: `<CTRL-C>`

4) Po vytvoření nového souboru je nutné nastavit přístupová práva tak, aby šel soubor spouštět.

```
chmod 755 davka
```

5) Nyní lze již soubor davka spustit stejně jako příkaz

`davka <Enter>`	*nebo*
`./davka <Enter>`	*není-li aktuální adresář nastaven v systémové proměnné PATH*

Spuštění na popředí a na pozadí

Příkaz (program nebo script) může být spuštěn na popředí nebo na pozadí. Rozdíl je v tom, že pokud je příkaz spuštěn na popředí, shell po spuštění příkazu čeká na jeho skončení. Uživatel musí proto počkat až příkaz skončí. Teprve potom se objeví prompt shellu, kterým je uživatel vyzván k zadání dalšího příkazu.

Pokud je příkaz spuštěn na pozadí, shell nečeká na jeho ukončení a je schopen okamžitě přijmout další příkaz. Spuštění příkazu na pozadí se provede tak, že se spouštěný příkaz ukončí znakem **`&`**.

http://kii.pef.czu.cz/Elvys

Příklad 3:

`ls -al` *Spuštění programu **ls** s parametry **-al** na popředí*

`sleep 100 &` *Spuštění programu **sleep** na pozadí. (Program **sleep** neudělá nic jiného, než že požádá operační systém o pozastavení na parametrem zadaný počet sekund. Po uplynutí této doby je proces řízený tímto programem systémem probuzen a po svém opětném spuštění okamžitě skončí)*

Příkazy:

<CTRL-C> *ukončení běžícího programu*

<CTRL-Z> *zastavení běžícího programu*

fg [%*číslo procesu*] *převedení programu na popředí*

bg [%číslo procesu] *převedení programu na pozadí*

Procesy

Proces je běžící program. Program je napsán v některém programovacím jazyce, tj. v assembleru, jazyku C nebo C++, Fortranu nebo některém dalším, je přeložen a na žádost uživatele spuštěn. Během své činnosti proces obvykle potřebuje od jádra operačního systému celou řadu **služeb**. Např. potřebuje:

- pracovat se souborem
- zaslat jinému procesu zprávu či signál
- použít některé I/O zařízení
- zjistit stav některého systémového prostředku (např systémového času)

V tom případě musí požádat o službu systému.

Systémové služby jsou v Unixu realizovány pomocí vnitřního přerušení. To znamená, že proces použije pro volání služby systému instrukci, která generuje vnitřní přerušení. Následuje přerušení činnosti procesu a skok do jádra systému a to do místa, ve kterém systém analyzuje, o kterou službu se jedná. Při skoku do jádra hardware automaticky uloží stavové slovo procesoru a přepne mód činnosti procesoru z uživatelského módu do módu jádra. Po zjištění, o kterou službu se jedná, je proveden skok na tu část kódu jádra, která požadovanou službu realizuje. Proces nadále běží v módu jádra.

Nejdříve se dokončí služba systému a teprve po návratu procesu z módu jádra do uživatelského módu je spuštěn **scheduler**, který rozhodne, zda bude proces pozastaven a zda bude spuštěn jeden z čekajících procesů a to ten, který má nejvyšší prioritu.

http://kti.pef.czu.cz/Elvys

Kontext procesu

V určitém okamžiku operační systém zpracovává řádově desítky procesů. Některé z nich plní systémové úkoly, některé řeší úlohy zadané uživateli. Aby procesy mohly být jednoznačně identifikovány, systém jim při jejich vzniku přiřadí uvnitř systému jednoznačné kladné číslo, tzv. ***identifikační číslo procesu*** (***PID, process identifier***).

Kontext procesu se skládá z **uživatelského** a **systémového** kontextu.

A) ***Uživatelský kontext*** tvoří:

1. Oblast paměti, ve které je uložen
   text procesu,
   data procesu,
   zásobník procesu.
2. Obsah ***pracovních a stavových registrů***.

Text procesu tvoří strojové instrukce spuštěného programu.

Oblast dat obsahuje v programu definovaná data. Ne všechna v programu definovaná data musí být umístěna v této oblasti. Některá z nich mohou být umístěna na zásobníku. Pokud byl program napsán v jazyce C, jsou v oblasti dat uloženy globální proměnné a statické lokální proměnné.

Na ***zásobník*** se ukládají parametry volání funkcí a návratové adresy. Na zásobník se také ukládají lokální proměnné, které nejsou statické.

B) ***Systémový kontext*** se skládá z obsahu ***systémových registrů***, obsahu ***systémového zásobníku***, který proces používá při běhu v módu jádra a ze záznamů, které si systém o procesu vede v ***tabulce procesů*** a v tak zvané ***u-oblasti***.

Tabulku procesů má systém neustále přístupnou v hlavní paměti a jsou v ní proto uloženy ty informace, které systém potřebuje o procesu znát i když proces právě neběží. Jsou to například informace nutné pro rozhodnutí, zda bude proces vybrán ke spuštění, tj. stav procesu, jeho priorita, dosud spotřebovaný procesorový čas atd. Dále je zde pole obsahující signály, které byly procesu zaslány atd.

Nejdůležitější údaje udržované v ***tabulce procesů*** jsou:

- *PID* (*process identification*) neboli *číslo procesu*. Operační systém přidělí každému procesu nezáporné číslo, které jej jednoznačně identifikuje.
- *Popis stavu procesu* (například proces je připraven ke spuštění, blokovaný atd.)
- *Deskriptor události*, která způsobila zablokování procesu.

http://kiwi.pef.czu.cz/Elvys

- *Pole signálů*, které byly procesu zaslány. Pro každý signál je v tomto poli vyhrazen jeden bit. Systém tedy není schopen rozlišit, zda byl procesu zaslán určitý signál jednou či vícekrát.
- *Ukazatel na tabulku stránek procesu* (tabulka stránek popisuje uložení procesu ve vnitřní paměti).
- *Ukazatel do u-oblasti na disku*.
- *Pole popisující vztah k jiným procesům* (je zde zaznamenáno *PID* procesu rodiče, *PID* procesů potomků , číslo skupiny procesů, do které proces patří atd.)
- *Čítače zaznamenávající využití zdrojů procesem*, především procesoru a paměti.
- *Parametry dovolující stanovit prioritu procesu*.

Uživatelská oblast procesu je v okamžiku, kdy proces neběží, uložena na disku. Proto jsou v ní uloženy ty údaje, které systém v případě, že proces neběží, nemusí znát. Například to jsou údaje o souborech a I/O zařízeních, se kterými proces pracuje a o stavu I/O přenosu, je zde zaznamenán pracovní adresář, řídící terminál, výsledek posledního volání služby systému atd. Jakmile je proces spuštěn, je obsah jeho uživatelské oblasti přepsán do hlavní paměti.

Nejdůležitější údaje udržované **v *u-oblasti*** jsou:

- *Reálné a efektivní jméno uživatele a skupiny* (tj. *UID, EUID,GID, EGID*)
- *Aktuální adresář a aktuální kořenový adresář.*
- *Tabulka deskriptorů otevřených souborů.*
- *I/O parametry* (adresa uvnitř procesu, odkud nebo kam se budou data přenášet, počet dosud přenesených bytů, počet bytů, které je třeba ještě přenést).
- *Ošetření signálu* (strojový kód, který se vykoná při příchodu signálu).
- *Řídící terminál*.
- *Výsledek a parametry volání systému*.

.

Součástí systémového kontextu procesu je dále **obsah systémových registrů** procesoru a **systémového zásobníku**. Systémový zásobník systém vytváří v jádře a proces jej využívá při běhu v módu jádra.

Stavy procesů

Proces může být během své existence v systému v různých stavech. Během své činnosti přechází z jednoho stavu do druhého. Všechny možné přechody mezi jednotlivými stavy se nejčastěji vyjadřují stavovým grafem. Uzly grafu označují možné stavy procesu, orientované hrany možné přechody mezi nimi. U hran se ve stavovém diagramu obvykle uvádí příčina přechodu z jednoho stavu do druhého. Stavový graf podle monografie (viz [2]) je uveden na obr. 5.

http://kiv.zcu.cz/vyuka/

Možné stavy procesu jsou:

1. ***Běh v uživatelském módu*** (***User running***). Proces běží v uživatelském módu.
2. ***Běh v módu jádra*** (***Kernel running***). Proces běží v módu jádra.
3. ***Připraven k běhu v paměti*** (***Ready to run in memory***). Proces je připraven ke zpracování. Je zařazen do fronty scheduleru a scheduler ho časem spustí.
4. ***Zablokován v paměti*** (***Asleep in memory***). Proces je z nějakého důvodu zablokován, to jest provedl algoritmus *sleep_on()*. Je v paměti, tj. zatím nebyl swapperem odložen na disk. Do fronty scheduleru bude zařazen až po svém probuzení.
5. ***Připraven k běhu na disku*** (***Ready to run swapped***). Proces je připraven ke zpracování, ale byl swapperem odložen na disk (nejspíše byl předtím nějaký čas zablokován). Nefiguruje ve frontě scheduleru a nemůže být proto schedulerem spuštěn. Swapper jej časem převede z disku do paměti a zařadí do fronty sheduleru.
6. ***Zablokován na disku*** (***Asleep and swapped***). Proces je zablokován a swapper jej odložil na disk, aby uvolnil paměť jiným procesům.
7. ***Pozastaven po preemci*** (***Preempted***). Proces vyčerpal své časové kvantum a scheduler jej pozastavil a spustil proces s vyšší prioritou. Proces zůstává zařazen ve frontě scheduleru a bude po čase buď spuštěn nebo v případě nedostatku místa v paměti swapperem odložen na disk Z tohoto hlediska je stav procesu 7 stejný jako stav 3.
8. ***Nový proces*** (***Created***). Proces byl nově vytvořen voláním služby systému *fork()*. Ještě pro něj nebyly vytvořeny všechny systémové struktury, které ke spuštění potřebuje a proto zatím není zařazen do fronty scheduleru a nemůže být spuštěn.
9. ***Proces zombie*** (***Zombie***). Proces končí svou činnost. Provedl algoritmus *exit()* a proto systém uzavřel všechny jím otevřené soubory a uvolnil jím obsazenou paměť. Proces již nemůže být spuštěn. Má ale ještě záznam v tabulce procesů. Čeká až proces rodič zpracuje jeho údaje uchované v tabulce procesů (především spotřebovaný čas procesoru) a vyřadí jej z tabulky procesů.

Použité příkazy:

`ps` [*volby*]

vypisuje informace o systémových a uživatelských procesech.

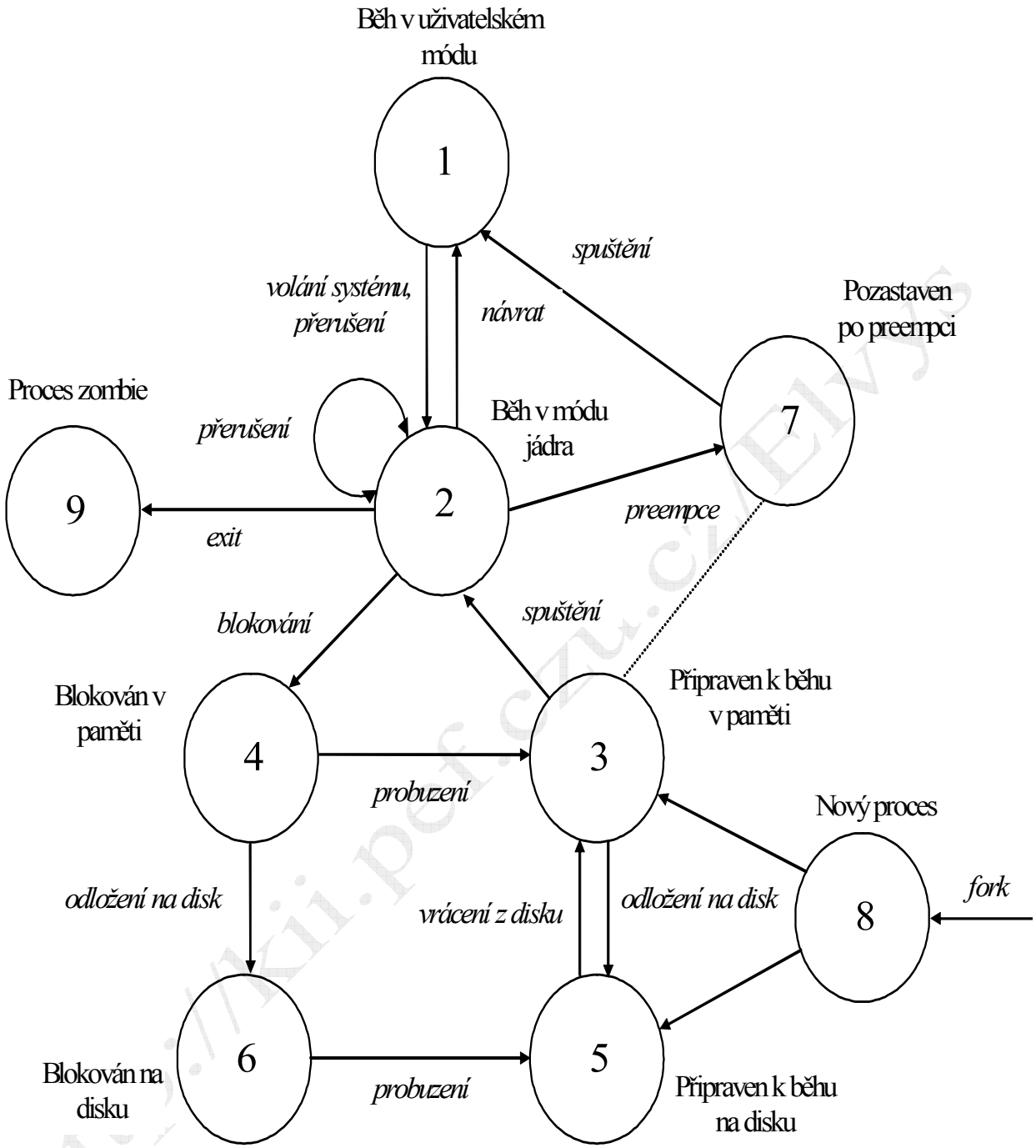
Volby:

`-l`	dlouhý výpis
`-u` *`uživatel`*	vypíší se všechny procesy uživatele

`jobs`

zobrazí stavy procesů

http://kii.fei.zcu.cz/EIOS



Obr. 5 Stavy procesů.

http://kii.pef.czu.cz/Elvis

Signály

Základním prostředkem, pomocí kterého lze ovlivnit další činnost procesu, jsou signály. ***Signál*** může procesu zaslat jádro systému nebo jiný proces. Signály jsou označeny kladnými čísly a je jich obvykle 32.

Pokud proces potřebuje zaslat signál jinému procesu, zavolá službu systému ***kill()***. Procesu lze také zaslat signál systémovým programem **kill** .

kill [-*signal*] *PID procesu(ů)*	*(podle výpisu příkazem ps)*
kill [-*signal*] %*číslo procesu*	*(podle výpisu příkazem jobs)*

Pokud v programu **kill** není uvedeno žádné číslo signálu, program procesu zašle signál 15.

Signály lze použít i pro rušení procesů. Pokud totiž příchod zaslaného signálu není ošetřen, proces skončí. Pokud chceme mít jistotu, že zasíláme procesu signál, který není ošetřen a tudíž, že proces bude určitě ukončen, použijeme signál 9. Příchod tohoto signálu totiž ošetřit nelze. Podobně nelze ošetřit signál 19. Po příchodu tohoto signálu je ale proces zablokován. Následným zasláním signálu 18 může být odblokován a posléze opět zařazen k zpracování.

Uživatel může zasílat signály pouze těm procesům, které vlastní, tj. pouze těm procesům, které mají jeho *UID*. Superuživatel může zasílat signály všem procesům, kromě několika základních systémových procesů, jejichž zrušení by vedlo k havárii systému (např. nemůže zrušit proces **swapper** a **scheduler**).

Každý proces má svého ***vlastníka-uživatele*** (***individuálního vlastníka***) a ***vlastníka-skupinu*** (***skupinového vlastníka***). Vlastník-uživatel je zaznamenán v položce *UID* a vlastník-skupina v položce *GID*. Po připojení uživatele program **login** nastaví své *UID* na *UID* uživatele, který se připojil a své *GID* na *GID* jeho základní skupiny. Proto **login shell** a následně i všechny procesy, které uživatel spustí jsou vlastněny tímto uživatelem a skupinou, do které uživatel patří.

Příklad 4:

Pomocí systémového programu **sleep** spusťte na pozadí 3 procesy. Po spuštění procesů si příkazem **ps** ověřte, že jsou běžící. Zjistěte název a *PID* programu, který je spustil. Nakonec všechny procesy zrušte příkazem **kill**.

Návod:

Program **sleep** se spustí takto:

sleep *time*

Program po svém spuštění požádá systém o zablokování na zadaný počet sekund. Po uplynutí této doby je systémem odblokován a skončí.

http://kiv.zcu.cz/~vys

Příklad 5: Tento příklad slouží k procvičení práce s procesy. Na pozadí spusťte dva procesy řízené programem **sleep** . Poprvé spusťte program **sleep** s parametrem 1000 a podruhé s parametrem 2000. Potom:

1. Převeďte běh procesu sleep 2000 na popředí.
2. Zastavte jeho běh na popředí.
3. Spusťte jej opět na pozadí.
4. Zobrazte aktuální stav obou procesů.
5. Proces sleep 1000 převeďte na popředí
6. Ukončete jeho běh na popředí

Řešení:

sleep 1000 &	*spuštění procesu sleep 1000 na pozadí*
[1] 100	
sleep 2000 &	*spuštění procesu sleep 2000 na pozadí*
[2] 101	
fg %2	*převedení procesu sleep 2000 na popředí*
<CTRL-Z>	*pozastavení procesu sleep 2000 na popředí*
bg %2	*spuštění procesu sleep 2000 na pozadí*
jobs	*zobrazení stavu procesů*
fg %1	*převedení procesu sleep 1000 na popředí*
<CTRL-C>	*ukončení procesu sleep 1000*

Příklad 6: Vyzkoušejte pozastavování a spouštění procesů zasíláním signálů.

Návod:

kill -l	*zobrazení seznamu signálů*
kill -SIGSTOP *PID*	*zastavení procesu*
kill -SIGCONT *PID*	*spuštění zastaveného procesu*
kill –SIGKILL PID	*ukončení procesu*

Před zasíláním signálů je třeba zjistit název nebo číslo signálů SIGSTOP, SIGCONT a SIGKILL. V různých systémech mohou být různé.

http://kii.pef.czu.cz/Elvys

Jazyk C

Základním programovacím prostředkem v unixovém prostředí je jazyk C. V něm je napsána podstatná část operačního systému a v něm je také realizována většina větších programovacích projektů (překladače, databázové systémy atd.).

Převod zdrojového programu v jazyce C do strojového programu probíhá ve třech fázích:.

1. ***Zpracování preprocesorem***. Spočívá především v náhradě symbolických konstant číselnými hodnotami, v rozvoji maker a v připojení dalších zdrojových souborů. Připojují se především tzv. **hlavičkové soubory**, které obsahují deklaraci standardních knihovních funkcí a definici symbolických konstant. Hlavičkové soubory jsou obvykle v adresáři `/usr/include` . Názvy souborů, ve kterých jsou uloženy, končí znaky `.h` a připojují se direktivou `#include`. Například

```
#include <stdio.h>
```

Název zdrojových programů v jazyce C musí končit řetězcem znaků `.c` .

2. ***Překlad kompilátorem do relativních modulů***. Jména relativních modulů mají koncovku `.o` . Relativní moduly mohou být organizovány do souborů se speciální strukturou, tzv. knihoven. Sestavovací program tyto knihovny prohledává a při sestavování výsledného programu požadované funkce připojuje. Název knihoven musí končit znaky `.a`. Knihovny jsou v adresáři `/usr/lib`. Zakládání a organizace knihoven se provádí systémovým programem **ar** .

3. ***Sestavení*** sestavovacím programem **ld**.

Překlad zdrojového programu do strojového kódu se provádí programem **cc**. Tento program provede preprocesorové zpracování a kompilaci a nakonec zajistí zpracování zkompilovaného programu sestavovacím programem **ld**. Pokud není specifikováno jinak, je cílový program nazván `a.out`. Při sestavování programu je automaticky prohledávána pouze základní knihovna jazyka C `/usr/lib/libc.a`. Prohledávání dalších knihoven je třeba programu **cc** zadat. Např. pokud zdrojový program obsahuje funkci pro výpočet funkce sinus, tj. funkci *sin()* , která je uložena v matematické knihovně `/usr/lib/libm.a` , je třeba program **cc** spustit s těmito parametry:

```
cc prog.c -lm
```

nebo

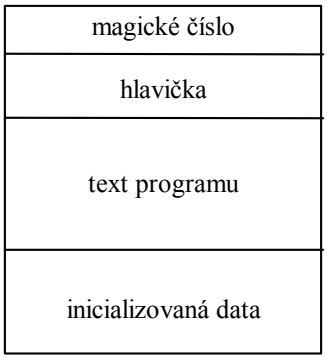
```
cc prog.c  -L/usr/lib/libm.a
```

Pokud požadujeme, aby přeložený program měl jiné jméno než `a.out`, např. `prog.exe`, lze **cc** parametrizovat takto:

```
cc prog.c -lm -o prog.exe
```

http://kip.zcu.cz/Elvis

Po překladu je cílový program uložen na disku. Struktura cílového programu je uvedena na obr. 6. Magické číslo tvoří první dva byty. Podle hodnoty magického čísla systém pozná, zda se jedná o cílový modul. Při požadavku na spuštění cílového modulu (služba *exec()*) systém nejdříve zkontroluje magické číslo. V případě nesouhlasu program nespustí. Hlavička obsahuje údaje o velikosti textu programu a inicializovaných a neinicionalizovaných dat a některé další informace, nutné pro ladění programu. Oblast neinicializovaných dat není součástí cílového modulu. Při umístění do hlavní paměti jí jádro reservuje potřebné místo a obsah celé, pro ni reservované paměti, vynuluje.



Obr. 6 Struktura cílového programu na disku.

Příklad 7:
Vytvořte program v jazyce C, který vypíše na obrazovku pozdrav „*Ahoj, já jsem program v jazyce C*“. Výsledný program přeložte do spustitelné podoby a uložte pod jménem `Ahoj`.

Řešení:

```
cat > program.c

#include <studio.h>
main()
{
  Printf(„Ahoj, já jsem program v jazyce c“);
}

<CTRL+C>

cc program.c            - výsledek bude v souboru a.out

cc program.c –o Ahoj    - výsledek bude v souboru Ahoj

./Ahoj
```

http://kii.pef.czu.cz/Elvys