SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI ZADATAK br. 1548

RAZVOJ BIBLIOTEKE ZA PRISTUP SKENERIMA PREKO SUČELJA SANE

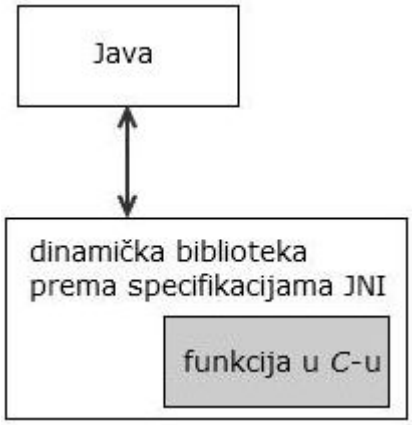
Tomislav Lukinić

Zagreb, srpanj 2010.

Sadržaj

Uvod

Cilj zadatka je razviti dinamičku biblioteku za operativni sustav Linux koja obavlja komunikaciju sa skenerima preko sučelja *Scanner Access Now Easy* (skraćeno SANE) i omogućiti korištenje njene funkcionalnosti iz aplikacije napisane u programskom jeziku *Java*. U sklopu okruženja SANE već postoji skup razvijenih upravljačkih programa za pojedine uređaje te ugrađeno sučelje preko kojeg je moguće do te iste funkcionalnosti doći iz programa u *C*-u. Povezivanje koda pisanog u *C*-u s *Javom* omogućava *Javino nativno sučelje* (engl. *Java Native Interface,* skraćeno JNI) koje služi kao svojevrstan posrednik (engl. *proxy*) preko kojeg se prenose argumenti s jedne strane na drugu i obrnuto, ovisno o ideji programske realizacije (Sl. 1).



Sl. 1: Povezivanje *Jave* i kôda u *C*-u

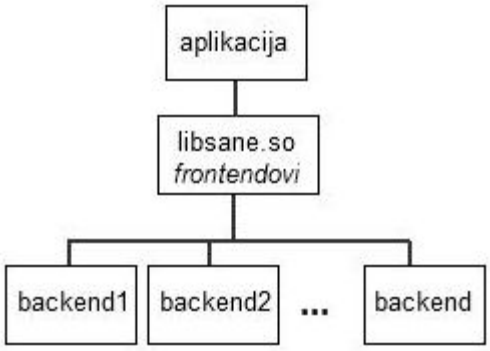
Iz *Jave* će se, dakle, pozivati *C* funkcija iz dinamičke biblioteke kreirane alatima JNI-a. Zadatci koje ta funkcija obavlja su sljedeći:

- učitavanje podataka o slici iz skenera,
- spremanje slike u *međuspremnik* (engl. *buffer*),
- zapisivanje podataka iz međuspremnika u zadanu datoteku i

- vraćanje obavijesti o tome je li operacija uspjela natrag u glavni *Java* program

1. SANE

Scanner Access Now Easy (skraćeno SANE) je sučelje za komunikaciju s uređajima za skeniranje u Linuxu čija je glavna svrha omogućiti da završni dio određene aplikacije koji prikuplja podatke od korisnika (engl. *frontend*) i predaje ih upravljačkom programu (engl. *backend*) ima univerzalni pristup različitim modelima skenera, kamera ili sličnih uređaja. SANE nudi podršku za velik broj različitih uređaja u obliku upravljačkih programa koji su prilagođeni njihovom specifičnom sklopovlju (Sl. 1.1), a pristup tim upravljačkim programima od strane bilo kojeg *frontenda* je standardiziran *sučeljem za razvijanje aplikacija* (engl. *Application Programming Interface*, skraćeno API) o kojem će biti riječi u prvom potpoglavlju ovog poglavlja. Takav standard dopušta da je u svrhu razvoja programske podrške za neki novi uređaj dovoljno napisati samo jedan upravljački program, što je mnogo bolje nego da se za svaki novi uređaj mora razviti po jedan upravljački program i jedan *frontend.*



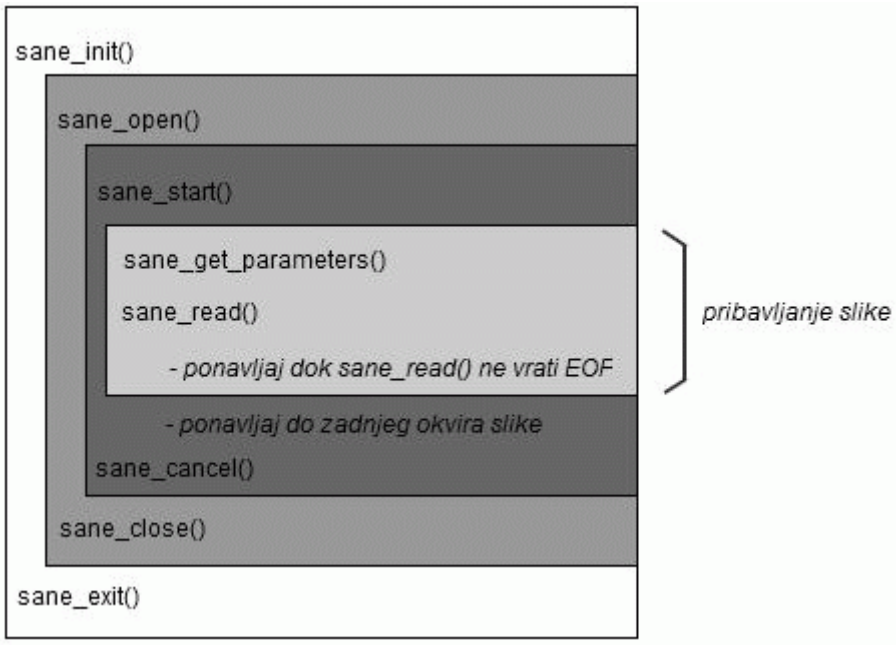
Sl. 1.1: Struktura sučelja SANE

Uzmimo za primjer da moramo razviti tri različite aplikacije koje trebaju moći funkcionirati na četiri različita uređaja. Koristeći okruženje SANE dovoljno je razviti samo 7 programa (jedan *frontend* po aplikaciji i jedan upravljački program po uređaju). U suprotnom, morali bismo za svaku od 3 aplikacije napisati jedan *frontend* te za svaki *frontend* napisati i po 4 upravljačka programa kako bi odgovarali hardveru uređaja i, simultano, strukturi aplikacije, što daje zbroj od 12 programa. Naravno, ušteda na kodu postaje još i veća kako se dodaju novi uređaji i/ili aplikacije[1].

Osim navedenoga, značajna prednost standarda SANE je i njegova pažljivo stvorena infrastruktura koja dozvoljava portabilnost na druge operativne sustave poput *Maca* i *Windowsa.*

1.1. Programsko sučelje biblioteke SANE

API SANE-a sastoji se od biblioteke ugrađenih funkcija koje se na nižoj razini brinu za komunikaciju između aplikacije i skupa razvijenih *backenda*. Pomoću njih obavljaju se jednostavne operacije poput uspostavljanja veze s uređajom i njegove inicijalizacije, identifikacije uređaja kako bi se odredilo koji će se *backend* koristiti u toj komunikaciji (ako *backend* za taj uređaj uopće postoji) te pokretanje skeniranja i dobavljanje podataka o slici. Prototipovi tih funkcija su unaprijed određeni, kao i redoslijed njihovog pozivanja (Sl. 1.2) kako bi se komunikacija sigurno i uspješno obavila.



Sl. 1.2: Osnovni obrazac pozivanja ugrađenih funkcija

U nastavku su opisane funkcije koje su potrebne za uspješno obavljanje skeniranja u nekom osnovnom obrascu. Napomena: sve funkcije kao povratnu vrijednost vraćaju varijable okruženja koje opisuju uspješnost operacije i/ili trenutno stanje skenera.

```
SANE_Status sane_init (SANE_Int * version_code,
                       SANE_Authorization_Callback authorize);
```

Ovom funkcijom se obavlja prepoznavanje uređaja i odgovarajućeg *backenda* te se sprema šifra verzije tog *backenda* u obliku varijable `version_code`. Druga varijabla je zapravo poziv funkcije za autorizaciju koju neki *backendovi* koriste kako bi dobili pristup dodatnim resursima operativnog sustava. Realizacija te funkcije je opcionalna i u slučaju da u aplikaciji nije predviđena mogućnost autorizacije, umjesto drugog argumenta moguće je predati vrijednost `NULL`. `sane_init()` se mora pozvati prije bilo koje druge ugrađene funkcije SANE-a. Neposredno nakon obavljanja tog poziva potrebno je dobaviti ime uređaja s kojim će se u sljedećem pozivu uspostaviti komunikacija. Ovo je najlakše uraditi preko varijable okruženja `SANE_DEFAULT_DEVICE` koja sadržava naziv prvog dostupnog uređaja. No, ukoliko je više uređaja spojeno na računalo, moguće je učitati nazive od svakog od njih u niz znakovnih varijabli pomoću funkcije `sane_get_devices()` te zahtijevati od korisnika da odabere uređaj koji želi koristiti. Pretpostavimo, za potrebe kratkoće, da je samo jedan uređaj spojen na računalo ili da korisniku nije bitno koji će se uređaj koristiti (u tom slučaju, u varijabli okruženja bit će naziv prvog uređaja na koji sustav naiđe). Sljedeći korak je uspostavljanje komunikacije sa uređajem, što se postiže funkcijom čiji je prototip dan u nastavku:

```
SANE_Status sane_open (SANE_String_Const name,
                       SANE_Handle * h);
```

Preko varijable `name` predaje se prethodno dobavljeno ime uređaja s kojim se uspostavlja komunikacija, a povratna vrijednost `h` predstavlja identifikacijski kôd uređaja koji služi za internu uporabu sučelja SANE i njegova vrijednost se nužno prenosi prilikom poziva svih operacija koje slijede.

```
SANE_Status sane_start (SANE_Handle h);
```

Ovaj jednostavni poziv inicijalizira dohvaćanje slike s uređaja identificiranog vrijednošću `h`, odnosno *priprema teren* za sljedeća dva poziva koji će zapravo obaviti posao skeniranja slike.

```
SANE_Status sane_get_parameters (SANE_Handle h,
                                 SANE_Parameters * p);
```

Prvi poziv, `sane_get_parameters()`, dohvaća sve trenutno dostupne podatke o slici, poput dimenzija slike, dubine uzorkovanja (engl. *sample depth*), formata sustava boja u kojem se slika prenosi i informaciju o tome koji se okvir slike trenutno učitava. U slučaju višeokvirnog oblika sustava boja, gdje se crveni, zeleni i plavi dio spektra prenose svaki u posebnom okviru, zahtijeva se višestruki poziv funkcije `sane_start()`. Većina portabilnih skenerskih uređaja nemaju podatak o veličini slike prije početka skeniranja, zbog čega se javlja potreba za dinamičkim alociranjem međuspremnika (engl. *buffer*) u koji će se naknadno spremiti pribavljeni podaci (dinamičko alociranje međuspremnika je isto tako nužna predoperacija za skeniranje slike višeokvirnog oblika sustava boja). Ovime su svi potrebni podaci dobavljeni i može se krenuti u dohvaćanje same slike, sljedećim pozivom:

```
SANE_Status sane_read (SANE_Handle h, SANE_Byte * buf,
                       SANE_Int maxlen, SANE_Int * len);
```

Funkciji se, kao što se već moglo očekivati, predaje oznaka uređaja, `h`, dok se podaci o trenutno skeniranom dijelu slike vraćaju u međuspremnik `buf`, čija se maksimalna veličina predaje funkciji parametrom `maxlen`. Taj parametar je ili određen nekom pretpostavljenom vrijednošću (najčešće 32 * 1024 *byteova*) ili trenutnom veličinom oslobođenog prostora prilikom dinamičke alokacije međuspremnik. Parametrom `len` se predaje informacija o količini (u *byteovima*) trenutno učitanih podataka sa uređaja.

Nakon višestrukog pozivanja dvaju zadnje navedenih operacija, uređaj će u jednom trenutku javiti sučelju preko vrijednosti varijable okruženja da je došao do *kraja datoteke* (engl. *End of file*, skraćeno EOF), nakon čega je, uz oslobađanje alociranih međuspremnika, potrebno samo obaviti završne pozive koji, redom: prekidaju sve operacije koje se nekim slučajem još uvijek izvode, prekidaju komunikaciju s uređajem te prekidaju uporabu *backenda*.

```
void sane_cancel (SANE_Handle h);
void sane_close (SANE_Handle h);
void sane_exit (void);
```

1.2. Djelomična demonstracija razvijenog kôda

Pokažimo sada kako izgleda dio kôda (Kôd 1.1) koji je razvijen u sklopu ovog završnog zadatka i koji, naravno, koristi prethodno opisano sučelje SANE-a. Promotrit ćemo samo, da ne ulazimo previše u detalje, fazu aplikacije u kojoj se u *buffer* prihvaćaju podatci o slici i pretpostavit ćemo da se ne radi o višeokvirnom obliku sustava boja te da uređaj *zna* kakvih će dimenzija slika biti (odnosno da nije potrebno dinamički alocirati međuspremnik).

```
int i, j = 0;
SANE_Status status;
SANE_Handle device;
SANE_Parameters parm;
SANE_Byte *buffer;
SANE_Byte *buffer1;
size_t buffer_size = (32 * 1024);

.           /* pretpostavljamo da smo obavili sve
.              potrebne prethodne operacije
.              propisane API-jem SANE-a        */

buffer = malloc (buffer_size);
```

```
status = sane_get_parameters (device, &parm);
buffer1 = malloc (parm.bytes_per_line * parm.lines);

while (1) {
  status = sane_read (device, buffer, buffer_size, &len);

  if (status == SANE_STATUS_EOF) {
    break;
  } else if (status != SANE_STATUS_GOOD) {
    fprintf (stderr, "sane_read: %s\n",
             sane_strstatus (status));
    return retVal;
  }

memcpy(buffer1 + j, buffer, len);
j += len;
}
```

Kôd 1.1 - Djelomičan kôd za pribavljanje slikovnih elemenata i njihovo spremanje u međuspremnik

Krenimo, dakle, od deklaracija varijabli. Brojači `i` i `j` su deklarirani standardnim tipom *integer*, dok je ostatak varijabli deklariran pomoću tipova određenih u zaglavljima (engl. *header)* okruženja SANE (za koje je također pretpostavljeno da su uključena u izvorni kôd). Varijabla `status` će poprimiti vrijednost znakovnog niza koji opisuje stanje pojedinih operacija, a `device` vrijednost identifikatora uređaja nakon što se obave operacije inicijalizacije uređaja (koje su u ovom primjeru izostavljene). Varijabla `parm` će služiti kao pokazivač na strukturu podataka u koju će pisati funkcija `sane_get_parameters()`. Na kraju, deklariraju se dva međuspremnika tipa polje *byteova* u koje će se učitavati slika. Primijetimo da je, u nastavku, `buffer` alociran spomenutom pretpostavljenom veličinom međuspremnika, za razliku od `buffer1` koji je alociran veličinom određenom iz podataka o dimenzijama slike: `parm.lines` (broj horizontalnih linija slike) i `parm.bytes_per_line` (količini *byteova* od kojih se sastoji jedna horizontalna linije slike). Ti podaci su, kao što se može uočiti, dobiveni iz strukture `parm`. Dakle, `buffer` će se koristiti za spremanje trenutno učitanog dijela slike,

kojim će se `buffer1` postepeno puniti sve dok u njemu ne bude sadržana cijela slika. Dakle, vrtimo *while* petlju u kojoj se pomoću `sane_read()` u `buffer` učitava dio po dio slike, sve dok ta funkcija, ako je u međuvremenu sve prošlo u redu, ne vrati vrijednost `SANE_STATUS_EOF` (što znači da je traka skenera došla do kraja slike). Pomoću naredbe `memcpy()` se prebacuje `len` broj *byteova* iz `buffer` u `buffer1`, a pomoću brojača `j` se nakon tog prebacivanja se pokazivač od `buffer1` pomiče za taj isti `len` kako bi sljedeći skenirani dio slike bio dodan u taj međuspremnik neposredno nakon prethodnog. Stoga, nakon što se odvrti cijela slika, `buffer1` će sadržavati uzastopno sve dijelove slike. Važno je još i primijetiti kako smo, za razliku od prvotnog prikaza redoslijeda ugrađenih funckija (Sl. 1.2), poziv `sane_get_parameters()` stavili izvan petlje. Ovo je također zbog prethodno spomenutog pojednostavljenja primjera u kojem smo pretpostavili da ne skeniramo sliku višeokvirnog oblika sustava boja za koju bismo trebali ponoviti proces za svaki okvir (okvir crvenog, zelenog i plavog spektra RGB sustava) i u svakom ponavljanju ponovno dohvatiti parametre slike kako bismo zapravo znali koji dio spektra u tom okviru učitavamo.

2. Java Native Interface

JNI je okruženje koje omogućuje aplikacijama napisanim u programskom jeziku *Java* da pozivaju nativne funkcije u programskom jeziku *C* i obrnuto. Korištenjem JNI-a možemo koristiti razvijenu funkcionalnost operacija s vanjskim jedinicama koje se pišu u *C*-u iz *virtualnog Java stroja* (engl. *Java Virtual Machine*, skraćeno JVM) [2]. Slučaj u kojem JNI postaje koristan je potreba za povezivanjem *Java* aplikacije s nativnim kodom koji se izvršava u istom procesu. Neki od takvih scenarija su:

- Java API često ne podržava sve mogućnosti hardwarea na kojem se izvršava. U takvim slučajevima ako aplikacija želi koristiti neke specijalne operacije, efikasno je to napraviti u native kodu i povezati ga da s Javom radi u istom procesu.
- Postojanje nativnog koda koji nam se ne isplati ponovo programirati u Javi, a puno je efikasnije da se izvodi u istom procesu[5].

Stoga, u sklopu zadatka koji je potrebno obaviti, JNI je i više nego prikladno rješenje.

2.1. JNI mehanizam

Pretpostavimo da imamo funkciju u *C*-u te je želimo povezati preko JNI-a s jednostavnim programom u *Javi*. Prva stvar koju je potrebno napraviti je učitati dinamičku biblioteku u glavnom *Java* programu. To se postiže sljedećim komadom kôda koji se implementira unutar razreda (engl. *class*) u kojem želimo pozivati nativnu funkciju:

```
static {
     System.loadLibrary("libfunc");
}
```

Ovdje je `libfunc` naziv biblioteke koju želimo učitati. Unutar tog istog razreda, moramo definirati i prototip nativne funkcije. Pretpostavimo za ovaj primjer da je ta funkcija tipa *void* odnosno da ne vraća nikakvu vrijednost, te da joj se ne predaje nikakav argument.

```
public static void func();
```

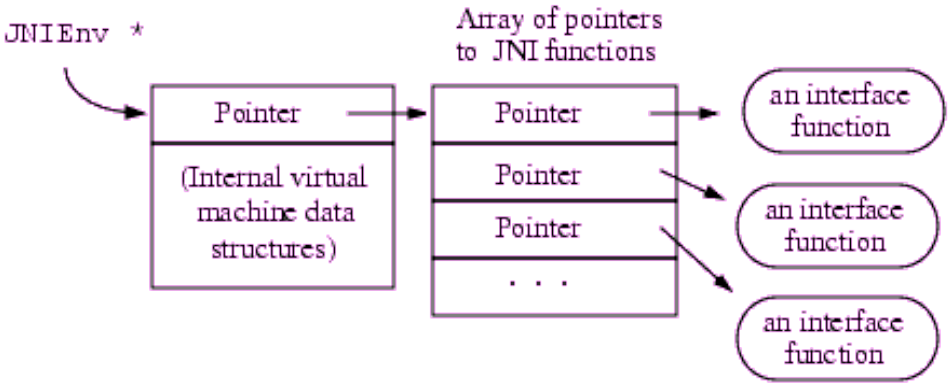
Funkciju unutar *Java* programa jednostavno koristimo kao da se radi o metodi tog razreda. Time smo osposobili *Java* stranu aplikacije i sljedeći korak je modifikacija *C* funkcije tako da veza bude potpuna. No, prije toga potrebno je saznati kako će izgledati novi prototip te funkcije kako bi odgovarao JNI zaglavlju. Kompajliramo *Java* program i generirajmo JNI zaglavlje pomoću sljedeće dvije naredbe:

```
javac JavaProgram.java
javah -jni JavaProgram
```

Nakon što se prvom naredbom uspješno obavi kompajliranje, druga naredba kreira datoteku JavaProgram.h koja predstavlja zaglavlje dinamičke biblioteke JNI-a. Datoteka sadržava prototip koji nam govori kako bi trebala izgledati deklaracija naše nativne funkcije da ona bude povezana u dinamičku biblioteku. U ovom slučaju taj prototip izgleda ovako:

```
JNIEXPORT void JNICALL
 Java_JavaProgram_func (JNIEnv *, jobject);
```

JNIEXPORT i JNICALL su pozivi koji osiguravaju uspješnu kompilaciju programa u okruženjima poput *Win32* koja zahtijevaju posebne ključne riječi za funkcije izvedene iz dinamički povezanih biblioteka [3]. Argument `JNIEnv` (Sl. 2.1) je pokazivač na pokazivač koji pak pokazuje na memorijsku tablicu određenih JNI funkcija. Pomoću njih nativne metode pristupaju strukturama podataka u JVM-u [4]. `jobject` je referenca na objekt preko kojeg je pozvana nativna metoda.



Sl. 2.1: Memorijski prikaz pokazivača JNIEnv [4]

Konačno, možemo modificirati nativnu funkciju. Pretpostavimo da je ovo originalni oblik funkcije:

```
#include <stdio.h>

void func() {
  printf("Ovo je funkcija!\n");
  return;
}
```

Pogledajmo sada modificiranu verziju:

```
#include <stdio.h>
#include <jni.h>
#include "JavaProgram.h"

 JNIEXPORT void JNICALL
 Java_JavaProgram_func(JNIEnv *env, jobject obj) {
   printf("Funkcija!\n");
   return;
 }
```

Jedino što preostaje je kompajlirati program i stvoriti dinamičku biblioteku pod imenom *libfunc.so* kako bi se mogla učitati u iz JVM-a. U ovom jednostavnom primjeru se nije

koristila funkcija koja prima argumente i vraća određenu vrijednost, no to je najbolje pojasniti na primjeru kôda razvijenog u sklopu praktičnog dijela ovog završnog zadatka.

2.2. Modifikacija razvijenog kôda

Pokažimo kako je ostvarena funkcionalnost kôda, čiji je dio pojašnjen u prvom poglavlju ovog rada (Kôd 1.1), pozivom iz *Jave* preko sučelja JNI. Za potrebe te implementacije, potrebno je prvo pokazati kako je uklopljen poziv funkcije u glavnom *Java* programu (Kôd 2.1).

```
import java.io.*;
import java.util.*;

class ScanIt {
 static {
  try {
   System.loadLibrary("scanit");
  }
  catch (UnsatisfiedLinkError e) {
   System.out.println("Could not load native code.");
   System.exit(1);
  }
 }

 public static void main(String [] args) throws IOException {
  String strImg;

  BufferedReader console =
     new BufferedReader(newInputStreamReader(System.in));

  for( ;; ) {
    System.out.print("\nImage name:");
    strImg = console.readLine();
    if(strImg != null && strImg.length() > 0) {
      if (scanit(strImg)) {
      System.out.print("\nScanned image to "
                       + strImg + ".pnm succesfully.\n");
```

```
      } else {
        System.out.print("Scanning failed.");
      }
    break;
   } else {
     System.out.print("\nYou must provide at least one
                      character   for the image name!\n");
   }
  }
  public static native boolean scanit(String strImg);
 }
```

Kôd 2.1 - Pojednostavljena verzija glavnog *Java* programa

Nije na odmet napomenuti da je i ovdje izbačeno nekoliko linija iz originalnog kôda zbog preglednosti. U klasi naziva `ScanIt` učitavamo biblioteku pomoću već prije prikazane naredbe `loadLibrary`, koja se naziva `scanit`. Glavni program zahtijeva od korisnika da upiše ime datoteke u koju će se skenirana slika spremiti. Nakon toga predaje taj znakovni niz nativnoj funkciji `scanit()`. Kao što se može primijetiti u nastavku opisa klase, nativna funkcija vraća vrijednost tipa *boolean*, odnosno 1 ako je operacija bila uspješna i 0 ako nije. Ta se vrijednost ispituje u glavnom *Java* programu i u slučaju pozitivnog rezultata se na ekran ispisuje poruka o uspješno obavljenom skeniranju. Pogledajmo sada kako izgleda isječak kôda modificirane nativne funkcije (Kôd 2.2) nakon što se kompajlira *Java* program i generira JNI zaglavlje.

```
JNIEXPORT jboolean JNICALL Java_ScanIt_scanit
(JNIEnv *env, jobject obj, jstring img) {
  const char *pImg = (*env)->GetStringUTFChars(env, img, 0);
  const char *format = 0;
  jint retVal = JNI_FALSE;
  char path[PATH_MAX];

  format = "%s.pnm";
  sprintf (path, format, pImg);

  fflush(stdout);              /* preusmjeravamo stdout
  fgetpos(stdout, &pos);          u datoteku */
  fd = dup(fileno(stdout));
```

```
  if (1 && NULL == freopen (path, "w", stdout)) {
    fprintf (stderr, "cannot open %s\n", path);
    sane_cancel (device);
    return retVal;
         }

.           /* ovdje se obavlja pribavljanje slike
.              u buffer1 na način kako je demonstrirano
.              u prošlom poglavlju (Kôd 1.1)           */

  fwrite (buffer1,
          1,
          parm.lines * parm.bytes_per_line,
          stdout);

  fflush(stdout);              /* ponovno preusmjeravamo
  dup2(fd, fileno(stdout));       ovaj puta iz datoteke
  close(fd);                      natrag u stdout */
  clearerr(stdout);
  fsetpos(stdout, &pos);

  retVal = JNI_TRUE;

  (*env)->ReleaseStringUTFChars(env, img, pImg);

  return retVal;
}
```

Kôd 2.2 - Isječak modificirane nativne funkcije

Prilikom poziva ovako modificirane funkcije, predaje joj se varijabla znakovnog niza pod imenom `img`. Linija kôda koja možda odmah nakon standardnog JNI poziva upada u oči je sljedeća:

```
const char *pImg = (*env)->GetStringUTFChars(env, img, 0);
```

Njome se obavlja pretvorba varijable znakovnog niza dobavljenog iz glavnog *Java* programa u tip znakovne varijable s kojom *C* može baratati i koja se sprema u `pImg`. Nakon toga zadajemo početnu vrijednost `JNI_FALSE` varijabli `retVal` koju ćemo nakon izvođenja funkcije vratiti natrag u glavni program. Nadalje, naredbom `sprintf()` pretvaramo taj dodajemo na kraj znakovnog niza `pImg` ekstenziju ".pnm" i sve spremamo u varijablu `path`. Ta varijabla se u nastavku kôda predaje funkciji `freopen()` kao datoteka u koju se preusmjerava *standardni izlaz* (engl. *standard output*) programa kako bi se podaci koji čine sliku u konačnici i zapisali na disk. U slučaju neuspješnog preusmjeravanja, glavnom se programu vraća vrijednost `retVal` koja je u ovom trenutku jednaka logičkoj nuli. Izostavljenim dijelom kôda koji je potpuno isti kao i prije nego što smo modificirali ovu funkciju (Kôd 1.1) međuspremnik `buffer1` je popunjen elementima skenirane slike. Taj se međuspremnik sada i ispisuje operacijom `fwrite()` u standardni izlaz preusmjeren u zadanu datoteku. Na kraju, funkcija postavlja povratnu vrijednost u logičku jedinicu i vraća je *Java* programu koji ju je pozvao. Bitno je napomenuti da je prije toga ponovno preusmjeren tok podataka iz datoteke u standardni izlaz, kako bi se potvrda o uspješnom skeniranju (Kôd 2.1) mogla ispisati na ekran. Prilikom kompajliranja ovog *C* programa, u komandnu liniju moramo uključiti opciju *-shared* koja omogućuje stvaranje zajedničkog objekta i navesti ime biblioteke koje je određeno u *Java* programu (Kôd 2.1) pomoću operacije `System.loadLibrary()`, a zadaje se uz prefiks *lib* (oznaka da se radi o biblioteci) i sufiks, odnosno ekstenziju, *.so*. Time smo povezali dinamičku biblioteku s *Javom* i omogućili korištenje njene funkcionalnosti preko jednostavnog pokretanja *Java* programa.

Zaključak

U sklopu završnog zadatka uspješno je razvijena dinamička biblioteka za komunikaciju sa skenerskim uređajima koja je poveziva s aplikacijama više razine napisanim u programskom jeziku *Java*. Biblioteka sadržava funkciju napisanu u programskom jeziku *C* koja pristupa već postojećem sučelju za razvoj aplikacija unutar okruženja SANE. Prednost korištenja SANE-a je što za različite vrste uređaja neće biti potrebno mijenjati aplikaciju, već samo razviti upravljački program specifičan za traženi uređaj. Iz ovoga je odmah jasan i glavni nedostatak, a to je da će uspješnost aplikacije ovisiti o tome postoji li u okruženju SANE upravljački program za uređaj na kojem se treba vršiti skeniranje. Povezivost s aplikacijama u *Javi* omogućena je modificiranjem funkcije u skladi s JNI standardom. To uključuje i obostrani prijenos varijabli, a ključna prednost ovakve *suradnje* je što se operacije s vanjskim jedinicama pisane u *C*-u mogu izravno pozvati iz *Jave*. Time je otvorena i mogućnost eventualne izgradnje grafičkog sučelja prema korisniku u svrhu povećanja intuitivnosti i pristupačnosti cijele aplikacije. Međutim, ne mogu se zanemariti ni nedostaci u takvoj *Java* aplikaciji od kojih je možda najznačajniji smanjena portabilnost, s obzirom da je za svaki operativni sustav potrebno *prevesti* nativni dio kôda kako bi se program u *Javi* mogao za njega koristiti.

Literatura

[1] Http://www.sane-project.org/Intro.html

[2] Http://En.Wikipedia.org/wiki/Java_Native_Interface

[3] Http://Codersger.de/mags/cscene/CS4/CS4-04.html

[4] Http://Java.Sun.com/docs/books/jni/html/ObjTypes.html

[5] BUČAR, D., BULOVIĆ, A., GRŽIČIĆ, S., HUCALJUK, J., KOVAČIĆ, B., PALAŠEK, P., POPOVIĆ, B., SAMBOL, A., Integracija dodatnih mogućnosti u programski sustav Marker, *dokumentacije predmeta Projekt*, FER, Zagreb, (2009), 10.

Sažetak

Razvoj biblioteke za pristup skenerima preko sučelja SANE

Sučelje SANE nudi standardizirani skup funkcija za pristup skenerima u Linuxu kako bi se razvijena aplikacija mogla primijeniti na više različitih uređaja. Ta primjenjivost je ograničena dostupnošću SANE-ove podrške za određene uređaje. Potreba za razvojem dinamičke biblioteke se javlja zbog mogućnosti korištenja operacija s vanjskim jedinicama pisanim u *C*-u iz programskog jezika *Java*. Razvijanje istog programa u cijelosti na JVM-u zahtijevalo bi programske akrobacije s obzirom na platformsku specifičnost okruženja SANE. Također, u *Javi* je moguće izraditi i grafičko sučelje kako bi konačna aplikacija bila pristupačnija korisniku. Standardom JNI se povezuju ta dva svijeta nativnog *C* kôda i programskog jezika *Java*, što sa sobom, doduše, donosi i nemogućnost pokretanja aplikacije na drugim operativnim sustavima bez izmjene nativnog dijela kôda.

Ključne riječi:

SANE, skener, Linux, dinamička biblioteka, C, nativna funkcija, Java, JVM, JNI

Summary

Development of a software library for accessing scanners over the SANE interface

The SANE interface provides a standardized set of functions for accessing scanner devices on Linux to make a program applicable to more different devices. This applicability is bound directly to whether the support for a certain device exists within the SANE enviroment. There is a need for developing a dynamically linked library in JNI in order to be able to use peripheral unit operations written in *C* directly from *Java*. By developing the application solely in JVM, it would require gruesome acrobatic programming to build it, considering how platform-specific the SANE interface is. Also, *Java* offers a possibility of developing a graphical interface to make the application more user-friendly. The JNI standard provides a way to connect these two worlds of *C* and *Java*, which unfortunately also makes it impossible to run the program on another OS without redeveloping its native code.

Keywords:

SANE, scanner, Linux, dynamic library, C, native function, Java, JVM, JNI

Skraćenice

SANE	*Scanner Access Now Easy*	
JNI	*Java Native Interface*	*Javino* nativno sučelje
API	*Application Programming Interface*	sučelje za razvoj aplikacija
JVM	*Java Virtual Machine*	virtualni *Java* stroj
RGB	*Red Green Blue (System)*	

Privitak

Instalacija programske podrške

Za instalaciju programske podrške, na računalu je potrebno imati prethodno instaliran operativni sustav Linux (neovisno o distribuciji). Isto tako, da aplikacija funkcionira, potrebno je instalirati SANE-ov skup alata – automatski pomoću *Update Managera* ili ručno, upisivanjem sljedeće naredbe u terminal pod administratorskim ovlastima (vrijedi za Ubuntu):

```
sudo apt-get install sane
```

Ova naredba može izbaciti poruku da SANE već postoji na tom računalu i da je eventualno moguće napraviti njegov *update*. U tom slučaju pređite na sljedeći korak.

Za pokretanje *Java* programa (i stvaranje dinamičke biblioteke za JNI) potrebno je instalirati po mogućnosti najnoviji razvojni paket *Jave* koji se može skinuti sa stranice *http://java.sun.com/javase/downloads/index.jsp*, ili ručno upisivanjem naredbe u terminal:

```
sudo apt-get install java-6-open-jdk
```

Nadomjestite naredbu za skidanje i instalaciju paketa s interneta s onom koja odgovara vašoj distribuciji Linuxa. Moguće je da je s obzirom na drugačiju distribuciju potrebno skinuti i drugačiji *Java* paket (npr. JRE ili JDK-RPM), no o tome možete pročitati u uputama na gore navedenom linku.

Nakon što ste uspješno instalirali SANE i razvojni *Java* paket, prekopirajte .tar.gz arhiv koji se nalazi na CD-u na svoje računalo u direktorij po želji. Nakon toga, otvorite terminal (ako već niste) i pozicionirajte se u direktorij gdje ste kopirali arhiv.

```
cd /put/do/direktorija
```

Možete provjeriti nalazi li se stvarno arhiv u direktoriju pomoću naredbe `ls`. Sljedeći korak je ektrahiranje arhiva, sljedećom naredbom:

```
sudo tar -xzf scanit.tar.gz
```

Ako ste mijenjali ime arhiva nakon kopiranja, promijenite ga i u gornjoj naredbi. Sada bi se u trenutnom direktoriju trebao pojaviti novi direktorij pod pretpostavljenim imenom *scanit*. Pozicionirajte se u taj direktorij pomoću `cd scanit` i nakon toga pokrenite sljedeću naredbu:

```
sudo gcc -L/usr/lib/sane -lsane -o libscanit.so scanit.c
-shared -fpic -I/put/do/trenutnog/direktorija/include
-I/put/do/java/direktorija/include
-I/put/do/java/direktorija/include/linux
```

Umjesto dijela podcrtanog punom linijom upišite put do direktorija u kojem se trenutno nalazite (provjerite put pomoću naredbe `pwd`), a umjesto dijela podcrtanog crtkanom linijm upišite put do direktorija gdje ste instalirali paket *Java*. Primjer takvog puta: `/usr/lib/jvm/java-6-openjdk`.

Završni korak instalacije je upisivanje sljedeće naredbe:

```
sudo cp libscanit.so /usr/lib && sudo ldconfig
```

Ovo osigurava da program pri pokretanju nađe JNI biblioteku koja mu je potrebna da se obavi komunikacija između *Java* i *C* dijela aplikacije.

Mogući problemi prilikom instalacije:

- Poruka o neuspješnom pronalasku datoteke libsane.so. Rješenje: pogledajte imate li u direktoriju /usr/lib/sane neku verziju datoteke u obliku libsane.so.x. Ako imate napravite simbolički link imena libsane.so koji će pokazivati na tu datoteku (naredba: `sudo ln -s /usr/lib/sane/libsane.so.x /usr/lib/sane/libsane.so`).
- Greške kompajlera koje uključuju .h datoteke. Rješenje: provjerite jeste li još uvijek u direktoriju u koji ste esktrahirali arhiv i odgovara li taj direktorij onom dijelu podcrtanom punom linijom kojeg ste upisali u gore navedenoj naredbi. Isto tako provjerite je li točan put do direktorija gdje se nalazi instalacija *Jave* (podcrtan crtkanom linijom)

- Greške pri povezivanju nativne biblioteke. Rješenje: U zadnje navedenoj naredbi u ovim uputama za instalaciju promijenite direktorij `/usr/lib` u `/usr/local/lib`.

Upute za korištenje programske podrške

Ostanite u istom direktoriju gdje ste izveli prijašnju naredbu te pokrenite program:

```
sudo java ScanIt
```

Poželjno je pokrenuti program s administratorskim ovlastima kako je gore naznačeno, s obzirom da neke distribucije (to se pogotovo odnosi na primjer na Ubunut) imaju problema s permisijama kada se pokreću neki programi koji komuniciraju s vanjskim jedinicama preko USB-a i slično. Ako je naredba prošla, u ovom trenutku trebao bi se pojaviti sljedeći upit u terminalu:

```
Image name:
```

Upišite naziv datoteke u koju želite da se skenira slika (bez ekstenzije i bez posebnih znakova) i nakon toga pritisnite enter. U slučaju da niste unijeli naziv, aplikacija će to od vas ponovno zatražiti. Sada pričekajte da program obavi komunikaciju sa skenerom i pribavi sliku, nakon čega će ispisati poruku da je skeniranje uspješno i naznačiti ime datoteke (s ekstenzijom .pnm) u koju je slika spremljena (unutar istog direktorija). Isto tako će dojaviti poruku o pogrešci ako skeniranje nije bilo uspješno.