REALIZACIJA VIŠESTRUKE KOMUNIKACIJE PC RAČUNARA SA PERIFERIJAMA PREKO SERIJSKIH KANALA U SPECIJALIZOVANOM REAL-TIME SISTEMU

Nikola Zogović, *Institut "Mihajlo Pupin", IMP-Računarski sistemi, Beograd*

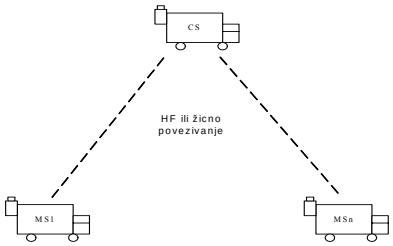
Sadržaj – *U ovom radu predstavljena je realizacija komunikacije, u sistemu koji radi u realnom vremenu, između personalnog računara i više periferijskih uređaja različitih generacija. Za komunikaciju je korišćena serijska multi-port PCI kartica, Windows 2000 operativni sistem i postojeći komunikacioni protokoli na perifernim uređajima.*

1. UVOD

U ovom radu prikazana je realizacija komunikacija PC računara sa periferijama preko *multi-port* serijske kartice na osnovu specifičnih zahteva. Opisan je sistem u okviru kojeg računar radi i naznačene su pojedine karakteristike sistema koje se odnose na komunikaciju. Predstavljena je arhitektura softvera na nivou glavnih komponenata, razvojno okruženje i operativni sistem pod kojim softver radi. Dat je pregled zahteva i specifikacija sprege sa komunikacionim modulom. Izložen je način rada serijskog porta, mehanizmi koje je moguće koristititi za komunikaciju preko njega i prikazana je struktura komunikacionih slojeva.

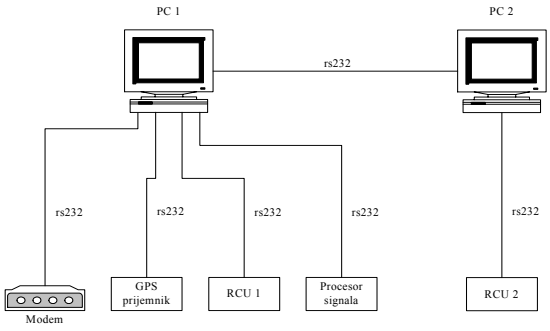
2. OPIS SISTEMA I OSNOVNI USLOVI ZA KOMUNIKACIONI DEO

Sistem za čije namene je realizovana komunikacija, namenjen je za prikupljanje podataka o EM (*electromagnetic*) aktivnosti u okolini. Sastoji se od više izviđačkih stanica (MS – *monitor station*) i jedne centralne stanice (CS – *command station*). Izgled razvijenog sistema prikazan je na slici 1.



Slika 1. *Izgled razvijenog sistema*

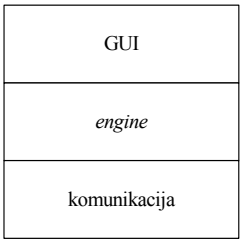
Svaka stanica se sastoji od dva PC računara i periferija sa kojima PC računari komuniciraju preko serijskih *multi-port* kartica (slika 2).



Slika 2. *Izgled jedne stanice*

Komunikacija između stanica realizovana je preko radnih stanica 1 kroz žičnu ili HF komunikacionu liniju. Za prilagođenje signala komunikacionoj liniji koristi se modem koji je povezan na PC računar preko serijskog porta. Osnovni procesi u sistemu su: obrada podataka, održavanje baze poznatih ciljeva, komunikacija unutar stanice i između stanica i prikaz podataka.

Na svakom PC računaru izvršava se softver koji realizuje određeni deo osnovnih procesa sistema. Postoje 4 različite pozicije PC računara: PC 1 i 2 na MS, kao i PC 1 i 2 na CS. Za svaki PC na bilo kojoj poziciji softver je organizovan na isti način u obliku troslojne arhitekture (slika 3). Grafički korisnički prikaz, GUI (*graphical user interface*), služi za prikazivanje podataka operateru i pruža mu mogućnost da zadaje komande. Na zadatu komandu koja se ne odnosi na GUI šalje se signal iz GUI-a ka *engine*u koju ovaj dalje procesira. *Engine* komunicira sa perifernim uređajima preko komunikacionog dela tako što ih periodično proziva po nekom redosledu ili na zahtev operatera, u zavisnosti od uređaja, vrši obradu podataka primljenih od periferija, priprema podatke za periferne uređaje i obrađene podatke predaje GUI-u. Komunikacioni deo prima podatke od *engine*a, pakuje ih u potreban format u zavisnosti od perifernog uređaja s kojim se komunicira, pripremljene podatke šalje na odgovarajući port; prima podatke sa portova, obavlja primarnu obradu nad njima i prosleđuje ih *engine*u.



Slika 3. *Organizacija softvera*

Softver se izvršava pod Windows 2000 operativnim sistemom. Za razvoj softver korišćeni su Microsoft Visual C++, MSDEV alati i MSDN biblioteka. Za razvoj komunikacionog dela korišćen je Commdll DLL (*dynamic link library*), razvijen u IMP-Računarski sistemi.

Komunikacija je realizovana tako da ostatak sistema može nesmetano da radi u odsustvu bilo kojeg uređaja. Za svaki uređaj je realizovan komunikacioni *driver* koga *engine* može nezavisno da pokrene ili prekine. Komunikacioni *driver* je definisan komunikacionim protokolom i formatom poruka uređaja sa kojim se komunicira (GPS prijemnik, RCU–*Remote Control Unit* 1, RCU 2, procesor signala), odnosno komunikacioni protokol i format poruka treba da budu definisani tamo gde ne postoje (modem, između PC računara). Da bi bila postignuta modularnost sistema, tj. da bi softver koji opslužuje komunikaciju bio odvojen od ostalog upravljačkog softvera na PC računarima *driver*i su realizovani kao DLL moduli. Osnovne funkcije koje

zadovoljavaju ovi *driver*i su podrška *multi-port*nog režima rada, podrška rada u *multi-tasking* okruženju i minimalno iskorišćenje procesorskog vremena prilikom komuniciranja sa uređajima.

Commdll je biblioteka razvijena za komunikaciju preko serijskog *multi-port*a sa namerom da omogući najniži sloj komunikacije i stojala je na raspolaganju za realizaciju prethodno navedenih *driver*a. Commdll omogućuje otvaranje komunikacinog kanala, zadavanje parametara komunikacije, povezivanje datoteke za razmenu podataka sa *driver*om porta sa otvorenim kanalom i alociranje bafera željene veličine za razmenu podataka sa aplikacijom. Za otvoreni komunikacioni kanal Commdll pruža mogućnost slojevima iznad sebe da na jednostavan način šalju i primaju informacije, a da pri tom ne vode računa o menadžmentu porta. Commdll ne pruža mogućnost slojevima iznad sebe da direktno pristupaju strukturama podataka operativnog sistema koje stoje na raspolaganju za komunikaciju preko serijskog porta. Ova osobina čini Commdll nepodesnim u primenama gde je potrebno da viši slojevi direktno komuniciraju sa portom. Ovakva situacija se sreće kod upotrebe starijih ili nestandardnih uređaja koji koriste serijsku liniju za komunikaciju. Prilikom inicijalizacije kanala pokreće se posebna nit za taj kanal koja ima zadatak da proverava da li su stigli neki podaci preko serijskog porta i ako jesu da ih smesti u kružni bafer namenjen za razmenu podataka sa komunikacionim slojem iznad sebe, da proverava sadržaj bafera u koje slojevi iznad Commdll-a upisuju podatke koje žele da pošalju i da ih prosledi u file.

3. MOGUĆNOSTI ZA KOMUNIKACIJU PREKO SERIJSKOG PORTA I MULTI PROCESNI RAD NA WINDOWS 2000 OPERATIVNOM SISTEMU

Windows 2000 OS vidi resurse kao *file*ove u skladu sa savremenom koncepcijom tretiranja resursa u OS-ima [1]. Na Windows 2000 OS–u za svaki komunikacioni resurs postoji servis podrška u vidu biblioteke ili *driver*a koja omogućuje aplikacijama da pristupe resursu.

Na osnovu podataka koji se nalaze u strukturi DCB (*device control block*) kontroler serijskog porta prepoznaje u kojem režimu treba da radi. Za dobijanje vrednosti parametara iz strukture i zadavanje parametara na raspolaganju stoje funkcije *SetCommState* i *GetCommState*. Za čitanje iz *file*a, preko kojeg serijski port i OS razmenjuju podatke, i upis u *file* na raspolaganju stoje funkcije *ReadFile* i *WriteFile* koje podržavaju sihnroni i asinhroni pristup *file*u, odnosno *ReadFileEx* i *WriteFileEx* koje podržavaju asinhroni pristup *file*u.

Da bi OS mogao da vidi događaje na samoj rs232 komunikacionoj liniji, postoji mogućnost zadavanja čekanja na događaje iz predefinisanog seta događaja [1]. Koji će događaj da bude detektovan, a koji ne, definisano je maskom koja se postavlja funkcijom *SetCommMask*. Pozivom funkcije *WaitCommEvent* vrši se provera da li se događaj desio i ako jeste, nit iz koje je funkcija pozvana nastavlja sa izvršavanjem, odnosno ako se događaj nije dogodio nit se blokira i čeka da se događaj dogodi.

Windows 2000 pruža mogućnost sinhronog i asinhronog pristupa *file*u u komunikaciji preko serijskog porta. Moguće je komunikaciju realizovati tehnikom *polling*a, odnosno periodičnim prozivanjem porta, naročito u prijemnom smeru i moguće je komunikaciju realizovati tehnikom prekida ali samo sa prekidima na događaje iz predefinisanog seta događaja.

Windows 2000 je multi procesni OS i pruža mogućnost konkurentnog izvršavanje više procesa i konkurentnog izvršavanje više niti u okviru jednog procesa. Za manipulaciju procesima i nitima u WINDOWS 2000 operativnom sistemu postoje funkcije, između ostalog, za:

1. kreiranje novog procesa,
2. otvaranje postojećeg procesa,
3. izlazak iz procesa,
4. terminiranje procesa,
5. kreiranje nove niti,
6. izlazak iz niti,
7. terminiranje niti,
8. prebacivanje niti u suspendovano stanje

CreateProcess funkcija kreira novi proces i njegovu osnovnu nit. *OpenProcess* funkcija otvara postojeći objekt procesa. *ExitProcess* funkcija završava proces i sve niti u okviru njega. Poželjno je koristiti ovaj metod za završavanje procesa jer pruža čisto završavanje. *TerminateProcess* funkcija terminira specificirani proces i sve niti u okviru njega. Ovaj metod se koristi za bezuslovno završavanje procesa i ne obaveštava priključene DLL-ove o završetku procesa. Po završetku procesa bilo kojim od ova dva metoda šalje se signal svim nitima koje su čekale bilo na završetak procesa ili neke niti u okviru njega.
CreateThread funkcija kreira nit koja će da se izvršava u okviru virtelnog adresnog prostora procesa iz kojeg je funkcija pozvana. Za kreiranje niti koja će da se izvršava u virtuelnom adresnom prostoru drugog procesa koristi se funkcija *CreateRemoteThread***.** *ExitThread* funkcija završava nit. Poželjno je koristiti ovaj metod za završavanje niti. Kada se pozove ova funkcija (bilo eksplicitno ili prilikom izlaska iz procedure niti), stek tekuće niti se dealocira i nit terminira. Ako je nit poslednja u okviru procesa pozivanjem ove funkcije terminira se i proces. *TerminateThread* funkcija terminira nit. Kada se pozove ova funkcija stek niti koja se terminira ne dealocira se. Pridruženi DLL-ovi se ne obaveštavaju o terminiranju. Ako je nit poslednja u okviru procesa pozivanjem ove funkcije terminira se i proces. Po završetku procesa bilo kojim od ova dva metoda šalje se signal svim nitima koje su čekale na završetak niti.

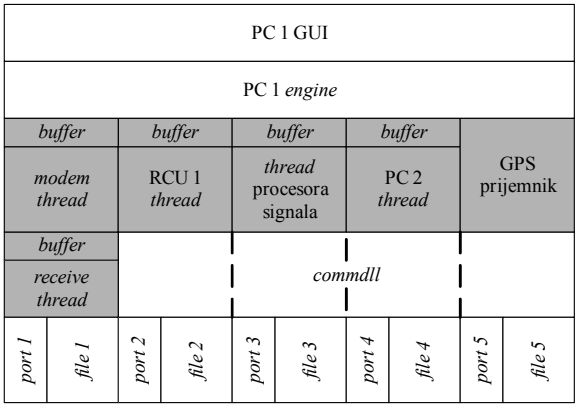
4. STRUKTURA I REALIZACIJA KOMUNIKACIONIH SLOJEVA

U cilju skraćenja vremena potrebnog za razvoj maksimalno su korišćenjeni postojeći softverski moduli. Na slici 4 prikazana je struktura komunikacionog dela softvera za računar PC 1, a na slici 5 za računar PC 2. Između nivoa OS-a kojim je realizovan sam port i pridružen mu *file* i nivoa aplikacije koja koristi komunikacioni deo (PC 1 *engine*, PC 2 *engine*) postoje dva nezavisna sloja pri čemu je svaki sloj realizovan kao posebna nit.

Niži komunikacioni sloj proverava sadržaj *file*a i ukoliko su neki podaci stigli na serijski port uzima ih iz *file*a i smešta u bafer. Ukoliko je potrebno da neki podaci budu poslati na serijski port te podatke upisuje u *file*. Ovaj sloj je realizovan

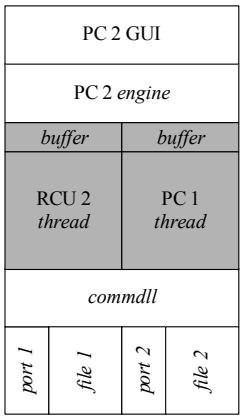
upotrebom Commdll modula za svaki kanal preko kojeg se komunicira sa uređajem koji za komunikaciju koristi rs232 protokol na standardan način. Ovaj sloj je realizovan posebno za kanal preko kojeg računar PC 1 komunicira sa modemom u vidu *receive thread*a. Za ovaj kanal je realizovan bafer za razmenu podataka između višeg i nižeg komunikacionog sloja. Za kanale koji koriste Commdll ovaj bafer je realizovan kao deo Commmdll modula.

Viši komunikacioni sloj realizuje komunikacione protokole za komunikaciju sa uređajima. U ovom sloju vrši se analiza pristiglih poruka na osnovu definisanih formata poruka. Analizirani podaci se pakuju u strukture podataka koje koristi *engine* i te strukture stavlja u bafer za razmenu podataka se *engine*om. Ovaj sloj proverava bafere za razmenu podataka sa *engine*om i ukoliko postoje podaci koje treba poslati pakuje ih u odgovarajuće formate. Za svaki uređaj je napravljena nit koja radi prethodno navedene poslove.



Slika 4. *Struktura komunikacionog dela softvera za računar PC1*

Modem thread za računar PC 1 na poziciji CS i *modem thread* za računar PC 1 na poziciji MS napravljeni su na istom princpu pri čemu svaki realizuje deo protokola za komunikaciju između stanica spram pozicije na kojoj se nalazi. Za komunikaciju između stanica koristi se protokol zasnovan na TDMA (*time division multiple access*) tehnici [2] sa centralizovanim raspoređivanjem koje obavlja računar PC 1 na poziciji CS.



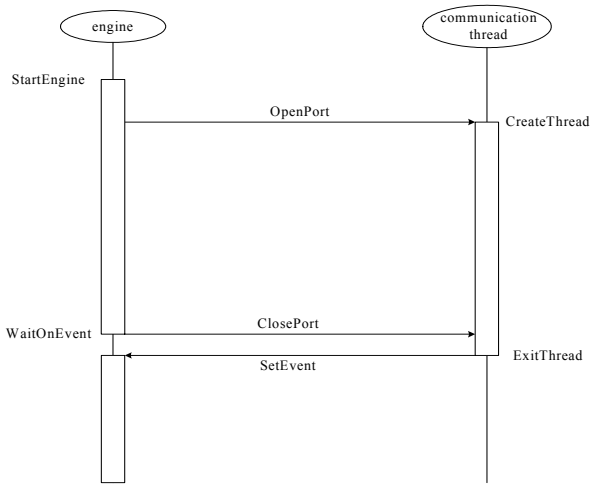
Slika 5 . *Struktura komunikacionog dela softvera za računar PC 2*

Komunikacija sa GPS prijemnikom ima jednostavan protokol [3] sa kritičnim vremenom odazivanja uređaja pa su za ovaj kanal realizovane samo funkcije za pakovanje podataka u odgovarajuće formate poruka i analizu pristiglih bajtova. Ovaj komunikacioni kanal nije realizovan kao posebna nit.

Za komunikaciju je bilo potrebno realizovati delove označene sivom bojom na slikama 4 i 5. Radi modularnosti sistema komunikacioni kanali za računar PC 1 su napravljeni kao jedan DLL osim kanala za komunikaciju preko modema. Kanal za komunikaciju preko modema napravljen je kao poseban DLL koji sadrži *modem thread*ove za pozicije CS i MS. Komunikacioni kanali za računar PC 2 napravljeni su kao jedan DLL.

U nastavku je detaljno objašnjen mehanizam korišćen za pokretanje i ukidanje niti i mehanizam korišćen za sinhronizaciju niti na deljenim resursima.

U ovom projektu korišćen je jedan proces i više niti koje se izvršavaju u okviru istog procesa. Stoga je u nastavku razmatran samo mehanizam za pokretanje i zaustavljanje niti unutar procesa. Kreiranje niti se vrši pozivom *CreateThread* funkcije iz neke niti. Nova nit se izvršava neposredno po pozivanju *CreateThread* funkcije ili se kreira kako *suspended* nit, u zavisnosti od parametra *dwCreationFlags* funkcije *CreateThread*. Za realizaciju niti koristi se funkcija na koju ukazuje parametar *lpStartAddress* funkcije *CreateThread*, a parmetri koje treba proslediti funkciji niti nalaze se u objektu na koji ukazuje parametar *lpParameter* funkcije *CreateThread*. Za zaustavljanje niti postoje dve mogućnosti: da je terminira druga nit (funkcija *TerminateThread*) i da se završi sama (eksplicitno, funkcija *ExitThread* ili kad funkcija kojom je realizovana nit dođe do kraja izvršavanja). Za zaustavljanje niti ovde je korišćeno zaustavljanje iz same niti pri čemu je akcija zaustavljanja inicirana iz druge niti. Iniciranje se vrši preko zajedničke promenjive koju nit koja treba da se zaustavi stalno proverava. Na slici 6 prikazan je mehanizam korišćen za pokretanje i zaustavljanje niti u ovom projektu.



Slika 6. *Pokretanje i zaustavljanje niti*

Iz osnovne niti *engine* procesa poziva se DLL funkcija *OpenPort* za kanal na kojem se nalazi uređaj sa kojim treba da se ostvari komunikacija. U okviru te funkcije poziva se funkcija *CreateThread* kojom se kreira nova komunikaciona nit i nadalje se ova nit konkurentno izvršava sa ostalim nitima. Unutar DLL-a koji realizuje komunikacione

komponente postoji *ExitThread* statička promenjive tipa *bool* koja inicijalno ima vrednost *false*. Ova promenjiva se prosleđuje rutini niti prilikom kreiranja niti. Sve rutine niti su realizovane kao petlje sa uslovom za izlazak na početku petlje. Uslov za izlazak iz petljeje je *true* vrednost *ExitThread* promenjive. Zaustavljanje komunikacione niti inicira osnovna nit *engine* procesa pozivom DLL funkcije *ClosePort*. U okviru ove funkcije promenjiva *ExitThread* se postavlja na vrednost *true* i počinje čekanje na završetak komunikacione niti. Pošto se funkcija *ClosePort* izvršava u kontekstu osnovne niti *engine* procesa ta nit je u stanju čekanja. Kada rutina kojom je realizovana komunikaciona nit izađe iz petlje poziva funkciju za izazak iz niti *ExitThread*. Prilikom izlaska iz komunikacione niti šalje se signal osnovnoj niti, koja se nalazi u *ClosePort* funkciji, da je završetak izvršen. U sledećem koraku *ClosePort* funkcije vrši se zatvaranje *handle*a komunikacione niti (*CloseHandle* funkcija) nakon čega se iz sistema uklanja objekt komunikacione niti.

Deljeni resursi u ovom projektu su baferi između niti koje realizuju komunikacione slojeve. Pošto se sve niti nalaze u okviru istog procesa korišćen je objekt *CriticalSection* za sinhronizaciju tipa međusobno isključenje. Ovaj metod ne garantuje redosled po kojem će niti da dobijaju vlasništvo nad deljenim resursom. Pre početka rada ovog mehanizma sinhronizacije potrebno je da objekt *CriticalSection* bude inicijalizovan. Za tu svrhu na raspolaganju stoji *API* funkcija *InitializeCriticalSection***.** Da bi se omogućio međusobno isključiv pristup deljenom resursu, svaka nit poziva *EnterCriticalSection* funkciju da bi tražila vlasništvo nad deljenim resursom pre izvršavanja koda koji se nalazi u štićenom delu. Povratak iz ove funkcije je kada nit dobije vlasništvo nad kritičnom sekcijom. Kada nit završi izvršavanje štićenog koda poziva funkciju *LeaveCriticalSection* da prepusti vlasništvo, omogućujući ostalim nitima da steknu vlasništvo i pristupe štićenom kodu. Nit mora da pozove *LeaveCriticalSection* funkciju svaki put kada je dobila pravo da pristupi kritičnoj sekciji. Bilo koja nit iz procesa može da koristi funkciju *DeleteCriticalSection* da bi oslobodila sistemski resurs koji je bio alociran kada je kritična sekcija inicijalizovana. Posle poziva ove funkcije objekt kritične sekcije više ne može da bude korišćen za sinhronizaciju. Ako je nit terminirana dok je kritična sekcija u njenom posedu stanje kritične sekcije postaje nedefinisano. Ako je nad kritičnom sekcijom izvršena funkcija *DeleteCriticalSection* dok je ona u posedu neke niti, stanje svih niti koje čekaju da dobiju vlasništvo nad tom kritičnom sekcijom postaje nedefinisano.

5. ZAKLJUČAK

Po planu i razmatranjima izloženim u ovom radu realizovan je softver koji je u eksploatacionim uslovima u potpunosti zadovoljio postavljene zahteve. Obzirom da su razmotrene osnovne mogućnosti *Windows 2000/XP* operativnih sistema za multiprogramiranje, ovaj rad može da posluži kao polazna tačka za projektovanje softvera sa sličnim zahtevima.

LITERATURA

[1] MSDN Library, Microsoft, januar 2001.

[2] ”Protokol za komunikaciju u distribuiranom računarskom sistemu male propusne moći komunikacionog kanala”, Nikola Zogović, Miroslav Petrović, Milovan Stamatović, konferencija ETRAN, jun 2003.

[3] ”Konstruktivna dokumentacija softvera za PAUK 3 verziju kontrolera”, IMP-Automatika, Beograd.

Abstract – In this paper a realisation of communication in a real-time system, between personal computer and some peripherial devices of various type and communication performances is presented. Serial multi-port PCI card, Windows 2000 operational system and the existing communication protocols on the peripherial devices are used for realisation.

A REALISATION OF A MULTI-CHANNEL COMMUNICATION BETWEEN PERSONAL COMPUTER AND PERIPHERIAL DEVICES OVER SERIAL CHANNELS IN A SPECIALIZED REAL-TIME SYSTEM

Nikola Zogovic