VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYLEPŠENÍ IMPLEMENTACE KONZOLY VE FREEBSD

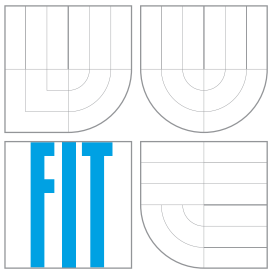
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR JURÁSEK

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYLEPŠENÍ IMPLEMENTACE KONZOLY VE FREEBSD

IMPROVING CONSOLE IMPLEMENTATION IN FREEBSD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR JURÁSEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. RUDOLF ČEJKA
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií
Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Petr Jurásek**
Obor: Informační technologie
Téma: **Vylepšení implementace konzoly ve FreeBSD**
Kategorie: Operační systémy

Pokyny:

1. Seznamte se se zdrojovým kódem operačního systému FreeBSD.
2. Vytvořte testovací skript, který umožní testovat nedostatky zjištěné u současné implementace konzoly ve FreeBSD.
3. Implementujte opravné změny, které zjištěné nedostatky odstraní.

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

Tato bakalářská práce se zaměřuje na implementaci konzoly v jádře FreeBSD. Snaží se odstranit její nedostatky. Zaměřuje se zejména pak na práci s vyznačenými bloky dat myší, kdy při větších změnách na obrazovce nezmizí, důslekem toho označuje úplně jiný text, než se kterým se pracovalo původně.

Klíčová slova

FreeBSD, jádro, konzole, oprava, implementace konzoly, C

Abstract

This bachelor's thesis consider problems of console implementation in FreeBSD. Study the implementation console in FreeBSD from sources code. Aspire solving deficiencies of console implementation. This bachelor's thesis focus on work with a marked section by mouse.

Keywords

FreeBSD, kernel, syscons, patch, C, console implementation

Citace

Petr Jurásek: Vylepšení implementace konzoly ve FreeBSD, bakalářská práce, Brno, FIT VUT v Brně, 2007

Vylepšení implementace konzoly ve FreeBSD

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rudolfa Čejky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Jurásek
15. května 2007

Poděkování

Děkuji svému vedoucímu práce panu Ing. Rudolfovi Čejkovi za poskytnutí odborné pomoci.

Obsah

Kapitola 1

Úvod

Hlavním cílem této bakalářské práce je analýza implementace konzoly ve FreeBSD a implementace oprav pro práci s vyznačeným blokem myší.

Kapitola 2 se zabývá popisem instalace operačního systému FreeBSD, nastavením operačního systému a seznámením se se zdrojovými kódy. Kapitola 3 se zabývá analýzou ESC sekvencí, testovacím skriptem a výsledky testování. Kapitola 4 se zabývá popisem podmínek, dále popisuje implementační změny v původním kódu a popisuje překlad jádra. Kapitola 5 shrnuje dosažené úspěchy při implementaci opravy chyb při práci s vyznačenými bloky. V dodatku A uvádím kompletní testovací skript pro otestování ESC sekvencí na konzoly.

Kapitola 2

Teoretická část

Tuto bakalářskou práci jsem si zvolil proto, abych se hlouběji seznámil s operačním systémem FreeBSD.

Práce se zaměřuje na implementaci konzoly v jádře operačního systému FreeBSD. Jedná se hlavně o odstranění nedostatků při práci s vyznačenými bloky dat myší, kdy při větších změnách na obrazovce nezmizí, důslekem toho označuje úplně jiný text, než se kterým se pracovalo původně.

Historie BSD systému začíná v počátcích AT&T unixu, pokračuje vývojem na kalifornské univerzitě v Berkley, kde vzniká první operační systém BSD. FreeBSD projekt odstartoval roku 1993 po převzetí zdrojových kódů z projektu 386BSD, který nebyl dlouho vyvíjen. Více o historii BSD systému si můžete přečíst např. v [1].

Mým prvním krokem, který jsem podnikl bylo seznámení se na stránkách projektu FreeBSD (http://www.freebsd.org) s příručkou k operačnímu systému FreeBSD [2].

Pokračoval jsem stažením instalačních CD z veřejně přístupného FTP, abych se mohl začít věnovat instalaci systému. Pro zájemce uvádím adresu odkud si můžete stáhnout ISO obraz instalačních CD operačního systému FreeBSD ftp://ftp.freebsd.org.

2.1 Instalace operačního systému FreeBSD

Před instalací je vhodné mít rozdělený disk na více částí, abyste nepřišli o již zaběhnutý systém, pokud ovšem nemáte v úmyslu na svém počítači provozovat jen operační systém FreeBSD.

Doporučuji instalovat operační systém FreeBSD za oddíly s operačním systémem Linux, aby nedošlo ke ztrátě dat, díky logickým oddílům disku v operačním systému FreeBSD. Pokud tak neučiníte zavadeč má problém zavést operační systém Linux, díky chybným přepočtům, kde se nachází bootovací oddíl.

Před zahájením samotné instalace je dobré vědět, jaký hardware váš počítač používá, abyste mohli zadat správnou konfiguraci. Pokud to nevíte je dobré si to zjistit a psát si poznámky, aby jste si vše nemuseli pamatovat. Je pravděpodobné že se vám zjištěné informace o vašem stroji budou hodit, a proto je dobré si papírek s poznačeným hardwarem počítače přípevnit ke stroji, aby jste je opětovně nemuseli vyhledávat.

Samotná instalace je velmi jednoduchá. Po vložení instalačního CD se načte zavaděč systému. Po naběhnutí zavaděče systému se zobrazí volba pro výběr země. Zvolíme položku Czech Republic. Pokračujeme výběrem mapování klávesnice, můžeme zvolit americkou či českou, potvrdíme výběr. Pokračujeme položkou Standard, která značí standardní in-

stalaci systému. Pomocí programu `fdisk` vybereme volné místo na disku pro instalaci. Následuje volba instalace zavaděče systému, můžeme si vybrat z možností nainstalovat zavaděč systému, zásah do MBR bez zavaděče systému, nebo žádnou. Výběr distribuce, zvolil jsem možnost kern-developer, protože budu upravovat zdrojové kódy jádra, spíše zdrojové kódy systémové konzoli. Následuje výběr instalačního média, volím možnost CD/DVD. Probíhá instalace, chvíli to trvá. Můžete si zatím odskočit na svačinu, než se to stihne nainstalovat. Následuje nastavení sítové karty a myši. Dále se zobrazí možnost vybrat si balíky které chceme mí nainstalované. Zajíždím do nabídky editorů a vybírám `vim`, abych mohl editovat zdrojové kódy. Zobrazí se výzva ke vložení druhého CD. Pokračuji vytvořením uživatelského účtu a zadáním hesla správce. Tím instalace končí.

2.2 Po instalační nastavení

Po instalační nastavení můžeme provést v programu `sysinstall`.

```
# sysinstall
```

Zde zvolíme volbu configure a můžeme nastavovat chování systému po startu, přiinstalovat softwarové balíčky, spustit fdisk, apod. Pro další práci je vhodné nastavit myš. Myš zapneme volbou *Mouse→Enable*.

Po nainstalování FreeBSD, se mi operační systém spouštěl velmi dlouhou dobu. Bylo to tím, že jsem měl zapnutý sendmail, i když jsem na svém počítači poštovní server neprovozoval. Po vypnutí sendmailu, jsem pocítil rychlejší náběh operačního systému. Sendmail vypneme připsáním následujících řádků do konfiguračního souboru /etc/rc.conf, zápis provádíme s oprávněním uživatele root:

```
sendmail_enable='NONE'
sendmail_flags=''
sendmail_outbound_enable='NO'
sendmail_submit_enable='NO'
sendmail_msp_queue_enable='NO'
```

Po znovu naběhnutí operačního systému, by se nemělo na obrazovce objevovat sm-mta démon, který funguje jako MTA a sm-msp-queue démon, který odesílá maily z fronty /var/spool/clientmqueue

2.3 Zdrojový kód

Zdrojové kódy operačního systému FreeBSD jsou dostupné v adresáři /usr/src, pokud je tento adresář prázdný, nebyly zdrojové kódy vybrány při instalaci. V tom případě je možné ze zkopírovat z instalačního cd, nebo stáhnout z internetu z ftp serveru.

Na instalačním cd nalezneme zdrojové kódy v adresáři /mnt/cdrom/verze/src. Nás bude zajímat větev /usr/src/sys, kde se nachází zdrojové kódy pro překlad jádra operačního systému a zároveň i pro systémovou konzoli. Potřebné zdrojové kódy větvě sys z cd nainstalujeme následovně:

```
# mount /dev/cd0 /mnt/cdrom
# cd /mnt/cdrom/verze/src
# sh install.sh sys
```

Po nainstalování zdrojových kódu jádra, můžeme podívat kde se nachází zdrojové kódy pro systémovou konzoli. Zjistíme, že zdrojové kódy pro konzoli se nachází v adresáři /usr/src/sys/dev/syscons.

Pokud si necháme pomocí příkazu `ls` vypsat obsah adresáře, zjistíme, že nachází několik dalších adresářů a souborů. V jednotlivých adresářích se nacházejí zdrojové kódy pro spořiče obrazovky.

```
# ls -F /usr/src/sys/dev/syscons
apm/    fade/   rain/       scterm-dumb.c scvesactl.c snake/    syscons.h
blank/  fire/   scgfbrndr.c scterm-sc.c   scvgarndr.c soubor    sysmouse.c
daemon/ green/  schistory.c scterm.c      scvidctl.c  star/     warp/
dragon/ logo/   scmouse.c   sctermvar.h   scvtb.c     syscons.c
```

Zdrojové soubory jsou logicky pojmenovány jak vidíme ve výpisu. Zdrojové soubory jsou v jazyce C, což můžeme odhadnout podle přípony c v názvech souborů. Můžeme se pokusit také odhadnout, co se ve kterém souboru nachází. Pro bližší zkoumání si soubory můžeme otevřít ve `vim`u, a blíže se seznámit s jejich obsahem.

```
# cd /usr/src/sys/dev/syscons
# vim *.c *.h
```

Pro bližším prohlédnutí souborů, se můžeme zaměřit na soubor `scterm-sc.c`, kde se nacházejí ESC sekvence, které budeme následně analyzovat. Po chvilce čtení zdrojového kódu zjistíme, že je velmi dobře čitelný. Nejsou v něm použity žádné podivné konstrukce jen jednoduchý jazyk C.

Kapitola 3

Analýza

3.1 ESC sekvence

Znak ESC (escape, 27, 033, 0x1b) se používá pro definici tzv. ESC sekvencí používaných pro rozšíření ASCII kódu pro různé účely. Jeden nebo několik znaků následujících znak ESC nejsou interpretovány jako ASCII kódy, ale mohou mít speciální význam, mohou definovat pozici kurzoru na obrazovce terminálu, definovat velikost fontu používaného tiskárnou. Organizace ANSI definovala sekvence určené pro ovládání znakových terminálů. Tyto sekvence zahrnují např. posun kurzoru na určitý sloupec a řádek obrazovky.

Příklad:

```
$ printf '\033[32m'
```

Výše uvedená ESC sekvence změní barvu textu terminálu na zelenou, kde je znak ESC zapsán osmičkově 033.

Ve zdrojovém souboru scterm-sc.c umístěném v adresáři /usr/src/sys/dev/syscons/ jsem našel tyto escape sekvence ve funkci `scterm_scan_esc()` pro načítání ESC sekvencí:

1. Byl zaregistrován znak ESC, a dále následuje jeden ze znaků viz tabulka 3.1:

Tabulka 3.1: Znaky následující po znaku ESC

Znak	Popis činnosti
7	ulož pozici kurzoru
8	obnov uloženou pozici kurzoru
[	začátek sekvence znaků
M	přesun kurzor výš o jeden řádek, roluj obrazovku pokud je na vrcholu
c	vymazání obrazovky
Q	není implemetováno
(	ISO-2022: kódování pro japonštinu, čínštinu, korejštinu

2. Byl zaregistrován znak [, dále následuje viz tabulka 3.2
3. Byla znamenána sekvence znaků ESC [0 - 9]+ = viz tabulka 3.3
4. Zaznamenaná sekvence ESC Q
5. Zaznamenaná sekvence ESC (

Tabulka 3.2: Znaky následující po znaku ESC [

Znak	Popis činnosti	Znak	Popis činnosti
0-9	čísla	P	smaž n znaků
;	oddělovač	@	vlož n znaků
=	jdi na sekci 3	S	scroluj n řádků nahoru
A	nahoru n řádků	T	scroluj n řádků dolů
B	dolů n řádků	X	vymaž n znaků v řádku
C	vpravo n sloupců	Z	přesuň se n tabulátorů zpět
D	vlevo n sloupců	‘	přesuň kurzor na sloupec n
E	kurzor na začátku řádku o n řádků dole	a	přesuň kurzor n sloupců vpravo
F	kurzor začíná o n řádků nahoře	d	přesuň kurzor na řádek n
f	pohyb kurzoru	e	přesuň kurzor n řádků dolů
H	pohyb kurzoru	m	změn atributy viz tabulka 3.4
J	vyčisti celou nebo část obrazovky	s	ulož pozici kurzoru
K	vyčisti celý nebo část řádku	u	obnov uložené souřadnice kurzoru
L	vlož n řádků	x	nastaveni barev viz tabulka 3.5
M	smaž n řádků	z	změn virtuální konzoli

Tabulka 3.3: Znaky následující po znaku ESC [0 - 9]+ =

Znak	Popis činnosti
0-9	čísla
;	oddělovač
A	nastav barvu hranicím obrazovky
B	nastav
C	nastav obecný typ a tvar kurzoru
F	nastav popředí adaptéru
G	nastav pozadí adaptéru
H	nastav reverzní popředí adaptéru
I	nastav reverzní pozadí adaptéru
S	nastav dočasný typ a tvar kurzoru

Tabulka 3.4: Atributy

Znak	Popis činnosti
0	vrátit se k výchozímu nastavení
1	tučný text
4	podtržený text
5	blikání
7	přehození
22	odstranň tučnost
24	odstraň podtržení
25	odstraň blikání
27	odstraň přehození
30-37	nastav barvu ansi popředí
39	obnov výchozí barevné ansi popředí
40-47	nastav ansi barvu pozadí
49	obnov výchozí barevné ansi pozadí

Tabulka 3.5: Nastavení barev

Znak	Popis činnosti
0	obnovení výchozího nastavení
1	nastavení ansi pozadí
2	nastavení ansi popředí
3	nastavení adaptéru
5	nastavení reverzního pozadí
6	nastavení reverzního popředí
7	nastavení reverzního adaptéru

3.2 Testovací skript

Testovací skript jsem vytvářel ve skriptovacím jazyce příkazového řádku. Použil jsem jednoduché konstrukce jako `tput, for, read, printf, exit`. Do skriptu jsem nepsal všechny sekvence, ale jen ty, které byly vhodné k otestování. Testování se zaměřilo hlavně na práci s blokem vyznačeným myší, který zůstává na obrazovce a neodpovídá vyznačenému textu. Ukázka testovacího skriptu pro řídící sekvenci ESC [4T – skrolování dolů o 4 řádky:

```
#!/bin/sh
tput clear
for i in 0 1 2 3 4 5 6 7 8 9 a b c d e f
do
  for j in 1 2 3 4 5 6 7 8 9 10
  do
    printf '$i'
  done
  printf '\n'
done
read Q # nacti vstup z klavesnice
tput cm 0 8 # umisteni kurzoru x y
read Q # nacti vstup z klavesnice
printf '\033[4T' # skroluj n radku dolu
read Q # nacti vstup z klavesnice
exit 0
```

Testovací skript můžeme spustit pomocí příkazu `sh` následovně:

```
$ sh skript-term.sh
```

Testovací skript se spustí v shellu. Pokud budeme chtít provést testovací skript v bashi provedeme to následovně:

```
$ ./skript-term.sh
```

Testování probíhalo spuštěním skriptu, vybráním volby ESC sekvence k testování. Vypsal se zkušební text na obrazovku. Myší jsem vybral náhodný blok. Zadal jsem vstup z klávesnice a čekal jestli na provedení sekvence a sledoval jsem jestli to bude mít vliv na chování konzoly.

3.3 Výsledek analýzy

Po otestování ESC sekvencí, jsem dospěl k závěru, že při mazání, vkládání nebo skrolování, kdy se většinou mění text na obrazovce, zůstává označen již neplatný blok dat. Při pohybu kurzoru se text nemění a nepovažuji za chybu, pokud zůstane blok dat vyznačen. Zůstává špatně vyznačený blok po provedení zadané sekvence znaků u těchto ESC sekvencí:

1. Vyčištění části nebo celého řádku
2. Vložení n řádků

3. Smazání n řádků
4. Smazání n znaků
5. Vložení n znaků
6. Skrolování nahoru
7. Skrolování dolů
8. Vymazání n znaků v řádku

Návrhy pro odstranění zjištěných nedostatků:

- Prvním návrhem pro odstranění zjištěných nedostatků bylo navrhnuto odstranění vyznačení.
- Dalším řešením bylo navrhnuto, aby se vyznačení posouvalo spolu s textem, dokud by nezmizel z obrazovky, označení by se zrušilo, ale jen v případě, že nebyl ovlivněn při mazání či vkládání vyznačený blok dat.

Zkusil jsem se podívat blíže na tyto sekvence a podívat se jak jsou implementovány. Zjistil jsem že volají následující funkce ze souboru `sctermvar.h`:

`sc_term_clr_eol()` funkce je volána pro vyčištění části nebo celého řádku

`sc_term_ins_line()` funkce je volána sekvencí pro vložení n řádků, nebo sekvencí pro skrolování nahoru, kdy je pozice y nula.

`sc_term_del_line()` funkce je volána pro sekvenci smazání n řádků, nebo skrolování nahoru, kdy je pozice y nula.

`sc_term_del_char()` funkce je volána pro sekvenci smazání n znaků

`sc_term_ins_char()` funkce je volána pro sekvenci vložení n znaků

Je zajímavé, že funkce `sc_term_ins_line()` a `sc_term_del_line()` jsou volány dvakrát jednou pro vložení čí smazání řádků a podruhé pro skrolování.

Všiml jsem si, že sekvence pro *vymazání n znaků v řádku* nevolá žádnou funkci ze souboru `sctermvar.h`, ale několik jiných funkcích které byly volány ze souboru `sctermvar.h`. Rozhold jsem se toto volání funkcí pro lepší přehlednost upravit a přidat novou in line funkci `sc_term_del_xchar()` do souboru `sctermvar.h`, kterou by tato sekvence volala.

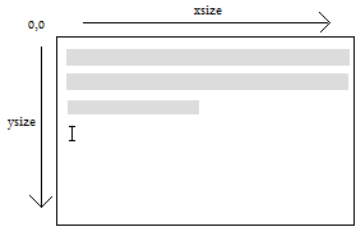
Kapitola 4

Implementace opravné změny

4.1 Určení podmínek

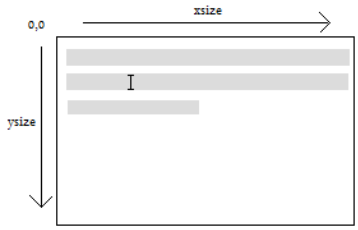
Při prvním pohledu na problém může dojít k následujícím situacím s vyznačeným blokem dat pomocí myši při vkládání/mazání:

- První situace nastane kdy vyznačený blok dat myší se nachází před kurzorem. Chování v této situaci je, že vyznačený blok dat není ovlivněn mazáním či vkládáním, vyznačený blok dat zůstává i nadále vyznačen. Nedochází k žádné změně viz obrázek 4.1.



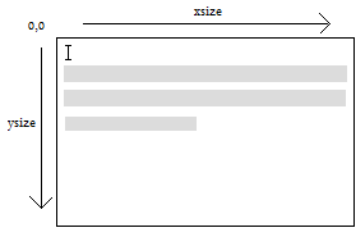
Obrázek 4.1: Kurzor se nachází za vyznačeným blokem

- Další situace nastane pokud vyznačený blok myší zasahuje ovlivněnou oblast. Očekávané chování v tomto případě je, že vyznačený blok dat zmizí. Text, který byl vyznačen, se změnil viz obrázek 4.2.



Obrázek 4.2: Kurzor se nachází uvnitř vyznačeného bloku

- Poslední situace nastane, že vyznačený blok dat myší leží za kurzorem. Vyznačený text není ovlivněn změnou vyznačeného bloku textu, ale posunem viz obrázek 4.3. Vyznačený blok dat se posune společně s vyznačeným textem, pokud část textu, která byla vyznačena, zmizí z obrazovky, zmizí i vyznačení bloku.



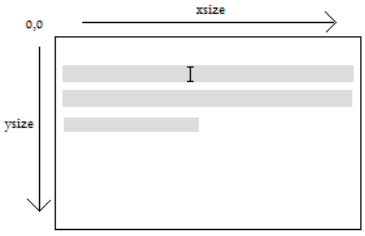
Obrázek 4.3: Kurzor se nachází před vyznačeným blokem

Při skrolování může dojít k následujícím situacím:

- Při skorolování se vyznačený blok myší posouvá po obrazovce spolu s textem. Pokud přesáhne vyznačená oblast hranice monitoru, vyznačení zmizí.

Uvedené podmínky nepokrývají všechny situace, ke kterým může dojít, jen nastiňují nejobvyklejší situace. Pro lepší určení podmínek je nutné se podívat na každou funkci zvlášť a určit pro ni podmínky.

Zaměřme se nyní na podmínky pro vkládání řádků. Podívejme se za jakých situací může dojít k posunu řádků. První situace nastane, kdy se kurzor nachází před vyznačenou oblastí viz obrázek 4.3. Další případ nastane, kdy se kurzor nachází na prvním řádku uvnitř vyznačeného bloku viz obrázek 4.4. Nový řádek se vkládá na řádek na němž se nachází kurzor, proto si posun můžeme dovolit i když se kurzor nachází ve vyznačeném bloku.



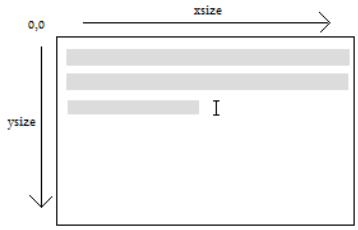
Obrázek 4.4: Kurzor se nachází na prvním řádku uvnitř vyznačeného bloku

Z uvedených obrázků můžeme určit podmínky pro posun řádku. K posunu dochází v případě, pokud se kurzor nachází na stejném řádku jako počátek vyznačeného bloku nebo výše než počátek vyznačeného bloku. Z toho vyplývá, že se u této funkce zaměříme na kontrolu řádků tzn. y hodnot. Matematicky tuto podmínku můžeme zapsat:

$$y_{cursor} \leq y_{mouse\_start}$$

Zbývá určit podmínky, za kterých se vyznačený blok zruší. Vyznačený blok se zruší pokud kurzor leží ve vyznačeném bloku, ale přesto neleží na prvním řádku vyznačeného

bloku jak je znázorněno na obrázku 4.2. Může nastat situace na obrázku 4.5, kdy se kurzor nachází na řádku za koncem vyznačeného bloku. Nový řádek se vloží před tento řádek a dojde k porušení vyznačeného bloku.

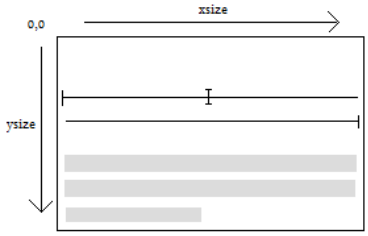


Obrázek 4.5: Kurzor se nachází na řádku za koncem vyznačeného bloku

Z uvedených situací určíme podmínku pro zrušení vyznačeného bloku. Blok zrušíme v případě, že konec vyznačeného bloku leží na stejném řádku jako kurzor, nebo se kurzor nachází uvnitř bloku, ale neleží na prvním řádku vyznačeného bloku. Matematicky tuto podmínku můžeme zapsat:

$$(y_{cursor} = y_{mouse\_end}) \vee ((y_{cursor} < y_{mouse\_end}) \wedge (y_{cursor} > y_{mouse\_start}))$$

Nyní určíme podmínky pro mazání řádků. Určeme za jakých podmínek může dojít k posunu řádků při mazání. K posunu řádků může dojít v případě, že konec mazané oblasti se nachází před začátkem vyznačeného bloku viz obrázek 4.6. Nejsou další situace, při kterých by mohlo dojít k posunu bloku.



Obrázek 4.6: Konec mazané oblasti se nachází před začátkem vyznačeného bloku

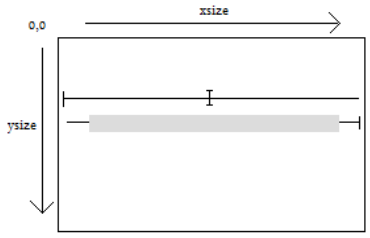
Matematicky tuto podmínku vyjádříme:

$$y_{cursor} + n - 1 < y_{mouse\_start}$$

Dále určíme podmínky pro zrušení vyznačeného bloku. U mazání si musíme dát pozor, že kromě pozice kurzoru, musíme kontrolovat konec mazané oblasti. Zrušení vyznačeného bloku nastane v případě, že mazaná oblast obsahuje celý vyznačený blok viz obrázek 4.7.
Tuto podmínku můžeme zapsat následovně:

$$(y_{cursor} < y_{mouse\_start}) \wedge (y_{cursor} + n > y_{mouse\_end})$$

Dále může nastat situace, že kurzor leží na stejném řádku, kde se nachází konec, či začátek vyznačeného bloku viz obrázek 4.5. V této situaci se vyznačený blok zruší. Tuto



Obrázek 4.7: Celý vyznačený blok leží v mazané oblasti

podmínku můžeme vyjádřit:

$$(y_{cursor} = y_{mouse\_start}) \vee (y_{cursor} = y_{mouse\_end})$$

V další situaci se může kurzor, nebo konec mazané oblasti nacházet uvnitř vyznačeného bloku. Podmínku vyjádříme:

$$((y_{cursor} > y_{mouse\_start}) \wedge (y_{cursor} < y_{mouse\_end})) \vee ((y_{cursor}+n > y_{mouse\_start}) \wedge (y_{cursor}+n < y_{mouse\_end}))$$

Určeme podmínky pro vkládání znaků. Při vkládání znaků dojde ke zrušení vyznačeného bloku pokud se kurzor nachází uvnitř vyznačeného bloku. K posunu bloku dochází při vkládání, kdy se kurzor leží na stejném řádku před začátkem vyznačeného bloku a zároveň konec vyznačeného bloku se nachází na stejném řádku jako začátek. Pokud se kurzor nachází výše o řádek než začátek vyznačeného bloku k posunu nedochází.

Určeme podmínky pro mazáni znaků. Při mazání znaků musí brát i v úvahu konec mazané oblasti. Pokud se konec mazané oblasti nebo kurzor nachází ve vyznačeném bloku dojde k odstranění vyznačeného bloku. Speciální případ nastává kdy, vyznačený blok je menší než počet mazaných znaků a leží uvnitř mazané oblasti, dojde opět k odstranění vyznačeného bloku. K posunu vyznačeného bloku dochází pokud leží na stejném řádku za koncem mazané oblasti a zároveň se konec vyznačeného blok se nachází na tomto řádku.

Podmínky pro mazání znaků v řádku. Při mazání znaků v řádku stačí určit podmínky pro odznačení bloku. K posunu znaků nedochází. Při mazání bereme v úvahu i konec mazané oblasti. Pokud se konec mazané oblasti nebo kurzor nachází ve vyznačeném bloku dojde k odstranění vyznačeného bloku. Speciální případ nastává kdy, vyznačený blok je menší než počet mazaných znaků a leží uvnitř mazané oblasti, dojde opět k odstranění vyznačeného bloku.

Podmínky pro vyčištění části nebo celého řádku. Zde opět bereme v úvahu dvě hodnoty. Záleží na případě kterou část řádku mažeme. V tomto případě nedochází k posunu textu. Určujeme podmínky jen pro odstranění vyznačeného bloku. Podmínky jsou podobné jako pro mazání znaků v řádku, jen se liší hodnoty, které budeme kontrolovat v jednotlivých případech:

Mažeme od kurzoru po konec řádku V tomto případě bereme v úvahu hodnoty kurzor a konec řádku

Mažeme od začátku řádku po kurzor V tomto případě bereme v úvahu hodnoty začátek řádku a kurzor

Mažeme celý řádek V tomto případe bereme v úvahu celý řádek, tedy hodnoty začátek a konec řádku.

4.2 Implementace

Prvním krokem implementace bylo přidání nové in line funkce `sc_term_del_xchar()` do souboru `sctermvar.h` s přidaným voláním funkce `sc_remove_cutmaking` pro odstranění vyznačeného bloku:

```
static __inline void sc_term_del_xchar(scr_stat *scp, int n, int ch, int attr);

static __inline void
sc_term_del_xchar(scr_stat *scp, int n, int ch, int attr)
{
  if (n < 1)
    n = 1;
  if (n > scp->xsize - scp->xpos)
    n = scp->xsize - scp->xpos;
  sc_vtb_erase(&scp->vtb, scp->cursor_pos, n, ch, attr);
  mark_for_update(scp, scp->cursor_pos);
  mark_for_update(scp, scp->cursor_pos + n - 1);
}
```

Tělo funkce jsem převzal se souboru `scterm-sc.c` a upravil jsem volání na:

```
case 'X': /* erase n characters in line */
  sc_term_del_xchar(scp, tcp->param[0], sc->scr_map[0x20], tcp->cur_attr);
  break;
```

Další změně ke které došlo ve zdrojových souborech bylo dopsání volání funkce pro odstranění vyznačeného bloku v souboru `sctermvar.h` do funkcí `sc_term_ins_line, sc_term_del_line, sc_term_del_char, sc_term_ins_char, sc_term_clr_eol`:

```
...
sc_remove_cutmakring(scp);
...
```

Funkce by měla zabránit, aby na obrazovce zůstával vyznačený blok myší, který by mohl časem označovat neplatný blok textu.

V dalším kroku jsem určil jak se budou v dané funkci implementovat podmínky. Podle podmínek jsem si určil funkce, které by vraceli polohu kurzoru. Jednalo by se o funkce, které by vracely typ int, který byl reprezentoval logickou pravdu či nepravdu. Po lepším prostudování zdrojového kódu, jsem objevil funkci `sc_inside_cutmark()`, která určuje zda kurzor, leží v oblasti bloku dat vyznačených myší. Pro určení polohy kurzoru jsem si stanovil návrh následujících funkcí:

`sc_inside_cutmark()` již implementovaná funkce v souboru `scmouse.c`. Funke zjišťuje zda zadaná poloha kurzoru se nachází uvnitř vyznačeného bloku myší.

`sc_after_cutmark()` funkce vrací hodnotu *true*, reprezentovanou typem int, pokud kurzor leží za vyznačenou oblastí myší.

`sc_before_cutmark()` funkce vrací hodnotu *true*, reprezentovanou typem int, pokud kurzor leží před vyznačenou oblastí myší.

Zdálo se mi zbytečné kontrolovat jestli má vyznačený blok zůstat v pořádku. Následně jsem implementoval funkci `sc_before_cutmark()`, pro zjištění zda kurzor leží před vyznačenou oblastí myší:

```
int
sc_before_cutmark(scr_stat *scp, int pos)
{
    int start;

    if (scp->mouse_cut_end < 0)
    return FALSE;
    if (scp->mouse_cut_start <= scp->mouse_cut_end) {
    start = scp->mouse_cut_start;
    } else {
    start = scp->mouse_cut_end;
    }
    return (start > pos);
}
```

Dále jsem si na implementoval funkci `sc_after_cutmark()`, pro zjištění zda zadaná pozice leží za vyznačenou oblastí myší:

```
int
sc_after_cutmark(scr_stat *scp, int pos)
{
    int end;

    if (scp->mouse_cut_end < 0)
    return FALSE;
    if (scp->mouse_cut_start <= scp->mouse_cut_end) {
    end = scp->mouse_cut_end;
    } else {
    end = scp->mouse_cut_start - 1;
    }
    return (end < pos);
}
```

Procházel jsem soubor `scmouse.c` jestli náhodou neobjevím funkci pro posun bloku. Po dlouhém hledání jsem žádnou funkci nenašel. Rozhodl jsem se na implementovat vlastní funkci pro posun bloku nazvanou `sc_move_cutmarking()`:

```
void
sc_move_cutmarking(scr_stat *scp, int n)
{
    if (scp->mouse_cut_end >= 0) {
      mark_for_update(scp, scp->mouse_cut_start);
      mark_for_update(scp, scp->mouse_cut_end);
      scp->mouse_cut_start = scp->mouse_cut_start + n;
      scp->mouse_cut_end = scp->mouse_cut_end + n;
```

```
      mark_for_update(scp, scp->mouse_cut_start);
      mark_for_update(scp, scp->mouse_cut_end);
    }
}
```

Pro funkci `sc_term_ins_line()` vkládání řádků kontroluji jestli se kurzor nachází za koncem oblasti vyznačeným blokem myší, následně dojde k posunu textu nebo kurzor leží v dané oblasti vyznačeným blokem myší, kdy dojde k odznačení bloku.

```
static __inline void
sc_term_ins_line(scr_stat *scp, int y, int n, int ch, int attr, int tail)
{
...
int y_start, y_end;

  if (scp->mouse_cut_start < scp->mouse_cut_end) {
    y_start = scp->mouse_cut_start/scp->xsize;
    y_end = scp->mouse_cut_end/scp->xsize;
  } else {
    y_start = scp->mouse_cut_end/scp->xsize;
    y_end = scp->mouse_cut_start/scp->xsize;
  }

  if ( y == 0 ) {
    if(y_end + n > scp->ysize)
      sc_remove_cutmarking(scp);
    else
      sc_move_cutmarking(scp, n*scp->xsize); /* move cutmarking */
  } else {
    if ((scp->ypos <= y_start)) {
      if(y_end + n > scp->ysize)
        sc_remove_cutmarking(scp);
      else
        sc_move_cutmarking(scp, n*scp->xsize); /* move cutmarking */
    }
    if ((scp->ypos <= y_end && scp->ypos > y_start))
      sc_remove_cutmarking(scp);
   }
...
}
```

Pro funkci `sc_term_del_line()` mazání řádků kontroluji jestli se kurzor nachází uvnitř vyznačeného bloku nebo konec mazané oblasti, nebo jestli kurzor leží před vyznačeným blokem a zároveň konec mazané oblasti leží za vyznačeným blokem myší, kdy dojde k odznačení bloku, nebo konec mazané oblasti leží před vyznačeným blokem myší, dojde k posunu vyznačeného bloku.

```
static __inline void
sc_term_del_line(scr_stat *scp, int y, int n, int ch, int attr, int tail)
```

```
{
...
int y_start, y_end;

  if (scp->mouse_cut_start < scp->mouse_cut_end) {
    y_start = scp->mouse_cut_start/scp->xsize;
    y_end = scp->mouse_cut_end/scp->xsize;
  } else {
    y_start = scp->mouse_cut_end/scp->xsize;
    y_end = scp->mouse_cut_start/scp->xsize;
  }

  if ( y == 0 ) {
    if (y_start - n < 0)
      sc_remove_cutmarking(scp);
    else
      sc_move_cutmarking(scp, n*scp->xsize*-1); /* move cutmarking */
  } else {
    if ((scp->ypos + n <= y_start))
      if (y_start - n < 0)
        sc_remove_cutmarking(scp);
      else
        sc_move_cutmarkig(scp, n*scp->xsize*-1);
    if ((scp->ypos <= y_end && scp->ypos >= y_start) ||
        (scp->ypos + n - 1 <= y_end && scp->ypos + n - 1 >= y_start) ||
        (scp->ypos < y_start && scp->ypos + n - 1 >= y_end))
      sc_remove_cutmarking(scp);
    }
...
}
```

Pro funkci `sc_term_ins_char()` vkládání znaků kontroluji jestli se kurzor nachází uvnitř vyznačeného bloku myší, kdy se vyznačení bloku zruší, nebo zda leží kurzor před vyznačeným blokem a dojde k posunu.

```
static __inline void
sc_term_ins_char(scr_stat *scp, int n, int ch, int attr)
{
...
int y_start, y_end;

  if (scp->mouse_cut_start < scp->mouse_cut_end) {
    y_start = scp->mouse_cut_start/scp->xsize;
    y_end = scp->mouse_cut_end/scp->xsize;
  } else {
    y_start = scp->mouse_cut_end/scp->xsize;
    y_end = scp->mouse_cut_start/scp->xsize;
  }
```

```
if (sc_inside_cutmark(scp, scp->cursor_pos - 1))
  sc_remove_cutmarking(scp);
if (sc_before_cutmark(scp, scp->cursor_pos - 1) &&
    (scp->ypos == y_start) && (scp->ypos == y_end))
  sc_move_cutmarkig(scp, n);
...
}
```

Pro funkci `sc_term_del_char()` mazání znaků kontroluji zda se kurzor nachází uvnitř vyznačeného bloku, nebo jestli se uvnitř vyznačeného bloku nachází konec mazané oblasti. Ještě je tu možnost, že mazaná oblast je vetší než vyznačený blok a proto kontroluji jestli mazaná oblast neobsahuje vyznačený blok, kdy dochází ke zrušení vyznačeného bloku. Kontroluji jestli nedochází k posunu vyznačeného bloku, kdy vyznačený blok leží za koncem mazané oblasti.

```
static __inline void
sc_term_del_char(scr_stat *scp, int n, int ch, int attr)
{
...
int y_start, y_end;

  if (scp->mouse_cut_start < scp->mouse_cut_end) {
    y_start = scp->mouse_cut_start/scp->xsize;
    y_end = scp->mouse_cut_end/scp->xsize;
  } else {
    y_start = scp->mouse_cut_end/scp->xsize;
    y_end = scp->mouse_cut_start/scp->xsize;
  }

if ((sc_inside_cutmark(scp, scp->cursor_pos) ||
     sc_inside_cutmark(scp, scp->cursor_pos + n)) ||
    (sc_before_cutmark(scp, scp->cursor_pos + n - 1) &&
     y_start != y_end) ||
    (sc_before_cutmark(scp, scp->cursor_pos) &&
     sc_after_cutmark(scp, scp->cursor_pos + n)))
    sc_remove_cutmarking(scp);

if (sc_before_cutmark(scp, scp->cursor_pos + n) &&
    (scp->ypos == y_start) && (scp->ypos == y_end))
    sc_move_cutmarkig(scp, n*-1);
...
}
```

Pro funkci `sc_term_del_xchar()` smazání znaků v řádku kontroluji jestli leží kurzor nebo konec mazané oblasti uvnitř vyznačeného bloku, či mazaná oblast v sobe obsahuje vyznačený blok. Zde nekontroluji jestli mazaná oblast leží před vyznačeným blokem, při mazání se text neposouvá.

```
static __inline void
```

```
sc_term_del_xchar(scr_stat *scp, int n, int ch, int attr)
{
...
if ((sc_inside_cutmark(scp, scp->cursor_pos) ||
     sc_inside_cutmark(scp, scp->cursor_pos + n - 1)) ||
    (sc_before_cutmark(scp, scp->cursor_pos) &&
     sc_after_cutmark(scp, scp->cursor_pos + n - 1)))
  sc_remove_cutmarking(scp);
...
}
```

Pro funkci `sc_term_clr_eol()` čištění části nebo celého řádku je to ještě trochu komplikovanější záleží zda mažu celý řádek nebo jen od kurzoru ke konci řádku, či od kurzoru k začátku řádku. Potřebuji si zde určit počátek a konec řádku. Při provádění této funkce nedochází k posunu. Definuji jen podmínku za které se má vyznačený blok smazat.

```
static __inline void
sc_term_clr_eol(scr_stat *scp, int n, int ch, int attr)
{
  switch (n) {
    case 0: /* clear form cursor to end of line */
      if (sc_inside_cutmark(scp, scp->cursor_pos) ||
          sc_inside_cutmark(scp, scp->cursor_pos + scp->xsize - 1 - scp->xpos)||
         (sc_before_cutmark(scp, scp->cursor_pos) &&
          sc_after_cutmark(scp, scp->cursor_pos + scp->xsize - 1 - scp->xpos)))
        sc_remove_cutmarking(scp);
      break;
    case 1: /* clear from beginning of line to cursor */
      if (sc_inside_cutmark(scp, scp->cursor_pos) ||
          sc_inside_cutmark(scp, scp->cursor_pos - scp->xpos) ||
         (sc_before_cutmark(scp, scp->cursor_pos - scp->xpos) &&
          sc_after_cutmark(scp, scp->cursor_pos)))
        sc_remove_cutmarking(scp);
      break;
    case 2: /* clear entire line */
      if (sc_inside_cutmark(scp, scp->cursor_pos - scp->xpos ) ||
          sc_inside_cutmark(scp, scp->cursor_pos +
            scp->xsize - 1 - scp->xpos) ||
         (sc_before_cutmark(scp, scp->cursor_pos - scp->xpos) &&
          sc_after_cutmark(scp, scp->cursor_pos + scp->xsize - 1 - scp->xpos)))
        sc_remove_cutmarking(scp);
      break;
  }
}
```

4.3 Překlad jádra

Pro otestování provedených změn ve zdrojových souborech je nutné provést překlad jádra, aby se změny projevily při testování provedených změn. Při překladu jádra je vhodné si

vytvořit vlastní konfigurační soubor. Konfigurační soubor nalezneme v adresáři /usr/src/sys/arch/conf, kde arch značí naši architekturu např. i386. Dobrým zvykem je pojmenovat konfigurační soubor podle názvu počítače. Je vhodné si konfigurační soubory ukládat do domovského adresáře superuživatele root, aby při aktualizaci zdrojových kódu nedošlo k jejich přepsání. Vše potřebné zařídíme podle následující skupiny příkazů, příklad je pro architekturu i386, musíte být přihlášen jako root:

```
cd /usr/src/sys/i386/conf
mkdir /root/kernels
cp GENERIC /root/kernels/HB01-223A
ln -s /root/kernels/HB01-223A
```

Nyní můžeme editovat konfigurační soubor HB01-223A

```
vim HB01-223A
```

Po úpravách konfiguračního souboru pokračujeme překladem jádra. Změníme adresář na /usr/src.

```
cd /usr/src
make buildkernel KERNCONF=HB01-223A
make installkernel KERNCONF=HB01-223A
```

Varianta pro inkrementální překlad jádra:

```
make buildkernel -DNO_KERNELCLEAN=yes KERNCONF=HB01-223A
```

Nové jádro se zkopíruje do adresáře /boot/kernel jako /boot/kernel/kernel a staré jádro se přesune do /boot/kernel.old/kernel. Před překladem jádra je vhodné si udělat zálohu a v případě potíží zavést známé funkční jádro.

```
cp -r /boot/kernel /boot/kernel.save
```

V zavaděči je potom možné zavést toto jádro pomocí napsáním těchto řádků:

```
unload all
load /boot/kernel.save/kernel
boot
```

Zde uvádím pro přehled jak dlouho trvá překlad jádra viz tabulka 4.1. Ke srovnání jsem měl k dispozici dva počítače.

Tabulka 4.1: Doba překladu jádra

Procesor	Frekvence	Paměť	Čas
Intel Celron	366 MHz	64 MB	1 h 58 m
AMD Turion64 X2	1,6 GHz	1 GB	15 m 28 s

Kapitola 5

Závěr

Práce se zabývala analýzou implementace konzoly ve FreeBSD a odstraněním jejích nedostatků. Hlavním cílem se staly analýza ESC sekvence pro řízení konzoly a implementace případné opravy pro práci s vyznačeným blokem myší. Můžeme v současné verzi implementace konzoly najít chyby, např. špatná práce s vyznačeným blokem myší.

Při implementaci opravy pro práci s vyznačeným blokem myší byly použity jednoduché konstrukce programovacího jazyka C. Implementace opravy funguje, avšak v některých situacích selhává. Nebyly pořád promyšleny všechny scénáře, ke kterým by mohlo dojít při práci na konzoly. Díky testování byly odhaleny slabiny implementace opravy.

Během implementace opravy se objevily různé problémy, např. překlepy v kódu, kdy nebylo snadné odhalit proč funkce nepracuje, jakby se od ní očekávalo a dává nesmyslné výsledky.

Dodatek A

Skript

Zde uvádím testovací skript, který byl použit při testování ESC sekvencí na konzole.

```
#!/usr/local/bin/bash

#nabidka
printf '1) Scroll up
        2) Scroll down
        3) Reset
        4) Move cursor up
        5) Up n rows
        6) Down n rows
        7) Right n columns
        8) Left n columns
        9) Cursor n lines down
        a) Cursor n lines up
        b) Cursor move
        c) Cursor move
        d) Clear all or part of display
        e) Clear all or part of line
        f) Insert n lines
        g) Delete n lines
        h) Delete n chars
        i) Insert n chars
        j) Erase n chraters in line
        k) Move n tabs to backwards
        l) Move cursor to column n
        m) Move cursor to n columns to the right
        n) Move cursor to row n
        o) Move cursor n rows down
        p) Switch to virtual console n'

read Volba #nacti volbu

# zaplneni obrazovky
tput clear
```

```
for i in 0 1 2 3 4 5 6 7 8 9 a b c d e f
do
  for j in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
  do
    printf '$i'
  done
  printf '\n'
done
# konec zaplneni obrazovky
tput cm 0 8 # nastav kurzor na tuto pozici
read -n1 #cti jakykoliv znak, ne jen 'enter'
#proved zvolenou volbu
case '$Volba' in
  '1') # Scroll up
    printf '\033[3S'
    ;;
  '2') # Scroll down
    printf '\033[3T'
    ;;
  '3') # Reset
    printf '\033c'
    ;;
  '4') # Move cursor up
    printf '\033M'
    ;;
  '5') # Up n rows
    printf '\033[3A'
    ;;
  '6') # Down n rows
    printf '\033[3B'
    ;;
  '7') # Right n columns
    printf '\033[3C'
    ;;
  '8') # Left n columns
    printf '\033[3D'
    ;;
  '9') # Cursor n lines down
    printf '\033[3E'
    ;;
  'a') # Cursor n lines up
    printf '\033[3F'
    ;;
  'b') # Cursor move
    printf '\033[3f'
    ;;
  'c') # Cursor move
    printf '\033[3H'
```

```
    ;;
  'd') # Clear all or part of display
    printf '\033[J'
    ;;
  'e') # Clear all or part of line
    printf '\033[K'
    ;;
  'f') # Insert n lines
    printf '\033[3L'
    ;;
  'g') # Delete n lines
    printf '\033[3M'
    ;;
  'h') # Delete n chars
    printf '\033[3P'
    ;;
  'i') # Insert n chars
    printf '\033[3@'
    ;;
  'j') # Erase n chraters in line
    printf '\033[3X'
    ;;
  'k') # Move n tabs to backwards
    printf '\033[3Z'
    ;;
  'l') # Move cursor to column n
    printf '\033[4''
    ;;
  'm') # Move cursor to n columns to the right
    printf '\033[3a'
    ;;
  'n') # Move cursor to row n
    printf '\033[3d'
    ;;
  'o') # Move cursor n rows down
    printf '\033[3e'
    ;;
  'p') # Switch to virtual console n
    printf '\033[3z'
    ;;
esac
read Q
exit 0
```

Literatura

[1] George V. Neville-Neil Marshall Kirk McKusick. *Design and Implementation of the FreeBSD Operating System.* Addison-Wesley Publishing Company, 2004. ISBN 0-201-70245-2.

[2] WWW stránky. Freebsd handbook. `http://www.freebsd.org/handbook/index.html`.