Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jure Krašna

Aplikacija za dostop do terminala preko spletnega vmesnika na operacijskem sistemu Linux

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX. Diagrami so izdelani z orodjem yEd.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00540 / 2013
Datum: 15.9.2013

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

Kandidat: **JURE KRAŠNA**

Naslov: **APLIKACIJA ZA DOSTOP DO TERMINALA PREKO SPLETNEGA VMESNIKA NA OPERACIJSKEM SISTEMU LINUX**
AN APPLICATION FOR ACCESSING A TERMINAL VIA THE WEB INTERFACE ON A LINUX OPERATING SYSTEM

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Razvijte program, ki omogoča dostop do ukazne lupine preko spletnega terminala ter zažene željeno aplikacijo EPICS kot prikriti proces. Program naj z zagnano aplikacijo komunicira preko njene ukazne lupine z uporabo tehnologije psevdo terminalov. Pri razvoju se omejite na operacijski sistem Linux.

Mentor:
izr. prof. dr. Patricio Bulić

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jure Krašna, z vpisno številko **63040080**, sem avtor diplomskega dela z naslovom:

Aplikacija za dostop do terminala preko spletnega vmesnika na operacijskem sistemu Linux

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricja Bulića
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. marca 2014 Podpis avtorja:

Zahvaljujem se izr. prof. dr. Patriciu Buliću za napotke in nasvete pri izdelavi diplomske naloge.

Iskreno se zahvaljujem tudi dekletu, družini in prijateljem, ki so me vzpodbujali in podpirali tekom študija.

Kazalo

Seznam kratic in simbolov

AJAX	Asynchronous JavaScript and XML.
ASCII	American Standard Code for Information Interchange.
BSD	Berkeley Software Distribution.
CA	Channel Access.
CGI	Common Gateway Interface.
CP/M	Control Program for Microcomputers.
CSS	Cascading Style Sheets.
DOM	Document Object Model.
DOS	Disk Operating System.
EPICS	Experimental Physics and Industrial Control System.
FastCGI	Fast Common Gateway Interface.
FIFO	First In First Out.
GNU	"GNU's not Unix".
HTML	Hyper Text Markup Language.
idr.	in drugi.

IOC	Input Output Controller.
JSON	JavaScript Object Notation.
JSP/Java	JavaServer Pages.
MSSQL	Microsoft Structured Query Language Server.
npr.	na primer.
PHP	Hypertext Preprocessor.
PLC	Programmable Logic Controller.
RTEMS	Real-Time Executive for Multiprocessor Systems.
SQL	Structured Query Language.
SSH	Secure Shell.
SUS	Single UNIX Specification.
TCP	Transmission Control Protocol.
TCP/IP	Internet protocol suite.
URL	Uniform Source Locator.
UTF	Unicode Transformation Format.
XML	Extensible Markup Language.

Povzetek

Okolje EPICS še vedno uporablja vrstične ukazne vmesnike za razvoj distribuiranih kontrolnih sistemov. Vsaka aplikacija razvita z okoljem vsebuje ukazno lupino, s katero ob zagonu prebere in izvede ukaze iz zagonske skripte. Ukazna lupina je med razvojem aplikacije nepogrešljivo orodje, ko pa želimo aplikacijo uvesti v produkcijsko okolje, nas lupina ovira. Aplikacija se mora izvesti kot prikriti proces, saj lahko le tako zagotovimo, da se izvaja neodvisno od preostalih procesov v sistemu in da ni povezana na noben nadzorni terminal. Ker okolje EPICS ne zadosti tem zahtevam, smo razvili aplikacijo, ki omogoča dostop do ukazne lupine preko spletnega terminala. Aplikacija zažene željeno aplikacijo EPICS kot prikriti proces in komunicira z njeno ukazno lupino z uporabo tehnologije psevdo terminalov. Pri razvoju smo se omejili na rešitev za operacijski sistem Linux.

Ključne besede:

Vrstični ukazni vmesniki, okolje EPICS, psevdo terminali, spletni terminal.

Abstract

EPICS framework still uses console line interfaces for distributed control systems development. Each EPICS application provides a command shell as part of its core functionality, which is used to parse and execute commands from the startup script. The EPICS shell is an invaluable tool, but becomes an obstacle once we want to deploy the application into a production environment. The application is required to run as a daemon, because we need to guarantee that it is independent of other processes and that it is dissociated from the controlling terminal. We developed a web application that can run EPICS framework applications as daemons, since they are unable to do that on their own. The application also connects to the EPICS shell and makes it accessible trough its web interface. Communication between the web interface shell and the EPICS application shell is implemented using pseudo terminals. Implementation of the solution is limited to Linux systems only.

Keywords:

Console line interfaces, EPICS framework, pseudo terminals, web terminal.

Poglavje 1

Uvod

Grafični in spletni vmesniki so v zadnjih letih prevzeli tako poslovni trg kot trg zabavne elektronike. Še vedno pa obstajajo področja, kot so distribuirani kontrolni sistemi, pri katerih prevladujejo vrstični ukazni vmesniki. Taki sistemi so v večji meri sestavljeni iz komponent, ki izvajajo kontrolne procese v ozadju, manjši del pa predstavljajo komponente v ospredju z grafičnim uporabniškim vmesnikom. Kot primer lahko vzamemo okolje EPICS (Experimental Physics and Industrial Control System), ki omogoča razvoj distribuiranih kontrolnih sistemov.

Glavna enota okolja EPICS je aplikacija, ki v jedru vsebuje ukazno lupino, s katero ob zagonu prebere in izvede ukaze iz zagonske skripte ter naloži bazo aplikacije v spomin. Poleg izvajanja zagonskih ukazov nam lupina z velikim naborom pomožnih ukazov omogoča dostop do podatkov v bazi. Med razvojem je ukazna lupina nepogrešljivo orodje, ko pa želimo aplikacijo uvesti v produkcijsko okolje, nas lupina ovira. Aplikacija se mora zagnati kot prikriti proces, saj lahko le tako zagotovimo, da se izvaja neodvisno od preostalih procesov v sistemu in da ni povezana na noben nadzorni terminal. Aplikacije EPICS v osnovi niso razvite za izvajanje kot prikriti proces, kar pomeni, da jih v tem načinu ne moremo neposredno izvajati.

Razvili smo spletno aplikacijo, ki zažene aplikacijo EPICS kot prikriti proces in se poveže na njeno ukazno lupino z uporabo tehnologije psevdo terminalov. V spletni vmesnik, imenovan tudi spletni terminal, smo vgradili funkcionalnosti asinhrone komunikacije s strežnikom in tako omogočili, da se vmesnik samodejno osvežuje, ko so na strežniku na voljo novi podatki. Za komunikacijo s spletnim

strežnikom smo uporabili protokol FastCGI (Fast Common Gateway Interface), saj se na ta način izognemo integraciji spletnega strežnika v našo rešitev. Omejili smo se na razvoj rešitve za operacijski sistem Linux.

V prvem delu naloge bomo opisali terminale z ukazno vrstico in njihovo delovanje ter psevdo terminale, ki nam omogočajo preusmeritev virtualnih terminalov aplikacij in posledično predajo nadzora zunanji aplikaciji. Pregledali bomo tudi programsko okolje za razvoj distribuiranih kontrolnih sistemov EPICS in njegove lastnosti, ki vplivajo na arhitekturo rešitve. Sledile bodo še tehnologije, ki so v uporabi pri izdelavi spletnih vmesnikov. V drugem delu naloge bomo zasnovali arhitekturo aplikacije, ki reši zastavljeni problem z uporabo omenjenih tehnologij in opisali izvedbo. Zaključili bomo s prednostmi in slabostmi rešitve ter pregledali možnosti izboljšav.

Poglavje 2

Vrstični ukazni vmesniki in psevdo terminali

Od izuma grafičnega uporabniškega vmesnika v šestdesetih letih dvajsetega stoletja se pogostost uporabe vrstičnih ukaznih vmesnikov postopno zmanjšuje. V grafičnih uporabniških vmesnikih so se kot nadomestilo pojavili virtualni terminali, ki omogočajo uporabo konceptov vrstičnih ukaznih uporabniških vmesnikov v novejših grafičnih okoljih. Na večini operacijskih sistemov, ki so danes v uporabi, lahko tako najdemo emulator terminala, ki omogoča dostop do storitev sistema z uporabo ukaznih lupin.

2.1 Vrstični ukazni vmesniki

Vrstični ukazni vmesniki, poznani tudi pod imenom konzolni vmesniki, omogočajo uporabniku izvajanje vrstičnih ukazov namenjenih specifičnemu računalniškemu programu imenovanem ukazna lupina, ki jih interpretira in izvede. Ukazi so v obliki zaporednih vrstic besedila.

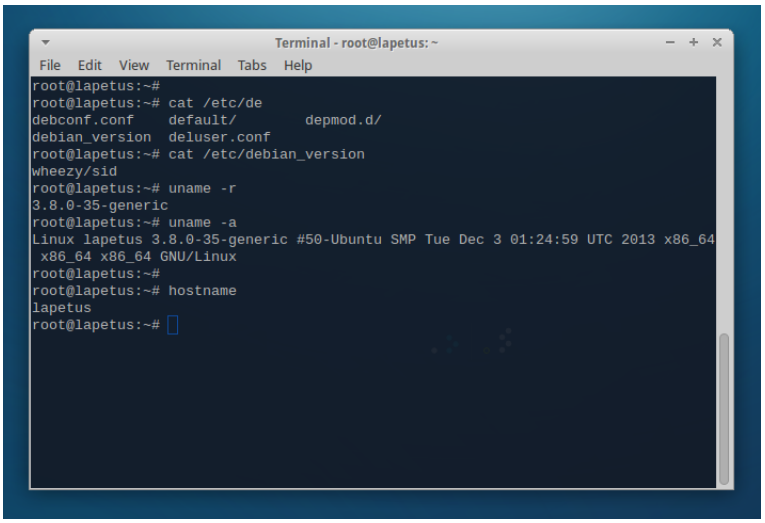
Razvili so se iz dialoga, ki so ga operaterji nekoč vodili preko naprav poimenovanih teleprinterji (slika 2.1, levo). Izmenjava informacij je med operaterji potekala tako, da so pošiljali in sprejemali vsako vrstico posebej. Teleprinterji so bili glavna metoda interakcije ljudi z računalniškimi sistemi do uvedbe video terminala (slika 2.1, desno) v sredini šestdesetih let. V sedemdesetih in osemdesetih

Slika 2.1: Teleprinter na levi in terminal VT100 na desni.

letih so bili vrstični ukazni vmesniki glavno sredstvo za dostop v operacijskih sistemih MS-DOS, CP/M in Apple DOS [3]. Kljub pojemajoči priljubljenosti so se v uporabi ohranili do današnjih dni.

Domači uporabniki se dandanes izogibajo uporabi vrstičnih ukaznih vmesnikov, saj so jim bližje grafični uporabniški vmesniki. Bolj razširjeni so med naprednimi uporabniki, ker zagotavljajo strnjen in zmogljivejši nadzor nad aplikacijo ali operacijskim sistemom. To se še posebej opazi na operacijskem sistemu Linux, kjer so emulatorji terminala (slika 2.2) in ukazne lupine, kot je na primer Bash, zelo razširjeni, saj imajo široke možnosti skriptiranja. K priljubljenosti ukaznih lupin prispevajo tudi možnost avtomatizacije z uporabo skriptnih jezikov in številne že obstoječe aplikacije in orodja.

Pomembna lastnost vrstičnih ukaznih vmesnikov je tudi njihova preprostost, zaradi česar so uporabni v primerih, kjer je razvoj kompleksnih uporabniških vmesnikov nesmiseln. To lahko opazimo predvsem pri distribuiranih kontrolnih sistemih, kjer se komponente sistema izvajajo neodvisno od nadzornih aplikacij z uporabniškimi vmesniki ali pa kot prikriti procesi. V takih primerih uporabljajo vrstične ukazne vmesnike za interakcijo med izvajanjem in v obdobju razhroščevanja programske komponente.



Slika 2.2: Emulator terminala na operacijskem sistemu Xubuntu.

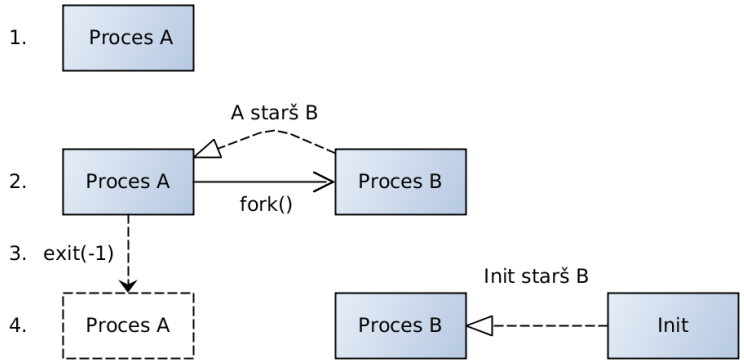
2.2 Prikriti procesi na operacijskem sistemu Linux.

Prikriti procesi (angl. daemon) so procesi, ki se izvajajo v ozadju in izvajajo generična sistemska opravila ali uporabniške naloge brez interakcije uporabnika. Običajno se izvedejo ob zagonu operacijskega sistema in se izvajajo s pravicami administrativnega uporabnika (angl. root user) ali uporabljajo pravice uporabnika posebej ustvarjenega zanje. Procesi v ozadju morajo izpolnjevati dve splošni zahtevi:

- Izvajati se morajo kot podproces primarnega procesa Linux imenovanega *"init"*.
- Ne smejo biti povezani na noben nadzorni terminal.

V splošnem mora vsak program, ki ga želimo izvajati kot prikriti proces, izvesti pravilno zaporedje sistemskih ukazov in klicev [4].

1. Najprej pokličemo sistemsko funkcijo *fork()*, ki ustvari nov proces. Po končanem klicu imamo starševski proces in otroški proces, ki ga bomo spremenili v prikriti proces (točki 1 in 2 na sliki 2.3).



Slika 2.3: Del postopka za ustvarjanje novega prikritega procesa. Na sliki lahko vidimo, kako glavni proces "Init"posvoji proces, če se starševski proces konča, preden se otroški proces zaključi.

2. V starševskem procesu izvedemo sistemski klic *exit()*. Starš starševskega procesa takrat privzame, da se je starševski proces končal izvajati in da prikriti proces ni več vodja nobene procesne skupine, kar je zahteva za naslednji korak. Ker otroški proces nima več starša, ga posvoji proces *"Init"* (točki 3 in 4 na sliki 2.3).

3. Klic funkcije *setsid()* dodeli prikritemu procesu novo identifikacijsko številko skupine in seje, tako da prikriti proces postane vodja obeh. S tem zagotavlja, da prikriti proces ni povezan na noben nadzorni terminal.

4. Sledi sprememba trenutnega delovnega imenika na korenski imenik z uporabo klica *chdir()*. To je potrebno, ker je podedovani delovni imenik lahko kjerkoli v datotečnem sistemu. Privzeti procesi delujejo celotni čas delovanja sistema, zato ne želimo imeti naključnih imenikov odprtih. Na ta način preprečujemo administratorju sistema, da bi izpel datotečni sistem, ki bi vseboval ta imenik.

5. Zapreti moramo vse opisnike datotek, ki so še odprti.

6. Na koncu odpremo opisnike datotek z indeksi 0, 1 in 2, ki so namenjeni standardnemu vhodu, standardnemu izhodu ter izhodu napak, in jih preusmerimo v */dev/null*.

Iz zgornjih navodil lahko opazimo, da je nemogoče izvajati programe z ukazno lupino kot prikriti proces, saj mora bit program že v osnovi implementiran tako, da to omogoča. Poleg tega prikriti procesi zahtevajo, da prekinemo dostop do vseh terminalov, s čimer onemogočimo dostop do uporabniškega vmesnika.

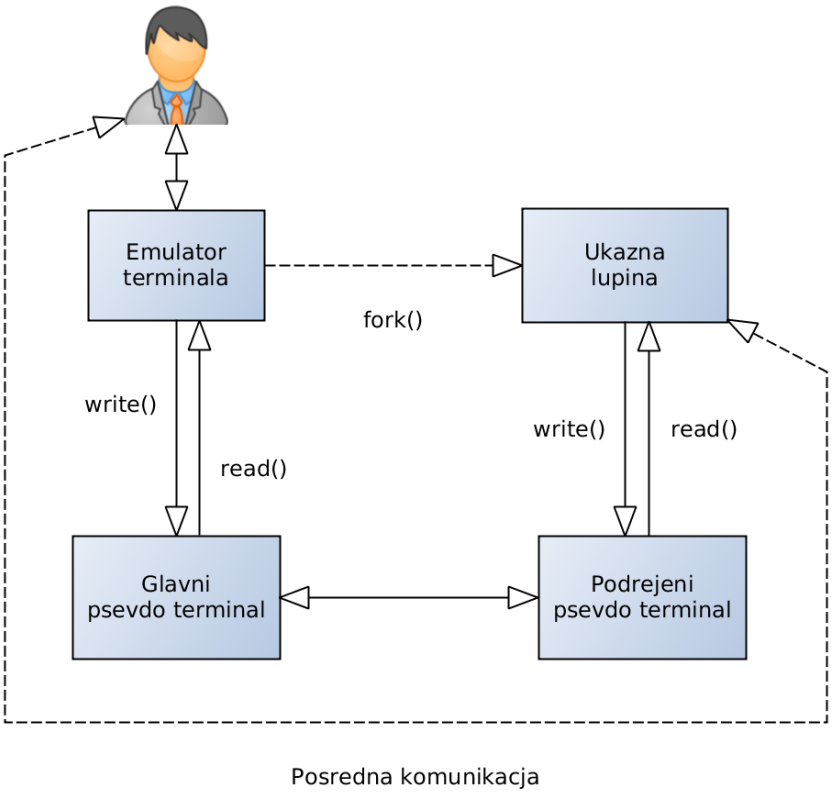
2.3 Psevdo terminali

Psevdo terminal je dvojica psevdo vmesnikov. Prvi je podrejen in emulira pravo napravo s tekstovnim terminalskim vmesnikom, medtem ko drugi, glavni vmesnik, omogoča emulatorjem terminala nadzor nad podrejenim vmesnikom.

Psevdo terminali so se razvili na operacijskem sistemu BSD (Berkeley Software Distribution). Prve implementacije so bile specifične za vsako distribucijo posebej in so omejevale količino psevdo terminalskih parov na voljo uporabniku. Sčasoma so jih med razvojem modernejših operacijskih sistemov standardizirali in vključili v specifikacijo SUS (Single UNIX Specification).

Implementacija psevdo terminalov, ki jih najdemo na novejših operacijskih sistemih, uporablja glavni razdelilnik psevdo terminalov (angl. pseudo-terminal master multiplexer). Do slednjega dostopamo preko opisnika datoteke */dev/ptmx*, kar nam omogoča dodeljevanje neomejenega števila podrejenih psevdo vmesnikov. Ko odpremo razdelilnik psevdo terminalov s funkcijo *open("/dev/ptmx")* ali *posix_openpt()*, nam ta vrne opisnik datoteke za glavni vmesnik ter poskrbi, da se ustvari povezani podrejeni vmesnik */dev/pts/N*.

Kot primer uporabe psevdo terminalov lahko navedemo emulatorje terminalov. Vloga emulatorja je interakcija z uporabnikom, psevdo terminali pa poskrbijo za povezavo med emulatorjem in ukazno lupino (slika 2.4). Ko uporabnik vnaša tekst, ga proces prebere in zapiše v glavni vmesnik, ukazna lupina pa ga prebere s podrejenega vmesnika. Ukazna lupina nato zapiše odgovor v podrejeni vmesnik, ki ga emulator terminala prebere in prikaže uporabniku. Med najbolj razširjenimi emulatorji terminala so: xterm, GNOME Terminal, Konsole in Mac OS X Terminal. Programi za oddaljeno prijavo, kot so strežniki za protokole SSH (Secure Shell) in Telnet, pa delujejo na enak način, le da komunicirajo z oddaljenim uporabnikom namesto z lokalnim [5].



Slika 2.4: Uporabnik posredno komunicira z aplikacijo preko psevdo terminalskega para.

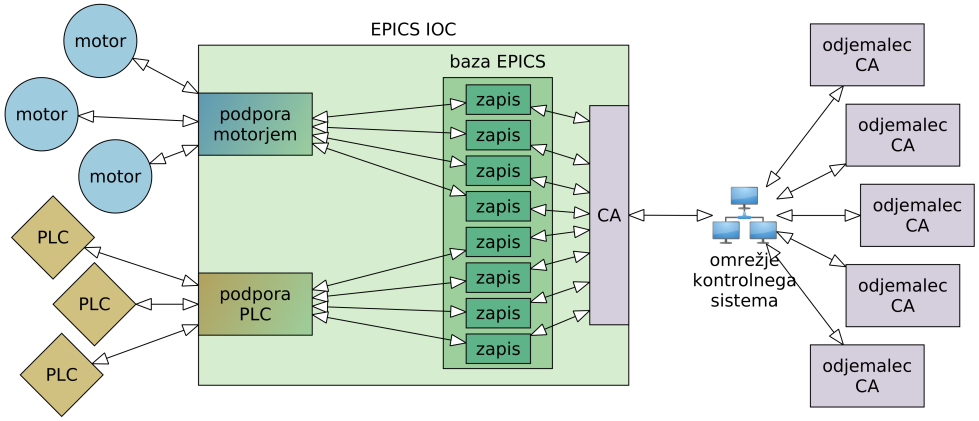
Poglavje 3

EPICS

EPICS (Experimental Physics and Industrial Control System) je programsko okolje, ki se uporablja za razvoj distribuiranih kontrolnih sistemov, ki običajno zajemajo veliko število računalnikov med seboj povezanih v omrežje. Taki kontrolni sistemi so namenjeni nadzoru pospeševalnikov delcev, teleskopov in drugih večjih eksperimentalnih naprav. Okolje omogoča preprost, modularen razvoj komponent sistema s standardiziranimi vmesniki ter medsebojni nadzor in prenos informacij med komponentami.

Uporablja se za zajemanje podatkov, avtomatizacijo in različne vrste nadzora nad napravami v distribuiranem kontrolnem sistemu. Arhitektura omogoča porazdeljeno procesiranje med napravami, ki so del lokalnega omrežja. Uporabljen je model odjemalec-strežnik, ki s protokolom CA (Channel Access) omogoči medsebojno naročanje in obveščanje. Strežniki, ki jih v okolju EPICS imenujemo IOC (Input Output Controller), izvajajo aplikacije EPICS, s pomočjo katerih v realnem času nadzirajo in zbirajo podatke z merilnih naprav, ki so priključene nanje. Zbrane informacije nato posredujejo odjemalcem preko protokola CA.

EPICS je rezultat sodelovanja med več organizacijami. Prvotno sta ga napisala ameriška laboratorija LANL (Los Alamos National Laboratory) in ANL (Argonne National Laboratory), danes pa ga uporabljajo mnoge znanstvene organizacije po celem svetu. Implementacija okolja je bila od začetka uporabe leta 1994 in nadaljnjih deset let na voljo samo za operacijski sistem vxWorks proizvajalca Wind River. Od leta 2004 dalje pa je dostopna tudi na operacijskih sistemih GNU/Linux, Solaris, Microsoft Windows, MacOS in RTEMS. V zadnjih letih se pojavlja tudi



Slika 3.1: Hipotetični primer implementacije naprave IOC. Na napravo so priključeni motorji in naprave PLC, ki so v okolje EPICS integrirani s podpornimi moduli.

programska oprema, ki omogoča drugim kontrolnim sistemom, da delujejo kot EPICS strežniki. Nasprotno pa je protokol CA že od nekdaj na voljo široki paleti sistemov. Knjižnice z implementacijo odjemalca protokola CA tako najdemo v programskih jezikih C/C++, Java, Python, LabView, Perl in mnogih drugih [6].

3.1 Delovanje in uporaba

Jedro vsake EPICS aplikacije je podatkovna baza, ki poleg samih podatkov hrani tudi informacije o strukturi podatkov. Glaven sestavni del baze so zapisi (angl. records), ki vsebujejo več polj. EPICS pozna različne tipe zapisov, kot so na primer: analogni vhod (`ai`), analogni izhod (`ao`) in še mnogo drugih. Vsak tip zapisa ima točno določena polja, vsako polje pa svoj namen. Nekatera polja so skupna vsem tipom zapisov, druga pa so specifična. Protokol CA je zasnovan tako, da moramo vsakemu ustvarjenemu zapisu dodeliti svoje ime, ki mora biti enolično na vseh napravah IOC, povezanih v isto omrežje TCP/IP (Internet protocol suite). Tudi polja so poimenovana, vendar so njihova imena določena z izbranim tipom zapisa.

Okolje EPICS je namenjeno nadzoru fizičnih naprav (npr. motorjev, tiskanih plošč za zajemanje podatkov, programabilnih logičnih krmilnikov itd.). Zapisi so v

tem kontekstu osnovni gradniki, s katerimi dosežemo abstrakcijo fizičnih signalov in parametrov na napravah, ki jih integriramo v kontrolni sistem. Problem s katerim se lahko srečamo je, da naprave uporabljajo različne načine komunikacije s kontrolnim sistemom. Nekatere omogočajo dostop in nadzor preko protokola TCP/IP, druge preko preprostih protokolov ASCII (American Standard Code for Information Interchange) in serijskih vmesnikov, kot so RS232, tretje, kot so na primer sistemi PLC (Programmable Logic Controller), pa uporabljajo protokol Modbus. EPICS zato omogoča razširjanje funkcionalnosti zapisov s programsko kodo napisano v jeziku C, kar uporabimo za integracijo različnih tehnologij v isti kontrolni sistem (slika 3.1). Funkcije tako razširjenih zapisov so v večini primerov hranjenje podatkov prebranih z naprave, vpis podatkov za zapis na napravo in izvajanje ukazov na napravi [7].

3.2 Struktura aplikacije

Aplikacije EPICS imajo standardizirano strukturo imenikov ter uporabljajo sistem *GNU Make* za gradnjo. Okolje nam nudi skripte, s katerimi ustvarimo ogrodje aplikacije, v katerega med razvojem vstavljamo baze z zapisi in datoteke z izvorno kodo. Vsaka aplikacija je že ob ustvarjanju opremljena z datotekami Makefile, kar nam olajša gradnjo aplikacije in vključevanje izvornih datotek. Ko se postopek konča, so nam na voljo trije glavni deli potrebni za izvajanje vsake aplikacije EPICS:

- **Baza z zapisi** je ena ali več tekstovnih datotek, ki vsebujejo zapise.
- **Zagonska skripta** je tekstovna datoteka z ukazi, ki se morajo izvesti ob zagonu aplikacije.
- **Izvršljiv računalniški program**, ki vsebuje jedrno funkcionalnost za obdelavo zapisov in opcijsko dodatno funkcionalnost potrebno za komunikacijo z napravami. Program izvede zagonsko skripto in naloži bazo z zapisi v spomin.

3.3 Ukazna lupina

Ukazna lupina okolja EPICS je preprost tolmač ukazov. Uporablja se za interpretiranje zagonskih skript, ki so del vsake aplikacije sistema EPICS in so sestavljene iz vrstičnih ukazov, ki jih tolmač ukazov sekvenčno prebere in izvede [9]. Aplikacija ob zagonu prebere zagonsko skripto in jo kot vhodne podatke preda ukazni lupini, ki nato prebere vsako vrstico posebej. Parametre razširi s spremenljivkami okolja, vrstico razbije na ukaz in argumente ter pokliče funkcijo, ki pripada zaznanemu ukazu. Ukazi in argumenti so med sabo ločeni s presledki, vejicami in oklepaji. Naslednja ukazna vrstica

```
dbLoadRecords("db/dbExample1.db", "user=mrk")
```

se interpretira kot ukaz `dbLoadRecords` z argumentoma `"db/dbExample1.db"` in `"user=mrk"`.

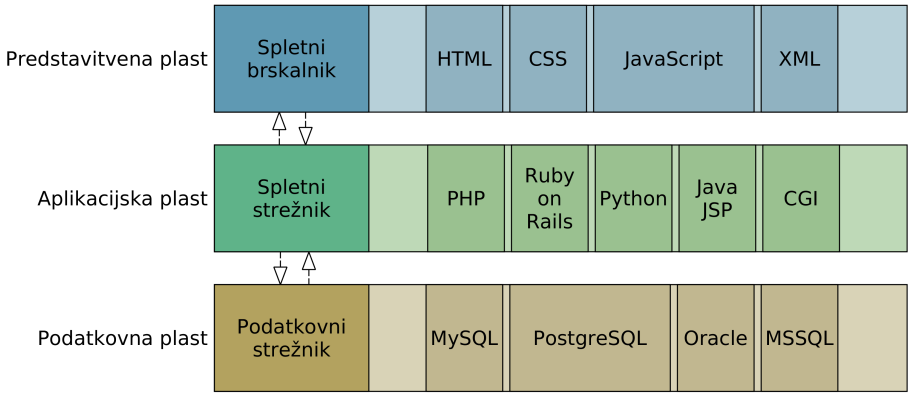
Če tolmač ukazov naleti na nepoznan ukaz, izpiše opozorilo in ukaz preskoči. Manjkajočim argumentom dodeli privzeto vrednost, ki je `0` za numerične argumente in `NULL` za znakovne argumente, odvečne argumente pa ignorira. Argumenti, ki vsebujejo presledke, morajo biti obdani z dvojnimi narekovaji, v nasprotnem primeru pa narekovaji niso potrebni.

Poglavje 4

Spletne aplikacije in tehnologije

Spletne aplikacije so priljubljene zaradi splošne razširjenosti spletnih brskalnikov in udobja, ki ga uporaba brskalnika ponuja. Poleg tega so v veliki večini primerov neodvisne od okolja, ki ga odjemalec uporablja, kar olajša razvoj. Ena ključnih lastnosti, zaradi katere zadnje čase nadomeščajo namizne aplikacije, pa je njihova zmožnost nadgradnje in vzdrževanja, brez potrebe po distribuciji in nameščanju programske opreme na sistemih odjemalcev.

Razdelimo jih lahko v logične plasti, kjer ima vsaka plast svojo vlogo. Klasične namizne aplikacije so praviloma sestavljene samo iz ene plasti, ki se nahaja na sistemu odjemalca, medtem ko spletne aplikacije težijo k večplastnosti. Najpogosteje je v uporabi triplastni model (slika 4.1), kjer plasti delimo na predstavitveno, aplikacijsko in podatkovno plast [10]. Prvo plast sestavljajo spletni brskalnik in tehnologije za predstavitev vsebine, kot so CSS (Cascading Style Sheets), HTML (Hyper Text Markup Language) in JavaScript. Aplikacijsko plast zaseda spletni strežnik, ki izvaja aplikacijski pogon implementiran s tehnologijami za dinamično generiranje spletne vsebine, kot so CGI (Common Gateway Interface), JSP/Java (Java-Server Pages), PHP (Hypertext Preprocessor), Python, Ruby On Rails in drugi. Na podatkovni plasti pa navadno najdemo podatkovni strežnik (npr. MySQL, PostgreSQL, MSSQL idr.). Brskalnik pošilja zahteve na aplikacijsko plast, ki generira uporabniški vmesnik in ga vrne brskalniku. Med generiranjem se manjkajoči podatki preko poizvedb preberejo iz podatkovne baze.



Slika 4.1: Triplastni model spletnih aplikacij.

4.1 Protokol CGI

CGI je standarden protokol na aplikacijski plasti, s katerim generiramo dinamične vsebine za spletne strani in aplikacije. Kadar ga uporabljamo skupaj s spletnim strežnikom, nam nudi vmesnik med spletnim strežnikom in programi, ki generirajo spletno vsebino [11]. Ti programi so običajno napisani v skriptnih jezikih in so poznani kot skripte CGI. Programi CGI se izvajajo v ločenih procesih, ki jih spletni strežnik ustvari ob začetku obdelave zahteve in zaustavi, ko se obdelava zahteve konča. Zaradi takega pristopa so CGI programi zelo preprosti za implementacijo, vendar pa ustvarjanje vsakega posameznega procesa zahteva veliko sistemskih sredstev, kar omejuje učinkovitost in razširljivost aplikacije. Pri visokih obremenitvah postane poraba sistemskih sredstev nezanemarljivo potratna. Poleg tega procesni model omejuje ponovno uporabo skupnih sredstev, kot so povezave do podatkovne baze [12]. Kljub temu ima CGI protokol precej prednosti:

- Protokol je preprost in enostaven za razumevanje.
- Protokol je jezikovno neodvisen, saj so lahko aplikacije CGI napisane v skoraj kateremkoli programskem jeziku.
- Aplikacije se med seboj ne morejo poškodovati ali pa povzročiti izpada spletnega strežnika, ker se izvajajo v ločenih procesih.
- Protokol je odprt in standardiziran in zato podprt na vseh spletnih strežnikih.

- Protokol je arhitekturno neodvisen, saj ni vezan na specifično arhitekturo spletnega strežnika.

4.2 Protokol FastCGI

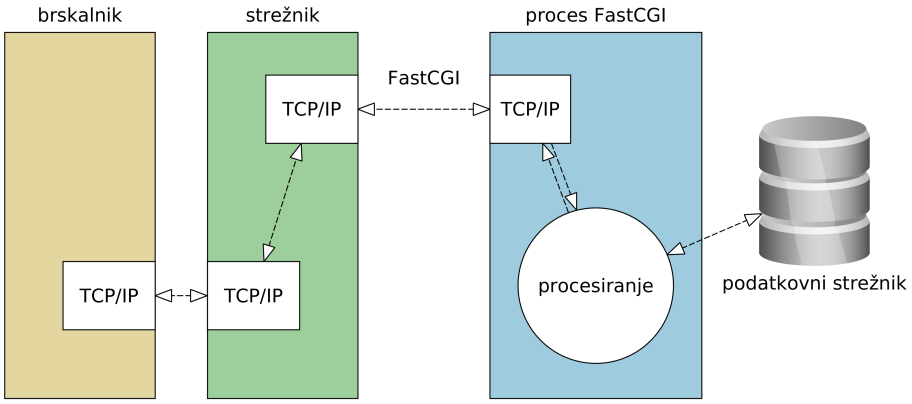
Protokol FastCGI se je pojavil kot odgovor na zmogljivostne probleme in omejitve protokola CGI. Protokol obdrži večino dobrih lastnosti protokola CGI in reši problematična področja [13]. Za obdelavo serije zahtev uporablja persistentni proces namesto ustvarjanja novega procesa ob vsaki zahtevi [14].

Proces, ki implementira protokol FastCGI (slika 4.2), se ustvari ali ob zagonu strežnika ali pa ga strežnik po potrebi ustvari ob prvi prispeli zahtevi. Ob zagonu se proces inicializira in čaka na povezavo s spletnega strežnika. Ko zahteva prispe na spletni strežnik, ta odpre novo povezavo TCP/IP do procesa FastCGI in posreduje prejeto zahtevo kot tudi vse informacije okolja. Proces odgovori na zahtevo preko iste povezave, jo zapre in s tem signalizira strežniku, da je obdelava zahteve zaključena. Nato čaka na ponovno povezavo s strežnika. Strežnik sprejme odgovor na obdelano zahtevo in počaka, da se povezava TCP/IP zapre, nato pa odgovor posreduje končnemu uporabniku.

Vsak proces FastCGI tako med izvajanjem obdela številne zahteve, s čimer se izognemo povečani porabi sistemskih sredstev, ki nastanejo zaradi ustvarjanja in zaključevanja ločenega procesa za vsako zahtevo. Več zahtev je naenkrat možno obdelati na več načinov. Prvi je uporaba ene povezave, ki jo multipleksiramo. Drugi je, da uporabimo več povezav med strežnikom in procesom. Tretji pa, da združimo obe tehniki. V vseh primerih lahko namestimo več procesov FastCGI, s čimer povečamo stabilnost sistema [12]. Ker so procesi FastCGI povezani s spletnim strežnikom preko TCP/IP, jih lahko izvajamo tudi na oddaljenih gostiteljih. Tako lahko porazdelimo obdelavo zahtev na sistem sestavljen iz več računalnikov, kar izboljša skalabilnost. Poleg tega nam omogoča, da posamezne procese FastCGI ponovno zaženemo ali nadgradimo, ne da bi povzročili izpad celotnega sistema.

Aplikacije FastCGI so na strani spletnega strežnika konfigurirane tako, da nastavitve vnesemo v konfiguracijsko datoteko. Nastavitve lahko razdelimo v dve skupini.

Prva obsega parametre aplikacijskega razreda, kot so:



Slika 4.2: Potek komunikacije med odjemalcem, strežnikom in aplikacijo FastCGI.

- Pot do izvršljive datoteke FastCGI aplikacije.
- Spremenljivke okolja, ki jih moramo nastaviti pred zagonom in argumente, ki jih moramo podati aplikaciji ob zagonu.
- Število procesov aplikacije, ki jih bomo pognali.

Pri aplikacijah, ki se izvajajo na oddaljenih gostiteljih, moramo poleg zgoraj naštetih parametrov vnesti še gostiteljev naslov in vrata TCP (Transmission Control Protocol). Strežnik predvideva, da se proces FastCGI že izvaja na gostitelju. Če strežnik sprejme zahtevo in se ne more povezati na aplikacijo na gostitelju, vrne uporabniku stran z napako [13].

Drugi logični del nastavitev zajema mapiranje aplikacijskega razreda določeni vlogi:

- Za vloge *odzivnika* (angl. responder role) administrator nastavi del naslova URL (Uniform Source Locator), iz katerega razberemo kateri proces FastCGI bo prejel zahtevo v obdelavo. Na primer vse naslove URL, ki se začnejo s serijo znakov `"/rollcall/"`, preusmerimo v aplikacijo, ki skrbi za dostop do podatkovne baze s podatki o zaposlenih.
- Za vloge *filtra* (angl. filter role) administrator nastavi pripono datoteke, ki jo bo obdelala filtrirna aplikacija. Na primer vse datoteke s pripono `"/.sql"` bi lahko obdelala aplikacija za iskanje poizvedb SQL.

- Za vloge *pooblastitelja* (angl. authorizer role) administrator nastavi aplikacijo za preverjanje avtorizacije in avtentikacije uporabnika. Na primer ko zahteva vsebuje podatke za identifikacijo uporabnika, jih aplikacija primerja s podatki v podatkovni bazi.

Večina spletnih strežnikov, ki jih danes najdemo na trgu, podpira uporabo protokola FastCGI [15]. Običajno so zasnovani tako, da omogočajo modularno dodajanje funkcionalnosti. Za FastCGI tako obstajajo komponente, ki dodajajo funkcionalnost protokola posameznim implementacijam spletnega strežnika. Kot primer lahko omenimo modul `mod_fastcgi` za spletni strežnik *Apache* [16] ali pa modul `ngx_http_fastcgi_module` spletnega strežnika *nginx* [17]. Spletni strežniki imajo zaradi ločenih modulov različno konfiguracijo protokola FastCGI.

4.3 Asinhroni JavaScript in XML

Asinhroni JavaScript in XML (AJAX) je skupina povezanih spletnih tehnik in tehnologij za implementacijo asinhronih spletnih aplikacij. Tehnologije v uporabi se izvajajo na strani odjemalca, torej v spletnem brskalniku, ki asinhrono komunicira s spletnim strežnikom. Kljub temu, da ime vsebuje XML (Extensible Markup Language), ta ni nujno potreben. Nadomestimo ga lahko s podobno strukturiranimi tekstovnimi formati, kot je na primer JSON (JavaScript Object Notation).

Vsebino, ki jo aplikacija prenese s strežnika, strukturiramo in jo slogovno opremimo s tehnologijami HTML in CSS. Z uporabo objektnega modela DOM (Document Object Model) in jezika JavaScript pa lahko vsebino prikažemo dinamično in se izognemo ponovnemu prenosu celotne spletne strani. Na ta način omogočimo uporabniku interakcijo z novo predstavljeno vsebino [18].

4.3.1 JSON

JSON je odprt standardiziran format, ki shranjuje podatke v tekstovni obliki. Podatki so shranjeni kot objekti, kjer je vsak objekt sestavljen iz atributa in vrednosti. Zaradi svoje oblike je preprost za branje in generiranje tako za ljudi kot za računalnike. Čeprav je implementacijsko neodvisen od programskih jezikov, najdemo podporo za format JSON v veliki večini programskih jezikov [20].

Format je zgrajen na temeljnih strukturah, ki jih poznamo v vseh programskih jezikih. Velikokrat se uporablja za serializacijo podatkov pri komunikaciji med aplikacijami, ki so napisane v različnih programskih jezikih. Glavne strukture jezika so:

- **število** z dvojno natančnostjo v zapisu s plavajočo vejico,
- **logična vrednost** `true` ali `false`,
- **niz znakov** v kodiranju UTF (Unicode Transformation Format), obdan z dvojnimi narekovaji,
- **objekt**, ki je zbirka parov sestavljenih iz atributa in vrednosti,
- **polje**, ki je urejen seznam vrednosti,
- **ničelna vrednost** `null`.

V objektih in poljih lahko na mesta za vrednosti vstavimo katerokoli od zgoraj naštetih struktur in jih rekurzivno razširimo. To nam omogoča, da lahko v format JSON pretvorimo poljubno kompleksne objekte [20].

Poglavje 5

Spletni vmesnik za dostop do konzole

V prejšnjih poglavjih smo se seznanili s problemi dostopa do konzole aplikacij, ki se izvajajo na oddaljenih računalnikih ter s tehnologijami razvoja bogatih spletnih aplikacij. Naš cilj je zasnovati in implementirati aplikacijo, ki bi omogočala zagon aplikacije EPICS kot prikriti proces in dostop do konzole aplikacije preko spletnega brskalnika.

5.1 Namen

Na svetovnem spletu lahko najdemo aplikacije, ki omogočajo isto funkcionalnost [21, 22], vendar je cilj naše aplikacije tesna integracija s kontrolnim sistemom EPICS. Odločili smo se, da ne bomo spreminjali obstoječih aplikacij, temveč zasnovali rešitev od začetka. Glavni razlog za to leži v načinu izvajanja komponent kontrolnega sistema, saj se lahko izvajajo po več mesecev ali tudi več let. Pri tem ustvarijo veliko izhodnih podatkov, ki jih je potrebno hraniti, vendar ne v pomnilniku, saj bi s tem neizogibno ogrozili stabilnost sistema. Ker se kontrolni sistem EPICS izvaja na lokalnem omrežju in ni javno dostopen, smo se izognili implementaciji varnostnih mehanizmov in omogočili dostop do konzole vsem uporabnikom na istem omrežju.

5.2 Zasnova rešitve

Da se izognemo terminološkim nesporazumom smo se odločili, da našo aplikacijo poimenujemo **nadzorna aplikacija**. Program, ki ga bo *nadzorna aplikacija* izvedla v ozadju in lahko predstavlja aplikacijo EPICS ali katero koli drugo aplikacijo z ukazno lupino, pa poimenujemo **aplikacija v ozadju**.

Po pregledu obstoječih rešitev in okolja EPICS smo za *nadzorno aplikacijo* predvideli naslednje osnovne zahteve:

1. *Nadzorna aplikacija* mora omogočati zagon *aplikacije v ozadju* kot prikriti proces.

2. *Nadzorna aplikacija* mora omogočati dostop in nadzor ukazne konzole *aplikacije v ozadju*.

3. *Nadzorna aplikacija* mora omogočati sprejemanje in pošiljanje podatkov brez osveževanja celotnega spletnega vmesnika.

4. *Nadzorna aplikacija* mora uporabljati čim manj sistemskih sredstev in ne sme ovirati delovanja *aplikacije v ozadju*.

Glavna naloga *nadzorne aplikacije* je generiranje spletnega terminala, preko katerega lahko uporabnik pošilja ukaze in sprejema odgovore iz *aplikacije v ozadju*. Uporabnikom moramo omogočiti dostop do spletnega vmesnika, zato smo v arhitekturo *nadzorne aplikacije* vključili tudi spletni strežnik, ki omogoča komunikacijo s spletnim brskalnikom na sistemu uporabnika.

Iz tretje zahteve lahko opazimo, da potrebujemo bogat spletni vmesnik, zato smo morali podpreti asinhrone prenose podatkov med strežnikom in *nadzorno aplikacijo*. Uporabili smo tehnologijo AJAX opisano v poglavju 4.3. Ker so taki prenosi lahko številni, smo za sporočila AJAX namesto formata XML uporabili JSON, saj so sporočila serializirana v tem formatu manjša po velikosti. Na ta način smo zmanjšali količino pasovne širine, ki jo aplikacija zaseda v omrežju kontrolnega sistema.

Da lahko dostopamo do podatkov vmesnika *aplikacije v ozadju*, mora *nadzorna aplikacija* vmesnik preusmeriti in se povezati z njim preko psevdo terminalov, ki smo jih opisali v poglavju 2.3. Implementacija preusmerjanja in komunikacije s

psevdo terminali je na voljo večini programskih jezikov, vendar je osnovna implementacija del standardne knjižnice jezika C na operacijskem sistemu Linux. Zaradi lažje integracije in večjega nadzora nad porabo pomnilniškega prostora smo se odločili, da *nadzorno aplikacijo* razvijemo v programskem jeziku C++.

Omenili smo že, da moramo *nadzorni aplikaciji* omogočiti komunikacijo z brskalnikom na uporabnikovem sistemu. Uporabimo lahko eno od naslednjih rešitev:

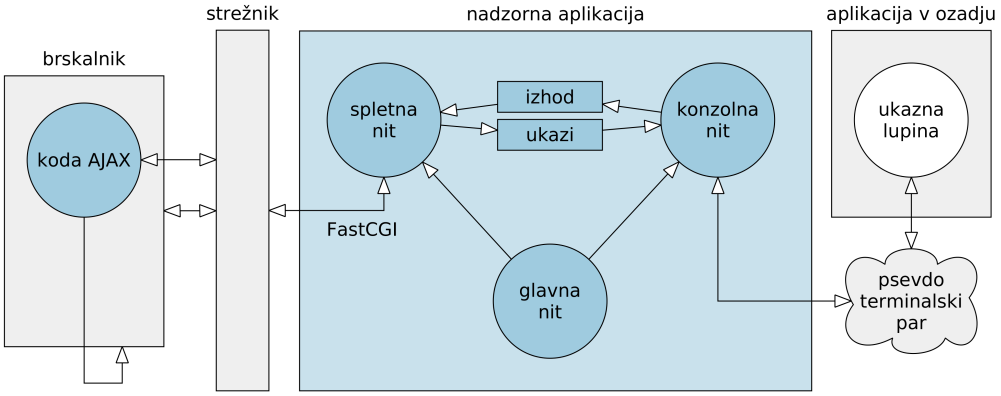
1. Napišemo preprosto implementacijo spletnega strežnika.
2. Integriramo obstoječo implementacijo spletnega strežnika.
3. Zasnujemo aplikacijo tako, da se izvaja neodvisno od spletnega strežnika.

Odločili smo se za tretjo možnost, ker na tržišču obstaja veliko dobrih spletnih strežnikov in ker vključevanje programske kode prinese veliko dodatnega vzdrževanja in nadgradenj ob novih verzijah. Za integracijo s spletnim strežnikom smo izbrali protokol FastCGI opisan v poglavju 4.2. Izbrali smo ga zaradi njegove preprostosti, dobre podpore v programskem jeziku C++ in možnosti integracije z večino modernih spletnih strežnikov.

5.3 Izvedba

Na sliki 5.1 je predstavljena arhitektura celotnega sistema. Na levi strani je brskalnik, v katerem se izvaja program, ki asinhrono komunicira z *nadzorno aplikacijo* in posodablja spletni vmesnik. Komunikacija poteka preko spletnega strežnika, ki izmenjuje sporočila protokola FastCGI z aplikacijo. V *nadzorni aplikaciji* imamo tri glavne niti. Te skrbijo za prenos izhoda ukazne lupine v brskalnik in za prenos ukazov iz brskalnika v ukazno lupino. Na desni strani lahko vidimo *aplikacijo v ozadju*, ki komunicira z *nadzorno aplikacijo* preko psevdo terminalskega para.

Nadzorno aplikacijo lahko ločimo na več funkcijsko različnih delov, ki pa delujejo enovito. Ker moramo neodvisno upravljati z dvema viroma, odjemalcem in ukazno konzolo, smo v aplikaciji uporabili več niti. Vsaka nit je del svojega razreda. Ti pa so implementirani tako, da omogočajo varen dostop do skupnih podatkovnih struktur, ki si jih niti delijo.



Slika 5.1: Potek komunikacije med posameznimi deli aplikacije in drugimi programskimi komponentami. Z modro barvo so označeni deli aplikacije, ki smo jih sami razvili.

5.3.1 Glavna nit

Glavna nit se začne izvajati ob zagonu *nadzorne aplikacije* in poskrbi, da se ostale niti inicializirajo in izvedejo. Med delovanjem sprejema signale operacijskega sistema in skrbi, da se signali pravilno obdelajo. Če na primer sprejme signal `SIGUSR1`, `SIGTERM` ali `SIGPIPE`, začne z varnim postopkom zaustavitve aplikacije.

5.3.2 Konzolna nit

Prva nit, ki jo *glavna nit* inicializira, je *konzolna nit* in skrbi za komunikacijo z *aplikacijo v ozadju*. Na začetku nit izvede ukaze, ki poskrbijo, da je komunikacija sploh možna, nato pa *aplikacijo v ozadju* izvede kot prikriti proces. Postopek, ki smo ga s pomočjo niti izvedli, je:

1. Pripravimo psevdo terminalski par tako, da odpremo glavni terminalski vmesnik s funkcijo `posix_openpt()`. Funkcija vrne opisnik datoteke glavnega terminala.

2. Odobrimo dostop do podrejenega psevdo terminala s funkcijo `grantpt()`. Kot parameter ji podamo opisnik datoteke glavnega terminala.

3. Odklenemo podrejeni psevdo terminal, ki ustreza glavnemu psevdo termi-

nalu s funkcijo `unlockpt()`. Kot parameter ji podamo opisnik datoteke glavnega terminala.

4. Pridobimo ime podrejenega psevdo terminala s funkcijo `ptsname()`. Funkcija vrne ime podrejenega terminala, kot parameter pa ji podamo opisnik datoteke glavnega terminala.

5. Odpremo podrejeni psevdo terminal in tako pridobimo njegov opisnik datoteke. To naredimo s funkcijo `open()`, ki kot parameter potrebuje ime podrejenega terminala, ki smo ga pridobili v prejšnjem koraku.

V tem trenutku ima nit tako opisnik glavnega kot podrejenega terminala in je pripravljena na zagon *aplikacije v ozadju*. To naredimo s klicem funkcije `fork(...)`, ki iz trenutnega procesa ustvari dva, starševski in otroški proces. Starševski proces prevzame nadzor nad glavnim psevdo terminalom, medtem ko bomo otroški proces spremenili v prikriti proces in ga zamenjali z *aplikacijo v ozadju*. Uporabimo naslednje zaporedje ukazov:

1. V otroškem procesu najprej še enkrat pokličemo funkcijo `fork()` in spremenimo proces v prikriti proces, kot je opisano v poglavju 2.2.

2. S funkcijo `dup2()` zamenjamo opisnike datotek 0, 1 in 2, ki predstavljajo standardni vhod, standardni izhod in standardni izhod napake, z opisnikom podrejenega psevdo terminala.

3. S funkcijo `chdir()` zamenjamo trenutno delovno pot na željeno vrednost.

4. S funkcijo `execvp()` ali `execv()` zamenjamo trenutni proces z *aplikacijo v ozadju*. Pri tem moramo podati pot do izvršljive datoteke *aplikacije v ozadju* in vse argumente, ki jih ta pri zagonu potrebuje. Če se izvršljiva datoteka ne nahaja na standardnih poteh izvršljivih aplikacij (`/bin`, `/usr/bin` itd.), potem moramo dodati tudi pot do izvršljive datoteke.

Aplikacija v ozadju se uspešno zažene in lahko komunicira z *nadzorno aplikacijo* preko opisnika datoteke podrejenega psevdo terminala. Razred, v katerem je definirana *konzolna nit*, vsebuje tudi dve podatkovni vrsti FIFO (First In First Out), izhodno in ukazno. Med izvajanjem vstavljamo podatke, ki jih preberemo

z glavnega psevdo terminala, v izhodno vrsto. Ukaze, ki jih beremo iz ukazne vrste, pa zapisujemo v glavni psevdo terminal. Razred omogoča drugim nitim varen dostop do obeh vrst. Mogoče je tudi dodajanje ukazov v ukazno vrsto in branje sporočil iz izhodne vrste.

5.3.3 Spletna nit

Glavna funkcija *spletne niti* je generiranje kode HTML za spletni vmesnik ter prenos datotek s strežnika odjemalcu. Nit tako čaka na zahteve, ki jih pošlje odjemalec strežniku. Vsaka zahteva se obdela in vrne se odgovor. V primeru kakršnekoli napake se vrne odgovor z napako. Aplikacija omogoča obdelavo štirih različnih tipov zahtev:

1. **Glavna zahteva** je zahteva za celotni spletni vmesnik, ki prispe v aplikacijo takrat, ko prvič naložimo spletni vmesnik ali ko v brskalniku osvežimo spletni vmesnik.

2. **Zahteva za datoteko** je zahteva za specifično datoteko (npr. predloge CSS, slike, datoteke JavaScript itd.).

3. **Asinhrona zahteva** je zahteva za prenos zadnjih izhodnih podatkov *aplikacije v ozadju* v formatu JSON. Zahteva vsebuje čas zadnjega prejetega sporočila, ki se uporabi pri generiranju odgovora tako, da se v odgovor vstavijo samo nova sporočila.

4. **Ukazna zahteva** vsebuje ukaz, ki ga uporabnik želi poslati *aplikaciji v ozadju*. Ukaz se izloči iz zahteve in doda v ukazno vrsto v *konzolni niti*.

5.3.4 Program za asinhrono komunikacijo

Del aplikacije je tudi preprost program napisan v jeziku JavaScript, ki se izvaja na strani odjemalca in skrbi za osveževanje uporabniškega vmesnika. Uporablja koncept AJAX, ki smo ga opisali v poglavju 4.3. Namesto tehnologije XML smo za serializacijo podatkov uporabili format JSON. Program smo napisali z uporabo knjižnice jQuery (http://jquery.com/), saj nam ta poenostavi programiranje na odjemalcu. Za implementacijo asinhrone komunikacije smo uporabili funkcijo

`jQuery.getJSON()`. Vhodni parameter funkcije je naslov URL, rezultat pa je vsebina odgovora na zahtevo v formatu JSON. Program se začne izvajati v ozadju takoj, ko se odjemalcu naloži spletna stran in preverja, ali so prispeli novi podatki s konzole. Podatke prejme, pretvori v kodo HTML in doda na konec trenutnega seznama sporočil.

Poleg osveževanja prejetih novih sporočil vsebuje program tudi del namenjen pošiljanju ukazov aplikaciji. Spletni vmesnik vsebuje obrazec sestavljen iz polja za vnos in gumba za pošiljanje podatkov obrazca. Knjižnica jQuery nam omogoča, da preprečimo privzeti odziv. Namesto, da bi se ob pritisku na gumb za pošiljanje podatkov ponovno naložila celotna stran, se pokliče funkcija po meri, ki smo jo sami registrirali. Funkcija pretvori ukaz v format JSON, ga pošlje aplikaciji in dobi odgovor o tem ali je bil ukaz uspešno dodan ali ne.

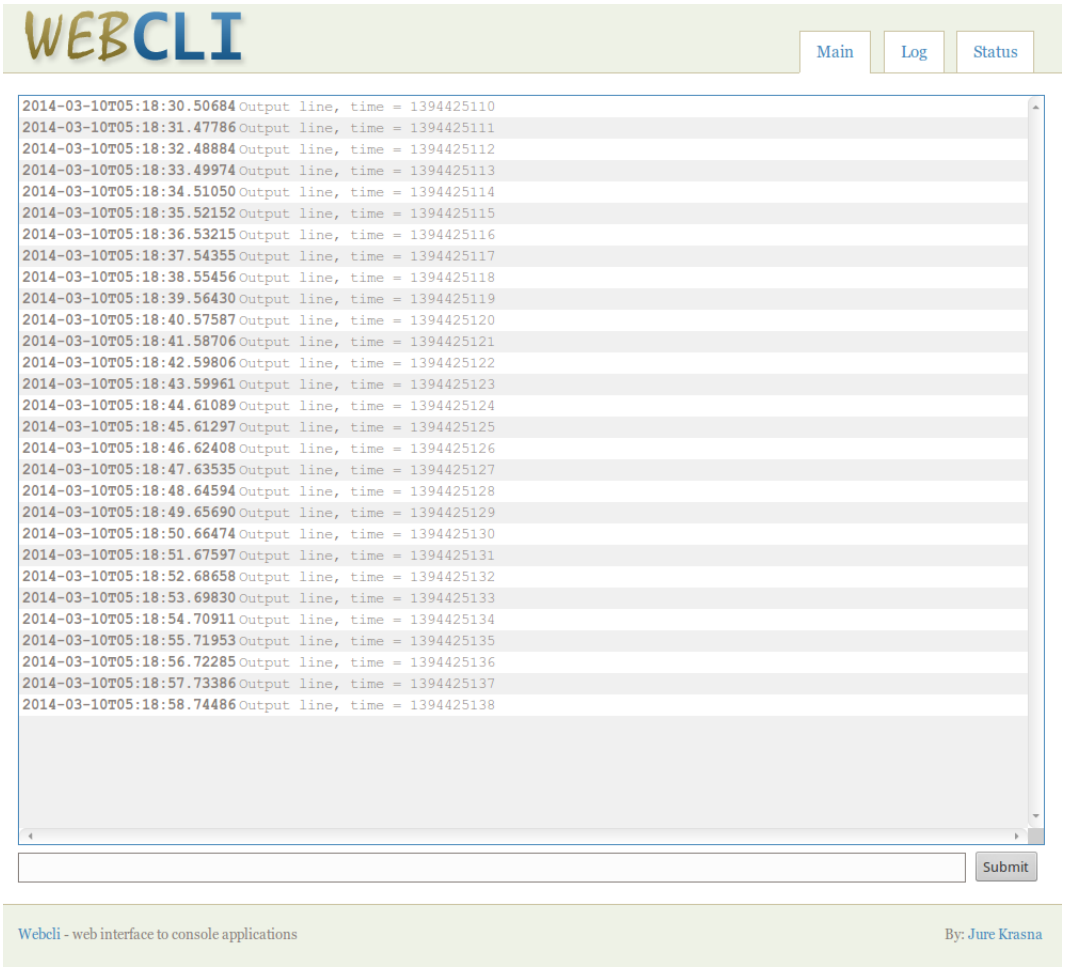
5.4 Izvorna koda in razvoj

Med razvojem smo za hranjenje in primerjanje različic izvorne kode aplikacije uporabljali orodje *Git* (http://git-scm.com/). Izvorno kodo smo objavili na spletnem portalu *GitHub* (https://github.com/). Najdemo jo lahko na tem naslovu URL: https://github.com/jkrasna/webcli.

5.5 Preizkus delovanja

Nadzorno aplikacijo smo preizkusili s spletnim strežnikom *nginx*. Kot *aplikacijo v ozadju* smo uporabili aplikacijo EPICS, ukazno lupino Bash in preprost program, ki smo ga sami implementirali in bere podatke z vhoda in izhoda ter izpisuje naključne vrstice teksta. V vseh primerih se je aplikacija odrezala solidno. Probleme smo naleteli le pri znakih, ki označujejo novo vrstico, saj različne aplikacije uporabljajo različne kombinacije znakov. Ker ima aplikacija trenutno samo eno nit za generiranje spletnega vmesnika, je včasih lahko počasna, saj ni zmožna vzporednega obdelovanja zahtev.

Na sliki 5.2 je prikazan uporabniški vmesnik spletne aplikacije. Na vrhnjem robu lahko najdemo logotip aplikacije in meni za izbiro pogleda. V sredini je spletni terminal z izpisom konzole *aplikacije v ozadju*. Na dnu je obrazec, ki nam



Slika 5.2: Uporabniški vmesnik *nadzorne aplikacije*.

omogoča vnos in pošiljanje ukazov *aplikaciji v ozadju*.

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo poskušali najti način, s katerim bi uporabniku omogočili dostop do konzole aplikacije preko spletnega vmesnika. Raziskali in uporabili smo znanje s področja razvoja večplastnih spletnih aplikacij. Trudili smo se, da bi bila aplikacija preprosta in hitra, pri čemer sta nam najbolj pomagala programski jezik C++ in protokol FastCGI. Ključen je bil predvsem slednji, saj smo lahko aplikacijo povezali v obstoječe spletne strežnike in se izognili direktni integraciji izvorne kode spletnega strežnika v aplikacijo. Aplikacijo smo poskušali čim bolj prilagoditi okolju EPICS, vendar ne na račun njene splošne uporabe, zato je še vedno primerna za vse aplikacije z ukazno vrstičnim vmesnikom. Kljub temu je šele v začetnih fazah razvoja in potrebuje določene izboljšave, s katerimi bi dosegla raven kvalitete, ki je potrebna za produkcijsko okolje.

Napake se lahko pojavijo predvsem, če bi več odjemalcev naenkrat dostopalo do vmesnika aplikacije, saj aplikacija zaenkrat ni zasnovana za večuporabniško izkušnjo. Izboljšave so možne še na področju dostopa do podatkov. Zaenkrat aplikacija v vsakem trenutku hrani največ 1024 sporočil *aplikacije v ozadju*, ki se vsa prenesejo, ko prvič odpremo vmesnik. Če bi želeli število shranjenih sporočil povečati, bi lahko zaradi velike količine podatkov obremenili omrežje. Rešitev tega problema bi bila uvedba asinhronega prenosa sporočil, saj bi se ob prvem dostopu do vmesnika preneslo manjše število sporočil. Uporabnik bi lahko kljub temu brskal po zgodovini sporočil, ki bi se mu asinhrono prenašala na zahtevo.

Aplikacijo smo namenoma zasnovali tako, da so povsod možne razširitve, kar nam bo pomagalo pri načrtovanih nadgradnjah. Med možnimi izboljšavami lahko

omenimo še možnost ponovnega zagona aplikacije v ozadju, možnost spreminjanja nastavitev zagona aplikacije v ozadju, zajem posebnih znakov (npr. Ctrl+C, Ctrl+D) itd. Razvita aplikacije deluje solidno in ima uporabno vrednost tudi v praksi. Razvoj bomo aktivno nadaljevali, saj je cilj nadgraditi aplikacijo do take mere, da bi jo lahko izvajali v produkcijskem okolju.

Literatura

[1] (2014) *Attribution-NonCommercial-ShareAlike 4.0 International.* Dosegljivo na: http://creativecommons.org/licenses/by-nc-sa/4.0/legalcode

[2] (2014) *GNU general public license.* Dosegljivo na: https://github.com/jkrasna/webcli/blob/master/LICENSE

[3] (2014) *Command-line interface.* Dostopno na: http://en.wikipedia.org/wiki/Command-line_interface

[4] R. Love, "Process Management" in Linux System Programming, O'Riley, 2007, str. 159–161.

[5] (2014) *Pseudo terminal.* Dostopno na: http://en.wikipedia.org/wiki/Pseudo_terminal

[6] (2014) *Experimental Physics and Industrial Control System.* Dostopno na: http://www.aps.anl.gov/epics/about.php

[7] L. R. Dalesio, M. R. Kraimer, A. J. Kozubal, "EPICS Architecture", Int. Conf. on Accelerator and Large Experimental Physics Control Systems, Tsukuba, Japonska, 1991, str. 278–281.

[8] L. R. Dalesio, J. O. Hill, M. Kraimer (2014) *The experimental physics and industrial control system architecture: past, present, and future.* Dostopno na: http://www.aps.anl.gov/epics/EpicsDocumentation/EpicsGeneral/epicsX5Farch-1.html

[9] M. R. Kraimer idr. (2014) *EPICS Application Developer's Guide.* Dostopno na: http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide.pdf

[10] (2014) *Web application.* Dostopno na: http://en.wikipedia.org/wiki/Web_application

[11] (2014) *Common Gateway Interface*. Dostopno na: http://en.wikipedia.org/wiki/Common_Gateway_Interface

[12] (2014) *FastCGI*. Dostopno na: http://en.wikipedia.org/wiki/FastCGI

[13] (2014) *FastCGI: A High-Performance Web Server Interface*. Dostopno na: http://www.fastcgi.com/drupal/node/6?q=node/15

[14] M. R. Brown (2014) *FastCGI Specification*. Dostopno na: http://www.fastcgi.com/drupal/node/6?q=node/22

[15] (2014) *Web server*. Dostopno na: http://en.wikipedia.org/wiki/Web_server

[16] (2014) *Apache HTTP server project – mod_fcgi*. Dostopno na: https://httpd.apache.org/mod_fcgid/

[17] (2014) *Module ngx_http_fastcgi_module*. Dostopno na: http://nginx.org/en/docs/http/ngx_http_fastcgi_module.html

[18] (2014) *Ajax (programming)*. Dostopno na: http://en.wikipedia.org/wiki/Ajax_(programming)

[19] (2014) *JSON*. Dosegljivo na: http://en.wikipedia.org/wiki/JSON

[20] (2014) *Introducing JSON*. Dosegljivo na: http://www.json.org/json-en.html

[21] (2014) *Web-based SSH — External links*. Dosegljivo na: http://en.wikipedia.org/wiki/Web-based_SSH#External_links

[22] (2014) *procServ Process Server*. Dosegljivo na: http://sourceforge.net/projects/procserv/