Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00001/2010
Datum: 01.10.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **IZIDOR MAKUC**

Naslov: **IGRALNI VMESNIK ZA SIMULATORJE VIDEO ARKADNIH AVTOMATOV**

A GAME INTERFACE FOR A VIDEO ARCADE MACHINE SIMULATOR

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Načrtujte in izdelajte igralni vmesnik za simulatorje video arkadnih avtomatov. Igralni vmesnik sestavljata strojna in programska oprema. Za realizacijo strojne opreme vmesnika uporabite 8-bitni ATmega mikrokrmilnik proizvajalca ATMEL. Za povezavo med strojno opremo ter osebnim računalnikom uporabite eno izmed danes standardnih serijskih povezav. Načrtujte gonilnik za igralni vmesnik za operacijski sistem Linux. Gonilnik naj igralni vmesnik predstavi operacijskemu sistemu, kot virtualno tipkovnico. Za testiranje vmesnika uporabite simulator arkadnih avtomatov M.A.M.E.

Mentor:
doc. dr. Patricio Bulić

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:
prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:
prof. dr. Andrej Likar

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 6

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Izidor Makuc

Igralni vmesnik za simulatorje video igralnih avtomatov

DIPLOMSKO DELO NA INTERDISCIPLINARNEM UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2011

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Izidor Makuc

Igralni vmesnik za simulatorje video igralnih avtomatov

DIPLOMSKO DELO NA INTERDISCIPLINARNEM UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Patricio Bulić

Ljubljana, 2011

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Izidor Makuc,
z vpisno številko 63040284,

sem avtor diplomskega dela z naslovom:

Igralni vmesnik za simulatorje video igralnih avtomatov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod vodstvom mentorja
  doc. dr. Patricia Bulića.

- so elektronska oblika diplomskega dela, naslov, povzetek ter ključne besede identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se mentorju doc. dr. Patriciu Buliću za vso pomoč pri pisanju diplomskega dela ter družini in ostalim, ki so mi med študijem stali ob strani.

KAZALO VSEBINE

KAZALO SLIK

DEFINICIJE POJMOV

V/I – vhodno izhodni

ghosting – pojav, ko se ob pritisku več tipk na tipkovnici pojavijo na izhodu dodatni simboli

keyboard encoder – naprava, ki simulira tipkovnico

USB (Universal serial bus)– USB standard za naprave, kot sta tipkovnica in miška

PS/2 (Personal System/2)– vtič, ki se uporablja za priklop tipkovnice na osebni računalnik

Vendor ID – številka, ki jo mora imeti vsak proizvajalec USB naprav

RS-232 (Recommended Standard 232) – skupina standardov za serijsko komunikacijo

DIP (Dual In-line Package) – vrsta ohišja za čipe, kjer nožice čipa gledajo skozi tiskano vezje

protoboard – plošča za povezovanje elektronskih elementov

pull-up upor– upor za dvig nivoja

bouncing – poskakovanje napetosti na kontaktu ob pritisku gumba

char device driver – vrsta gonilnika za znakovne naprave

firmware – operacijska programska oprema, ki teče na vgrajenih sistemih (operacijski softver)

1 POVZETEK

V diplomskem delu si ogledamo razvoj igralnega vmesnika za simulatorje video igralnih avtomatov, ki tečejo na osebnem računalniku. Najprej si ogledamo, kaj igralni vmesnik sploh je ter čemu smo se lotili njegove izdelave. Nato bolj podrobno definiramo, kaj od igralnega vmesnika pričakujemo ter se lotimo njegove izvedbe.

Izvedba igralnega vmesnika je sestavljena iz sledečih delov. Najprej je prikazan razvoj operacijske programske opreme (firmware) za igralni vmesnik, kjer si ogledamo osnovni algoritem za branje tipk. V drugem delu pride pod drobnogled strojna oprema, ki nam omogoča realizacijo igralnega vmesnika. Nazadnje si ogledamo še samo povezavo med igralnim vmesnikom ter računalnikom.

V nadaljevanju si ogledamo gonilnik za operacijski sistem GNU/Linux, ki nam omogoča, da igralni vmesnik sistemu predstavimo kot virtualno tipkovnico. Poleg gonilnika razvijemo tudi program, ki skrbi, da se gumbi na igralnem vmesniku pravilno interpretirajo glede na to, kateri simulator video igralnega avtomata je v uporabi.

Ker želimo omogočiti uporabo igralnega vmesnika širokemu krogu uporabnikov, je bila za vse programske rešitve v diplomskem delu uporabljena zgolj odprtokodna programska oprema ter operacijski sistem GNU/Linux. Prav tako je bilo diplomsko delo napisano ter urejeno v odprtokodnem pisarniškem paketu LibreOffice[4].

Ključne besede

Igralni vmesnik, simulator video igralnega avtomata, odprta-koda.

2 ABSTRACT

The purpose of this thesis is to develop a game interface for video arcade machine simulators. First the game interface is described in general. After that the known issues are pointed out, as well as expected goals of the project.

In the following sections the development and construction process is described. First the algorithms for the firmware are developed. After that the hardware and the connection between the game interface and a personal computer are described.

Next the driver for the GNU/Linux operating system is described, which allows the game interface to be presented as a virtual keyboard to the system, and a software to process the keystrokes from this virtual keyboard and interpret them according to the video arcade simulator in use.

In order to make this game interface widely available, all the software was developed with open-source tools and the GNU/Linux operating system was used. This thesis was also written in the word processor from open-source LibreOffice productivity suite.

Keywords: Game interface, video arcade machine simulator, open-source

3 UVOD

Slika 1: Primer igralnega vmesnika in igralnega avtomata

Ogledali si bomo potek izdelave igralnega vmesnika za igranje iger, kot ga poznamo iz video igralnih avtomatov (video arcade machine). Videz video igralnega avtomata in igralnega vmesnika je predstavljen na sliki 1.

3.1 VIDEO IGRALNI AVTOMAT

Video igralni avtomati oziroma krajše igralni avtomati so dosegli višek priljubljenosti med letoma 1980 in 1990, nato so jih izrinile domače igralne konzole ter osebni računalniki. Igralni avtomati so dandanes zelo popularni med zbiratelji ter navdušenci, ki jih izdelujejo doma v svojem prostem času. Ta igralni vmesnik je namenjen prav slednjim. Poglejmo si, kako danes navdušenci sestavljajo igralne avtomate.

Večino "pameti" igralnega avtomata dandanes predstavlja osebni računalnik. Le-ta skrbi za simulacijo celotne strojne opreme igralnega avtomata. Če želimo izdelati svoj igralni avtomat, potrebujemo poleg osebnega računalnika še ohišje ter igralni vmesnik, ki poskrbi za interakcijo med uporabnikom ter računalnikom.

4 MOTIVACIJA

Največje težave navdušencem pri izdelavi povzroča prav igralni vmesnik, saj je za izdelavo le-tega je potrebnega nekaj znanja s področja elektronike ter programiranja. Vsi ostali deli igralnega avtomata se namreč simulirajo programsko. Dotičnim programom pravimo simulatorji igralnih avtomatov, več o njih pa si bo moč prebrati v sledečih poglavjih.

V nadaljevanju si oglejmo, kaj igralni vmesnik sploh je.

4.1 IGRALNI VMESNIK

Igralni vmesnik si lahko predstavljamo kot ploščo, na kateri se nahajajo elementi za interakcijo med človekom ter video igralnim avtomatom. Najpogosteje uporabljeni elementi so:

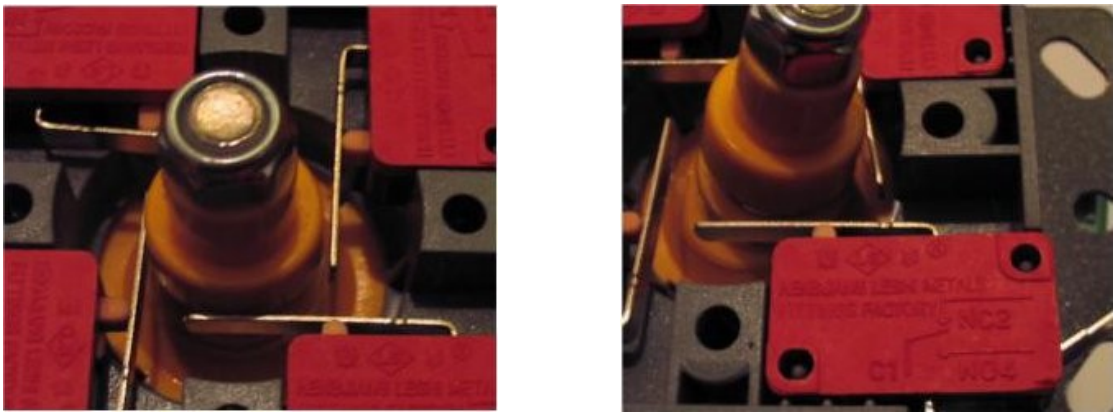
1. Gumb ali tipka. Primer prikazuje slika 2.

Slika 2: Primer gumba

2. Osemsmerna igralna palica (joystick). Osnovno delovanje igralne palice je prikazano na naslednji strani. Primer prikazuje slika 3.

Slika 3: Primer igralne palice

Osemsmerna igralna palica je sestavljena iz štirih gumbov. Ob premiku igralne palice v določeno smer le-ta pritisne na en oziroma dva gumba. Iz informacije, kateri gumbi so pritisnjeni, lahko razberemo smer premika palice. Prikaz premika ter pritiska gumba je viden na sliki 4.



Slika 4: Premik igralne palice

4.2 USTALJENA PRAKSA PRI IZDELAVI IGRALNIH VMESNIKOV

Glede na to, da so vsi elementi igralnega vmesnika dejansko gumbi, lahko igralni vmesnik simuliramo s tipkovnico.

Glavna problema, ki jih moramo rešiti pri izdelavi igralnega vmesnika, sta torej povezava igralnega vmesnika z osebnim računalnikom ter komunikacija med igralnim vmesnikom in programom, ki simulira igralni avtomat.

V nadaljevanju si oglejmo, kakšna je ustaljena praksa pri domači izdelavi igralnih vmesnikov.

Večina izdelovalcev za izdelavo igralnega vmesnika uporabi kar dele iz računalniške tipkovnice. Na prvi pogled enostavna rešitev hitro pokaže svoje zobe. Tipkovnica ima namreč mikrokrmilnik, na katerega so tipke povezane preko matrične vezave. Ob pritisku več gumbov hkrati lahko pride do dodatne informacije (ghosting), saj mikrokrmilnik ne more več enolično razbrati stanja na tipkovnici.

Poleg uporabe tipkovnice lahko izberemo tudi komercialno rešitev, tako imenovani kodirnik za tipkovnico (Keyboard encoder). Le-ta sicer nima težav s pritiski na več gumbov hkrati, nam pa zato najcenejša verzije olajša denarnico za slabih $50.

Obe zgoraj našteti možnosti imata eno slabost. Obe se predstavita sistemu kot standardna USB ali PS/2 tipkovnica. To sicer odpravi problem komunikacije med igralnim vmesnikom ter simulatorjem igralnega avtomata, saj zanjo poskrbi sam gonilnik operacijskega sistema, problem pa se pojavi, ker različni simulatorji igralnih avtomatov uporabljajo različne tipke na tipkovnici. To pomeni, da moramo za vsak simulator posebej sporočiti igralnemu vmesniku, kako naj zakodira vsako od tipk, da bo simulator dobil pravilne vhodne podatke. Za rešitev dotičnega problema bi potrebovali večjo kompleksnost igralnega vmesnika ter spremembo gonilnikov.

V nadaljevanju je tako predstavljena rešitev, ki odpravi pomanjkljivosti, ki jih imata zgoraj opisani rešitvi.

5 NAČRTOVANJE IGRALNEGA VMESNIKA

Sedaj ko smo si ogledali trenutne rešitve ter približno poznamo problematiko, lahko začnemo z načrtovanjem igralnega vmesnika.

Za začetek bomo privzeli nekaj omejitev:

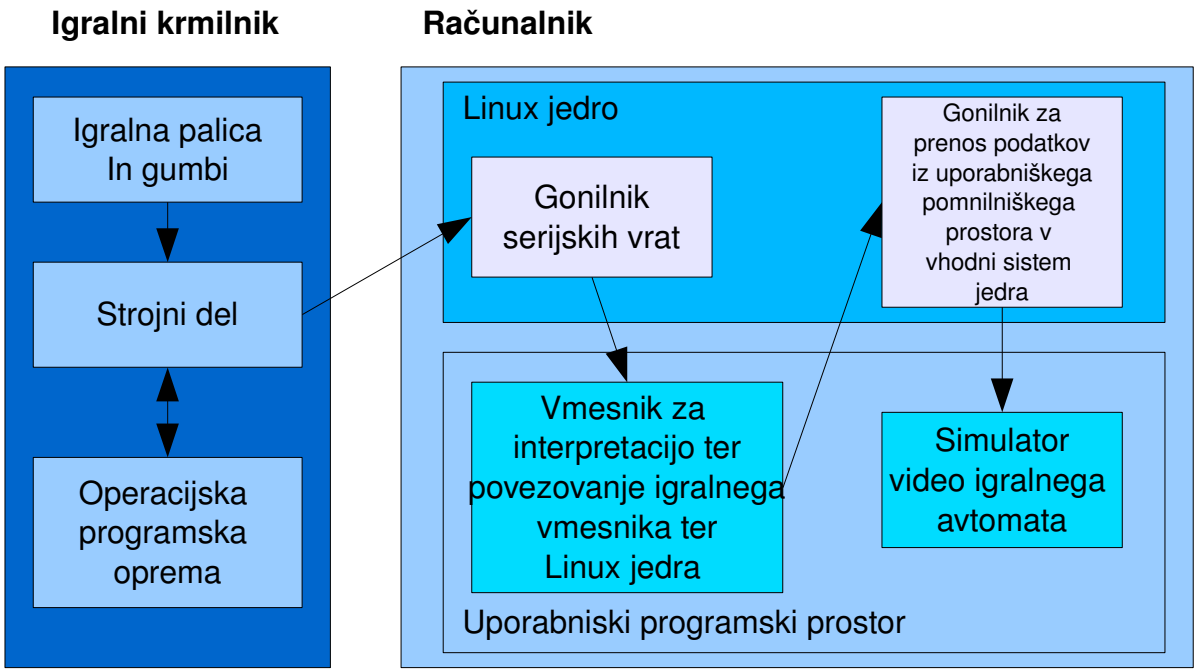
- uporabili bomo zgolj odprtokodna orodja ter odprtokodno programsko opremo,

- igralni vmesnik bo podpiral največ 8 digitalnih vhodov,

- fizični videz igralnega vmesnika nas ne zanima,

- pomembno vlogo pri izbiri rešitve bo imela njena cena.

Kot smo videli, je igralni vmesnik neke vrste tipkovnica, ki jo želimo priklopiti na osebni računalnik. Na žalost pa igralnega vmesnika računalniku ne moremo predstaviti kot tipkovnico, saj bi s tem omejili uporabo zgolj na skupino simulatorjev, ki uporabljajo isti nabor tipk za upravljanje svoje vsebine. Na drugi strani simulatorji igralnih avtomatov pričakujejo vhodne podatke, ki jih sistemu posreduje tipkovnica.



Slika 5: Celotni koncept igralnega vmesnika, kjer puščice prikazujejo pot informacije od igralnega vmesnika do simulatorja video igralnega avtomata

Zgornja problema sta si očitno nasprotujoča, saj se z rešitvijo enega pojavi drugi. Zato bomo za rešitev obeh morali uvesti vmesni člen, ki bo komuniciral z igralnim vmesnikom na eni strani ter z Linux jedrom na drugi strani. Ker z Linux jedrom ne moremo neposredno komunicirati, potrebujemo tudi gonilnik, ki nam bo želeno komunikacijo omogočil.

Celoten koncept predstavlja slika 5.

6 STROJNI DEL IGRALNEGA VMESNIKA

Sedaj ko poznamo koncept igralnega vmesnika, se lahko lotimo njegove realizacije. Realizirali ga bomo s pomočjo mikrokrmilnika, ki bo poskrbel, da se informacija o pritisku tipk prenese do računalnika v njemu razumljivi obliki.

6.1 IZBOR TER UPORABA MIKROKRMILNIKA

Pri izbiri mikrokrmilnika moramo paziti, da lahko zagotovimo branje vsaj osmih digitalnih vhodov ter implementiramo komunikacijo z računalnikom.

Priporočljiva izbira za implementacijo komunikacije je vsekakor standard USB. Kljub kompleksnosti je USB trenutno standardno vodilo za priklop V/I naprav na osebni računalnik. Poleg tega reši tudi težavo napajanja mikrokrmilnika. Kompleksnost pa uspešno skrijejo programske knjižnice oz. vse bolj pogosto kar strojno rešene implementacije vmesnika USB v samem mikrokrmilniku. USB je trenutno najbolj smotrna izbira komunikacije. Na žalost pa standard USB zahteva, da se vsaka naprava predstavi s svojim identifikatorjem (Vendor ID), ki nam ga mora dodeliti USB-IF (USB Implementers Forum, Inc). Seveda tega ne počnejo zastonj, tako da nas pridobitev Vendor ID-ja stane okoli $1000 letno [7].

Ker smo privzeli, da je za rešitev pomembna tudi cena, bomo za komunikacijo uporabili standard RS-232. Poleg tega nam standard RS-232 zmanjša kompleksnost samega vezja, kar je tudi ena od pomembnih iztočnic pri razvoju igralnega vmesnika.

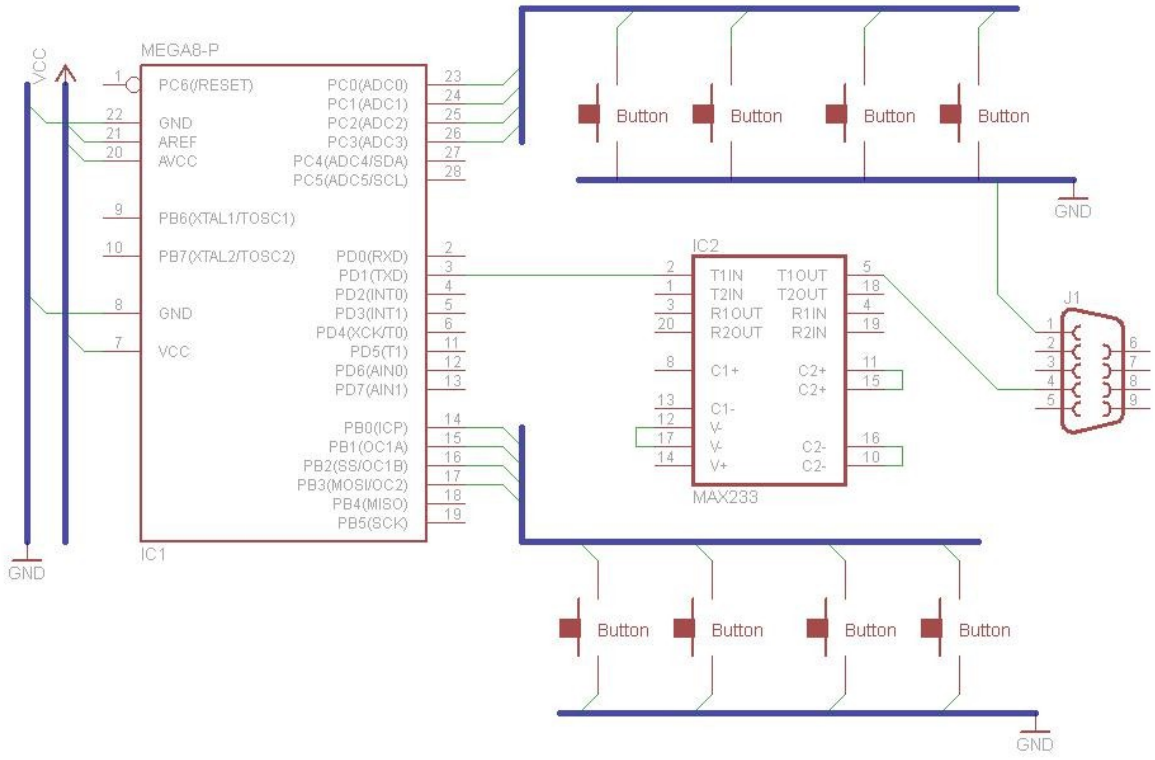
Poleg zgoraj navedenega je za nas ključnega pomena, da za izbrani mikrokrmilnik obstaja odprtokodno razvojno okolje. Žalostna resnica je, da večina proizvajalcev mikrokrmilnikov ponuja zaprta razvojna okolja, namenjena predvsem operacijskemu sistemu Windows. Na srečo obstaja za mikrokrmilnike proizvajalca Atmel odprtokodni programator ter različica prevajalnika GCC, ki prevaja programski jezik C/C++ v ustrezno strojno kodo. Poleg tega so poskrbeli tudi za obširno C knjižnico po imenu AVR Libc[3], ki je zelo okrnjena verzija knjižnice stdlib ter je seveda prilagojena za 8-bitne Atmelove mikrokrmilnike. Poleg tega imamo na voljo tudi programsko knjižnico, ki poskrbi za implementacijo vmesnika USB[6]. V primeru, da posedujemo veljaven USB identifikator, lahko tako brez težav komunikacijo rešimo tudi z uporabimo vmesnika USB.

Za igralni vmesnik bomo tako uporabili 8-bitni Atmelov mikrokrmilnik atmega8[2]. Le-ta nam ponuja zadostno število V/I nožic ter strojno podporo za komunikacijo preko vmesnika RS-232.

6.2 VEZJE

V nadaljevanju si bomo ogledali enostavni vezalni načrt vezja (slika 6), ki nam zagotavlja potrebne vhode za gumbe ter komunikacijo z računalnikom preko RS-232.

Same fizične realizacije vezja se tu ne bomo dotaknili, saj je odvisna od sposobnosti posameznika, tehnologij, ki jih ima na voljo, ter podnožij samih elementov. V primeru posedovanja elementov v ohišjih DIP lahko vezje realiziramo tudi na testni plošči (protoboard).



Slika 6: Vezalni načrt vezja

Tipke bomo priklopili na mikrokrmilnik s pomočjo pull-up uporov. Pull-up upor potrebujemo, da onemogočimo nedefinirano (floating) stanje vhodne nožice mikrokrmilnika, ko gumb ni pritisnjen. Uporabili bomo notranje pull-up upore, ki se nahajajo v samem čipu mikrokrmilnika, zato na vezalnem načrtu niso vidni. To za nas pomeni, da ob pritisku tipke mikrokrmilnik le-to zazna kot logično ničlo.

Čip MAX233[5] skrbi za pretvorbo logičnih nivojev. Standard RS-232 zahteva -15 V za logično enico ter +15 V za logično ničlo. Naš mikrokrmilnik pa deluje na standardnih TTL nivojih. MAX233 tako prilagodi napetost iz TTL nivoja v RS-232 napetostne nivoje.

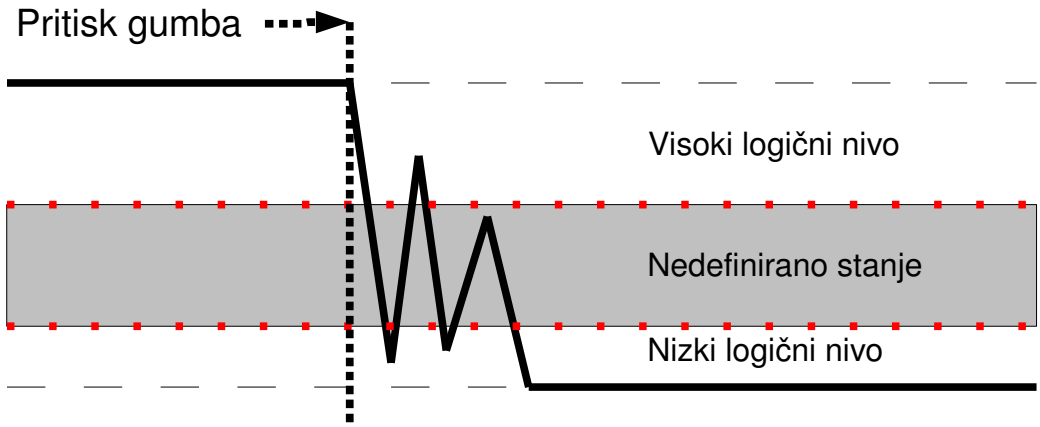
7 OPERACIJSKA PROGRAMSKA OPREMA IGRALNEGA VMESNIKA

Kot smo nakazali v prejšnjem poglavju, moramo programsko rešiti branje tipk ter komunikacijo z računalnikom.

7.1 BRANJE TIPK

Priklop tipk zgolj s pomočjo „pull-up“ uporov nam ne omogoča enostavnega branja le-teh, ker ne odstrani pojava, ki mu pravimo poskakovanje napetosti na kontaktu (bouncing). V nadaljevanju si oglejmo, kaj pravzaprav dotični pojav je, in algoritem, s katerim bomo problem rešili.

7.1.1 Poskakovanje napetosti na kontaktu



Slika 7: Primer žagastega nihanja napetosti ob pritisku gumba

Problem se pojavi ob pritisku tipke. Namreč namesto takojšnjega padca oziroma dviga napetosti, le-ta zaniha v žagasti obliki. To pri hitremu branju tipke lahko povzroči, da zaznamo več pritiskov tipke namesto le enega (slika 7).

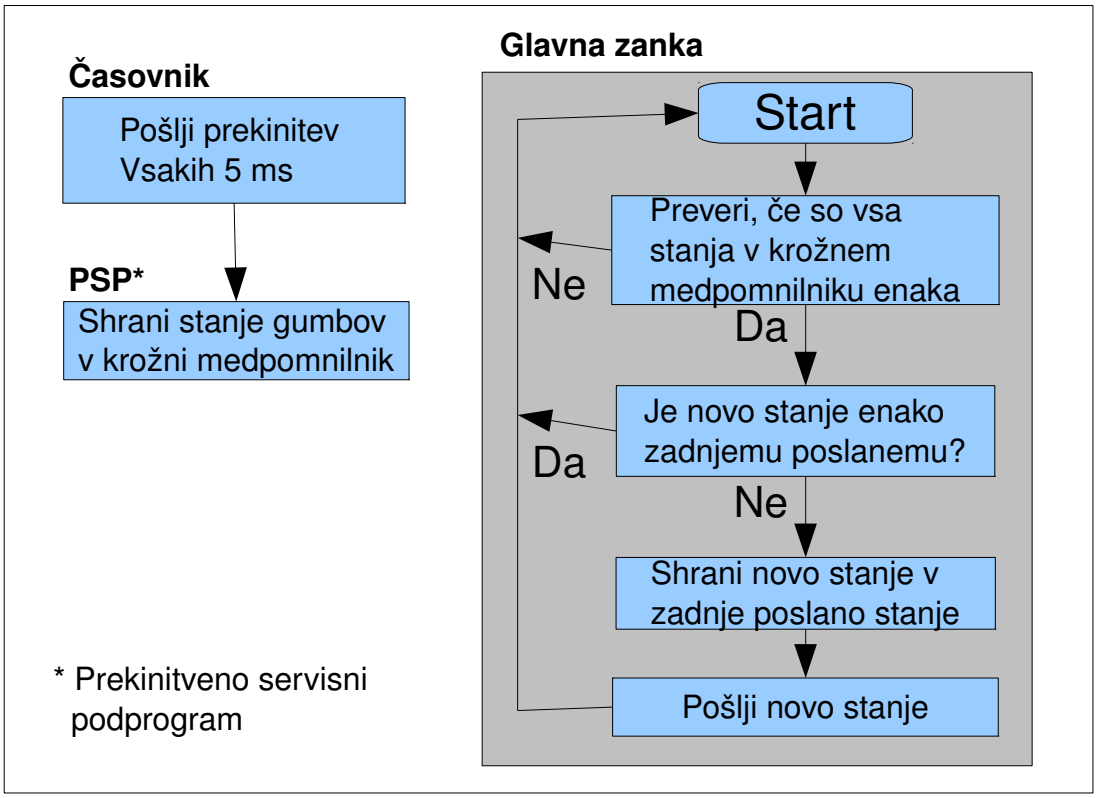
Tej težavi se lahko zoperstavimo na več načinov. Lahko jo rešimo strojno ali programsko. Mi si bomo ogledali programsko rešitev, ki je za nas cenovno ugodnejša ter manj kompleksna za realizacijo.

Učinkovita in dokaj enostavna rešitev je branje tipke v določenem časovnem intervalu T ter s pomočjo zadnjih N odčitanih branj ugotoviti, ali je prišlo do pritiska gumba. Edini problem,

ki ga imamo, je določanje vrednosti T ter N, saj ju določata dva nasprotujoča si kriterija. Prvi je čas, ki je potreben, da se poskakovanje kontakta umiri. Drugi je odzivnost sistema. Uporabnik namreč ne sme opaziti zamika med pritiskom gumba ter odzivom sistema. Ker sam nimam opreme, ki je potrebna za izvajanje meritev, sem uporabil meritve iz članka. [10]

V članku je bilo ugotovljeno, da so imeli vsi testni gumbi čas umiritve pod 10 ms. Zakasnitev, manjša od 50 ms, pa je za človeka praktično neopazna.

Kot vidimo, nam meritve dajejo veliko možnosti za izbiro parametrov T in N. Mi bomo uporabili T=5 ms in N=6, kar pomeni, da bomo novo stanje gumba dobili v najslabšem primeru z zamikom 40 ms, kjer 10 ms vzame umiritev poskakovanja napetosti na kontaktu, ostalih 30 ms pa 6 stabilnih zaporednih branj tipk. Sam algoritem za branje tipk lahko vidimo v sledečem diagramu (slika 8).



Slika 8: Diagram algoritma za branje tipk

Na levi strani diagrama je prikaz delovanja časovnika, ki vsakih 5 ms proži prekinitev. Prekinitev pokliče prekinitveni servisni podprogram, ki shrani stanje gumbov na igralnem vmesniku. Ker so gumbi priklopljeni na različne vhodne registre, jih najprej zloži v 1 B dolgo

besedo, tako da prihranimo na prostoru. Nato to besedo shrani v krožni medpomnilnik velikosti 6 B. Tako hranimo zadnjih 6 stanj gumbov na igralnem vmesniku.

V glavni zanki programa se preverja, ali je prišlo do spremembe stanja tipk. Najprej preverimo, če je stanje tipk stabilno. Da to ugotovimo, primerjamo vse vrednosti v krožnem pomnilniku. V primeru, da so vse vrednosti enake, je stanje tipk stabilno zadnjih 30 ms, kar pomeni, da lahko preverimo, ali je prišlo do novega stabilnega stanja. To ugotovimo s primerjavo zadnjega poslanega stanja s trenutnim stabilnim stanjem. V primeru, da se stanji razlikujeta, pošljemo novo stanje računalniku in ga shranimo v zadnje poslano stanje.

7.2 KOMUNIKACIJA Z RAČUNALNIKOM

Ker smo privzeli, da imamo na igralnem vmesniku največ 8 gumbov, lahko stanje le-teh enostavno zakodiramo v 1 B dolgi besedi, kjer vsak bit pove stanje enega gumba. Komunikacija se tako prevede v pošiljanje ene besede, ko se pojavi novo stanje na igralnem vmesniku.

Izbrani mikrokrmilnik ima strojno podporo za serijsko komunikacijo, tako je uporaba le-te dokaj trivialno opravilo. Vse, kar moramo storiti, je, da nastavimo pravilne parametre v konfiguracijske registre mikrokrmilnika. Več o standardu RS-232 si je moč prebrati na Wikipediji [9]. V nadaljevanju si bomo zgolj pogledali, katere parametre moramo nastaviti, da vzpostavimo želeno komunikacijo.

V našem primeru gre za asinhronsko serijsko komunikacijo. To pomeni, da imata pošiljatelj in prejemnik ločeni uri, ki ju je ob začetku prenosa potrebno sinhronizirati. Da uri ostaneta sinhronizirani, morata prejemnik in pošiljatelj delovati z isto hitrostjo, ki jo merimo v bitih na sekundo oz. bps (bits per second).

Poleg hitrosti je pomembna tudi dolžina prenosa, ki jo merimo v bitih. Mi pošiljamo osem bitov podatkov, pred katerimi moramo poslati začetni bit (start bit), ki poskrbi za sinhronizacijo ur ter na koncu še končni bit (stop bit), da sporočimo konec prenosa oziroma omogočimo novo pošiljanje začetnega bita. Tako dejansko pošljemo 10 bitov vsakič, ko pošljemo en bajt oziroma eno besedo podatkov.

Sedaj ko poznamo dolžino paketa, lahko izračunamo minimalno potrebno hitrost v bps, ki nam bo zagotovila, da se podatek pošlje, preden se spremeni stanje na igralnem vmesniku.

Časovnik na vsakih 5 ms pokliče prekinitveno servisni podprogram, v katerem se preveri stanje gumbov. Ker imamo več kot en gumb, to pomeni, da se v najslabšem primeru lahko stanje spremeni v 5 ms. Torej moramo biti zmožni poslati 10 bitov v 5 ms oziroma zagotoviti hitrost prenosa vsaj 2000 bps, da se izognemo izgubi podatkov.

V našem primeru bomo izbrali hitrost prenosa 19200 bps, ki spada med standardne hitrosti. Tako bomo lahko brez težav uporabljali knjižnice ter standardne gonilnike na operacijskem sistemu GNU/Linux.

8 PROGRAMSKA OPREMA ZA OSEBNI RAČUNALNIK

Predpostavimo, da uporabljamo osebni računalnik, osnovan na arhitekturi x86 z operacijskim sistemom GNU/Linux. Le-ta je v času pisanja diplomskega dela imel največji tržni delež med odprtokodnimi operacijskimi sistemi. Poleg tega za GNU/Linux obstaja mnogo različnih simulatorjev igralnih avtomatov, zaradi katerih smo pravzaprav začeli razvijati svoj igralni vmesnik. Nenazadnje imamo za GNU/Linux tudi odprtokodne knjižnice ter prevajalnik GCC za programski jezik C/C++. Vse to zadosti vsem našim potrebam za implementacijo povezljivosti med računalnikom ter igralnim vmesnikom.

Kot smo omenili že v uvodu, vsi simulatorji podpirajo tipkovnico. Nekateri poleg tipkovnice podpirajo tudi igralni plošček ali kako podobno vhodno napravo. Ker hočemo, da je igralni vmesnik uporaben za kar se da veliko število simulatorjev, ga bomo predstavili sistemu kot tipkovnico.

Najenostavnejši način je, da igralni vmesnik sistemu predstavimo kot standardno PS/2 ali USB tipkovnico. Na ta način bi lahko uporabili že obstoječe gonilnike Linux jedra. Na prvi pogled enostavna rešitev pa skriva en problem, in sicer vsak simulator definira svoje kontrolne tipke na tipkovnici, kar pomeni, da bi moral igralni vmesnik pošiljati različne kode tipk ob pritisku na isti gumb, glede na uporabljani igralni vmesnik. Ker nočemo povečati kompleksnosti našega igralnega vmesnika, bomo to problematiko raje rešili na strani računalnika. Tako bomo vpeljali vmesni člen v obliki programa, ki bo komuniciral z igralnim vmesnikom na eni strani ter z Linux jedrom na drugi strani. Vmes bo ustrezno prekodiral dobljeno informacijo o pritisku gumba na igralnem vmesniku ter jo posredoval jedru v njemu razumljivi obliki. Navodila, v katere črke je potrebno prekodirati gumbe na igralnem vmesniku, bomo programu podali skozi konfiguracijsko datoteko. Na ta način bo vse delo opravil program, ki teče v uporabniškem programskem prostoru. Operacijska programska oprema igralnega vmesnika in gonilnik, ki bo skrbel za komunikacijo med programom ter Linux jedrom, bosta tako ostala karseda enostavna. To je za razvijalca rešitve pomembno, saj je razvoj gonilnikov ter operacijske programske opreme, predvsem pa njihovo testiranje, precej kompleksnejše opravilo kot razvoj in testiranje programov, ki tečejo v uporabniškem pomnilniškem prostoru.

8.1 PROGRAM ZA KOMUNIKACIJO MED IGRALNIM VMESNIKOM TER LINUX JEDROM

Koncept dotičnega programa si bomo najlažje ogledali, če ga razdelimo na tri dele: komunikacijo z igralnim vmesnikom, osrednji del ter komunikacijo z Linux jedrom.

8.1.1 Komunikacija z igralnim vmesnikom

Ker uporabljamo RS-232, mora biti serijski vmesnik na računalniku nastavljen enako kot na igralnem vmesniku. Razlika je zgolj v tem, da tu ne moremo direktno dostopati do serijskih vrat, zato moramo uporabiti vmesnik, ki nam ga nudi operacijski sistem.

8.1.2 Osrednji del

Tu si bomo pogledali glavni del programa. Ta skrbi za ustrezno interpretacijo stanja tipk na igralnem vmesniku. Program bo najprej prebral konfiguracijsko datoteko, ki je sestavljena iz vrstic, opisanih s sledečim regularnim izrazom definiranim v razširjeni notaciji POSIX:

"([0-7]) *, *([A-Z_]+) *, *([a-zA-Z _]+) *"

Vsaka vrstica tako opisuje, kateri biti v bajtu, ki ga dobimo iz igralnega vmesnika, predstavljajo katero kodo tipke. Konfiguracijska datoteka je prikazana na sliki 9.

```
5, KEY_W, up
7, KEY_X, down
4, KEY_A, left
6, KEY_D, right
0, KEY_RIGHTSHIFT, magic
1, KEY_S, attack
2, KEY_TAB, jump
```

Slika 9: Primer konfiguracijske datoteke.

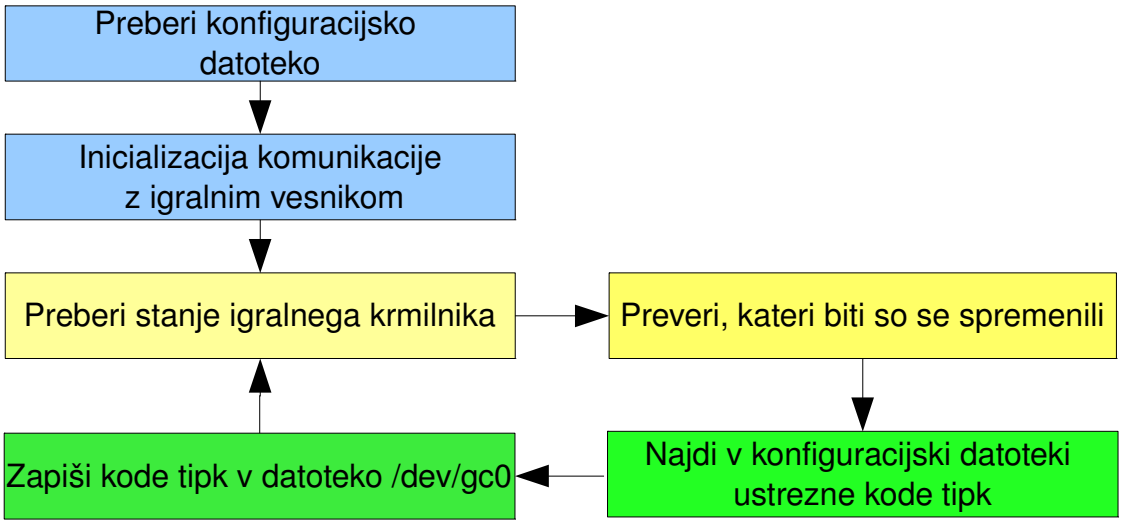
Drugi stolpec predstavlja ime za kodo tipke, ki jo moramo poslati Linux jedru, če se spremeni bit na mestu, ki ga določa prvi stolpec. Tretji stolpec je zgolj namenjen kot opomba za uporabnika.

Ob prejemu novega stanja iz igralnega vmesnika, program primerja novo stanje s prejšnjim. Ob ugotovitvi spremembe bo Linux jedru poslal informacijo o tem, katera tipka je spremenila stanje in ali smo tipko pritisnili ali izpustili.

8.1.3 Komunikacija z Linux jedrom

Ker naš program teče v uporabniškem pomnilniškem prostoru, ne more neposredno komunicirati z ostalimi programi, razen če le-ti tega ne podpirajo. Na slednje se ne moremo zanašati, zato je najbolje, da za to poskrbi kar vhodni sistem Linux jedra. Ker Linux zaradi varnosti ne podpira neposredne komunikacije med programi, ki tečejo v uporabniškem programskem prostoru, ter Linux jedrom, moramo za to napisati ustrezen gonilnik, ki bo poskrbel za prehod informacije med uporabniškim prostorom ter vhodnim sistemom Linux jedra. Več o gonilniku si bomo pogledali v naslednjem poglavju; trenutno nas zanima zgolj to, kako bo program komuniciral z gonilnikom. Program bo videl gonilnik kot datoteko naprave (device file), v katero bo zapisal po 2 bajta za vsak pritisk ali sprostitev gumba na igralnem vmesniku. Prvi bajt bo predstavljal kodo tipke, drugi pa, ali je šlo za pritisk oziroma sprostitev tipke. Kode tipk so definirane v linux/input.h datoteki, ki je del odprte kode Linux jedra.

Zgoraj smo razčlenili osnovno delovanje programa. Ponazorimo sedaj delovanje programa še z diagramom, ki je prikazan na sliki 10 ter si podrobneje oglejmo za nas zanimive dele.



Slika 10: Diagram programa

Program najprej prebere konfiguracijo, zapisano v datoteki, ter jo shranili v seznam, katerega elementi so strukture struct Key_bind. Vsaka struktura hrani podatek, kateri gumb na igralnem vmesniku opisuje, ter katero kodo tipke moramo poslati gonilniku ob pritisku tega gumba.

Po konfiguraciji sledi inicializacija povezave z igralnim vmesnikom preko serijskega vmesnika. Tu poskrbimo za ustrezno konfiguracijo serijskih vrat, ki mora biti enaka konfiguraciji na igralnem vmesniku.

Operacijski sistem GNU/Linux ima za delo s serijskimi vrati že vgrajen gonilnik, ki je del Linux jedra in poenostavi konfiguracijo serijskih vrat ter branje in pisanje preko le-teh. Gonilnik tako uporabniku ponudi funkcijo read, ki lahko deluje v dveh načinih. Funkcija lahko bere celotne vrstice, kar pomeni, da hrani znake v medpomnilniku in čaka, dokler ne dobi znaka za novo vrstico. Nato vrne kazalec na prvi znak te vrstice. Druga možnost je, da funkciji nastavimo, koliko znakov mora sprejeti, preden vrne kazalec na prvi prebrani znak.

V fazi branja je uporabljena druga možnost. Funkcijo smo omejili, da čaka, dokler ne sprejme enega znaka. Med čakanjem program spi, tako da ne obremenjuje sistema. Ko igralni vmesnik pošlje novo stanje, funkcija read vrne en bajt in zbudi program, tako da lahko ta nadaljuje z naslednjim stanjem.

Ko prejmemo novo stanje, ga bit za bitom primerjamo s prejšnjim, da ugotovimo, kje je prišlo do spremembe. Ko ugotovimo spremembo bita, poiščemo v seznamu strukturo, ki opisuje dotični bit, ter pošljemo ustrezno kodo tipke gonilniku.

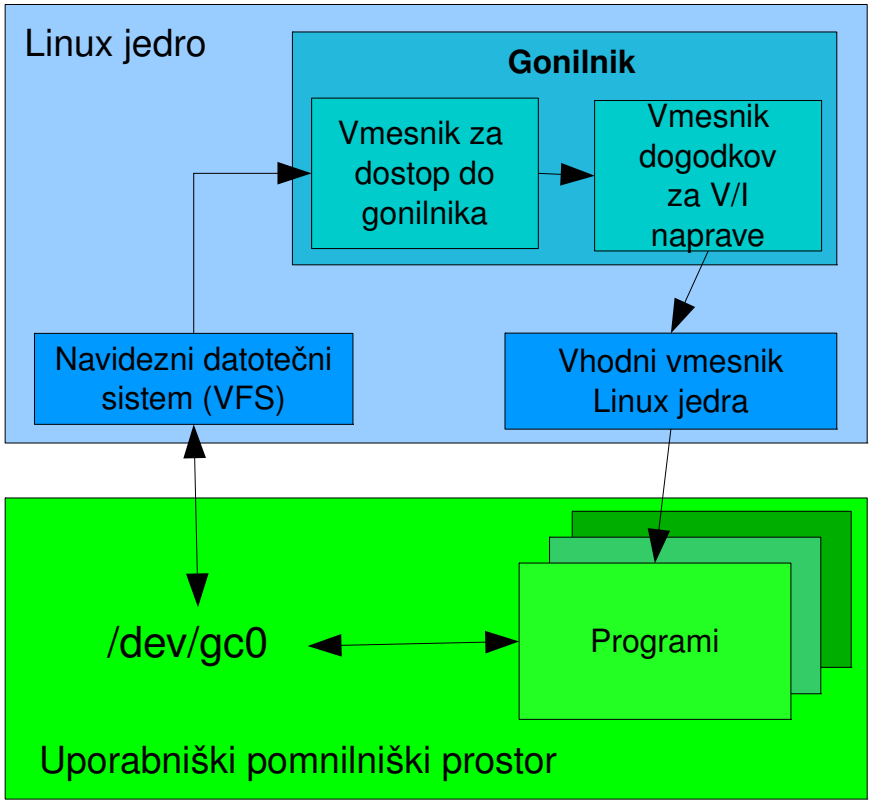
8.2 GONILNIK

Do sedaj smo uspešno povezali igralni vmesnik s sistemom ter napisali program, ki skrbi za komunikacijo in interpretacijo le-te. Kot poslednja naloga ostane še implementacija gonilnika, ki bo omogočil komunikacijo med Linux jedrom ter programom, ki skrbi za komunikacijo med računalnikom in igralnim vmesnikom.

Linux že ima vgrajen sistem, ki poskrbi, da se V/I naprave enotno predstavijo jedru s pomočjo tako imenovanih dogodkov. Dogodke jedro nato obdela ter posreduje naprej v uporabniški delovni prostor programu, ki trenutno teče v ospredju.

Glede na to, da je dotični mehanizem prav to, kar potrebujemo, je najbolj smotrna izbira napisati gonilnik, ki se z Linux jedrom poveže preko vmesnika dogodkov, v uporabniškem prostoru pa se prikaže kot datoteka naprave. Datoteke naprave so navidezne datoteke, katerih namen je prikazati napravo kot datoteko v uporabniškem programskem prostoru. S tem se poenoti vmesnik za delo z datotekami ter napravami. Koncept gonilnika je viden na sliki 11.

Mi se s podrobnostmi Linux jedra ne bomo ukvarjali, saj to presega namen diplomske naloge. V nadaljevanju si bomo tako zgolj ogledali dele gonilnika, ki so specifični za reševanje problema komunikacije med programom ter vhodnim vmesnikom Linux jedra.



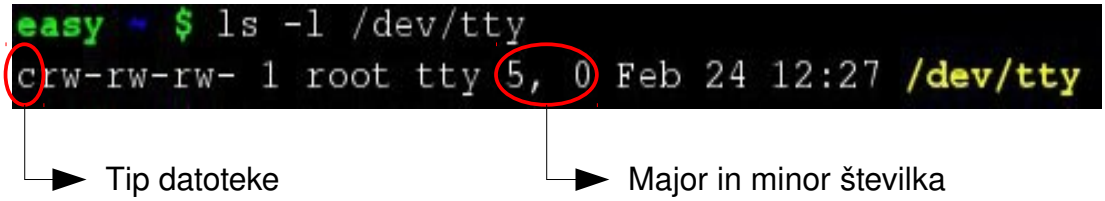
Slika 11: Koncept gonilnika

8.3 VMESNIK ZA DOSTOP DO GONILNIKA

Uporabili bomo vmesnik, ki za komunikacijo z gonilnikom uporabi znakovno datoteko naprave (character device file). Znakovna datoteka naprave je ena od vrst datotek naprave, ki jo uporabljamo večinoma za komunikacijo z gonilnikom, ki prenaša manjšo količino podatkov med uporabniškim prostorom ter jedrom. Za program, ki teče v uporabniškem

prostoru, je datoteka naprave vidna kot navadna datoteka in tako omogoča komunikacijo z Linux jedrom preko standardnih vhodno-izhodnih sistemskih klicev. Datoteka naprave ni del gonilnika, temveč jo mora ustvariti skrbnik sistema ter jo povezati z želenim gonilnikom. Le-to naredi s sistemskim ukazom *mknod [ime datoteke naprave] [tip datoteke naprave] [major številka] [minor številka]*. Major številka je ključnega pomena za Linux jedro, kajti le-ta določa povezavo med gonilnikom ter datoteko naprave. V nadaljevanju si poglejmo primer znakovne datoteke naprave.

V operacijskem sistemu GNU/Linux z ukazom *ls -l [ime datoteke]* dobimo osnovne informacije o datoteki. Če ukaz uporabimo na datoteki naprave, dobimo izpis, prikazan na sliki 12.



Slika 12: Primer datoteke device

Kot je razvidno iz slike 12, nam črka c, ki je obkrožena z rdečim krogom, pove, da gre za znakovno datoteko naprave. Naslednji dve obkroženi številki predstavljata major in minor številko. Major številko že poznamo, minor številka pa za nas ni pomembna.

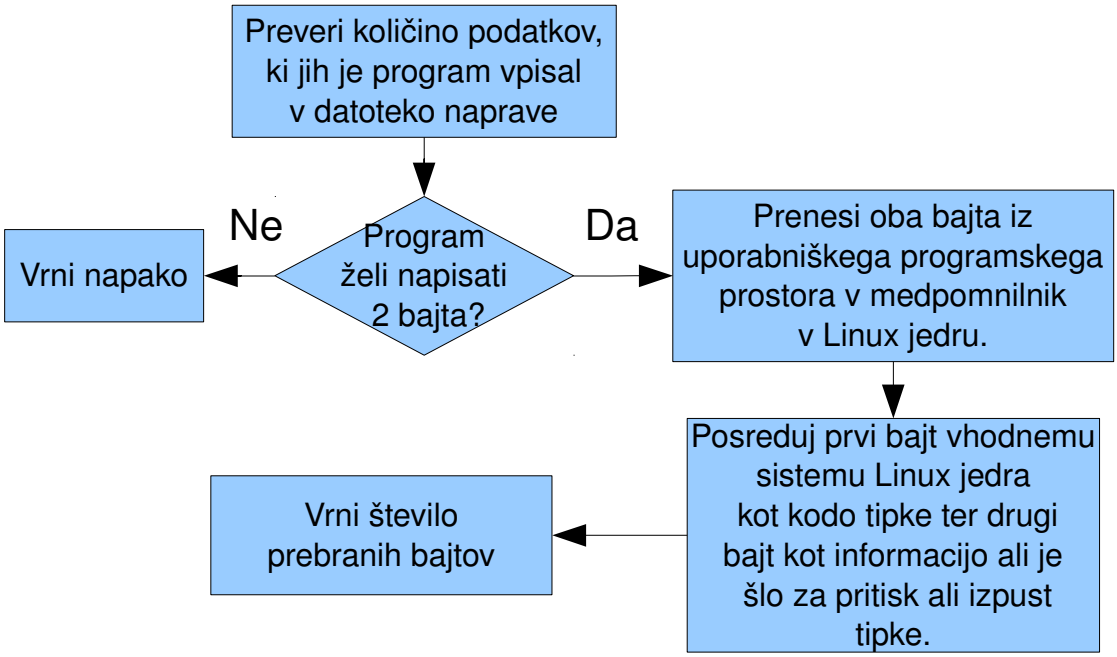
V Linux jedru različice 2,6 je ustaljena praksa, da uporabimo dinamični način določanja major številke, kjer nam major številko določi sistem sam ob nalaganju gonilnika. Pri uporabi dinamičnega določanja major številke, moramo tako gonilnik naložiti, preden ustvarimo datoteko naprave. Katero major številko uporablja kateri gonilnik, lahko vidimo v datoteki */proc/devices*.

Postopek implementacije gonilnika za znakovno datoteke naprave je na dolgo in široko opisan v [1], zato si ga tu ne bomo ogledali. Za nas je zanimiva le funkcija, ki jo moramo deklarirati, da gonilnik ve, kaj mora narediti, ko začne kak program pisati v znakovno datoteko naprave. Slika 13 nam prikaze osnovni koncept dotične funkcije.

Najprej preverimo, koliko podatkov je bilo vpisanih. Če osvežimo spomin, program, ki skrbi

za komunikacijo med igralniškim vmesnikom in gonilnikom, vedno piše dva bajta, zato v primeru, da je bilo vpisanih več oziroma manj bajtov, javimo napako. V primeru, da sta bila zapisana dva bajta, ju moramo najprej prenesti iz uporabniškega programskega prostora v pomnilniški prostor jedra. Na srečo nam za to operacijo Linux jedro ponuja svoje funkcije. Vsekakor pa velja opozoriti, da ne gre zgolj za navadno kopiranje podatkov, saj se uporabniški programski prostor nahaja v drugem segmentu kot pomnilniški prostor Linux jedra.

Po uspešnem prenosu podatkov v pomnilniški prostor jedra moramo le-te poslati vmesniku dogodkov Linux jedra. Več o tem si bomo ogledali v naslednjem poglavju. Za konec lahko se omenimo, da funkcija vrne število prenesenih bajtov.



Slika 13: Diagram funkcije, ki skrbi za prenos podatkov

8.4 VMESNIK DOGODKOV ZA V/I NAPRAVE

Za konec si poglejmo, kako deluje dotični vmesnik. Tako kot gonilnik za datoteko naprave je tudi vmesnik dogodkov standardni del Linux jedra. Zelo lepo je dokumentiran v sami dokumentaciji Linux jedra [8], tako da si bomo tu ogledali zgolj za nas pomembne stvari.

Gonilnik, ki želi uporabiti vmesnik se mora prijaviti v Linux jedro kot gonilnik vhodne naprave. Le-to pomeni, da moramo nastaviti strukturo, ki opisuje vhodno napravo, tako da

Linux jedro ve, kaj od vhodne naprave lahko pričakuje. Ker mi pošiljamo kode tipk, moramo gonilnik predstaviti kot tipkovnico. Tipke oziroma gumbi tvorijo vrsto dogodkov, ki jim pravimo dogodki EV_KEY. Poleg skupine dogodkov EV_KEY obstaja recimo tudi skupina EV_REL, ki predstavlja dogodke, ki jih tvori miška ali njej podobne naprave. Slednjo skupino smo navedli zgolj kot primer, saj za nas ne pride v poštev. Ko določimo skupino dogodkov, ki jih gonilnik podpira, moramo določiti še katere dogodke iz skupine podpiramo. V skupini dogodkov EV_KEY so tako tudi dogodki, ki jih prožijo igralni ploščki ter podobne naprave. Nas ti dogodki ne zanimajo, zato določimo, da gonilnik podpira zgolj dogodke KEY_*. To so dogodki, ki jih podpira standardna PC tipkovnica. Na ta način lahko gonilnik simulira standardno tipkovnico in tako dobimo želeno funkcionalnost.

9 SKLEPNE UGOTOVITVE

9.1 POVZETEK OPRAVLJENEGA DELA

V diplomskem delu sem prikazal korake razvoja igralnega vmesnika ter programske opreme, ki omogoča njegovo polno funkcionalnost. Diplomska naloga tako ni ozko usmerjena v določen problem, temveč sestoji iz vrste različnih enostavnih problemov. Med reševanjem le-teh se tako dotaknemo razvoja rešitev za namenske sisteme kot tudi razvoja programske opreme za osebni računalnik.

Ker ne gre zgolj za teoretični izdelek, je med razvojem prišlo do določenih odločitev, ki mogoče na prvi pogled niso optimalne. Do le-teh je prišlo predvsem zaradi časovne omejitve izdelave diplome ter opreme, ki sem jo imel na voljo. Prav zaradi slednjega sem večino kompleksnosti rešil na osebnem računalniku, tako da je igralni vmesnik lahko ostal karseda enostaven. Na ta način sem tudi povečal obseg interesentov, ki bi ponovili postopek in si izdelali svoj primerek igralnega vmesnika.

9.2 OSEBNA OCENA SISTEMA

Priznati moram, da sem bil dokaj skeptičen, ko se mi je porodila ideja za koncept igralnega vmesnika. Predvsem me je skrbela odzivnost sistema, saj moramo poleg zamika, ki ga povzroči odčitavanje stanja na igralnem vmesniku ter pošiljanja podatkov na osebni računalnik, upoštevati tudi pot, ki jo informacija opravi na osebnem računalniku. Tu informacijo najprej prebere gonilnik za serijska vrata, ki jo nato posreduje v uporabniški programski prostor, bolj natančno programu, ki jo obdela ter posreduje nazaj Linux jedru. Tu zaostanke težko določimo, saj so odvisni od obremenjenosti sistema v danem trenutku.

Za testiranje sem tako izbral igranje iger v simulatorju igralnih avtomatov M.A.M.E (http://mamedev.org/), ki spada med procesorsko bolj zahtevne simulatorje igralnih avtomatov. Med testiranjem ni bilo opaziti neodzivnosti, zato osebno menim, da je sistem izpolnil svojo nalogo.

9.3 MOŽNE NADALJNJE IZBOLJŠAVE

Kot smo že omenili, je tu opisani igralni vmesnik zelo okrnjen, saj smo morali nalogo prilagoditi vnaprej določenemu časovnemu okvirju. Iz tega sledi, da imamo mnogo možnosti za razvoj izboljšane verzije.

V diplomski nalogi smo se omejili na osem digitalnih vhodov, kar zadostuje za enega igralca. Pri prvi izboljšavi lahko tako povečamo število digitalnih vhodov. Na prvi pogled enostavna izboljšava nam prinese kup preglavic. Najprej bi morali preveriti, ali ima trenutno uporabljeni mikrokrmilnik sploh zadostno število vhodnih nožic. Poleg tega bi morali spremeniti celotno komunikacijo med mikrokrmilnikom ter računalnikom. Trenutno prenašamo zgolj en bajt, ki opiše celotno stanje gumbov na igralnem vmesniku. V primeru povečanja števila digitalnih vhodov to ne bi bilo več mogoče.

Pri naslednji izboljšavi lahko dodamo možnost uporabe analognih vhodov, ki jih potrebujemo v primeru uporabe analogne igralne palice. To pomeni, da bi morali razširiti funkcionalnost prav vseh v diplomi opisanih delov igralnega vmesnika. Kljub tako velikemu številu sprememb pa lahko ohranimo v diplomskem delu opisani koncept igralnega vmesnika.

Možnih izboljšav je tako res veliko. Seveda se je pri vsaki od le-teh potrebno na prvem mestu vprašati, kaj dejansko z njo pridobi uporabnik. Poleg tega moramo imeti vedno v mislih kompleksnost sistema. Čim enostavnejši sistem imamo, večja je verjetnost, da ga bo kdo uporabil.

SEZNAM UPORABLJENIH VIROV

[1] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, Linux Device Drivers, Third Edition, O'Reilly Media, 2005, pogl. 3

[2] (09.04.2011) ATmega8 Datasheet. Dostopno na:http://www.atmel.com/dyn/products/product_card.asp?part_id=2004

[3] (09.04.2011) AVR Libc. Dostopno na: http://www.nongnu.org/avr-libc/

[4] (09.04.2011) Libre Office. Dostopno na: http://www.libreoffice.org/

[5] (09.04.2011) MAX233 Datasheet. Dostopno na: http://www.maxim-ic.com/datasheet/index.mvp/id/1798

[6] (09.04.2011) V-USB. Dostopno na: http://www.obdev.at/products/vusb/index.html

[7] (1.3.2011) Getting a Vendor ID. Dostopno na: http://www.usb.org/developers/vendor/

[8] (1.3.2011) Programming input drivers. Dostopno na: http://lxr.linux.no/#linux+v2.6.37.2/Documentation/input/input-programming.txt

[9] (22.2.2011) RS-232. Dostopno na: http://en.wikipedia.org/wiki/RS-232

[10] (3.6.2008) A Guide to Debouncing. Dostopno na: http://www.ganssle.com/debouncing.htm