SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1742

Upravljanje prozorima u Windows XP okruženju

Hrvoje Dagelić

Zagreb, lipanj 2008.

Sadržaj

1. Uvod 1

2. Prozori i upravljanje prozorima 2

2.1 Vrste i struktura prozora 2
2.2 Stvaranje prozora 4
2.3 Poruke i komunikacija 7
2.4 Upravljanje prozorima 10
2.5 Osnove iscrtavanja sadržaja 17
2.6 Prednosti i nedostaci upravljanja prozorima 22

3. Izvedba panoramske radne površine 23
3.1 Opis značajki programa 23
3.2 Osnovna struktura i problemi implementacije 24
3.3 Modul za dohvaćanje stanja radne površine i detekciju promjena 25
3.4 Modul za uzimanje slika prozora 35
3.5 Modul za detekciju povlačenja prozora 43
3.6 Modul za upravljanje radnom površinom 46
3.7 Modul za prikaz 49

4. Zaključak 52

5. Literatura 53

1. Uvod

Moderna grafička okruženja današnjih operacijskih sustava temelje se na prikazu sadržaja aplikacija u prozorima. Prozor je najčešće pravokutno područje na ekranu koje služi kao sučelje između aplikacije i korisnika. Operacijski sustavi omogućuju prikaz više prozora odjednom koji se mogu postavljati na razna mjesta na radnoj površini, svesti na ikonični prikaz ili prikazati preko cijelog ekrana. Na ovaj se način korisniku omogućava prirodno prikazivanje i rad s više aplikacija odjednom. Iako grafička okruženja postoje već dugo i dobro su osmišljena ipak još uvijek ima mjesta za poboljšanja. Jedan od problema sa gotovo svim grafičkim okruženjima je nepreglednost i teškoće u snalaženju kada se odjednom otvori više prozora. Tako se, na primjer, u operacijskom sustavu Windows svi prozori prikazuju u traci zadataka kao gumb sa ikonom i dijelom naslova na prilično skučenom prostoru. Kada se otvori već i nekoliko prozora može se dogoditi da se teško raspoznaju na traci zbog identične ikone ili naslova, a kada se otvori velika količina prozora snalaženje postaje vrlo komplicirano. I sama radna površina ograničena je na pravokutno područje pa je čak ponekad nemoguće postaviti i dva prozora jedan uz drugi i, na primjer, usporediti njihov sadržaj. Već za vrijeme pojave prvih grafičkih okruženja pojavila su se rješenja kako bi se ovi problemi umanjili. Tako se još 1985. na računalima Amiga pojavljuje koncept virtualnih radnih površina. S virtualnim radnim površinama korisnik može prebacivati ili pomicati pogled na ekranu na zasebne radne površine i na njima imati otvorene različite skupine aplikacija. Ovaj koncept postao je kasnije i standard na grafičkim okruženjima porodice operacijskih sustava Unix. U posljednje vrijeme pojavljuje se i sustav Exposé na računalima Apple Macintosh koji na naredbu miša omogućuje privremeni prikaz otvorenih prozora jedan uz drugi i izbor jednoga od njih jednostavnim pritiskom tipke miša.

U ovome radu prikazuje se još jedan sustav kojim se pokušava umanjiti problem nepreglednosti na radnoj površini. Ideja se temelji na konceptu virtualnih radnih površina ali s razlikom da se ne koristi diskretna promjena radne površine već se radna površina prikazuje na prirodniji način – linearno. Program je predviđen za operacijski sustav Windows XP i u prvom se dijelu rada opisuje način na koji ovaj sustav upravlja prozorima.

2. Prozori i upravljanje prozorima

Za razliku od tradicionalnih aplikacija koje kao sučelje koriste naredbenu liniju, a najčešće se izvode slijedno, komunikacija korisnika sa aplikacijama koje koriste prozore interaktivna je i odvija se pomoću gumba, izbornika, polja za unos, dijaloških okvira itd. Zbog ovoga načina komunikacije i same aplikacije se uvelike razlikuju od tradicionalnih jer moraju reagirati na bilo koji od mogućih korisničkih unosa u bilo kojem trenutku te se koristi paradigma programiranja vođenog događajima. Ovo je osnovni koncept za programiranje aplikacija s prozorima a također se i samim prozorima upravlja na ovaj način koristeći API funkcije operacijskog sustava. U ovome dijelu rada prvo se opisuje osnovna struktura i vrste prozora u operacijskom sustavu Windows XP, zatim način stvaranja, poruke i model vođen događajima i na kraju API funkcije za upravljanje prozorima sa naglaskom na one koje se koriste u praktičnom dijelu rada.

2.1 Vrste i struktura prozora

U ovome poglavlju opisane su glavne vrste prozora u operacijskom sustavu Windows XP te struktura tipičnoga prozora.

2.1.1 Vrste prozora

Sučelje operacijskog sustava Windows sastoji se od prozora, a oni se mogu podijeliti na tri osnovne vrste:

1. vršni prozori (*eng. top-level windows*),
2. prozori djeca (*eng. child windows*) i
3. prozori bez prikaza (*eng. message only windows*).

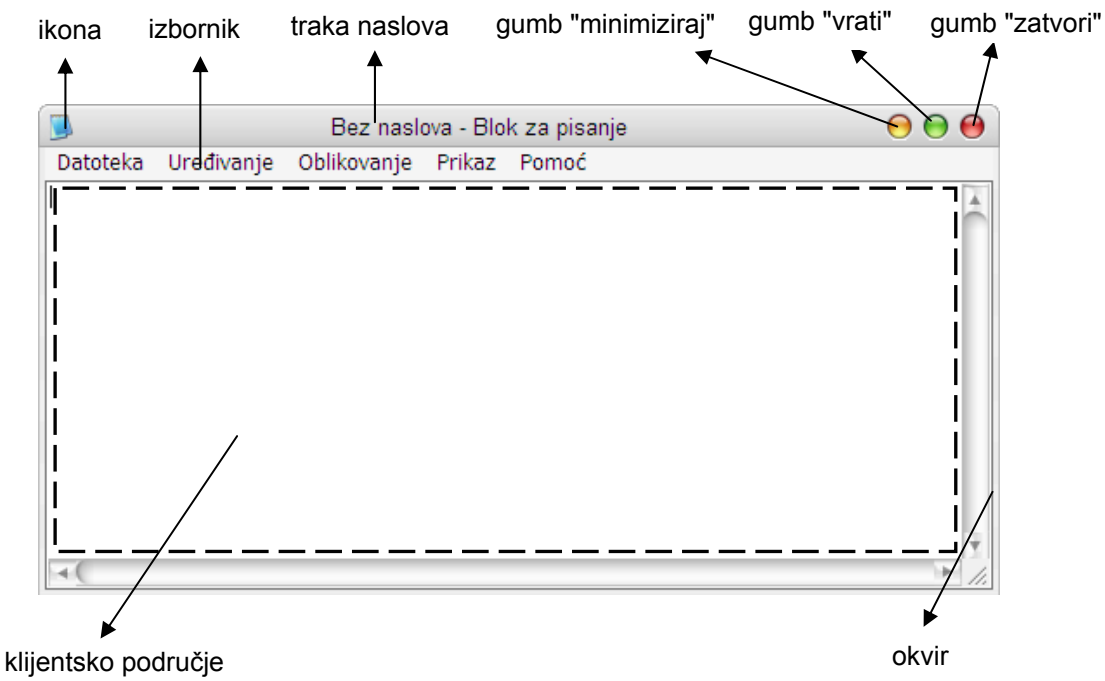
Vršni prozori su prozori koji služe kao glavno sučelje aplikacije a mogu se premještati po radnoj površini dok su prozori-djeca unutrašnji prozori aplikacije i ne mogu se prikazati izvan vršnog prozora aplikacije. U ovu vrstu prozora spadaju kontrole kao što su gumbi, meniji ili polja za unos. Posebna vrsta prozora-djece su i prozori sučelja višestrukih dokumenata (MDI *child windows*) koji imaju sve karakteristike vršnih prozora, jedino im je pomak ograničen na klijentsko područje vršnoga prozora. Prozori bez prikaza poseban su slučaj prozora koji mogu komunicirati sa drugim prozorima, ali nemaju reprezentaciju na ekranu. Osim ove podjele, prozori se mogu podijeliti i po drugim karakteristikama kao što su način iscrtavanja ili oblik prozora. Po obliku prozori se dijele na kvadratne i nekvadratne a po načinu iscrtavanja na:

1. uslojene prozore,
2. neuslojene prozore.

Uslojeni prozori pojavili su se prvi puta u operacijskom sustavu Windows 2000, a razlikuju se po tome što omogućuju drugačiji postupak iscrtavanja sadržaja na ekran, a omogućuju i efekt prozirnosti. Uslojeni prozori koriste se i u praktičnome dijelu ovoga rada, a njihovo postojanje omogućilo je dohvaćanje slike prozora u memoriju na čemu se i temelji praktični dio. Zbog toga se ova vrsta prozora detaljnije opisuje u poglavlju 2.4.

2.1.2 Struktura tipičnog prozora

Kao što je već rečeno, sve kontrole u sučelju operacijskog sustava od prozora aplikacija do gumbi i polja za unos su prozori te se stvaraju koristeći istu API funkciju *CreateWindow*. Zato su prozori vrlo raznoliki i jedino što im je zajedničko je da imaju područje za prikaz sadržaja i mogu obrađivati poruke. Tipičan aplikacijski prozor prikazan je na slici 2.1.



Slika 2.1 *Osnovna struktura prozora*

Na prikazu prozora razlikuju se dva osnovna dijela: klijentsko područje i neklijentsko područje. Klijentsko područje je dio prozora kojim upravlja operacijski sustav, točnije, dio operacijskog sustava koji se naziva upravitelj prozora, a izvodi se u jezgrenom načinu rada. Neklijentsko područje je dio kojim upravlja aplikacija te na njemu prikazuje proizvoljni sadržaj. Na gornjem dijelu slike označeni su glavni dijelovi i kontrole, a budući da je njihova funkcionalnost poznata nema ih potrebe opisivati.

2.2 Stvaranje prozora

U operacijskom sustavu Windows, svi prozori stvaraju se pozivom API funkcije CreateWindow ili CreateWindowEx. Ova posljednja proširena je inačica prve, a ona je zadržana zbog kompatibilnosti sa prethodnim inačicama operacijskog sustava. Prije nego što se ova funkcija može pozvati, prvo je potrebno odrediti parametre za prikaz novoga prozora, a oni se mogu podijeliti na dvije skupine:

1. parametri koji se prenose izravno funkciji za stvaranje prozora,
2. parametri koji se definiraju razredom prozora.

Razred je skupina atributa koji služe kao predložak za stvaranje prozora. Razredom se definiraju parametri koji su više općeniti i najčešće zajednički za više prozora, a parametri koji se prenose izravno su oni koji su specifični za svaki stvoreni prozor kao što su, na primjer, naslov i dimenzije. Za prozore kao što su gumbi, izbornici itd. postoje već unaprijed definirani razredi koji se samo prenose funkciji *CreateWindowEx,* a također je moguće definirati vlastite razrede te ih nakon registracije u sustavu pozivom funkcije *RegisterClassEx* koristiti za stvaranje vlastitih prozora. Razred je definiran zapisom tipa *WNDCLASSEX* koji je objašnjen u nastavku. Deklaracija strukture opisana je sa:

```
typedef struct {
    UINT cbSize;
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HINSTANCE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
    HICON hIconSm;
} WNDCLASSEX, *PWNDCLASSEX;
```

- Polje *cbsize* – ovo je pomoćno polje i u njega se zapisuje veličina ove strukture u bajtovima.

- Polje *style* – služi za definiciju kombinacije konstanti stilova razreda koristeći operator *ili*. Ovi stilovi određuju, na primjer, da li prozor ima pozadinsku sjenu, da li se sadržaj obnavlja automatski poslije promjene dimenzija, da li se spremaju bitovi područja koje prekrivaju drugi prozori itd. Ostali stilovi prenose se izravno kao parametar funkcije za stvaranje prozora.

- Polje *lpfnWndProc* – služi za spremanje pokazivača na proceduru prozora koja obrađuje poruke poslane prozoru. Procedura prozora je detaljnije opisana kasnije.

- Polja *cbClsExtra i cbWndExtra* – definiraju koliko se alocira dodatne memorije u zapisu razreda odnosno u instanci prozora. Polja se koriste samo u posebnim slučajevima, a pretpostavljena vrijednost je 0.

- Polje *hinstance* – u ovo se polje postavlja pokazivač modula programa koji sadrži kod procedure prozora. Kako jedan program može imati više učitanih modula (sam program, dll-ovi) ovo je polje bitno kako bi sustav mogao pronaći kod procedure prozora.

- Polja *hIcon, hIconSm hCursor i hbrBackground* – u ova se polja upisuju vrijednosti upravljačkih varijabli (eng. *handle*) ikone, kursora i kista (eng. *brush*) pozadine. Kist je struktura koja definira način crtanja i ispisa sadržaja, boje itd. Umjesto pokazivača na kist može se koristiti i konstanta koja definira samo boju pozadine. Ako se ova polja ne definiraju, koriste se pretpostavljene vrijednosti.

- Polja *lpszMenuName* i *lpszClassName* – u ova se polja upisuje pokazivač na niz imena pretpostavljenog izbornika i imena razreda. Ako se polje *lpszMenuName* ne postavi, prozor nema pretpostavljeni izbornik.

Nakon što se razred definira i registrira pozivom funkcije *RegisterClassEx,* prozor se stvara pozivom funkcije *CreateWindowEx,* kojoj se prenose ostali parametri važni za prikaz prozora. Deklaracija funkcije dana je sa:

```
HWND CreateWindowEx(
    DWORD dwExStyle,
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    HMENU hMenu,
    HINSTANCE hInstance,
    LPVOID lpParam
);
```

Povratna vrijednost funkcije je upravljačka varijabla prozora koja se koristi kao jedinstveni identifikator i omogućuje korištenje drugih API funkcija za rad i upravljanje s prozorima. Opis parametara dan je u nastavku.

- Parametar *dwExStile* – određuje prošireni stil prozora koji se sastavlja *ili* kombinacijom od 26 raspoloživih konstanti. Ovdje se među ostalim određuje da li je prozor uslojen (WS_EX_LAYERED), da li propušta akcije miša (WS_EX_ TRANSPARENT), da li je uvijek iznad normalnih prozora (WS_EX_TOPMOST), da li se može aktivirati (WS_EX_NOACTIVATE) itd. Koristeći konstantu WS_EX_TOOLWINDOW postiže se da se prozor ne prikazuje u traci zadataka (eng. *taskbar*). U praktičnom dijelu rada koriste se sve od navedenih konstanti, na primjer, prozirni prozor koristi se za efekt postepenog prijelaza između dva stanja

radne površine kod promjene pogleda, a taj je prozor ujedno iznad ostalih prozora i ne može se aktivirati.

- Parametar *lpClassName* – u ovaj se parametar prenosi pokazivač na niz koji sadrži ime registriranog imena razreda prozora, a u parametru *lpWindowName* pokazivač na niz sa naslovom prozora. Ako prozor ima traku naslova, ovaj će se niz na njoj prikazati a također se prikazuje i u ikoničnom prikazu na traci zadataka.

- Parametar *dwStyle* – ovaj parametar služi za definiciju osnovnog stila prozora. Također se koristi *ili* kombinacija konstanti od kojih se najveći broj odnosi na strukturu prozora. Tako, na primjer, konstanta WS_CAPTION određuje da li prozor ima traku naslova, konstante WS_MAXIMIZEBOX i WS_MINIMIZEBOX određuju da li ima gumbe za maksimizaciju i minimizaciju, a konstanta WS_SIZEBOX da li mu se mogu mijenjati dimenzije. Grupa konstanti određuje i početno stanje prozora, tako WS_VISIBLE određuje da li je prozor skriven ili vidljiv, WS_ICONIC određuje da li je početno minimiziran, a WS_MAXIMIZE da li je početno maksimiziran. Konstanta WS_CHILD koristi se za stvaranje prozora-dijeteta.

- Parametri *x, y, nWidth i nHeight* – ovi parametri služe za određivanje početnog položaja i dimenzija prozora. Korištenjem konstante CW_USEDEFAULT na mjestu nekoga od njih sustav postavlja pretpostavljene vrijednosti.

- Parametar *hWndParent – služi za* određivanje roditelja prozora, a postavlja se na upravljačku varijablu prozora roditelja ako se stvara prozor dijete.

- Parametar *hInstance* – u ovaj se parametar postavlja pokazivač na modul programa koji je povezan s prozorom.

- Parametar *lParam* – služi za prijenos parametara za stvaranje prozora putem poruke WM_CREATE koja se obrađuje u proceduri prozora za vrijeme njegovog stvaranja. U *lParam* postavlja se vrijednost pokazivača na strukturu tipa CREATESTRUCT koja sadrži polja identična pojedinim parametrima funkcije *CreateWindowEx*. Na ovaj se način omogućuje dinamičko postavljanje svojstava prozora. Ako se ne koristi, ovome se parametru prenosi vrijednost NULL.

Nakon što se pozove funkcija *CreateWindowEx*, prozor je stvoren i spreman za korištenje te se mogu koristiti API funkcije za upravljanje, dohvaćanje i promjenu njegovog sadržaja i svojstava. Prozor također počinje primati poruke i komunicirati sa operacijskim sustavom kroz proceduru prozora, a može komunicirati i sa drugim prozorima koji postoje na sustavu. Ova komunikacija opisuje se u sljedećem poglavlju. Kako bi se prozoru dodale kontrole, to jest prozori-djeca, koristi se opisani postupak stvaranja samo što se funkciji *CreateWindowEx* prenosi upravljačka varijabla prozora roditelja u parametru *hWndParent* te se parametru *dwStyle* dodaje stil WS_CHILD. Pozivi za stvaranje prozora-djece obično se dodaju u kod za obradu poruke WM_CREATE u proceduri prozora koja je vezena za glavni prozor.

2.3 Poruke i komunikacija

Komunikacija sustava sa prozorom odvija se putem poruka. Obrađivanjem ovih poruka prozori dobivaju informacije o akcijama korisnika ili sustava, a odgovorima na poruke mogu utjecati na upravljanje sustava. U ovome dijelu najprije se opisuje način komunikacije sustava sa prozorima te osnovne programske i podatkovne strukture, a zatim se u sljedećem poglavlju opisuju najvažnije poruke za upravljanje prozorima.

2.3.1 Red poruka i procedura prozora

Kada se pokrene program, operacijski sustav stvara red poruka u koji se spremaju poruke namijenjene nekom od njegovih prozora. Ako se program izvodi u više dretvi, posebni red poruka pridružuje je svakoj od njih nakon pokretanja. Ovi redovi čitaju se u petlji koja se treba izvoditi cijelo vrijeme dok je dretva pokrenuta, a naziva se petlja poruka (eng. *message loop*). U glavnoj dretvi programa petlja poruka obično se nalazi u početnoj funkciji *WinMain* i ima sljedeću strukturu:

```
while (GetMessage(&msg, NULL, 0, 0))
{
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
```

API funkcija *GetMessage* čeka dok se u redu poruka dretve ne pojavi poruka, a nakon toga ju uklanja iz reda te stavlja u prvi parametar *msg*. U drugi parametar funkcije može se prenijeti upravljačka varijabla prozora i tada će funkcija dohvaćati samo one poruke namijenjene tome prozoru. Ako se ovaj parametar postavi na NULL, dohvaćaju se poruke za sve prozore. Pomoću posljednja dva parametra poruke se mogu filtrirati po vrsti, to jest, može se postaviti opseg vrijednosti identifikatora poruka koje će se dohvaćati. Ako se ovi parametri postave na 0, dohvaćaju se sve poruke. Prvi je parametar pokazivač na strukturu MSG koja je osnovna struktura za spremanje poruka i komunikaciju sa sustavom. Deklaracija strukture dana je sa:

```
typedef struct tagMSG
{
   HWND hwnd;
   UINT message;
   WPARAM wParam;
   LPARAM lParam;
   DWORD time;
   POINT pt;
} MSG, *PMSG;
```

- Polje *hwnd* – u ovo polje sustav sprema upravljačku varijablu prozora a provjeravanjem ove vrijednosti program može otkriti kojem je prozoru poruka namijenjena.

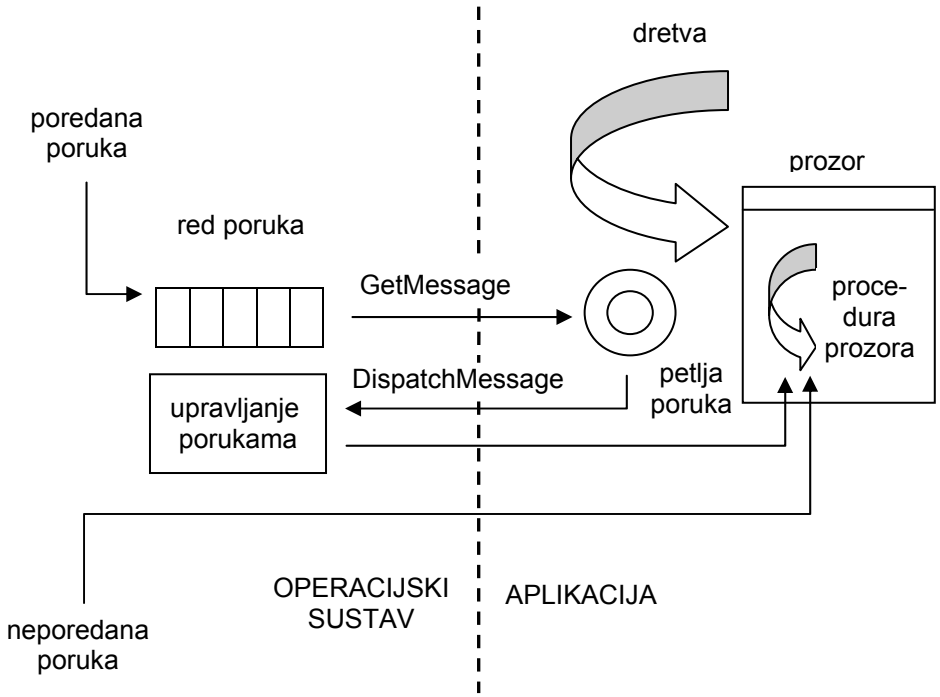
- Polje *message* – varijabla koja predstavlja jedinstveni identifikator poruke, to jest, poruku koja se šalje. Za svaku poruku u sustavu definirane su konstante koje imaju prefiks WM. Tako je, na primjer, WM_PAINT poruka kojom sustav traži obnavljanje sadržaja prozora, a WM_SIZE poruka kojom sustav dojavljuje da su se promijenile dimenzije prozora.

- Polja *wParam* i *lParam* – služe za prijenos parametara poruke i imaju posebno značenje za svaku poruku.

- Polje *time* – služi za prijenos vremena kada je poruka stavljena u red poruka, a u polju *pt* prenose se koordinate pokazivača miša u tom trenutku.

Nakon dohvaćanja poruke funkcijom *GetMessage*, sljedeći korak u petlji poruka je poziv funkcije *TranslateMessage*. Ova se funkcija koristi za pretvorbu poruka korisničkog unosa. Tako se, na primjer, parametri poruke WM_KEYDOWN, koja sadrži informaciju o pritisku neke tipke na tastaturi, nakon dohvaćanja nalaze u formatu "virtualne tipke" (eng. *virtual key*). Ovaj format ne sadrži informaciju o tome koje je slovo ili znak unesen jer to ovisi o formatu tipkovnice i odabranom jeziku unosa, a to su parametri sustava. Pretvorbu u ovaj znakovni format obavlja funkcija *TranslateMessage*. Sljedećim pozivom *DispatchMessage*, poruka se opet prenosi sustavu te ju sustav šalje odgovarajućoj proceduri prozora.

Međutim, opisanim se načinom ne obrađuju sve poruke, nego samo takozvane poredane poruke (eng. *queued messages*). Postoji i druga vrsta, neporedane poruke, koje sustav ne stavlja u red poruka dretve, već se one izravno šalju proceduri prozora. Na slici 2.2 prikazana su ova dva načina slanja poruka.



Slika 2.2 *Poruke u operacijskom sustavu Windows*

Poredane poruke većinom su poruke vezane za korisnički unos kao što su WM_KEYDOWN ili WM_MOUSEMOVE, a neporedane većinom šalju pozivi funkcija operacijskog sustava. Tako funkcija *CreateWindow* nakon stvaranja prozoru šalje neporedanu poruku WM_CREATE, a funkcija *UpdateWindow*, kojom se traži obnavljanje sadržaja prozora, šalje neporedanu poruku WM_PAINT. Postoje i dvije API funkcije pomoću kojih se može poslati proizvoljna poredana ili neporedana poruka. To su *SendMessage*, kojom se šalje neporedana poruka i *PostMessage*, kojom se šalje poredana. Ove funkcije služe za korisničku komunikaciju između dretvi i prozora na način da se prvo definiraju i registriraju korisničke poruke koje se zatim uz upravljačku varijablu prozora prenose kao parametar ovim dvjema funkcijama. Prozor ove poruke, kao i sve druge, dohvaća u proceduri prozora, a dretva koristeći već opisanu petlju poruka. Postupak definicije procedure prozora već je opisan u prethodnom poglavlju – kod definicije razreda prozora u strukturi razreda postavlja se i pokazivač na funkciju koja će se koristiti kao procedura prozora. Budući da se ova funkcija veže uz razred prozora, svi prozori određenog razreda imati će istu proceduru prozora. Njena deklaracija dana je sa:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
```

Parametri procedure prozora identični su poljima strukture MSG. Kako procedura može biti zajednička za više prozora, ispitujući prvi parametar – upravljačku varijablu prozora kojem je poruka namijenjena, obrada se može proslijediti modulu programa koji obrađuje poruke za određeni prozor. Način pozivanja od strane operacijskog sustava je takav da se za svaku poruku namijenjenu za neki prozor procedura poziva jedanput, a za vrijeme obrade jedne poruke ona se ne može prekinuti pozivom za neku drugu poruku. Ovo se odnosi samo na poruke koje proceduri šalje operacijski sustav a nisu rezultat poziva API funkcija iz procedure prozora. Ali slučaju kada se iz procedure prozora za vrijeme obrade neke poruke poziva neka API funkcija koja rezultira generiranjem nove poruke, a funkcija je takva da blokira dok se poruka ne obradi, procedura će se ponovno pozvati za vrijeme obrade, to jest, prekinut će se prvi poziv dok ne završi drugi. Zbog ove mogućnosti proceduru treba konstruirati na način da se vodi računa o mogućim promjenama varijabli zbog drugog poziva i ne može se računati da će varijable uvijek zadržati postavljenu vrijednost. Povratna vrijednost procedure prozora utječe na ponašanje upravitelja prozorima a vrijednosti i njihovo značenje definirane su posebno za pojedine poruke.

Procedura ne mora obrađivati sve poruke za koje se pozove, samo one koje su bitne, na primjer, za prikaz sadržaja ili obradu korisničkog unosa. Poruke koje ne obrađuje trebaju se proslijediti pretpostavljenoj proceduri prozora koja je dio operacijskog sustava pozivom funkcije *DefWindowProc*. Iako se pretpostavljenoj proceduri mogu prenijeti sve poruke i ona će ih obraditi, ovo nema smisla za poruke koje su bitne za sadržaj prozora i interakciju sa korisnikom. Neke poruke moraju se prenijeti pretpostavljenoj proceduri kako bi prozor normalno funkcionirao. Ove poruke, kao i druge koje služe za upravljanje prozorom i njegovim sadržajem, opisuju se u nastavku.

2.4 Upravljanje prozorima

U ovom poglavlju opisuje se način na koji operacijski sustav upravlja prozorima, to jest, slijed poruka i komunikacija između prozora i sustava za pojedine akcije kao što su stvaranje i uništavanje prozora, promjena položaja i dimenzija itd. Sve ove akcije izvodi dio operacijskog sustava koji se naziva upravitelj prozorima, a izvodi se u jezgrenom načinu rada. Tako se, na primjer, promjena dimenzija prozora povlačenjem okvira u potpunosti izvodi u upravitelju prozorima, a aplikaciji se porukom dojavljuju podaci o promjeni i novim dimenzijama kako bi mogla obnoviti sadržaj prozora. Međutim, postoje i slučajevi kada aplikacija komunicira sa sustavom kako bi se izvela neka akcija upravljanja te prozor aktivno sudjeluje u njenom izvođenju tako da šalje parametre upravitelju prozora.

Budući da je kod operacijskog sustava Windows zatvoren nije niti poznat način na koji radi upravitelj prozorima već samo njegove mogućnosti i protokol komunikacije s aplikacijama. Zbog toga nije moguće opisati dio upravljanja koji se odvija u upravitelju prozorima nego samo onaj dio koji se odvija u aplikaciji te način na koji aplikacija komunicira s upraviteljem prozorima.

2.4.1 Stvaranje prozora

U prethodnim poglavljima već je bilo govora o načinu na koji se prozor stvara te je opisana API funkcija *CreateWindowEx* i procedura prozora. U ovome dijelu opisuje se komunikacija između upravitelja prozora i aplikacije za vrijeme stvaranja prozora.

Komunikacija za vrijeme stvaranja prozora odvija se na sljedeći način:

1. sustav šalje WM_NCCREATE i WM_CREATE,
2. sustav šalje WM_GETMINMAXINFO,
3. ako je postavljen stil WS_VISIBLE, sustav šalje WM_PAINT,
4. ako nije postavljeno WS_EX_NOACTIVATE, sustav šalje WM_ACTIVATE i WM_SETFOCUS.

Prve dvije poruke služe za dojavu prozoru da se upravo stvara. Najprije se šalje poruka WM_NCCREATE kojom se dojavljuje stvaranje neklijentskog područja, pa WM_CREATE kojom se dojavljuje stvaranje klijentskog područja. Parametar *lParam* definira se kod poziva *CreateWindowEx*, a parametar *wParam* se ne koristi. Ove poruke u aplikacijama se većinom koriste za obavljanje inicijalizacije i stvaranje prozora-djece. Sljedećom porukom GETMINMAXINFO prozoru se nakon stvaranja dojavljuje da će mu se promijeniti stanje minimizacije i maksimizacije. Ova dojava služi kako bi prozor mogao promijeniti pretpostavljeno

stanje, a to radi tako da postavi polja strukture MINMAXINFO koja se prenosi kao parametar poruke *lParam*. Sljedeća poruka koja se šalje je WM_PAINT, kojom sustav traži obnavljanje sadržaja prozora. Aplikacija reagira tako da pozove funkcije za crtanje sadržaja i pozivima *BeginPaint* i *EndPaint* učini sadržaj prozora važećim. O ovome se više govori kasnije u opisu iscrtavanja sadržaja prozora. Sljedećim dvjema porukama, WM_ACTIVATE i WM_SETFOCUS prozoru se dojavljuje da postaje aktivni prozor i da dobiva mogućnost korisničkog unosa.

2.4.2 Uništavanje prozora

Nakon što aplikaciji prozor više nije potreban poziva se API funkcija *DestroyWindow* kojom se prozor deaktivira i skriva s ekrana te se oslobađaju sve strukture koje on zauzima u memoriji. Ova funkcija ima jedan parametar – upravljačku varijablu prozora. Ako prozor ima prozore-djecu oni se također uništavaju. Operacijski sustav prije uništavanja šalje poruke WM_DESTROY i WM_NCDESTROY kako bi aplikacija mogla osloboditi zauzetu memoriju i obaviti druge akcije prije uništavanja prozora. Ove dvije poruke ne koriste niti *lParam* niti *wParam* parametar. Na kraju obrade poruke WM_DESTROY prozor može pozvati funkciju *PostQuitMessage* koja kao argument uzima izlazni status i uzrokuje stavljanje poruke WM_QUIT u petlju poruka dretve. Petlja poruka dretve, kao što je već navedeno ima sljedeći oblik:

```
while (GetMessage(&msg, NULL, 0, 0))
{
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
```

Kada funkcija *GetMessage* dohvati poruku WM_QUIT vratit će vrijednost 0 i tako će petlja i dretva u kojoj se izvodi završiti.

Osim izravnog pozivanja funkcije *DestroyWindow* postoji i drugi način uništavanja prozora, kada korisnik zatvori prozor pritiskom na gumb "zatvori" ili kombinacijom ALT+F4. Kada se prozor zatvara na ovaj način on se ne uništava se automatski. Komunikacija u slučaju pritiska na gumb "zatvori" odvija se na sljedeći način:

1. sustav proceduri prozora šalje poruku WM_NCHITTEST,
2. aplikacija prosljeđuje poruku pretpostavljenoj proceduri prozora,
3. obrada u pretpostavljenoj funkciji uzrokuje slanje poruke WM_SYSCOMMAND,
4. aplikacija prosljeđuje poruku pretpostavljenoj proceduri prozora,
5. obrada u pretpostavljenoj funkciji uzrokuje slanje poruke WM_CLOSE,
6. aplikacija prosljeđuje poruku pretpostavljenoj proceduri prozora,
7. obrada u pretpostavljenoj funkciji uzrokuje slanje poruke WM_DESTROY.

Poruka WM_NCHITTEST šalje se uvijek kada se miš pomakne iznad područja prozora ili se pritisne tipka miša a omogućuje aplikaciji da njenom obradom definira koja je kontrola na prozoru izabrana. U ovom slučaju, ako procedura prozora vrati HTCLOSE upravitelj prozorima će reagirati kao da je pritisnut gumb "zatvori" i generirati sljedeću poruku. Za pretpostavljene gumbe na traci naslova ovu je poruku dovoljno prenijeti pretpostavljenoj proceduri prozora koja će automatski generirati odgovarajuću povratnu vrijednost. Parametar *wParam* sljedeće poruke WM_SYSCOMMAND postavlja se na SC_CLOSE, a prosljeđivanjem ove poruke pretpostavljenoj proceduri, sustav generira poruku za zatvaranje prozora WM_CLOSE, a nakon njene obrade WM_DESTROY. Navedeni koraci su pretpostavljeni način zatvaranja prozora i kada procedura ne bi prosljeđivala poruke WM_SYSCOMMAND i WM_CLOSE pozivom *DefWindowProc* niti ih obrađivala, zatvaranje prozora ne bi funkcioniralo. Ako se ne želi da se prozor odmah uništi na pritisak gumba "zatvori", procedura prozora treba obraditi jednu od ovih poruka i vratiti vrijednost 0 bez prosljeđivanja pretpostavljenoj proceduri prozora.

2.4.3 Promjena stanja, položaja i dimenzija prozora

Prozor se, ovisno o parametrima postavljenim kod stvaranja, može nalaziti u više stanja: može biti u normalnom stanju, minimiziran ili maksimiziran, može biti aktivan ili neaktivan, skriven ili vidljiv. Također mu se mogu kontinuirano mijenjati dimenzije i položaj na ekranu kada je u normalnom stanju. Ove promjene rade ili ljuska operacijskog sustava tako da prozoru šalje određene poruke ili upravitelj prozorima koji se izvodi u jezgri. Tako traka zadataka (eng. *taskbar*) koja je dio ljuske operacijskog sustava omogućuje minimizaciju ili vraćanje u normalno stanje tako da šalje poruku WM_SYSCOMMAND, a upravitelj prozorima ovu poruku šalje automatski nakon pritiska miša na ikone u traci naslova prozora. Bez obzira na način na koji se poziva, izvođenje same minimizacije ili vraćanja obavlja upravitelj prozorima. Upravitelj prozorima također omogućuje promjenu dimenzija povlačenjem okvira prozora i promjenu položaja na ekranu povlačenjem trake naslova. Sve ove akcije mogu se izvesti programski pozivima API funkcija i bilo koja aplikacija pozivima ovih funkcija može utjecati na prozore bilo koje druge. Ovo je posebno važno za praktični dio rada i zbog toga se u ovome dijelu opisuju i te API funkcije.

Osnovna komunikacija sustava i procedure prozora u slučaju minimizacije odvija se u sljedećim koracima:

1. ako se minimizacija obavlja pritiskom miša na području prozora, prvo se obrađuje poruka WM_NCHITTEST,

2. sustav ili ljuska šalje poruku WM_SYSCOMMAND sa *wParam* = SC_MINIMIZE,

3. aplikacija prosljeđuje poruku pretpostavljenoj proceduri prozora,

4. pretpostavljena procedura minimizira prozor pozivom API funkcije *ShowWindow* sa parametrom SW_MINIMIZE,

5. sustav šalje poruke WM_GETMINMAXINFO, WM_SIZE i WM_WINDOWPOSITIONCHANGING.

Porukom WM_SYSCOMMAND sustav obavještava prozor o naredbi za minimizaciju i omogućuje prozoru da ju obadi i da na taj način kontrolira svoju minimizaciju. Poruka GETMINMAXINFO omogućuje prozoru da promijeni pretpostavljene dimenzije i napredne parametre neposredno prije minimizacije na način da se u proceduri prozora postave parametri strukture MINMAXINFO koji se prenose kao *lParam* ove poruke. Sljedeće dvije poruke WM_SIZE i WM_WINDOWPOSITIONCHANGING obavještavaju prozor o promjeni dimenzija i položaja, a rezultat su načina na koji upravitelj prozorima obavlja minimizaciju: prozoru se mijenjaju visina širina na vrijednost 0 te se prozor postavlja u kut ekrana.

Maksimizacija i vraćanje u normalno stanje obavlja se na jednak način uz razliku što je parametar poruke WM_SYSCOMMAND postavljen na SC_MAXIMIZE odnosno SC_RESTORE, a prozoru se šalje i dodana poruka WM_PAINT kojom se zahtjeva obnavljanje njegovog sadržaja. Također se izvode koraci potrebni za aktivaciju prozora:

1. nakon aktivacije sustav šalje poruke WM_NCACTIVATE WM_ACTIVATE,

2. ako prozor pripada drugoj aplikaciji od one čiji je prozor trenutačno aktivan, šalje se poruka WM_ACTIVATEAPP,

3. sustav šalje poruku WM_SETFOCUS.

Prvim dvjema porukama sustav obavještava prozor da je aktiviran. U slučaju deaktivacije također se šalju ove poruke, a razlika je jedino u parametru *wParam* koji je u slučaju aktivacije postavljen na WA_ACTIVE ili WA_CLICKACTIVE, a u slučaju deaktivacije na WA_INACTIVE. Porukom WM_ACTIVATEAPP sustav dodatno obavještava prozor da je promijenjena i aktivna aplikacija. Parametar *wParam* postavljen je na TRUE u slučaju aktivacije, a na FALSE u slučaju deaktivacije. Porukom WM_SETFOCUS sustav obavještava prozor da je dobio mogućnost unosa sa tipkovnice. U slučaju deaktivacije šalje se poruka WM_KILLFOCUS. Parametar *wParam* ovih dviju poruka postavlja se na vrijednost upravljačke varijable druge aplikacije koja je izgubila, odnosno dobila mogućnost unosa. Aktivacija prozora može se izvesti i pozivom API funkcija *SetActiveWindow* ili *SetForegroundWindow*. Obje funkcije kao argument uzimaju vrijednost upravljačke varijable prozora koji se želi aktivirati, a razlika je u tome što se koristeći *SetActiveWindow* može aktivirati jedino prozor koji se izvodi u dretvi koja ju poziva, a koristeći *SetForegroundWindow* može se aktivirati i prozor druge aplikacije.

Pomak prozora povlačenjem trake naslova odvija se u sljedećim koracima:

1. sustav proceduri prozora šalje poruku WM_NCHITTEST

2. aplikacija prosljeđuje poruku pretpostavljenoj proceduri prozora,

3. obrada u pretpostavljenoj funkciji uzrokuje slanje poruke WM_SYSCOMMAND,

4. aplikacija prosljeđuje poruku pretpostavljenoj proceduri prozora,

5. obrada u pretpostavljenoj proceduri uzrokuje pomak prozora i slanje poruke WM_ENTERSIZEMOVE,

6. za vrijeme pomaka sustav šalje poruke WM_MOVE, WM_MOVING, WM_WINOWPOSCHANGING i WM_WINDOWPOSCHANGED,

7. kada pomak prestane sustav šalje poruku WM_EXITSIZEMOVE.

Slično kao i u već opisanom slučaju zatvaranja prozora, obradom poruke WM_NCHITTEST aplikacija može promijeniti način povlačenja prozora. Ako bi se, na primjer, htjela postići funkcionalnost povlačenja prozora pritiskom na bilo koji njegov dio, procedura prozora treba za obradu ove poruke uvijek vratiti vrijednost HTCAPTION. Prenošenjem poruke pretpostavljenoj proceduri ona će automatski vratiti odgovarajuću vrijednost a ako je pritisak miša bio na traci naslova, vratit će HTCAPTION. Nakon toga upravitelj prozorima generira poruku WM_ SYSCOMMAND sa parametrom *wParam* postavljenim na SC_MOVE. Nakon obrade ove poruke upravitelj prozorima započinje pomak prozora te ga obavještava porukom WM_ENTERSIZEMOVE, a za vrijeme pomaka nastavlja slati poruke koje mu omogućuju praćenje trenutačnog položaja. Poruke WM_MOVE i WM_MOVING specifične su za pomicanje prozora i kao parametar *lParam* prenose koordinate prozora dok je poruka WM_WINOWPOSCHANGING općenitija i šalje se uvijek u slučaju promjene položaja dimenzija ili z-položaja prozora. Promjena dimenzija odvija se na jednak način uz razliku da se proceduri prozora šalju poruke WM_SIZE i WM_SIZING koje prenošenjem dimenzija prozora omogućuju praćenje nove veličine. Također se u WM_SYSCOMMAND prenosi vrijednost SC_SIZE. Budući da se za vrijeme promjene dimenzija mijenja i sadržaj prozora, sustav šalje i poruku WM_PAINT.

Položaj i dimenzije prozora također se mogu promijeniti pozivanjem API funkcija *MoveWindow* i *SetWindowPos*. Funkcija *MoveWindow* omogućuje promjenu položaja i dimenzija, a kao argumente uzima upravljačku varijablu, nove x i y koordinate gornjeg lijevog kuta prozora te novu visinu i širinu. Funkcija *SetWindowPos* je općenitija i uz promjenu dimenzija i položaja omogućuje i promjenu z-položaja te skrivanje i prikazivanje prozora, a detaljnije se opisuje u sljedećem poglavlju.

Osim u navedenim stanjima prozor se također može nalaziti u skrivenom stanju. Kada je prozor skriven on ima sve strukture kao i prikazani prozor jedino nema reprezentaciju na ekranu. Prozor se ne može dovesti u ovo stanje koristeći ljusku operacijskog sustava i ono se većinom koristi unutar aplikacija kada se prozor ne želi uništiti nego samo privremeno sakriti za kasniju uporabu. Prozor se također može stvoriti u skrivenom stanju ako se kod poziva funkcije *CreateWindowEx* u parametru *dwStyle* ne koristi vrijednost WS_VISIBLE. Nakon što je prozor stvoren može se sakriti ili prikazati pozivom API funkcije *ShowWindow* sa parametrom SW_HIDE odnosno SW_SHOW. Pozivom ove funkcije aplikacija može skrivati i

prikazivati i prozore drugih aplikacija, a ova mogućnost koristi se u praktičnom dijelu rada.

2.4.4 Z-raspored i pobrojavanje prozora

Budući da na radnoj površini u isto vrijeme može biti otvoreno više prozora koji se mogu slobodno pomicati, prozor se može nalaziti ispred ili iza nekog drugog prozora. Ovaj odnos između prozora naziva se z-raspored, a položaj određenog prozora z-položaj. Stvaranjem i prikazivanjem novoga prozora on se stavlja na vrh z-rasporeda, aktivira se i dobiva mogućnost unosa s tipkovnice. Koristeći mogućnosti ljuske operacijskog sustava z-raspored se mijenja izborom, to jest, aktivacijom prozora koji tada dolazi na vrh z-rasporeda. Ovo je jedini način promjene z-položaja koristeći ljusku i prozor nije moguće umetnuti na bilo koji položaj u z-rasporedu. Međutim, korištenjem API funkcija operacijskog sustava ovo je moguće, a također je moguće pobrojati sve prozore i pročitati cjelokupni z-raspored. Ove mogućnosti koriste se u praktičnom dijelu rada za spremanje i ponovno prikazivanje stanja radne površine.

U operacijskom sustavu Windows z-raspored se dijeli na dvije razine: normalnu razinu i visoku razinu. Većina prozora nalaze se u normalnoj razini, a visoka razina koristi se kako bi prozor bio uvijek iznad (normalnih) prozora. Primjer ovakvog prozora je Windows upravitelj zadataka. U kojoj razini z-rasporeda se prozor nalazi određuje se kod stvaranja postavljenjem vrijednosti WS_EX_TOPMOST u parametar *dwExStyle* funkcije *CreateWindowEx*, a može se promijeniti i naknadno dodavanjem ili ukanjanjem ove vrijednosti iz proširenog stila prozora. Za ovo se koriste dvije API funkcije – *GetWindowLong* i *SetWindowLong*. Ove dvije funkcije omogućuju i dinamičku promjenu svih stilova prozora od vidljivosti do pretvorbe iz neuslojenog prozora u uslojeni a također se mogu zvati i mijenjati prozore drugih aplikacija. Deklaracija ovih funkcija dana je sa:

```
LONG GetWindowLong(HWND hWnd, int nIndex);

LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);
```

Funkcije su komplementarne, prva dohvaća određenu skupinu parametara prozora a druga ih postavlja na novu vrijednost. U prvi parametar funkcija prenosi se upravljačka varijabla prozora čiji se parametri čitaju odnosno postavljaju, a u drugi identifikator skupine parametara. Ovaj identifikator može biti jedna od 10 definiranih konstanti, a u slučaju postavljanja razine z-rasporeda prenosi se GWL_EXSTYLE. Treći parametar druge funkcije je nova vrijednost stila. Kada se želi promijeniti stil prozora, funkcije se zovu u paru – prvo se dohvaća postojeći stil funkcijom *GetWindowLong* koji se mijenja *ili* kombinacijom konstanti kao što je WS_EX_TOPMOST pa se prenosi funkciji *SetWindowLong*.

Za postavljanje z-položaja prozora na određeno mjesto u z-rasporedu koristi se API funkcija *SetWindowPos*. Funkcija osim ove mogućnosti omogućuje još i postavljanje prozora na proizvoljni položaj te proizvoljnu promjenu dimenzija. Deklaracija funkcije dana je sa:

```
BOOL SetWindowPos(
    HWND hWnd,
    HWND hWndInsertAfter,
    int X,
    int Y,
    int cx,
    int cy,
    UINT uFlags
);
```

U prvi parametar, kao i kod ostalih funkcija za upravljanje prozorima, prenosi se vrijednost upravljačke varijable prozora koja identificira prozor nad kojim se obavlja operacija. U drugi parametar *hWndInsertAfter,* prenosi se upravljačka varijabla prozora ispod kojeg se ovaj prozor želi postaviti i na ovaj se način postavlja položaj u z-rasporedu. Ako se prozor želi postaviti na vrh z-rasporeda normalnih prozora u ovaj se parametar prenosi konstanta HWND_TOP odnosno HWND_TOPMOST ako se postavlja na vrh z-rasporeda u visokoj razini. Sljedeća četiri parametra određuju novi položaj i dimenzije prozora, a posljednji parametar *uFlags* razne zastavice koje utječu na ponašanje funkcije. Na primjer, zastavica SWP_HIDEWINDOW uzrokuje skrivanje prozora, SWP_NOZORDER sprječava promjenu z-položaja a SWP_NOSIZE sprječava promjenu dimenzija. Moguće je prenijeti više zastavica koristeći *ili* kombinaciju.

Kako bi se pročitao z-položaj svih prozora na radnoj površini koriste se funkcije *GetTopWindow* i *GetNextWindow*. Deklaracija ovih dviju funkcija dana je sa:

```
HWND GetTopWindow(HWND hWnd);
```

```
HWND GetNextWindow(HWND hWnd, UINT wCmd);
```

Prva funkcija dohvaća upravljačku varijablu najvišeg prozora u z-rasporedu ako se njen parametar *hWnd* postavi na NULL, a ako se ovaj parametar postavi na vrijednost upravljačke varijable konkretnog prozora, ispituje se z-položaj njegovih prozora-djece. Fuknkcija *GetNextWindow* dohvaća vrijednost upravljačke varijable prethodnog ili sljedećeg prozora u z-rasporedu ovisno o tome da li je parametar *wCmd* postavljen na GW_HWNDNEXT ili GW_HWNDPREV. Koristeći ove dvije funkcije u petlji moguće je pročitati cijeli z-raspored na radnoj površini, to jest, dohvatiti upravljačke varijable poredane po z-položaju, a pomoću drugih API funkcija mogu se dohvatiti ostali parametri prozora. Budući da se u z-rasporedu mogu nalaziti samo vidljivi prozori i oni koji nisu minimizirani, koristeći ove funkcije nije moguće dohvatiti sve postojeće prozore. Ovo omogućuje funkcija *EnumWindows*, a njena deklaracija dana je sa:

```
BOOL EnumWindows(
    WNDENUMPROC lpEnumFunc,
    LPARAM lParam
);
```

U parametar *lpEnumFunc* prenosi se pokazivač na funkciju koju sustav poziva za svaki prozor koji je trenutačno otvoren. Ova funkcija ima oblik:

```
BOOL CALLBACK EnumWindowsProc(
    HWND hwnd,
    LPARAM lParam
);
```

Kada se pozove funkcija *EnumWindows*, sustav poziva *EnumWindowsProc* za svaki prozor koji trenutačno postoji na sustavu i prenosi joj vrijednost njegove upravljačke varijable kao prvi parametar. Parametar *lParam* funkcije *EnumWindows* prenosi se kod svakog poziva funkciji *EnumWindowsProc*. Ako se kao ovaj parametar, na primjer, postavi pokazivač na listu, ovo omogućuje spremanje podataka za svaki prozor izravno iz funkcije *EnumWindowProc*.

2.5 Osnove iscrtavanja sadržaja

U ovom poglavlju ukratko se opisuje način na koji operacijski sustav iscrtava sadržaj prozora na ekran i API funkcije koje se za tu svrhu koriste. U dosadašnjem dijelu rada već je spomenuta podjela prozora na uslojene i neuslojene i poruka WM_PAINT, a ovdje se ovi pojmovi uz ostale detaljnije objašnjavaju.

2.5.1 GDI i kontekst uređaja

Osnovni sustav za crtanje sadržaja na ekran u operacijskom sustavu Windows naziva se *Graphics Device Interface* ili skraćeno GDI. GDI je dio jezgre operacijskog sustava i omogućuje razne grafičke operacije kao što su crtanje osnovnih oblika, teksta, stvaranje, kopiranje i konverziju formata slika itd. Crtanje nije ograničeno samo na ekran već podržava pisače i druge vanjske uređaje. Kako su prozori grafički objekti na ekranu i oni se za svoj prikaz koriste ovim sustavom, a jedina iznimka su prozori koji se za prikaz koriste protokole sa sklopovskim ubrzanjem kao što su OpenGL ili Direct3D. U tom slučaju sustav GDI koristi se samo neizravno i za postavljanje parametara, a prikaz se obavlja koristeći odgovarajuće module jezgre koji komuniciraju izravno sa upravljačkim programom grafičke kartice. Ovo predstavlja problem u praktičnom dijelu rada jer neke API funkcije kao što je na primjer funkcija za uzimanje slike prozora podržavaju samo GDI.

Osnovni koncept u sustavu GDI naziva se kontekst uređaja (eng. *device context*) i predstavlja osnovnu strukturu za GDI operacije. Tako se kod inicijalizacije upravljačkog programa grafičke kartice automatski stvara kontekst uređaja ekrana sa odgovarajućim parametrima, a svaki prozor ima svoj kontekst uređaja kompatibilan s njime. Kontekst uređaja sadrži mnoge parametre i strukture koje se koriste za razne grafičke operacije, a korisniku API sučelja one se prikazuju na apstraktan način. Ove strukture među ostalima uključuju *olovku* za operacije crtanja s linijama, *kist* za crtanje u boji i popunjavanje, *paletu* koja definira dostupne boje i *sliku* (eng. *bitmap*) za dohvaćanje i kopiranje sadržaja. Budući da

se u praktičnom dijelu koristi samo nekoliko mogućnosti GDI sustava sve ove strukture neće se opisivati. Za gotovo sve grafičke operacije u GDI sustavu mora se koristiti kontekst uređaja na način da se ili dohvati postojeći (na primjer, kontekst prozora) ili stvori novi. U praktičnom dijelu rada konteksti uređaja koriste se samo kako bi se mogle izvesti operacije kopiranja slika prozora u memoriju a to se izvodi u sljedećim koracima:

1. dohvaća se kontekst uređaja prozora,
2. stvara se novi kontekst uređaja kompatibilan s njime,
3. stvara se kompatibilna slika i selektira u novi kontekst,
4. slika se kopira iz konteksta uređaja prozora u stvoreni kontekst.

Kontekst uređaja prozora dohvaća se koristeći API funkciju *GetDc* koja kao argument uzima upravljačku varijablu prozora čiji se kontekst uređaja dohvaća. Kompatibilni kontekst uređaja stvara se pozivom *CreateCompatibleDC* sa dohvaćenim kontekstom kao parametrom. Kako bi se operacija kopiranja slike mogla izvesti, u kontekst uređaja treba selektirati sliku koja po formatu i dubini boja mora biti kompatibilna sa slikom u kontekstu s kojeg se kopira. Slika se stvara koristeći funkciju *CreateCompatibleBitmap* kojoj se kao parametri prenose dohvaćeni kontekst uređaja te visina i širina a selektira se funkcijom *SelectObject* kojoj se prenose novi kontekst uređaja i stvorena slika. Samo kopiranje obavlja se funkcijom *BitBlt* koja je deklarirana na sljedeći način:

```
BOOL BitBlt(
  HDC hdcDest, // handle to destination DC
  int nXDest,  // x-coord of destination upper-left corner
  int nYDest,  // y-coord of destination upper-left corner
  int nWidth,  // width of destination rectangle
  int nHeight, // height of destination rectangle
  HDC hdcSrc,  // handle to source DC
  int nXSrc,   // x-coordinate of source upper-left corner
  int nYSrc,   // y-coordinate of source upper-left corner
  DWORD dwRop  // raster operation code
);
```

Prvih pet parametara definiraju odredišni kontekst uređaja te položaj i dimenzije područja na odredišnoj slici u koje se kopira a sljedeća tri kontekst i položaj izvorišnog područja (područje prozora). Ova funkcija osim kopiranja podržava i druge operacije sa pikselima pa se u slučaju kopiranja kao zadnji parametar prenosi vrijednost SRCCOPY. Ako se ovoj vrijednosti *ili* kombinacijom doda konstanta CAPTUREBLT kopirat će se i slika uslojenih i prozirnih prozora.

Na ovaj se način može kopirati samo pravokutno područje na ekranu, a ne sadržaj prozora zbog načina na koji operacijski sustav izvodi iscrtavanje: kontekst uređaja prozora nema vlastitu sliku već je ona samo područje ukupne slike ekrana. Prekrivanjem nekog područja prozora drugim prozorom sadržaj slike se gubi zbog

prikazivanja sadržaja drugog prozora. Ovaj problem trebala bi riješiti API funkcija *PrintWindow* koja je uvedena u inačici Windows XP međutim ova funkcija ima brojne greške koje su detaljnije opisane u poglavlju 3.4. Funkcija je deklarirana sa:

```
BOOL PrintWindow(
  HWND hwnd,
  HDC  hdcBlt,
  UINT nFlags
);
```

Prvi parametar je upravljačka varijabla prozora čija se slika želi dohvatiti, a drugi vrijednost upravljačke varijable konteksta uređaja u koji će se kopirati slika. Treći parametar *nFlags* omogućuje kopiranje samo klijentskog područja prozora ako se prenese vrijednost PW_CLIENTONLY. Funkcija radi na sljedeći način:

1. pretvara prozor u uslojeni,

2. stvara praznu sliku povezanu s prozorom te traži da prozor iscrta svoj sadržaj koristeći funkciju *UpdateWindow*,

3. dohvaća kontekst uređaja novog prozora koristeći *GetDcEx,*

4. kopira sliku iz dohvaćenog konteksta koristeći *NtGdiBitBlt,*

5. oslobađa novi kontekst uređaja i vraća prozor u normalno stanje.

Iz načina na koji funkcija radi vidi se zašto proizvodi neželjene efekte. Budući da preusmjerava obnavljanje prozora može se dogoditi kvarenje sadržaja na ekranu ako se prozor u tom trenutku obnovi. Funkcija koristi *UpdateWindow* kako bi obnovila sadržaj prozora, a na ovaj način ne mogu se obnoviti uslojeni prozori niti prozori koji koriste sklopovsko ubrzanje. Funkcija vjerojatno čeka iscrtavanje prozora određeno vrijeme što objašnjava sporo izvođenje i crna područja u slici jer je moguće da prozor ne stigne obnoviti svoj sadržaj do isteka vremenskog ograničenja.

2.5.2 Mehanizam iscrtavanja prozora

Od operacijskog sustava Windows 2000 podržavaju se dva potpuno različita načina iscrtavanja sadržaja prozora ovisno o vrsti prozora, to jest, da li je prozor uslojeni ili klasični (normalni). Prvo se opisuje klasični način iscrtavanja pa onda način iscrtavanja uslojenih prozora. Zanimljivo je to da je baš zbog postojanja uslojenih prozora i mogućnosti dinamičkog pretvaranja normalnih u uslojene prozore omogućena izvedba API funkcije *PrintWindow*, ali u ovu funkciju nije ugrađena podrška za uzimanje slika samih uslojenih prozora.

2.5.2.1 Klasični prozori

Budući da je u vrijeme nastanka prvih inačica operacijskog sustava memorija bila vrlo ograničena, a slike zauzimaju veliku količinu memorije, glavna koncentracija

kod osmišljavanja sustava bila je ušteda memorije. Zbog toga se u klasičnom iscrtavanju za razliku od, na primjer, sustava MacOS slika prozora ne sprema u memoriju nego se koristi zajednička slika ekrana u koju svi prozori iscrtavaju sadržaj. Kada se prekrije područje prozora nekim drugim prozorom sadržaj prekrivenog prozora se gubi i zbog toga svaki prozor treba u svakom trenutku biti u stanju obnoviti svoj sadržaj na zahtjev sustava. Zahtjev za obnavljanje sadržaja obavlja se u sljedećem koracima:

1. sustav označava područje prozora kao nevaljano,
2. sustav šalje neporedanu poruku WM_PAINT,
3. procedura prozora obnavlja sadržaj nevaljanog područja,
4. procedura prozora označava cijeli prozor valjanim pozivom *DefWindowProc*.

Sustav izvodi ove akcije kod svakog prekrivanja dijela prozora, a za vrijeme stvaranja i prikazivanja prozora, njegovo cijelo područje označava se kao nevaljano. Na svaku poruku WM_PAINT prozor treba odgovoriti ili obnavljanjem cjelokupnog sadržaja ili obnavljanjem nevaljanog područja te ponovno označiti područje kao valjano prenošenjem poruke WM_PAINT pretpostavljenoj proceduri prozora. Kako bi se otkrilo koje je područje nevaljano poziva se API funkcija *GetUpdateRect* koja vraća koordinate i dimenzije nevaljanog područja za traženu vrijednost upravljačke varijable prozora.

Postoje i API funkcije kojima se može obnoviti sadržaj prozora na zahtjev. Kako bi se označilo nevaljalo područje koristi se funkcija *InvalidateRect*:

```
BOOL InvalidateRect(
  HWND hWnd,
  CONST RECT* lpRect,
  BOOL bErase
);
```

U prvi se parametar prenosi upravljačka varijabla prozora čije se područje želi označiti kao nevaljano, a u drugi relativni položaj i dimenzije toga područja. Treći parametar označava treba li izbrisati pozadinu nevaljanog područja. Nakon što je područje definirano potrebno je poslati poruku WM_PAINT, a ovo omogućuje funkcija *UpdateWindow* koja kao parametar prihvaća upravljačku varijablu prozora čiji sadržaj treba obnoviti. Problem u ovom načinu je što se ne može utjecati na obnavljanje neklijentskog područja prozora budući da se ono obnavlja na poseban način i koristi se poruka WM_NCPAINT. Također se ne može utjecati na obnavljanje prozora-djece. Kako bi se ovo postiglo može se koristiti funkcija *RedrawWindow:*

```
BOOL RedrawWindow(
  HWND hWnd,
  CONST RECT *lprcUpdate,
  HRGN hrgnUpdate,
  UINT flags
);
```

Ova funkcija omogućuje definiciju nevaljanog područja i slanje poruka za obnavljanje u jednom koraku i podržava definiranje kvadratnog područja parametrom *lprcUpdate* ili nekvadratnog parametrom *hrgnUpdate*. Posljednjim parametrom definira se *ili* kombinacija zastavica koje utječu na rad funkcije. Ovim parametrom može se izabrati i da li je uključeno i ne-klijentsko područje i svi prozori-djeca. Nedostatak ove funkcije je što u slučaju da se izabere obnavljanje i neklijenskog područja i prozora-djece, funkcija ignorira definirano područje te obnavlja cijeli prozor. Ovo predstavlja problem i u praktičnom dijelu rada jer je kod ispravljanja kvarenja sadržaja zbog greške u funkciji *PrintWindow* baš potrebno obnoviti točno definirano područje prozora. Budući da bi funkcija RedrawWindow ponekad uzrokovala treperenje cijeloga prozora koristi se trik da se privremeno prikaže "nevidljivi" prozor iznad problematičnog područja, pa sustav automatski obnovi prekriveno područje nakon njegovog uklanjanja.

2.5.2.2 Uslojeni prozori

Uslojeni prozori imaju drugačiji način iscrtavanja sadržaja od klasičnih i podržavaju efekt prozirnosti. Razlika je u tome što aplikacija može u potpunosti upravljati iscrtavanjem sadržaja, a operacijski sustav se brine o očuvanju sadržaja prozora pa se ne moraju obrađivati poruka WM_PAINT. Kako bi se stvorio uslojeni prozor potrebno je u parametar *dwExStyle* funkcije *CreateWIndowEx* dodati zastavicu WS_EX_LAYERED. Također je moguće i dinamički postaviti ovu zastavicu funkcijom *SetWindowLong* i tako pretvoriti normalni prozor u uslojeni. U tom slučaju sam sustav preusmjerava iscrtavanje i interno koristi funkcije za uslojene prozore, ali podržava klasični način iscrtavanja koristeći poruku WM_PAINT. Drugi način koji je moguće koristiti samo u uslojenim prozorima je iscrtavanje funkcijom *UpdateLayeredWindow*. Prije poziva funkcije potrebno je stvoriti vizualnu reprezentaciju sadržaja prozora u memoriji koristeći kontekst uređaja u koji je selektirana slika. Deklaracija funkcije *UpdateLayeredWindow* dana se sa:

```
BOOL UpdateLayeredWindow(
    HWND hwnd,
    HDC hdcDst,
    POINT *pptDst,
    SIZE *psize,
    HDC hdcSrc,
    POINT *pptSrc,
    COLORREF crKey,
    BLENDFUNCTION *pblend,
    DWORD dwFlags
);
```

U prvi se parametar prenosi upravljačka varijabla uslojenog prozora čiji se sadržaj obnavlja. Za drugi parametar *hdcDst* obično se postavlja vrijednost NULL i tada funkcija automatski dohvaća kontekst uređaja ekrana. Pomoću sljedeća dva parametra *pptDst* i *psize* mogu se postaviti novi položaj i dimenzije prozora, a prenošenjem vrijednosti NULL položaj i dimenzije se ne mijenjaju. U parametar *hdcSrc* prenosi se kontekst uređaja koji sadrži sliku prozora, a u *pptSrc* točka u

slici od koje se slika prenosi na prozor. Sljedećim parametrom *crKey* može se definirati RGB vrijednost prozirne boje. Parametrom *pblend* određuje se količina prozirnosti, a *dwFlags* određuje da li funkcija koristi vrijednost definiranu u *pblend* (ULW_ALPHA), prozirnu boju (ULW_COLORKEY) ili se crta neprozirni prozor (ULW_OPAQUE).

Nakon poziva ove funkcije slika i kontekst uređaja iz kojeg se prozor obnavlja mogu se do potrebe promjene obrisati i sustav će sam održavati ispravnu reprezentaciju prozora na ekranu. Prednosti uslojenih prozora jesu bolje iscrtavanje na ekranu (manje treperenja) i mogućnost korištenja prozirnosti. Nedostatak je to što ako se koristi funkcija *UpdateLayeredWindow* nije moguće na jednostavan način prikazati standardne kontrole (prozore-djecu) kao što su gumbi ili polja za unos. Kako bi se ovo postiglo mora se koristiti klasični način iscrtavanja koji na uslojenom prozoru koristi više memorije jer sustav cijelo vrijeme održava sliku prozora u memoriji.

2.6 Prednosti i nedostaci upravljanja prozorima

Prednost upravljanja prozorima u operacijskom sustavu Windows je bogata biblioteka API funkcija kojima se podržavaju praktički sve operacije koje izvodi jezgra sustava. Tako postoje funkcije za postavljanje dimenzija i položaja i svih mogućih stanja prozora, obnavljanje sadržaja na zahtjev, slanje poruka, a također je moguće dohvaćati i postavljati sve parametre prozora. Sustav je i vrlo kompatibilan sa prethodnim inačicama i funkcije ne zastarijevaju i ne mijenjaju se iz inačice u inačicu nego se dodaju njihova proširenja. Jedna velika prednost za praktični dio ovoga rada je mogućnost poziva gotovo svih funkcija za upravljanje prozora nad prozorima drugih aplikacija. Ovo omogućuje programsko upravljanje cjelokupnom radnom površinom pa čak i izgradnju alternativne ljuske sustava. Međutim ovo se može promatrati i kao nedostatak jer bi se zloćudnim programom moglo potpuno onemogućiti rad korisnika tako da se na primjer prozori automatski skrivaju i gase ili nekontrolirano pomiču po ekranu.

Glavni nedostatak sustava pa tako i dijela za upravljanje prozora proizlazi iz zatvorenosti koda i teškom ili nemogućem prijavljivanju grešaka Microsoftu. Tako je, na primjer API funkcija *PrintWindow* toliko loše izvedena i puna grešaka da je gotovo neupotrebljiva. Unatoč tome nema niti napomene o bilo kakvom problemu niti iznimkama za korištenje ove funkcije, a od prvog izdanja operacijskog sustava Windows XP nije se ništa napravilo kako bi se funkcionalnost poboljšala.

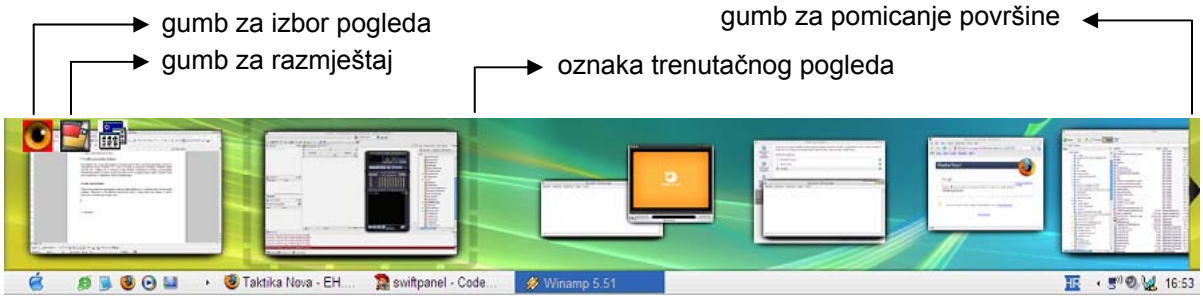
Iako se ne može nazvati nedostatkom budući da svi moderni operacijski sustavi koriste sličan način prikaza, predstavljanje prozora u traci zadataka sa ikonom i dijelom naslova dosta je nepregledno u slučaju da je otvoreno više prozora. Ovaj problem pokušat ću umanjiti programom za prikaz radne površine na panoramskoj traci koji opisujem u sljedećem poglavlju.

3. Izvedba panoramske radne površine

Kao praktični dio ovoga rada napisan je program koji bi trebao poboljšati upravljanje prozora na operacijskom sustavu Windows XP. Ideja se temelji na postojećim rješenjima virtualnih radnih površina, ali s razlikom da je virtualna površina linearna i beskonačna. Površina je predstavljena beskonačnom trakom na koju se mogu postavljati prozori te mijenjati njihov raspored. Svaki prozor predstavlja se umanjenom sličicom (*eng. thumbnail*).

3.1 Opis značajki programa

Traka koja predstavlja panoramsku radnu površinu prikazuje se u donjem dijelu ekrana iznad trake zadataka. Prikazuje se dovođenjem pokazivača miša u donji desni kut ekrana, a skriva pritiskom i držanjem lijeve tipke miša. Na slici 3.1 prikazan je izgled trake i označene su kontrole.



Slika 3.1 *Izgled i kontrole trake za prikaz panoramske radne površine*

Kod prvog prikazivanja s lijeve strane se okruženo crtkanom linijom prikazuje trenutačno stanje radne površine tj. prozori vidljivi na radnoj površini. Prikaz u crtkanom području dinamički preslikava stanje prozora na radnoj površini.

Pritiskom na gumb za razmještaj omogućuje se promjena razmještaja prozora na traci na način kao i na radnoj površini s razlikom da se prozori mogu "primiti" i vući pritiskom na bilo koji dio njihove površine. Promjena položaja prozora na traci odražava se i na radnoj površini, a ako se prozor povuče izvan crtkane linije nestaje sa radne površine i iz trake zadataka. Drugi način postavljanja prozora je izravno dovlačenje sa radne površine na bilo koje mjesto na traci.

Pritiskom na gumb za izbor pogleda prikazuje se izbornik pogleda koji se pomiče lijevo i desno po traci prateći pokazivač miša. Pritiskom lijeve tipke miša pogled se mijenja, a promjena se odražava na radnoj površini:

- prozori koji su u novom pogledu prikazuju se na radnoj površini i na traci zadataka,
- prozori iz starog pogleda uklanjaju se sa radne površine i trake zadataka.

Pogled je moguće i kontinuirano mijenjati tako da se za vrijeme pomicanja izbornika pogleda drži lijeva tipka miša.

Gumbima za pomicanje površine može se pomaknuti cijeli prikaz i osloboditi dodatno mjesto na traci koje je praktički neograničeno. Prikaz se također pomiče i povlačenjem prozora na rubove trake.

3.2 Osnovna struktura i problemi implementacije

Funkcionalnost programa može se podijeliti u 5 osnovnih cjelina koje moraju zajedno surađivati:

1. modul za dohvaćanje stanja radne površine i detekciju promjena,
2. modul za uzimanje slika prozora,
3. modul za detekciju povlačenja prozora,
4. modul za upravljane radnom površinom,
5. modul za prikaz.

Moduli se izvode paralelno i potrebno je riješiti i problem sinkronizacije među njima.

Budući da program manipulira sa slikama visokih rezolucija osnovni je zahtjev ostvariti relativno malo korištenje resursa računala takvo da ne utječe na brzinu rada ostalih programa a s druge strane omogućiti brz odziv programa. Drugi problem proizlazi iz vrlo ograničene podrške operacijskog sustava za dohvaćanje slika prozora. Ovaj problem niti jedno od autoru poznatih komercijalnih ili *Open Source* rješenja koja zahtijevaju ovu funkcionalnost ne rješava u potpunosti. Ostali problemi su algoritamske prirode, na primjer, pronalaženje načina za brzu detekciju promjene sadržaja prozora kako bi se izbjeglo skupo periodičko uzimanje slika. Također je potrebno riješiti probleme vezane za brojne greške u API funkciji za uzimanje slike prozora koje se manifestiraju kvarenjem sadržaja prozora na radnoj površini i drugim neželjenim efektima koji ometaju rad korisnika. Rješenja svakog od ovih problema opisuju se u sljedećim poglavljima u kojima se govori o funkcionalnosti pojedinih modula programa.

U izvedbi se koristi programski jezik Delphi, API funkcije operacijskog sustava, biblioteke jezika Delphi, OpenGL te biblioteka za operacije sa slikama FreeImage. Paradigma je proceduralna uz korištenje osnovnih objekata (lista, hashtablica) i objekata za čuvanje podataka te osnovne operacije nad njima.

3.3 Modul za dohvaćanje stanja radne površine i detekciju promjena

Ovaj modul izvodi se kontinuirano u posebnoj dretvi, prati stanje prozora na radnoj površini, detektira promjene i održava strukturu podataka potrebnu za prikaz. Također se poziva i iz modula za upravljanje radnom površinom kako bi se obnovilo stanje podataka neposredno prije promjene pogleda.

3.3.1 Struktura podataka

Svi podaci potrebni za prikaz spremaju je u jedan objekt razreda *TWinBitmaps* deklariranom u datoteci winbitmaps.pas. U ovoj datoteci nalazi se i cjelokupni kod modula za dohvaćanje stanja radne površine i detekciju promjena te modula za uzimanje slika prozora.

Deklaracija razreda *TWinBitmaps* dana je sa:

```
TWinBitmaps = class
  WinList : TList;
  hsh : THashedStringList;
  zlista : TStringList;
  tmp_zlista : THashedStringList;
  x1lista, x2lista : TList;

  function DajPoZ(z : integer) : TWinBitmap;

  constructor Create;
  destructor Destroy;  override;

  function Kopiraj : TWinBitmaps;
end;
```

Razred ima samo jednu instancu u kojoj se održava reprezentacija stanja radne površine i svih prozora koji se prikazuju na panoramskoj traci a instancira se u modulu za dohvaćanje stanja nakon pokretanja programa. Pristup objektu imaju svi moduli programa osim modula za detekciju povlačenja prozora koji održava svoju strukturu podataka. U nastavku su opisana pojedina polja i funkcije razreda.

- Polje *WinList* – lista u koju se spremaju objekti tipa *TWinBitmap* od kojih svaki predstavlja stanje pojedinog prozora te sadrži sve podatke potrebne za njegovo predstavljanje na panoramskoj traci i komunikaciju sa stvarnim prozorom. Za svaki prozor koji postoji na radnoj površini, uključujući i prozore koji su skriveni održava se zapis u polju *WinList* te se postavljaju oznake ovisno o njegovoj vrsti.

- Polje *hsh* – hash tablica koja omogućuje brzi dohvat zapisa za pojedini prozor pomoću njegove upravljačke varijable. Zapisi su oblika *handle-indeks*, gdje je *indeks* indeks zapisa prozora u polju *WinList*. Koristi se u mnogim dijelovima programa za operacije kao što su dohvaćanje i obnavljanje parametara prozora,

obnavljanje lista te operacije s njima, računanje presjeka prozora, pronalaženje prozora za prikaz kod povlačenja na traku itd.

- Polje *zlista* – pomoćna lista koja sadrži zapis redoslijeda prozora u trenutačnom pogledu poredanih po z-rasporedu. Koristi se u modulu za dohvaćanje stanja i detekciju promjene za vrijeme dohvaćanja stanja radne površine i generiranja drugih lista. Također se koristi i za računanje presjeka prozora.

- Polje *tmp_zlista* – hash tablica u koju se sprema z-raspored svih prozora koji se prikazuju na panoramskoj traci. Tablicu održavaju modul za dohvaćanje stanja koji ju usklađuje sa novim stanjem trenutačnog pogleda tj. sa promjenama na radnoj površini i modul za prikaz koji mijenja stanje tablice kod pomicanja prozora po traci. Zapisi su oblika *handle:indeks u WinList*.

- Polja *x1lista i x2lista* – pomoćne liste koje održava modul za dohvaćanje stanja a koriste se kod inicijalizacije prikaza panoramske trake za određivanje da li će se početni pogled prikazati na krajnjem lijevom tj. krajnjem desnom rubu trake ili na sredini da bi se osiguralo da je prikaz na traci maksimalno popunjen. *x1lista* sadrži indekse prozora u polju *WinList* sortirane po x koordinati lijevog ruba, a *x2lista* indekse sortirane po x koordinati desnog ruba prozora.

- Funkcija *DajPoZ* – funkcija koja vraća zapis prozora na traženom z-položaju.

- Funkcija *Kopiraj* – kopira objekt *WinBitmaps* u novu instancu (koristi se za efikasniju sinkronizaciju između dretve modula za prikaz i postavljanje i dretve za dohvaćanje stanja.

U nastavku je opisana struktura podataka za spremanje stanja jednog prozora. Deklaracija je dana sa:

```
TWinBitmap = class
  slika_pfi : PFIBITMAP;
  slika_pfi2 : PFIBITMAP;
  tex : gluint;
  tex2 : gluint;
  tekstura_vezana, tekstura2_vezana : boolean;
  tekstura_bila_vezana, tekstura2_bila_vezana : boolean;
  vrijeme : TDateTime;
  vrijeme_zadnjeg_shota : longint;
  minmax : TSProz;
  dimenzije : TDimenzije;
  dimenzije_tmp : TDimenzije;
  prave_dimenzije : TDimenzije;
  postoji : boolean;
  handle : HWND;
  promjena : smallint;
  ceka_shot : boolean;
  tmp_matrica : TProzMatrica;
  ontop : boolean;
  smrzni_polozaj : boolean;
```

```
  blt : boolean;

  na_desktopu : boolean;
  privremeno_restoriran : boolean;
  namjerno_skriven : boolean;
  rest_dimenzije : TDimenzije;
  tmp_gl_xoff, tmp_gl_yoff : integer;
  otisak : TOtisak;
  tmp_otisak : TTmpOtisProz;
  title : tcharr;
  classname : tcharr;
  pomaknut : boolean;
  ne_diraj : boolean;
  ne_prikazuj : boolean;
  function Kopiraj : TWinBitmap;

  constructor Create;
  destructor Destroy;  override;
end;
```

- Polje *slika_pfi* – sadrži sliku prozora u punoj rezoluciji to jest u najbližoj rezoluciji originalnoj gdje su širina i visina svedeni na potenciju od 2, zbog korištenja u OpenGL prikazu. Slika je tipa PFIBITMAP koji je osnovni format slike koji se koristi u biblioteci Freeimage. Slika u punoj rezoluciji ne koristi se za prikaz i predviđena je za neke druge primjene koje još nisu implementirane kao na primjer, animacija pomicanja i prijelaza između pogleda. Ovaj je podatak najveći potrošač memorije budući da slika nije komprimirana.

- Polje *slika_pfi2* – sadrži sliku prozora u smanjenoj rezoluciji a dobiva se bilinearnom promjenom rezolucije originalne slike u modulu za uzimanje slika. Visina i širina se također svode na potenciju od 2 zbog korištenja u OpenGL prikazu. Ova se slika koristi za prikaz sličica na traci a ovisno o postavljenim parametrima konfiguracije može biti 2, 4 ili 8 puta manja (po jednoj dimenziji) od slike u punoj rezoluciji. Slika se ne uzima za one prozore koji se ne prikazuju na traci tj. imaju postavljen parametar *ne_prikazuj* na *true*.

- Polja *tex i tex2* – ova dva polja predstavljaju OpenGL teksture koje su potrebne kako bi se slika prozora mogla prikazati na traci.

- Polja *tekstura_vezana i tekstura_bila_vezana* – pomoćne varijable koje se koriste u modulu za prikaz za vezanje i oslobađanje OpenGL tekstura kod promjene slike prozora. Postavljaju se i u modulu za dohvaćanje stanja kada se detektira promjena slike.

- Polje *vrijeme* – polje koje sadrži vrijeme dohvaćanja stanja prozora.

- Polje *vrijeme_zadnjeg_shota* – sadrži vrijeme u Unix formatu kada je zadnji put uzeta slika prozora a služi za ograničenje učestalosti uzimanja slika.

- Polje *minmax* – sadrži informaciju o tome da li je prozor u normalnom stanju ili je minimiziran odnosno maksimiziran.

- Polje *dimenzije* – sadrži informaciju o dimenzijama i položaju prozora na panoramskoj traci.

- Polje *dimenzije_tmp* – pomoćno polje koje se koristi kod zapisivanja novog stanja prozora.

- Polje *prave_dimenzije* – sadrži prave dimenzije prozora na radnoj površini. Razlika u odnosu na polje dimenzije je da se u x koordinati ne uzima u obzir položaj na panoramskoj traci. Polje se postavlja i za prozore koji se ne prikazuju na traci a služi za operacije u kojima je potrebno odrediti točan raspored i prekrivanje prozora na radnoj površini.

- Polje *postoji* – pomoćno polje koje se koristi kod usklađivanja zapisa sa novim stanjem radne površine (kod dohvaćanja stanja).

- Polje *handle* – upravljačka varijabla prozora tj. jedinstveni identifikator prozora koji se koristi za sve funkcije operacijskog sustava koje dohvaćaju ili mijenjaju njegovo stanje.

- Polje *promjena* – pomoćno polje u koje se sprema vrsta promjene prozora u odnosu na minimizaciju i maksimizaciju a koristi se kod dohvaćanja novog stanja radne površine.

- Polje *ceka_shot* – pomoćno polje u koje se sprema informacija da prozor čeka uzimanje slike ukoliko se slika nije mogla uzeti odmah zbog ograničenja učestalosti uzimanja slika ili drugih razloga.

- Polje *tmp_matrica* – crno-bijela slika prozora svedena na 1 bajt po pikselu. Koristi se za detekciju grešaka i problema nastalih zbog korištenja funkcije *PrintWindow*.

- Polje *ontop* – sadrži informaciju da li je prozor u takvom stanju da je uvijek iznad normalnih prozora. Koristi se za ograničenje promjene z-rasporeda kod razmještaja prozora po panoramskoj traci.

- Polje *smrzni_polozaj* – pomoćno polje za sinkronizaciju koje služi da bi se onemogućilo istovremeno pomicanje prozora pomoću panoramske trake i dohvaćanje njegovog stanja u modulu za dohvaćanje stanja radne površine i detekciju promjena.

- Polje *blt* – pomoćno polje koje sadrži informaciju da li se za uzimanje slike prozora koristi funkcija *BitBlt* (izravno kopiranje bitova sa ekrana) ili *PrintWindow*. Postavlja se ovisno o tome da li je prozor prekriven s drugim prozorima ili izlazi izvan ekrana.

- Polje *na_desktopu* – označava da li je prozor predviđen da se stavlja na panoramsku traku, to jest omogućava da se neki prozori izdvoje iz pomaka i da budu stalno na istome mjestu.

- Polje *privremeno_restoriran* – označava da je prozor privremeno prebačen iz maksimiziranog u normalno stanje što se događa kod promjene pogleda ako je u trenutačnom pogledu maksimizirani prozor. Postavljanjem ovoga polja pamti se da je prozor originalno maksimiziran i da se treba vratiti u to stanje kod postavljanja pogleda na njega. Polje postavlja i čita modul za upravljanje radnom površinom.

- Polje *namjerno_skriven* – označava da je prozor skriven kod promjene pogleda. Ovo se događa kada prozor izađe izvan pogleda. Polje postavlja modul za upravljanje radnom površinom, a čita modul za dohvaćanje stanja kako bi prozor bio izuzet iz detekcije.

- Polje *rest_dimenzije* – polje sadrži dimenzije prozora u normalnom stanju, čak i kad je prozor minimiziran ili maksimiziran.

- Polja *tmp_gl_xoff, tmp_gl_yoff*: - pomoćne varijable koje se koriste u OpenGL prikazu (modulu za prikaz) za izvedbu pomicanja pojedinih prozora po traci.

- Polje *otisak* –polje koje služi za detekciju vizualne promjene sadržaja prozora a sadrži matricu sa brojčanim reprezentacijama pojedinih kvadratnih područja na prozoru tj. zbrojeve vrijednosti piksela na tim područjima.

- Polje *tmp_otisak* – pomoćno polje koje se koristi kod generiranja novog otiska.

- Polje *title* – naslov prozora.

- Polje *classname* – ime razreda prozora.

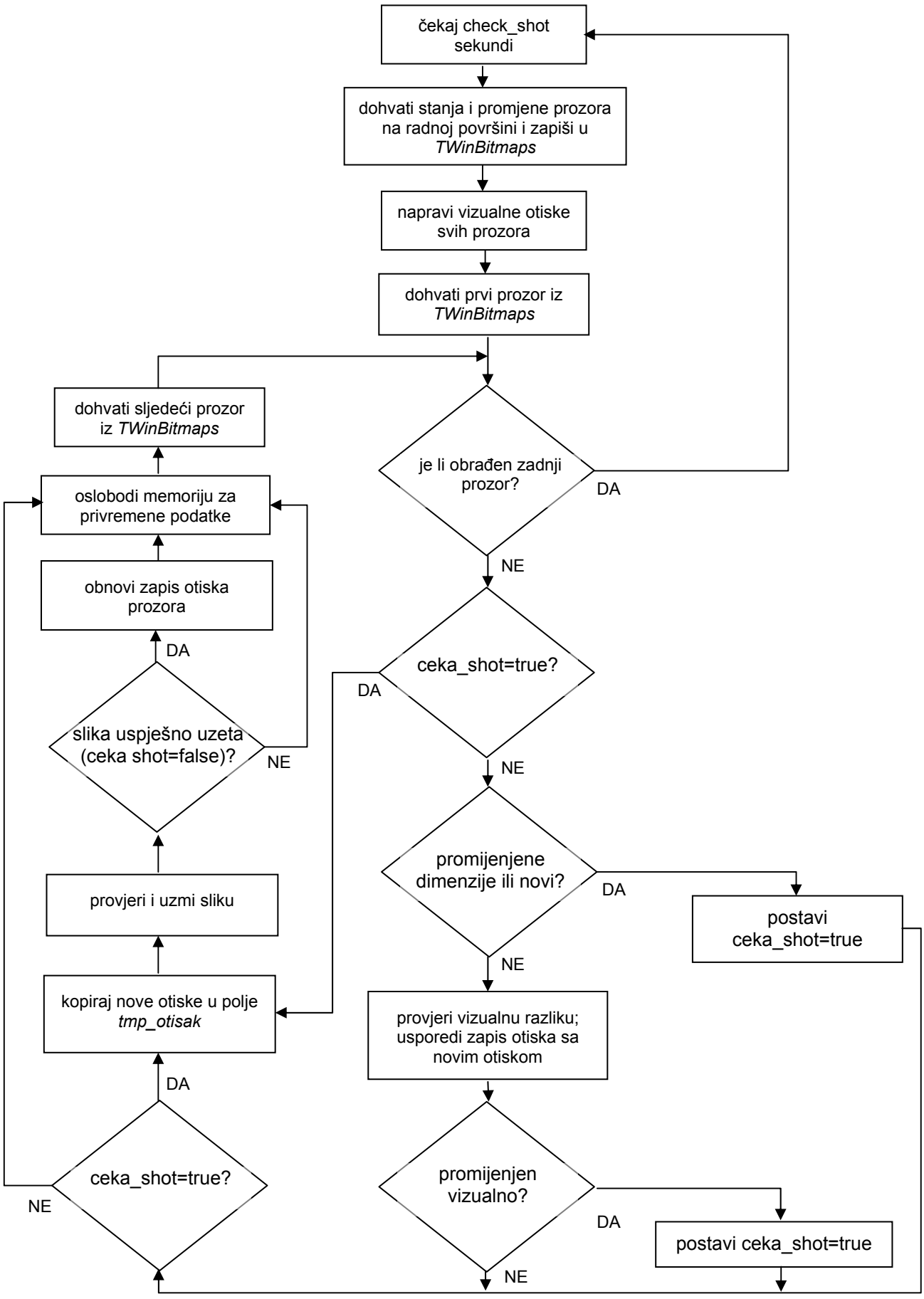
- Polje *pomaknut* – pomoćno polje koje se koristi kod izrade nove liste z-rasporeda u modulu za detekciju promjena.

- Polje *ne_diraj* – polje koje se postavlja ako je prozor izvan ekrana, to jest ako je izašao izvan ekrana zbog promjene pogleda.

- Polje *ne_prikazuj* – označava da li je prozor predviđen da se stavlja na panoramsku traku, to jest omogućava da se neki prozori izdvoje iz pomaka i da budu stalno na istom mjestu.

3.3.2 Opis funkcionalnosti

U ovom poglavlju prvo se opisuje osnovna funkcionalnost modula za dohvaćanje stanja radne površine i detekciju promjena dijagramom toka. Zatim se daje opis pojedinih modula iz dijagrama toka.



Slika 3.2 *Dijagram toka modula za dohvaćanje stanja radne površine i detekciju promjena*

Petlja koju prikazuje dijagram toka na slici 3.2 započinje odmah nakon pokretanja programa i izvodi se kontinuirano za vrijeme njegovog rada. Prije početka petlje inicijalizira se objekt *TWinBitmaps* koji služi za spremanje svih podataka za prikaz panoramske radne površine. Nakon toga pokreću se dretve modula za prikaz i modula za detekciju povlačenja prozora. Kao što se vidi iz dijagrama, detekcija stanja prozora i uzimanje slika obavlja se kontinuirano i neovisno o tome je li panoramska traka prikazana. Razlog ovakvog rješenja je to što je nemoguće brzo uzimanje slike prozora neposredno prije prikaza panoramske trake prvenstveno zbog ograničenja API funkcije *PrintWindow* i grešaka u njenoj implementaciji od kojih se neke mogu ispraviti jedino da se funkcija zove dva ili više puta za redom. Tada se vrijeme izvođenja funkcije povećava i na više od sekunde i uzrokuje neprihvatljivo spor odziv programa. Ovi problemi su opisani u opisu modula za uzimanje slika prozora. U nastavku su opisane pojedine funkcije iz dijagrama, a to su: dohvaćanje stanja i promjena prozora na radnoj površini, izrada vizualnih otisaka i provjera vizualne razlike. Provjera i uzimanje slika nije opisana jer obuhvaća modul za uzimanje slika prozora koji se posebno opisuje.

3.3.2.1 Dohvaćanje stanja i promjena prozora na radnoj površini

U ovome koraku obavlja se pobrojavanje svih prozora koji postoje na radnoj površini te usklađivanje podataka u *WinBitmaps* sa novim stanjem uključujući i detekcija bitnih promjena za svaki prozor. Ovaj korak nalazi se u funkciji *dohvati_stanja*. Funkcija omogućava kontinuirano obnavljanje prikaza na panoramskoj traci ovisno o promjenama na radnoj površini. Detektiraju se sljedeće promjena pojedinih prozora:

- promjena položaja,
- promjena dimenzija,
- promjena stanja (minimizacija, maksimizacija, normalizacija),
- stvaranje novoga prozora i nestanak prozora,
- promjena z-položaja prozora.

Jezgra ove funkcije je API funkcija *EnumWindows* koja omogućava dohvaćanje upravljačkih varijabli svih postojećih prozora. Za svaku se upravljačku varijablu zatim u *callback* proceduri koju za svaki prozor poziva operacijski sustav, pomoću drugih API funkcija dohvaćaju parametri pojedinih prozora. To su funkcije *GetWindowPlacement* koja dohvaća atribute prozora kao što stanje minimizacije/ maksimizacije, funkcija *GetWindowRect* koja dohvaća stvarne dimenzije i položaj prozora, *GetWindowLong* koja dohvaća stilove prozora uključujući i podatak da li je prozor uvijek iznad svih prozora, *GetWindowText* koja dohvaća naslov i *GetClassName* koja dohvaća ime razreda prozora. Z-raspored se dobiva korištenjem funkcija *GetTopWindow* i *GetNextWindow*. Ove funkcije opisane su detaljnije u poglavlju 2.3.

Na početku funkcije za dohvaćanje stanja pretpostavlja se da ne postoji niti jedan prozor iz *WinBitmaps.WinList* koji nije namjerno skriven zbog promjene pogleda.

Ako je prozor namjerno skriven postavlja se oznaka *prozor.ne_diraj* = *true*. Koristeći API funkciju *EnumWindows* zatim se pobrojavaju svi prozori na radnoj površini i ako *ne_diraj* nije postavljen na *true* dohvaćaju se nove dimenzije i ostali parametri te se obnavlja zapis u *WinBitmaps.Winlist*. Prethodno se uspoređuju staro i novo stanje, te se postavlja podatak o promjeni stanja prozora (minimizacija, maksimizacija, promjena veličine itd.) koji se kasnije koristi za odluku da li treba uzeti novu sliku ili ne. Također se za svaki nađeni prozor označava da postoji. Ako zapis prozora nije postojao u starom stanju, dodaje se novi zapis, a ovisno o tome da li prozor zadovoljava uvjete za prikaz na panoramskoj traci, postavlja se parametar prozora *ne_prikazuj*. Prozori kojima se ovaj parametar treba postaviti su, na primjer, prozor radne površine, prozor start-menija te prozori koje definira korisnik zato što želi da uvijek ostanu na istom mjestu kod promijene pogleda. Primjer ovakvoga prozora je "sidebar" u Windows Visti. Budući da se x-koordinata prozora treba zapisati ovisno o položaju na panoramskoj traci za sve prozore se zapisuje dohvaćena x-koordinata zbrojena sa položajem trenutačnog pogleda (varijablom *polozaj_panorame).*

Nakon što završi pobrojavanje prozora provjerava se da li je uspješno obavljeno, a zatim se redom prolaze svi prozori u *WinBitmaps.Winlist*, a oni koji su još uvijek označeni da ne postoje, brišu se iz liste. Sada se generira hashtablica *WinBitmaps.hsh* koja se u raznim dijelovima programa koristi za dohvat zapisa prozora po upravljačkoj varijabli. Zatim se generiraju *x1lista* i *x2lista*. Također se i koristeći funkcije *GetTopWindow* i *GetNextWindow* prelaze svi prozori po z-rasporedu, te se generira lista z-rasporeda *Winlist.zlista* koja sadrži z-raspored svih prozora vidljivih na radnoj površini (u trenutačnom pogledu).

Nakon toga se usklađuje *WinBitmaps.tmp_zlista* – hashtablica koja sadrži z-raspored svih prozora na panoramskoj traci uključujući i one koji su izvan trenutačnog pogleda. Ovo je najsloženija operacija funkcije i mora zadovoljiti sljedeće uvjete:

- svaka promjena z-rasporeda na radnoj površini treba se odraziti i na promjenu z-rasporeda na panoramskoj traci,

- skrivanje i prikazivanje prozora koji izlaze izvan pogleda (za vrijeme promjene pogleda) ne smije uzrokovati promjenu u z-rasporedu na traci.

Da bi se ovaj problem riješio potrebno je prije generiranja nove z-liste sačuvati stare vrijednosti kako bi se one usporedile sa starim stanjem. Usporedba se radi na sljedeći način:

- prelaze se svi prozori A u novom z-rasporedu radne površine te se traži isti prozor u starom z-rasporedu, a ako se prozor ne nađe, dodaje se u privremenu listu,

- ako se prozor A nađe traži se da li postoji prozor B koji je sada ispod njega a bio je iznad. Ako postoji postavlja se oznaka da je z-položaj prozora A promijenjen i on se dodaje u privremenu listu,

- na kraju se prozori iz privremene liste premještaju na vrh z-liste vodeći računa da se normalni prozor ne stavi iznad prozora koji ima stil WS_EX_TOPMOST

Ovakva izvedba funkcije oslanja se na očekivano ponašanje operacijskog sustava koji prozore uvijek stavlja na vrh z-rasporeda ili na vrh svih prozora koji nemaju WS_EX_TOPMOST. Iako je takvo ponašanje neočekivano i nikada se ne događa u normalnom radu ipak je moguće promijeniti z-položaj prozora na bilo koje mjesto u z-rasporedu. Ovo se može izvesti, na primjer, korištenjem API funkcije *SetWindowPos*. U takvim slučajevima funkcija neće ispravno raditi.

3.3.2.2 Izrada vizualnih otisaka prozora

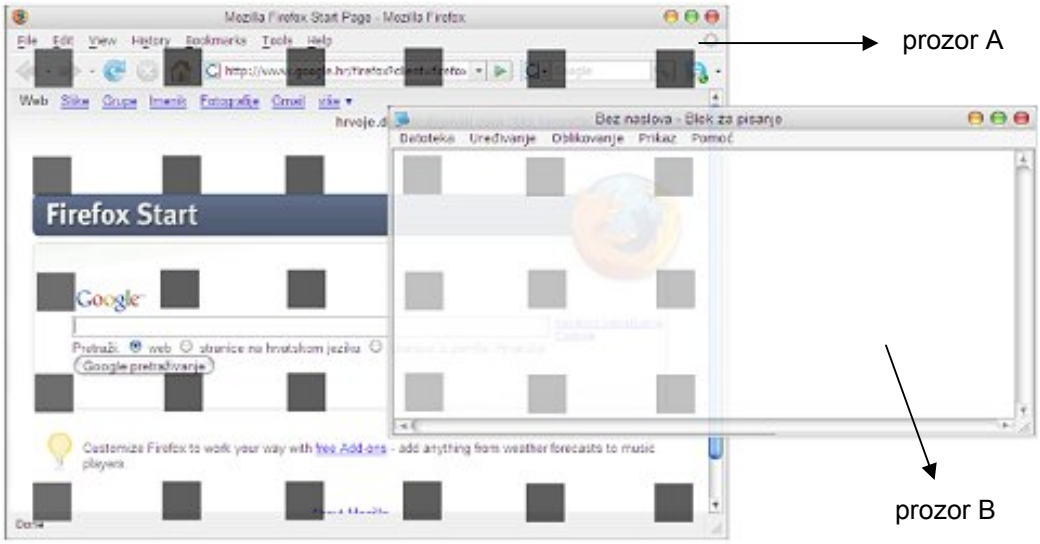
Budući da slike prozora treba uzimati kontinuirano zbog ograničenja API funkcije *PrintWindow* i sporih operacija nad slikama, potrebno je osmisliti način kako detektirati kada sliku treba uzeti. Provjera se mora vršiti periodički i ne smije značajno koristiti resurse računala. Najjednostavniji slučaj detekcije je promjena dimenzija prozora, to jest, minimizacija, maksimizacija ili promjena veličine. Kod svih ovih slučajeva treba uzeti novu sliku prozora, ali oni ne obuhvaćaju najčešći način promjene sadržaja prozora – promjenu zbog korisničkog unosa. Može se reći da korisnički unos u aplikacijama s prozorima gotovo uvijek uzrokuje promjenu sadržaja prozora, međutim, postoje i slučajevi kada to nije tako ili je promjena sadržaja zanemariva. Na primjer, pritiskom desne tipke miša na područje web-stranice na kojem ne postoji link neće se ništa promijeniti u izgledu prozora web pretraživača. Također postoje mnoge aplikacije koje se mijenjaju kontinuirano kao što su aplikacije za prikaz video sadržaja. Ovakav način detekcije je izvediv na način da se ubacivanjem dll-a u svaku aplikaciju filtriraju poruke korisničkog unosa. Drugi način koji koristi ubacivanje dll-a bi bio prisluškivanje API funkcija koje zove prozor i detekcija funkcija poput *BeginPaint* ili *UpdateLayeredWindow* koje obnavljaju vizualni sadržaj prozora. Međutim niti ovaj način nije dobar budući da se *BeginPaint* zove svaki put kada se otkrije područje prozora koje je prekrivao drugi prozor kao odgovor na poruku WM_PAINT koju šalje operacijski sustav, iako se sadržaj prozora nije promijenio. Uz to, rješenje pomoću ubacivanja dll-a i prisluškivanja prilično je složeno i moguće je da uzrokuje nestabilnost aplikacija koje se prisluškuju te reakcije antivirusnih programa. Treći i najbolji način je programsko "gledanje" prozora to jest detekcija promjene temeljena na operacijama nad slikom prozora, a ovaj se način koristi i u izvedbi programa. Osnovni problemi koje treba riješiti u ovakvoj izvedbi su:

1. potrebno je pronaći način da se detekcija obavi uz zanemarivo korištenje procesorskog vremena i memorije,

2. detekcija mora raditi pouzdano i za prozore koje preklapaju drugi prozori ili izlaze izvan ekrana,

3. poželjno je da zanemarive promjene u sadržaju prozora ne uzrokuju uzimanje nove slike.

Ovi se problemi rješavaju pomoću otisaka prozora. Otisak je dvodimenzionalno polje u kojem svaki element sadrži srednju vrijednost piksela određenog kvadratnog područja na prozoru. Prvi se problem rješava tako da se dohvaća samo relativno mali dio sadržaja slike i to korištenjem brze funkcije *BitBlt*. Treći problem može se riješiti tako da se promjena detektira samo kada se promijeni određeni broj elemenata otisaka (područja na prozoru). Drugi problem ne može se u potpunosti riješiti bez da se za dohvaćanje slike ne koristi funkcija koja daje i dio slike koji je prekriven. Budući da je jedina takva funkcija API funkcija *PrintWindow* koja je zbog grešaka i sporog izvođenja glavni uzrok problema u drugim dijelovima programa, nije ju moguće koristiti. Ipak je moguće naći zadovoljavajuće rješenje koristeći samo vidljive dijelove slike prozora tako da se za detekciju jednostavno zanemare promjene u dijelovima prozora koji nisu vidljivi.



Slika 3.3 Grafički prikaz otisaka prozora

Na slici 3.3 prikazan je primjer uzimanja otisaka prozora. Otisci se uzimaju za prozor A dok prozor B prekriva prozor A. Za svako područje označeno tamnim kvadratom uzima se srednja vrijednost piksela i sprema u polje otiska dok se na područjima označenim svijetlim kvadratima zanemaruju vrijednosti te se u polju označava da vrijednost nije pročitana. Kasnije se kod usporedbe vizualne razlike uspoređuju pročitane vrijednosti za svaki prozor sa vrijednostima spremljenim u *WinBitmap.otisak*. U ovoj varijabli pamte se pročitani otisci iz prošle iteracije modula za detekciju promjene, a vrijednosti koje nisu pročitane zbog prekrivanja popunjavaju se u modulu za uzimanje slike nakon uzimanja nove slike prozora. Kod za uzimanje otisaka nalazi se u funkciji *napravi_otiske,* a njena funkcionalnost opisuje se u nastavku.

Funkcija vraća objekt *otisci* u koji upisuje matricu srednjih vrijednosti piksela za pročitana kvadratna područja ili oznaku da područje nije bilo vidljivo. Sadrži petlju koja iterira po polju *zlista* tj. obavljaju se operacije nad svim prozorima u trenutačnom pogledu počevši od najviših u z-rasporedu prema nižima:

Prvo se za prozor pomoću funkcije *GetWindowDC* dohvaća kontekst uređaja koji omogućuje grafičke operacije nad prozorom i poslije će se koristiti za dohvaćanje kvadratnih područja njegove slike. Zatim se stvara novi kontekst uređaja kompatibilan s ovim kontekstom i pomoću njega kompatibilna slika za što se koriste funkcije *CreateCompatibleDC* i *CreateCompatibleBitmap*. Slika se stvara sa dimenzijama *sirina_otiska* i visina_otiska i služit će za kopiranje kvadratnih područja slike prozora.

Nakon ovoga se iz konfiguracije određuje korak iteracije za čitanje kvadratnih područja tj. razmak i broj područja. Uvijek se računa tako da kvadratna područja obuhvate cijeli prozor a ovisno o postavljenom broju smanjuje se ili povećava razmak između njih. Nakon što se odredi razmak pomoću petlji se prelaze koordinate svih područja na prozoru i ovisno o tome da li je područje vidljivo poziva se funkcija *BitBlt* koja kopira sličicu područja u pomoćnu sliku. U svrhu otkrivanja vidljivosti područja koristi se funkcija *podrucje_vidljivo* koja vraća da li je područje prekriveno drugim prozorom ili izlazi izvan ekrana.

Ako je područje vidljivo računa se srednja vrijednost piksela iz pomoćne slike koristeći funkciju *GetPixel*, a ako nije stavlja se oznaka *vidljiv* = *false*.

Budući da je API funkcija *GetPixel* vrlo spora i neefikasna ovaj dio postaje zamjetno spor ako se broj otisaka postavi na oko 200 po prozoru uz dimenzije 8x8. Ovo ne predstavlja problem budući da je sasvim zadovoljavajuće koristiti i manje vrijednosti, ali bi se umjesto ove funkcije mogla koristiti neka druga koja bi čitala memoriju sličice i radila srednju vrijednost.

3.4 Modul za uzimanje slika prozora

Ovaj modul se izvršava u istoj dretvi kao i modul za dohvaćanje stanja i detekciju promjena i poziva se iz njega kada se detektira promjena sadržaja prozora. Strukturu podataka nije potrebno opisivati jer se koriste isti podaci koji su već opisani u opisu prethodnog modula. Modul bi trebao obavljati jednostavnu funkciju: uzeti sliku prozora koristeći API funkciju *PrintWindow*, promijeniti veličinu slike za prikaz na panoramskoj traci i spremiti ju u objekt *TWinBitmap* sa ostalim podacima o prozoru. Međutim, zbog brojnih problema i grešaka funkcije *PrintWindow* ovu operaciju nije moguće jednostavno izvesti i potrebno je napisati kod za ispravak ovih grešaka. Također je nemoguće ostvariti potpunu funkcionalnost zbog toga što neke aplikacije kao što su Java, GTK ili OpenGL/DirectX ne podržavaju uzimanje slike pomoću ove funkcije. Još jedan problem u izvedbi ovoga modula je i problem konverzije Windows formata slike u OpenGL format te konverzije dimenzija slike na potenciju broja 2 kako bi se ona mogla prikazati pomoć OpenGL-a.

3.4.1 Problemi sa API funkcijom PrintWindow

API funkcija *PrintWindow* uvedena je prvi put u operacijskom sustavu Windows XP i omogućuje dohvaćanje slike prozora u proizvoljni kontekst uređaja. Dohvaćanje ne ovisi o tome da li je prozor prekriven drugim prozorom ili izlazi izvan ekrana. Međutim ova funkcija ima brojne greške na sustavu Windows XP i ne podržavaju ju sve aplikacije što je čini gotovo neupotrebljivom. U Windows Visti većina grešaka je riješena, ali kako program treba raditi na sustavu Windows XP potrebno je naći način kako riješiti ove probleme. Jedini drugi način uzimanja slike je izravno kopiranje piksela sa konteksta uređaja prozora koristeći funkciju *BitBlt*, ali ako se koristi ovaj način prozor ne smije biti prekriven jer se dobiva slika na ekranu a ne slika prozora. Problemi u funkciji *PrintWindow* i način njihovog rješavanja opisuje se u nastavku.

1. problem brzine: Vrijeme izvođenja funkcije vrlo je veliko i to, uz sporost operacije promjene veličine slike, čini nemogućim generiranje i prikazivanje slika na zahtjev već se to treba odvijati kontinuirano. U tablici 3.1 prikazana su izmjerena vremena uzastopnog izvođenja funkcije *PrintWindow* na prozoru rezolucije 1600x1200.

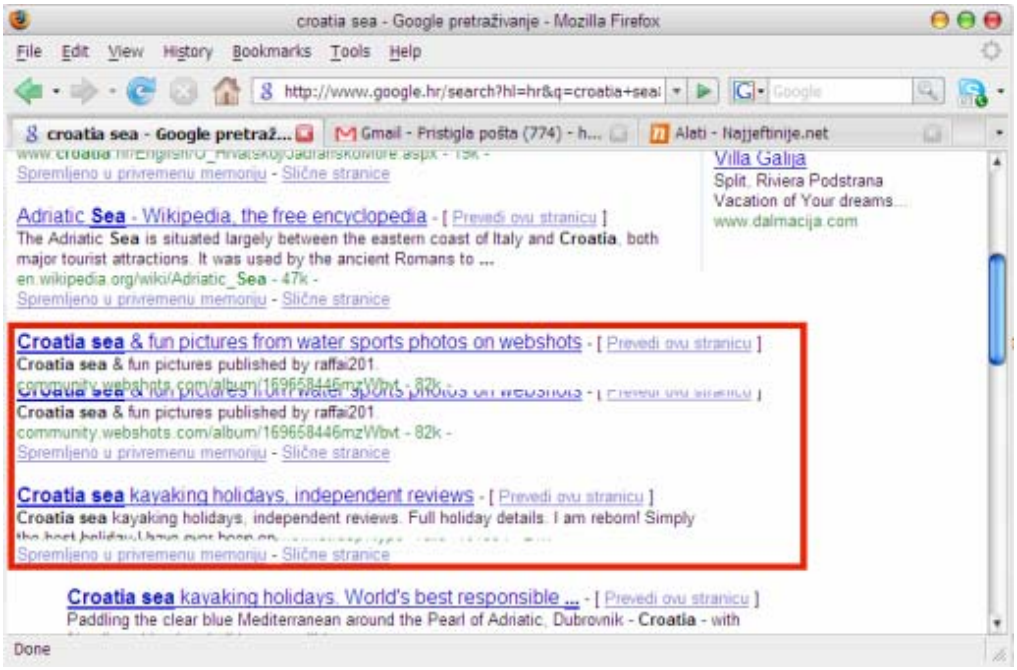
	1	2	3	4	5	6
vrijeme izvođenja (ms)	171	312	172	203	140	156

Tablica 3.1 *Mjerenje trajanja izvođenja API funkcije PrintWindow*

Što daje srednju vrijednost od 192 milisekunde i samo za 5 prozora može se očekivati zastoj od jedne sekunde za prikazivanje slika samo zbog ove funkcije, a kada se uzmu u obzir i operacije promjene veličine i konverzije vrijeme je i trostruko veće. Vrijeme izvođenja funkcije ne poboljšava se značajno na računalima sa boljim performansama.

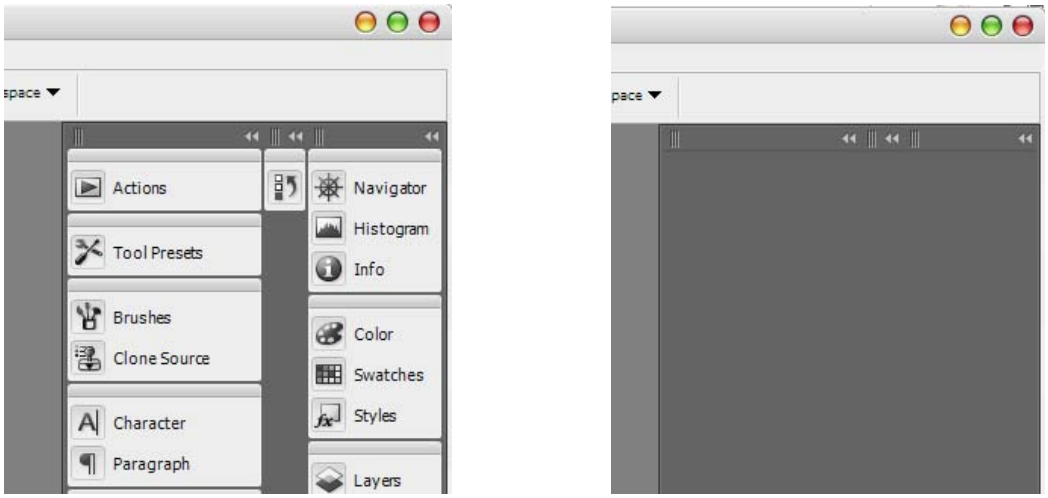
2. problemi pogrešnog obnavljanja prozora: Pozivanje funkcije *PrintWindow* može ponekad uzrokovati kvarenje sadržaja prozora nad kojim se poziva. Ovo je posebno neugodan problem pogotovo kada se funkcija poziva kontinuirano za vrijeme rada računala. Događa se u dva slučaja:

Uvijek kada se sadržaj prozora promijeni za vrijeme izvođenja funkcije: prozor se ne obnovi na ekranu nego na kontekstu uređaja prenesenom funkciji (slika 3.4). Ovo se događa jer funkcija za vrijeme izvođenja preusmjerava obnavljanje prozora na sliku u memoriji i ne šalje zahtjev da se prozor obnovi na ekranu. Ovaj problem može se riješiti jedino tako da se korištenjem funkcije *BitBlt* uzme slika ili dio slike prozora prije izvođenja *PrintWindow* i usporedi sa slikom nakon izvođenja. U slučaju da postoji promjena treba za prozor pozvati funkciju *RedrawWindow*.



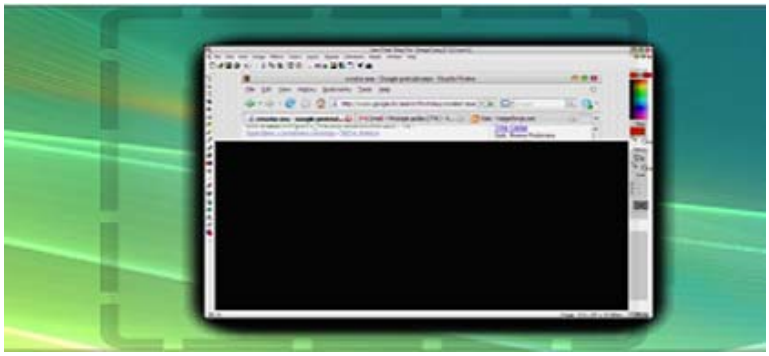
Slika 3.4 *Primjer djelomičnog obnavljanja prozora zbog greške u funkciji PrintWindow*

Kada se poziva na nekim uslojenim prozorima: Kako uslojeni prozori mogu koristiti dva načina za obnavljanje sadržaja, klasični pomoću poruke WM_PAINT i pomoću funkcije *UpdateLayeredWindow*, moguće je da koriste oba načina. Budući da funkcija *PrintWindow* šalje zahtjev za obnavljanjem porukom WM_PAINT prozor se može djelomično obnoviti na ekranu (slika 3.5). Kada se to dogodi nema načina da se popravi sadržaj prozora jer operacijski sustav ne podržava drugi način obnavljanja na zahtjev osim pomoću WM_PAINT.



Slika 3.5 *Uslojeni prozor prije i nakon pozivanja funkcije PrintWindow*

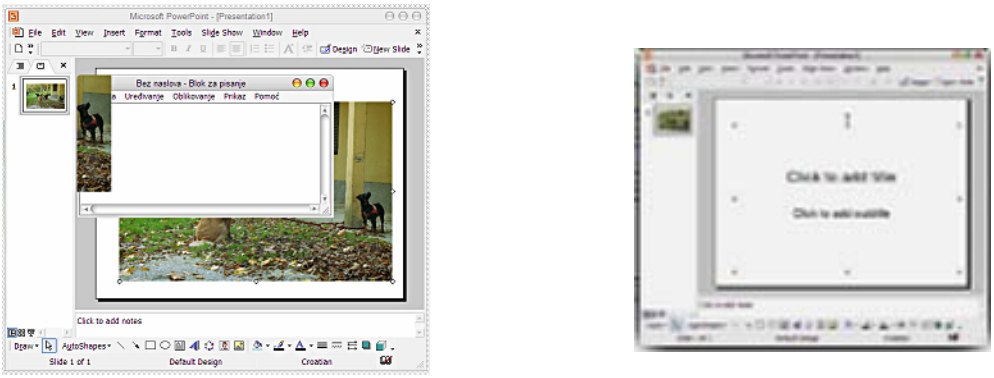
3. problem crnih područja u slici: Funkcija *PrintWindow* ponekad vraća crna područja u dobivenoj slici prozora vjerojatno zbog toga što se prozor ne stigne potpuno obnoviti prije nego što funkcija završi (slika 3.6). Ovaj problem moguće je u velikom broju slučajeva riješiti uzastopnim pozivanjem funkcije dva ili tri puta nakon detekcije crnih područja u dobivenoj slici ili detekcije razlike u slikama dobivenih funkcijom *BitBlt* i *PrintWindow*.



Slika 3.6 *Slika sa crnim područjima zbog greške u funkciji PrintWindow*

4. problem kvarenja sadržaja prozora sadržajem drugoga prozora: Neke aplikacije ne koriste standardni način iscrtavanja sadržaja na ekran tako da se pozivom funkcije *PrintWindow* sadržaj ne obnovi na memorijskom kontekstu nego na ekranu. Tada se događa da se slika ili neki dio slike prozora koji je prekriven nekim drugim prozorom "zalijepi" na prozor koji ga prekriva i ostane tako sve dok se taj dio prozora ne obnovi (slika 3.7). To se događa u Java i GTK aplikacijama i u nekim drugim aplikacijama kao što je na primjer PowerPoint XP. Ovaj problem kvarenja sadržaja može se riješiti prikazivanjem pomoćnih prozora sa pročitanim sadržajem ekrana na dijelovima koji su prekriveni za vrijeme uzimanja slike. Tako da se slika "zalijepi" na pomoćni prozor koji se poslije sakrije. Drugi mogući način je pozivanje funkcije *RedrawWindow* na prozoru na koji se posumnja da je pokvaren, ali ovaj način nije dovoljno dobar jer često dovodi do treperenja sadržaja prozora, a obnavljanje se može tražiti samo za cijeli prozor, ne za njegov dio. Unatoč rješenju problema kvarenja ostaje problem što je dobivena slika nepotpuna, a u slučaju Java i GTK aplikacija i potpuno bez sadržaja.

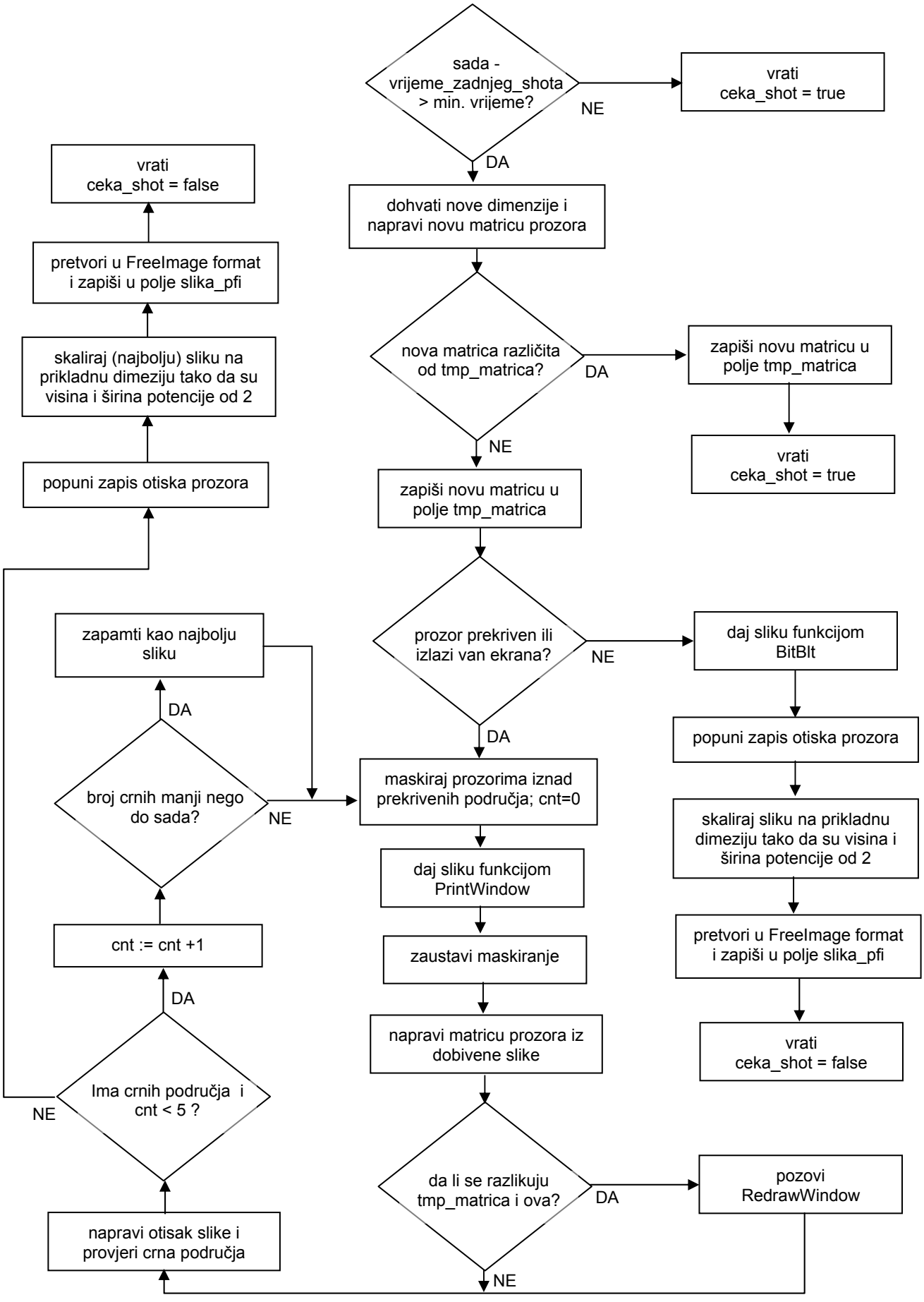
Od navedenih problema nije moguće riješiti problem sa aplikacijama koje ne podržavaju uzimanje slika pomoću WM_PAINT (uslojeni prozori) i aplikacijama koje ne daju sliku zbog nestandardnog načina crtanja sadržaja na ekran. Budući da se većina takvih aplikacija može detektirati pomoću parametara prozora kao što su ime klase (za Java i GTK aplikacije) one se mogu izuzeti iz uzimanja slike pomoću *PrintWindow*.



slika 3.7 *Lijevo – kvarenje sadržaja prozora zbog poziva PrintWindow, desno – slika dobivena funkcijom (bez fotografije u prezentaciji)*

3.4.2 Opis funkcionalnosti

U ovom poglavlju prvo je opisana osnovna funkcionalnost modula za uzimanje slika prozora dijagramom toka. Zatim se daje opis pojedinih modula iz dijagrama toka. Dijagram je prikazan na slici 3.8 a predstavlja djelomično rješenje uzimanja slika problematičnom funkcijom *PrintWindow*. U potpunosti se rješava problem pogrešnog obnavljanja prozora koristeći matricu prozora. Kao što je opisano u strukturi podataka, ovo polje sadrži crno bijelu sliku prozora u kojoj se svaki piksel svodi na jedan bajt, a sprema se u matricu. Moguće je koristiti i reprezentaciju u kojoj se uzima svaki drugi ili svaki n-ti piksel, ali je bitno da sadrži što više informacije jer služi za detekciju kvarenja prozora (i najmanjih razlika). Naravno, ovo je samo privremeno rješenje jer bi bilo puno efikasnije pamtiti hash tablicu slike a ne sliku. Polje *tmp_matrica* koje sadrži matricu sprema se u podatke prozora u svakom ciklusu uzimanja slike. Prethodno se matrica (iz prošlog ciklusa) uspoređuje sa novom matricom kako bi se utvrdilo da li ima razlike u izgledu prozora u odnosu na prije. Ukoliko ima, prekida se uzimanje slike te se odgađa na sljedeći ciklus. Ovakvo rješenje osigurava veću vjerojatnost da se sadržaj prozora neće promijeniti za vrijeme funkcije *PrintWindow* pa se tako smanjuje vjerojatnost kvarenja sadržaja. Međutim, na ovaj način onemogućava se obnavljanje slike prozora koji se kontinuirano mijenja, pa je potrebno ili isključiti ovu provjeru ili uvesti ograničenje na maksimalni broj odustajanja od uzimanja slike. Drugi način na koji se rješava ovaj problem je izrada matrice iz slike dobivene funkcijom *PrintWindow* te pozivanje funkcije *RedrawWindow* ako ima razlike između te matrice i polja *tmp_matrica*. Ovaj način je gotovo 100% efikasan ali može izazvati pojavu kratkotrajnog treperenja u prozoru koji se obnavlja. Algoritam također rješava i problem kvarenja sadržaja prozora sadržajem drugoga prozora tako da se računaju presjeci prozora za kojeg se uzima slika sa prozorima iznad njega te se za vrijeme poziva *PrintWindow* ta područja maskiraju. Maskiranje se vrši tako da se na tim mjestima privremeno prikažu pomoćni prozori u slučaju detekcije promjene slike. Tako se pokvareni sadržaj sakrije, a nakon što se pomoćni prozor skloni, operacijski sustav automatski obnavlja sadržaj aplikacije ispod. Problem crnih područja djelomično je riješen njihovom detekcijom i uzastopnim uzimanjem slike.

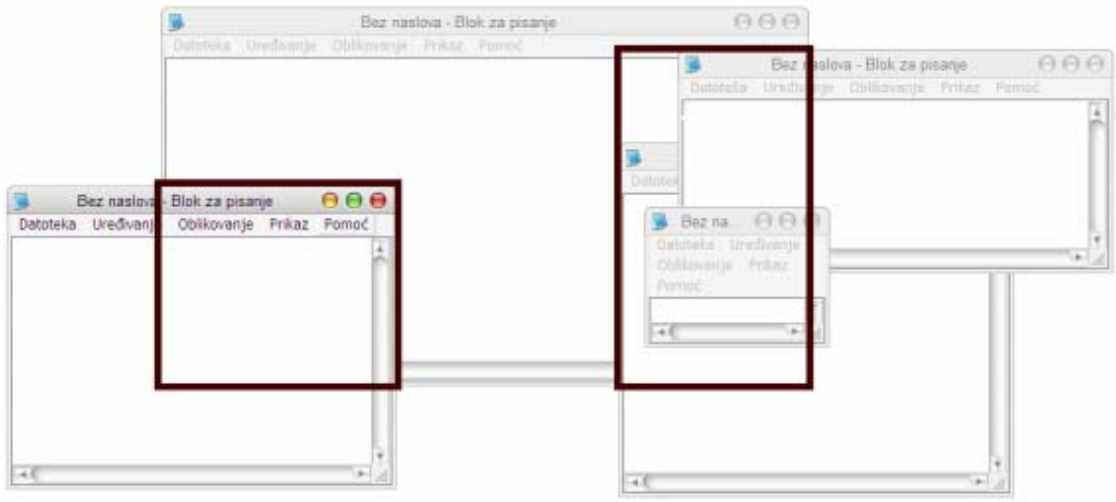


Slika 3.8 *Dijagram toka modula za uzimanje slika prozora*

Osnovni problem ovakve izvedbe jesu veliki zahtjevi na procesorsko vrijeme budući da se izvodi više operacija sa slikama. Ali kada se uzme u obzir da operacije koje se moraju izvesti, kao što je skaliranje, zahtijevaju veće procesorsko vrijeme kao i to da nije potrebno uzimanje slika u realnom vremenu, ovakvo rješenje je prihvatljivo. Moguće bi bilo i dinamički mijenjati prioritet dretve ovisno o opterećenju sustava uz uvođenje maksimalnog vremena trajanja operacije.

3.4.2.1 Detekcija i maskiranje prekrivenih područja

Kako bi se riješio problem kvarenja sadržaja drugih prozora za vrijeme poziva funkcije *PrintWindow* nad nekim prozorom A, potrebno je za vrijeme poziva prekriti područja prozora A koja prekrivaju drugi prozori, ali na način neprimjetan korisniku (slika 3.9). Ovo se mora napraviti za svaki poziv funkcije *PrintWindow* jer nema načina detekcije koji prozori i pod kojim uvjetima mogu uzrokovati kvarenje.



Slika 3.9 *Područja koja treba maskirati kod izvođenja PrintWindow na srednjem prozoru*

Funkcija prvo računa presjeke pravokutnika na sljedeći način: U polju *zlista* traži se prozor A, pa se prelaze svi prozori iznad i računaju se presjeci koji se spremaju u polje *presjeci*. Nakon toga se prelazi polje i uspoređuje se svaki presjek *p1* sa svakim drugim *p2* te otkriva da li prekriva ili obuhvaća *p1*. Ako prekriva, *p1* se zaboravlja, a ako obuhvaća *p1* se skraćuje sa te strane tako da se *p1* i *p2* ne sijeku. Na kraju usporedbe sa svim *p2*, presjek *p1* se pamti. Tako dobiveno polje zapamćenih presjeka koristi se kasnije za određivanje koordinata pomoćnih prozora za maskiranje. Broj presjeka može se ograničiti i odustati od uzimanja slike ako je njihov broj prevelik, ali u normalnim situacijama to se neće dogoditi.

Da bi se područje prekrilo prozorom za svaki presjek iz konačnog polja zove se funkcija *maskiraj_podrucje*. Ova funkcija prvo kopira sliku *s1* kvadratnog područja na ekranu zadanog koordinatama presjeka:

1. dohvaća se kontekst uređaja ekrana pozivom *GetDC(0),*

2. stvara se kompatibilni kontekst pozivom *CreateCompatibleDC (kontekst_ ekrana),*

3. stvara se kompatibilna slika (HBITMAP) dimenzija presjeka pozivom *Create CompatibleBitmap (kontekst_ekrana)* pa se slika selektira u u stvoreni kontekst pozivom *SelectObject(kompat_ kontekst, slika),*

4. Sada se funkcijom *BitBlt* kopira slika iz konteksta ekrana u stvoreni kontekst (i stvorenu sliku).

Nakon toga se periodički uzima slika istog područja *s2* i uspoređuje sa početnom i ako se detektira razlika na području se prikazuje i periodički obnavlja prozor koji sadrži sliku *s1*. Prozor se stvara koristeći funkciju *CreateWindowEx* sa postavljenom zastavicom WS_EX_TOOLWINDOW i praznim nizom naslova kako ne bi imao traku naslova, zastavicom WS_EX_TOPMOST da bi bio prvi u z-rasporedu i WS_EX_NOACTIVATE da se ne bi mogao aktivirati i deaktivirati drugu aplikaciju.

Nakon završenog poziva svi se pomoćni prozori uklanjaju, a operacijski sustav automatski aplikaciji šalje poruku da obnovi sadržaj područja koje je bilo prekriveno. Ovo je jedini način da se obnovi proizvoljno područje aplikacije iako operacijski sustav ima funkcije kao *RedrawWindow*, *UpdateWindow* i *InvalidateRect* koje naizgled ovo omogućavaju. Funkcija *InvalidateRect* je ograničena samo na klijentsko područje prozora, a kvarenje sadržaja može se dogoditi i na ne-klijentskom području (na traci naslova i alatnim trakama). Funkcijom *RedrawWindow* može se postići obnavljanje cijeloga prozora, ali ovo nije prihvatljivo jer često uzrokuje primjetno treperenje.

3.4.2.2 Detekcija crnih područja

Način na koji se detektiraju crna područja opisan je u dijagramu na slici 3.7; u slučaju detekcije ponavlja se dohvaćanje slike, pa se opet provjerava da li ima crnih područja u njoj. Broj ponavljanja ograničen je konstantom. Sama detekcija izvodi se pomoću otisaka prozora. Prvo se izrade otisci slike na jednak način na koji se to radi u modulu za detekciju promjena. Jedina razlika je da se ne radi kopiranje bitova iz konteksta uređaja ekrana ili prozora, nego se stvara novi kontekst kompatibilan sa kontekstom ekrana koristeći API funkciju *CreateCompatibleDC* i u njega se selektira slika koristeći funkciju *SelectObject*. Nakon toga se vrši kopiranje slike iz tog konteksta pomoću *BitBlt* funkcije. Funkcija detektira pokvarenu sliku ako je srednja vrijednost nekoliko kvadratnih područja slike jednaka nula. Ovaj način može uzrokovati pogrešnu detekciju jer je normalno da prozor može imati crno područje na nekom dijelu. U tom slučaju nepotrebno se ponavlja uzimanje slike nekoliko puta. Ovo se može poboljšati tako da se detekcija

crnih područja vrši zajedno sa detekcijom kvarenja sadržaja pomoću matrice prozora. Na ovaj se način ne bi mogle detektirati crne slike prozora koji su potpuno prekriveni ali budući da se i tako ne detektira promjena sadržaja prekrivenog prozora i promjene dimenzija takvih prozora se vrlo rijetko događaju, za njih se ne bi gotovo nikada niti uzimale slike.

3.4.2.3 Popunjavanje zapisa otiska, promjena dimenzija i formata slike

Kao što je već navedeno u opisu dijela za detekciju vizualne razlike modula za dohvaćanje stanja detekciju promjene otisak prozora popunjava se u modulu za uzimanje slike prozora. Kod generiranja otiska slike (skupa srednjih vrijednosti nekih kvadratnih područja) zanemaruju se dijelovi prozora koji nisu vidljivi i označava se u zapisu otiska da je na ovim dijelovima vrijednost nepoznata. Ako se u sljedećem ciklusu modula za dohvaćanje stanja i detekciju promjene dogodi da ova područja postanu vidljiva, nema načina da se vrijednosti usporede, pa tako niti da se detektira eventualna promjena sadržaja na tim područjima. Zbog toga se ove vrijednosti kopiraju iz otiska dobivene slike u ovome modulu.

Budući da se slike prozora trebaju prikazati u OpenGLu, potrebno je izvršiti i konverziju slike u kompatibilni format. Početno je dobivena slika u Windows formatu HBITMAP koji nije kompatibilan sa OpenGLom a također i same dimenzije slike predstavljaju problem za OpenGL prikaz budući da se podržavaju samo one slike kojima je visina i širina potencija broja dva. Za sve konverzije koristi se biblioteka FreeImage, a u svrhu svođenja na potenciju broja dva napisana je funkcija koja nalazi najbolju potenciju broja dva za zadanu dimenziju slike. U funkciji se može definirati konstanta koja određuje koliko dimenzija mora biti blizu nižoj potenciji broja u odnosu na višu da se na nju svede. Ovo se radi u svrhu izbjegavanja prevelike promjene dimenzija na više za vrijeme prikaza. Još jedan problem predstavlja i ograničenje veličine slike u OpenGLu, a rješava se na način da se dohvati OpenGL konstanta GL_MAX_TEXTURE_SIZE i ne dozvoljavaju se veće dimenzije. Za operaciju skaliranja slike može se koristiti funkcija iz FreeImage biblioteke ili API funkcija *SetWorldTransform*. Obje funkcije podržajavaju bilinearni filter, a mjerenjem je utvrđeno da je API funkcija *SetWorldTransform* brža, pa se koristi ona.

3.5 Modul za detekciju povlačenja prozora

Jedna od mogućnosti koju program treba omogućiti je i izravno povlačenje prozora sa radne površine na panoramsku traku i zbog toga je potrebno pronaći način za detekciju povlačenja prozora. Ovu funkciju obavlja modul za detekciju povlačenja prozora. Modul se izvodi u posebnoj dretvi koja je aktivna stalno za vrijeme rada programa i na poziv funkcije *daj_prozore_koji_se vuku* vraća listu prozora koji se povlače. Ova lista se kasnije koristi u modulu za prikaz. Logično se čini da bi funkcija trebala vratiti jedan prozor jer se samo jedan prozor može odjednom pomicati mišem, međutim to nije istina. Postoje aplikacije kao što je na primjer

Nullsoft Winamp koje su sastavljene od više prozora te se povlačenjem jednog od njih programski pomiču ostali. Jedno moguće rješenje detekcije povlačenja moglo bi biti ubacivanje dll-a u sve aplikacije i prisluškivanje svih poruka bitnih za povlačenje prozora. Ovim načinom bi bilo kompliciranо riješiti problem detekcije programskog povlačenja a i ubacivanje dll-a u sve aplikacije nije preporučljivo jer bi to mogli detektirati neki antivirusni programi. Način koji se koristi u ovom modulu koristi samo API pozive a rješava i problem detekcije programskog povlačenja.

3.5.1 Opis strukture podataka

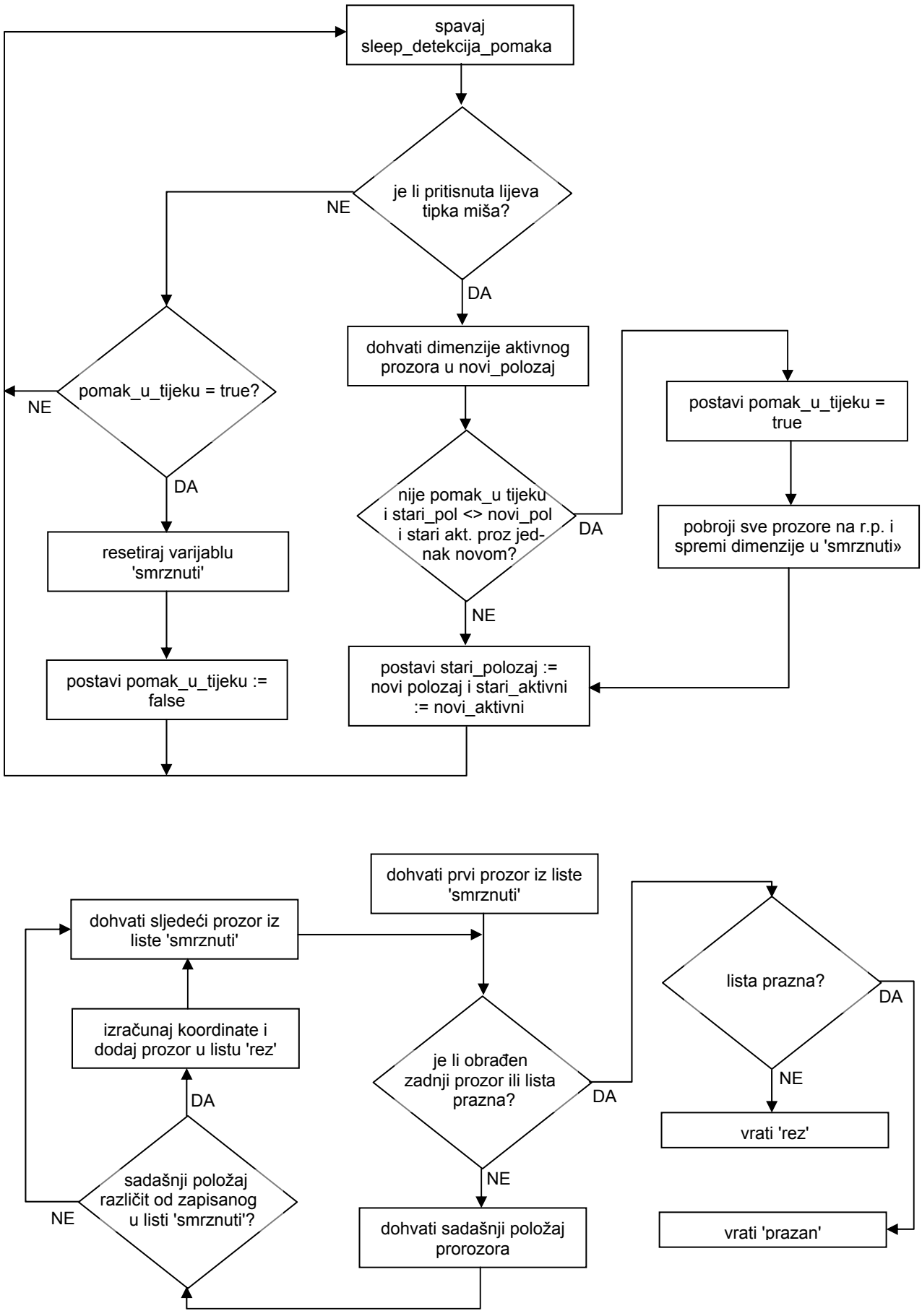
Podaci bitni za rad ovoga modula su dvije varijable: varijabla *pomak_u_tijeku* tipa boolean koja se postavlja na *true* kada se detektira početak pomaka nekog prozora na radnoj površini, a na *false* kada se detektira prestanak pomaka. Druga varijabla je objekt tipa *TSviSmrznuti* koji je zapravo dinamička lista. U varijablu se sprema stanje svih prozora na radnoj površini u trenutku kada se detektira pomak nekog prozora. Stanje se sprema u objekt čiji je razred deklariran sa:

```
TSmrznuti = class
  handle : HWND;
  x, y : integer;
end;
```

Gdje je *handle* upravljačka varijabla prozora, a *x* i *y* njegova x odnosno y koordinata.

3.5.2 Opis funkcionalnosti

Modul obavlja detekciju na način da periodički provjerava položaj trenutačno aktivnog prozora i to da li je pritisnuta lijeva tipka miša. Kada se detektira pomak, pobrojavaju se svi prozori koristeći funkciju *EnumWindows* i dohvaća se njihov položaj koristeći API funkciju *GetWindowRect*. Zatim se svi prozori spremaju u varijablu *smrznuti* tipa *TSviSmrznuti*. Kada se zatraže pomaknuti prozori funkcija *daj_pomaknute* ponovno dohvaća sve prozore te uspoređuje nove položaje sa zapisima u varijabli *smrznuti*. Ako je postavljeno *pomak_u tijeku*, funkcija vraća listu svih prozora za koje se položaji razlikuju od zapisanih, u listi nizova u formatu *handle:x_relativno_mišu, y_relativno_mišu*. Opis funkcionalnosti dretve za detekciju i funkcije za dohvaćanje dan je dijagramima toka na slici 3.10.



Slika 3.10 *Dijagrami toka dretve za detekciju (gore) i funkcije za dohvaćanje (dolje)*

3.6 Modul za upravljanje radnom površinom

Modul za upravljanje radnom površinom je skupina funkcija koje se pozivaju iz modula za prikaz a služe za postavljanje prikaza na radnoj površini prema izabranom pogledu na panoramskoj traci i za postavljanje pojedinih prozora kada se njihove sličice povlače po panoramskoj traci. Ove funkcije moraju biti sinkronizirane sa modulom za dohvaćanje stanja, to jest, ne smije se dozvoliti istovremena detekcija stanja i postavljanje prozora. Ovo se ostvaruje koristeći kritične odsječke, ali tako da se pozivi *EnumWindows* u njima ne nalaze jer se u protivnom događa potpuni zastoj. Dretva u kojoj se izvode funkcije ovoga modula je dretva modula za prikaz. Iz funkcije za postavljanje pogleda također je potrebno pozvati i funkciju za dohvaćanje stanja radne površine koja se normalno izvodi u modulu za dohvaćanje stanja te sinkronizirati pozive. Ovaj poziv je potreban jer se neposredno prije promjene pogleda treba uzeti trenutačno stanje kako novi prozori koji su otvoreni od zadnje periodičke detekcije ne bi bili izgubljeni.

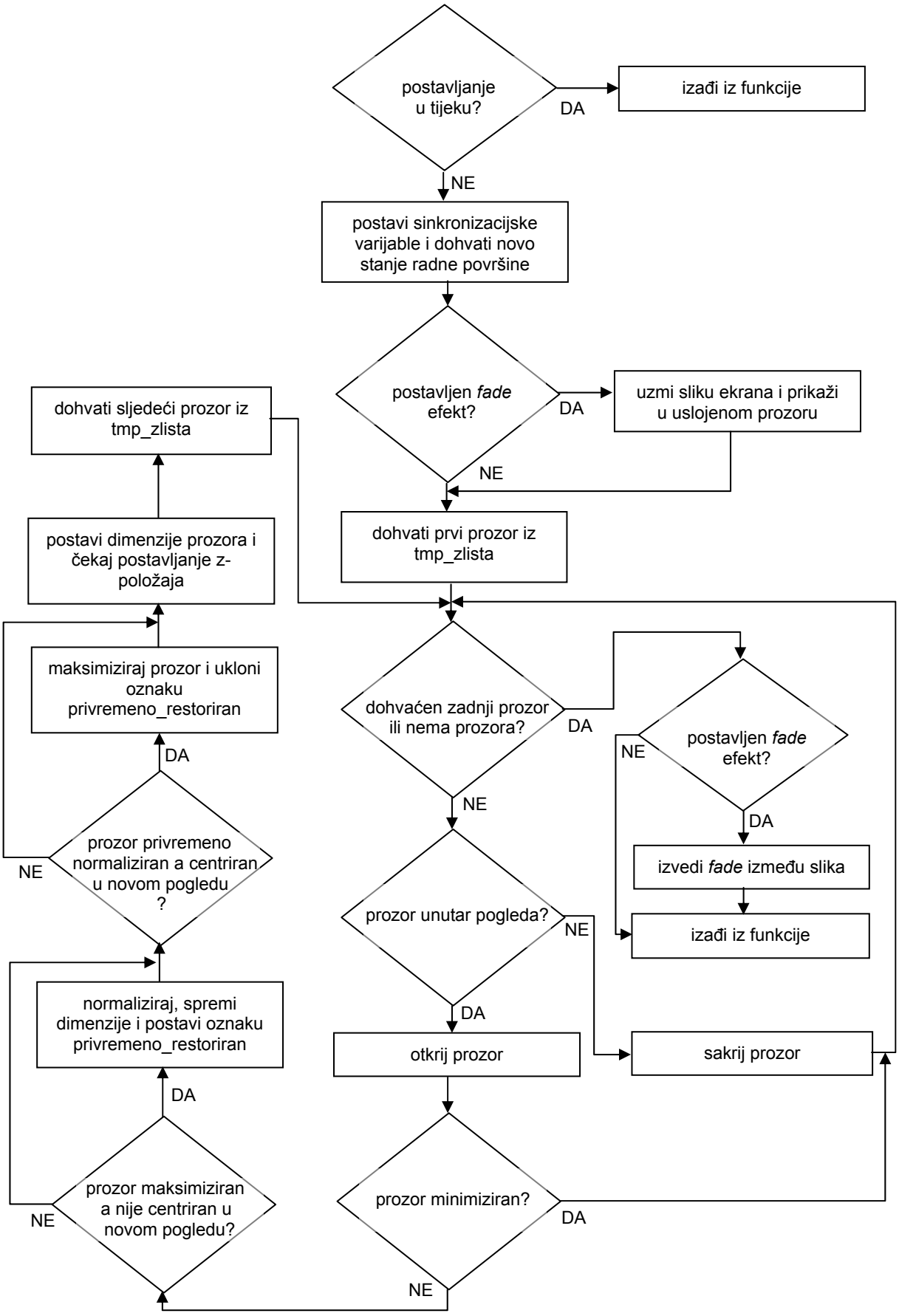
3.6.1 Funkcija za postavljanje radne površine

Funkcija *postavi_desk_panorama* modula za upravljanje radnom površinom poziva se kada je potrebno postaviti radnu površinu na novi pogled koji je izabran na panoramskoj traci. Funkcija kao ulazne parametre uzima x koordinatu izabranog pogleda, širinu ekrana *w*, polje *tmp_zlista* koje sadrži z-raspored svih prozora na panoramskoj traci i listu svih parametara svih prozora tipa *TWinBitmaps*. Funkcija je opisana dijagramom toka na slici 3.11.

Na početku funkcije provjerava se da li je postavljanje već u tijeku, a ako je funkcija se ne izvodi. Ovo je potrebno kako se ne bi dozvolilo prekidanje funkcije istom funkcijom u slučaju postavljanja vođenog događajima. Zatim se poziva funkcija *dohvati_stanja* modula za dohvaćanje stanja i detekciju promjena ali tako da se izvrši sinkronizacija između dva modula. Ovaj problem još nije potpuno riješen jer je sinkronizaciju potrebno provesti na više mjesta u modulu za dohvaćanje ali na način da se ne dogodi primjetan zastoj u radu modula za postavljanje radne površine a s njime i modula za prikaz.

Sljedeće što funkcija radi je uzimanje slike cijele radne površine u slučaju da je postavljen *fade* efekt. Ovim se efektom postiže ljepši prijelaz kod promjene pogleda i sadržaja radne površine. Slika se uzima funkcijom *BitBlt* i sprema se u varijablu tipa HBITMAP na način koji je već opisan u opisu modula za uzimanje slike. Jedina razlika je što se ne dohvaća kontekst pojedinih prozora nego kontekst prozora radne površine čija se upravljačka varijabla dobiva API funkcijom *GetDesktopWindow*. Ova slika se odmah prikazuje u uslojenom prozoru koji se prikazuje iznad svih ostalih.

Nakon toga izvršava se petlja za postavljanje novoga sadržaja radne površine prema izabranom pogledu na način da se prozori koji nisu u pogledu sakriju



Slika 3.11 *Dijagrami toka funkcije za postavljanje radne površine*

pozivom funkcije *ShowWindow* sa zastavicom SW_HIDE, a oni koji jesu se prikažu pozivom iste funkcije sa zastavicom SW_SHOW. Rezultat ovoga je da će se na traci sa zadacima nalaziti samo oni prozori koji su u izabranome pogledu. Ako je neki prozor u minimiziranome stanju on se samo prikazuje, odnosno skriva, a petlja se nastavlja za sljedeći prozor. Ostali prozori postavljaju se na određeni z-položaj određen redoslijedom u listi *tmp_zlista* i dimenzije zapisane u *WinBitmaps*. Za ovo se koristi funkcija *SetWindowPos* pomoću koje se mogu postaviti i dimenzije i z-položaj prozora u jednom pozivu. Za postavljanje z-položaja koristi se drugi parametar *InsertAfter* koji se postavlja na upravljačku varijablu prethodnog prozora odnosno na HWND_TOP ili HWND_TOPMOST za prvi prozor ovisno o tome da li prozor ima postavljenu zastavicu WS_EX_TOPMOST i nalazi se uvijek iznad "običnih" prozora. Problem razdvajanja ove dvije vrste prozora rješava se u funkciji za postavljanje tako da se održavaju dvije liste za postavljanje i ne dozvoljava se postavljanje prozora koji je "uvijek iznad" ispod onoga koji nije i obratno. Drugi problem kod postavljanja prozora u z-raspored proizlazi iz činjenice da je postavljanje prozora asinkrono i prozor se ne stigne postaviti prije poziva postavljanja za sljedeći prozor. Zato se nakon postavljanja čeka postavljanje pozivima *GetNextWindow* sa vremenskim ograničenjem kako bi se u slučaju greške izbjegla beskonačna petlja i zastoj programa. Nakon što završi postavljanje prozora u slučaju da je postavljen, izvodi se *fade* efekt. Ovaj efekt omogućuju uslojeni prozori koji podržavaju prozirnost, a izvodi se na sljedeći način:

Prije postavljanja novoga stanja stvoren je novi uslojeni prozor funkcijom *CreateWindowEx* sa parametrima WS_EX_LAYERED i WS_EX_TRANSPARENT i uzeta je slika ekrana u varijablu tipa HBITMAP. Kada završi postavljanje prozoru se mijenja prozirnost Koristeći funkciju *UpdateLayeredWindow* tako da se dobije efekt polaganoga pojavljivanja novoga stanja. Problem ovog načina je što se ne koristi sklopovsko ubrzanje, pa iščezavanje slike preko cijelog ekrana koristi dosta procesorskog vremena te je brzina izvođenja ovisna o brzini procesora. Zbog toga se broj koraka petlje za iščezavanje i pauza u njoj određuje dinamički mjereći vrijeme tako da iščezavanje uvijek traje približno jednako na svim računalima.

3.6.2 Funkcija za postavljanje prozora

Na panoramskoj traci moguće je povlačiti prozore kao i na radnoj površini. Kada se neki prozor povlači unutar trenutačnoga pogleda, pomak se mora odraziti i na pomicanje pravog prozora na radnoj površini. Kako bi se ovo ostvarilo, koristi se funkcija za postavljanje prozora. Funkcija se poziva iz modula za prikaz kada se detektira povlačenje nekog prozora na traci a prenosi joj se indeks u listi *WinList* prozora koji se povlači i varijabla tipa T*WinBitmaps* sa svim podacima o stanju svih prozora. Funkcija obavlja relativno jednostavan zadatak: prvo određuje da li je prozor "iznad svih" tako da provjerava vrijednost polja *OnTop* i ovisno o tome postavlja prozor na vrh z-rasporeda ili iznad svih "normalnih" prozora pozivom funkcije *SetWindowPos*. U istome pozivu postavlja se i položaj prozora prema zapisanim podacima.

3.7 Modul za prikaz

Ovaj modul prikazuje panoramsku traku sa sličicama prozora te omogućuje promjenu razmještaja prozora na njoj te promjenu pogleda. Za prikaz se koristi OpenGL kako bi se omogućile brze operacije sa slikama i ostavila mogućnost korištenja animacija. Budući da se radi o OpenGL prikazu, prikazivanje se obavlja periodičkim iscrtavanjem nekoliko desetaka puta u sekundi, a za sada se za iscrtavanje koristi tajmer koji okida funkciju za crtanje. Modul se izvršava u posebnoj dretvi i poziva modul za postavljanje radne površine za vrijeme pomicanja prozora ili promjene pogleda. Budući da cijela funkcionalnost ovoga modula nije zanimljiva sa stanovišta upravljanja prozorima on se samo ukratko opisuje.

3.7.1 Opis strukture podataka

Za prikaz sličica prozora i zapisivanje promjena koristi se struktura tipa *TWinBitmaps* koja je već opisana u modulu za dohvaćanje stanja a uz nju koriste se i ostale strukture, a važnije među njima opisuju se u nastavku.

Razred *TPanKontrole* sadrži OpenGL teksture i podatke o položaju i dimenzijama kontrola na panoramskoj traci. Instanciranje i pridruživanje vrijednosti obavlja se u fazi inicijalizacije kod pokretanja programa, a objekt se instancira u globalnu varijablu *pan_kontrole*. Razred *TPomaknutiProz* služi za spremanje podataka o prozorima koji se izravno povlače sa radne površine na traku. Jedna instanca služi za spremanje podataka o jednom prozoru za vrijeme povlačenja sa radne površine, a budući da je moguće i povlačenje i više prozora odjednom, za spremanje podataka koristi se lista. Objekt se instancira u globalnu varijablu *indeksi_pomaknutih*. Većina podataka sprema se u globalne varijable, a uz varijable koje sadrže instance opisanih razreda one važnije opisuju se u nastavku.

- Varijabla *polozaj_panorame* – cijeli broj koji označava položaj pogleda na traci. Kod promjene pogleda u ovu se varijablu zapisuje vrijednost koja označava pomak u broju piksela u odnosu na početni pogled. Zapisuje se broj piksela u odnosu na dimenzije radne površine, a ne u odnosu na dimenzije na traci. Ova se varijabla koristi i u modulu za dohvaćanje stanja radne površine za određivanje početnog položaja novih prozora.

- Varijable *pan_scaling_faktor i pan_1_kroz_scaling* – realni brojevi koji označavaju omjer veličine prikaza na radnoj površini i prikaza na traci. Konverzija se obavlja za određivanje položaja i veličine sličica na traci jer se u strukturi *WinBitmaps* koriste stvarne dimenzije na radnoj površini. Obrnuta konverzija obavlja se kod pomicanja prozora po traci kada je potrebno zapisati promjenu u strukturu *WinBitmaps*.

- Varijable *mx, my, mis_usao* itd. – prenose stanje i promjene miša a koriste se u raznim operacijama kao što su izbor kontrola, povlačenje prozora i promjena pogleda.

- Varijable *_visina, _sirina* – varijable u koje se sprema visina odnosno širina radne površine u pikselima.

- Varijabla *panorama_aktivna* – varijabla koja označava da li je traka prikazana ili nije. Ovisno o vrijednosti ove varijable pokreće se ili zaustavlja OpenGL prikaz.

- Varijabla *_mod* – označava da li je izabran način promjene pogleda ili način razmještaja prozora.

3.7.2 Opis funkcionalnosti

Funkcionalnost modula može se podijeliti u dvije faze:

1. faza inicijalizacije,

2. faza provedbe.

U fazi inicijalizacije prvo se iz posebne dretve koja sadrži petlju poruka poziva funkcija *glStvoriProzor* kojoj se prenose dimenzije i položaj na kojem se treba prikazati panoramska traka. Funkcija stvara prozor bez neklijentskog područja koristeći *CreateWindowEx* u skrivenom stanju te inicijalizira OpenGL kontekst. Sljedeća se poziva funkcija *InicijalizirajPanoramu*. Ova funkcija inicijalizira sve varijable na početne vrijednosti uključujući i postavljanje početnog načina rada na način razmještaja. Koristeći podatak o dimenzijama trake računaju se i faktori skaliranja te se upisuju u varijable *pan_scaling_faktor* i *pan_1_kroz_scaling.* Na kraju se poziva funkcija *vezi_proz_teksture* kojom se učitavaju početne slike prozora iz *WinBitmaps* te se generiraju OpenGL teksture kako bi se kasnije mogle prikazati sličice prozora. Kasnije se u fazi provedbe kontinuirano ispituju promjene slika prozora i ako je do promjene došlo, stara se tekstura oslobađa te se generira nova. Druga funkcija koja se poziva je *ucitaj_pan_kontrole* koja iz konfiguracije čita i postavlja položaj i dimenzije kontrola na traci te iz datoteka učitava slike koje se koriste za prikaz kontrola. Ovi se podaci zapisuju u objekt razreda *TPanKontrole*.

Faza provedbe izvodi se kontinuirano nakon pozivanja funkcije *glPrikaziProzor* koja prvo inicijalizira tajmer i nakon toga prikazuje traku pozivom *Show Window(handle, SW_SHOW).* Nakon inicijalizacije tajmer periodički proceduri prozora šalje poruku WM_TIMER te se iz nje poziva funkcija *glCrtaj* koja sadrži sve pozive potrebne za prikaz i upravljanje prikazom na panoramskoj traci:

1. ispituje se stanje miša i u varijable prenose oznake naredbi,

2. poziva se funkcija za crtanje i ispitivanje kontrola,

3. poziva se funkcija za vezivanje novih tekstura za sličice prozora,

4. poziva se funkcija za crtanje prikaza radne površine.

Podaci o stanju miša prenose se preko globalnih varijabli u sljedeće korake. Iz ovih se podataka u funkciji za crtanje i ispitivanje kontrola *gl_CrtajPanSkin* postavljaju način rada *_mod* na MOD_RAZMJESTAJ ili MOD_POGLED i ostale naredbe kao što je pomak trake lijevo ili desno koje se kasnije izvode u funkciji za crtanje prikaza radne površine. Funkcija za vezanje novih tekstura (*vezi_proz_ teksture)* ispituje parametar *tekstura2_vezana* i ovisno o njegovoj vrijednosti veže novu OpenGL teksturu iz polja *slika_pfi2* objekta *WinBitmaps* koji sadrži skaliranu sličicu prozora. Parametar *tekstura2_vezana* zatim se postavlja na *true*. Kada se u modulu za dohvaćanje stanja i detekciju promjena detektira promjena sadržaja prozora ovaj se parametar nakon uzimanja nove slike postavlja na *false*.

Glavni dio funkcionalnosti modula za prikaz nalazi se u funkciji *crtajPanoramu* koja služi za crtanje prikaza radne površine. Kako bi se prikazalo stanje radne površine, u petlji se prelazi polje *tmp_zlista* objekta *WinBitmaps* te se redom prikazuju pripadne teksture prozora. Lista sadrži indekse zapisa prozora poredane po z-rasporedu i crtanjem od najnižeg prema najvišem postiže se prikaz koji odgovara zapisanom z-rasporedu. U funkciji se također ovisno o postavljenom načinu rada i naredbama miša pozivaju funkcije za postavljanje radne površine i postavljanje prozora modula za upravljanje radnom površinom. Kada se u funkciji detektira pritisak miša nad sličicom prozora izravno se mijenja polje *tmp_zlista* kako bi se sličica prozora postavila na vrh z-rasporeda. Također se za vrijeme povlačenja prozora po traci mijenja zapis njegovog položaja. Ove operacije sinkronizirane su, naravno, sa modulom za dohvaćanje stanja.

4. Zaključak

U ovome radu prikazan je način upravljanja prozorima u Windows XP okruženju, te je opisana je struktura prozora i način komunikacije prozora sa sustavom odnosno korisnikom. Napravljen je i program koji nudi novi način upravljanja prozorima tako da prikazuje sličice prozora na panoramskoj traci te omogućuje promjenu razmještaja na njoj i postavljanje prikaza na radnoj površini.

Prilikom izrade programa naišao sam na nekoliko problema. Najteži problem koji je nemoguće u potpunosti otkloniti proizlazi iz načina na koji operacijski sustav iscrtava prozore i nedovoljne podrške za dohvaćanje slika prozora. Tako je, na primjer, na operacijskom sustavu Windows XP nemoguće postići dohvaćanje slika prozora za Java, GTK ili aplikacije koje za prikaz sadržaja koriste sklopovsko ubrzanje. Ovaj bi se problem ipak mogao djelomično riješiti budući da je uvijek moguće uzeti sliku dijela ekrana pa tako i slike ovih prozora u slučaju da nisu prekriveni. Tako bi program koristeći kompleksniji algoritam mogao pratiti stanje ovakvih prozora i u slučaju da nisu prekriveni kopirati njihovu sliku sa ekrana. Također bi se moglo analizom slike detektirati djelomične promjene prekrivenih prozora i kombinirati sa prethodnom slikom. Unatoč svim ovim "trikovima" nikada se neće moći postići savršeno obnavljanje sličica za ove vrste prozora. Kako bi program postao korisniji također se mogu napraviti brojna poboljšanja. Tako bi se mogao uvesti kontekstni meni povezan sa sličicama prozora sa opcijama za zatvaranje i razne promjene stanja. Korisno bi bilo i omogućiti funkcionalnost kopiraj-zalijepi povlačenjem sadržaja mišem s prozora na radnoj površini izravno na sličice na panoramskoj traci.

Naravno, kako bi program postao upotrebljiv potrebno je najprije ispraviti i greške u sadašnjoj izvedbi. Ove greške uključuju nemogućnost detekcije promjene razlučivosti ekrana, nemogućnost oporavka programa nakon izlaska računala iz pričuve, greške sa upravljanjem akcijama miša i kvarenje sadržaja radne površine zbog toga što se slike za Java, GTK i slične prozore uzimaju funkcijom *PrintWindow*. Program je također potrebno optimizirati i smanjiti korištenje procesorskog vremena i memorije. U sadašnjoj izvedbi program pamti slike u punoj razlučivosti, a to je za funkcionalnost koju omogućuje potpuno nepotrebno. Kako bi se smanjilo korištenje procesorskog vremena algoritam uzimanja slike mogao bi se promijeniti na način da se slike ne uzimaju kod promjene sadržaja bilo kojega prozora nego da se dozvoli uzimanje slika za jedan ili dva prozora neposredno prije prikazivanja trake. Tako bi se kontinuirano moglo održavati aktualno stanje slika svih prozora na radnoj površini osim, na primjer, za aktivni prozor čija bi se slika dohvatila neposredno prije prikaza. Na ovaj bi se način smanjilo korištenje procesorskog vremena jer se slike najčešće uzimaju upravo za aktivni prozor budući da se zbog akcija korisnika njegov sadržaj najčešće mijenja.

5. Literatura

[1] MSDN biblioteka, http://msdn.microsoft.com

[2] Charles Petzold, Programming Windows, fifth edition, Microsoft press (1998.)

[3] Feng Yuan, Windows Graphics Programming Win32 GDI and DirectDraw, Prentice Hall (2000.)

[4] Leo Budin, Predavanja iz predmeta Operacijski sustavi 2, FER Zagreb (2003.)

[5] Siwu's blog, Reverse Engineering the Windows XP Window Manager, http://siwu.info

[6] Feng Yuan, Window Contents Capturing using WM_PRINT Message, http://www.fengyuan.com

Dostupnost linkova provjerena je 23.6.2008.