

Gimnazija "Sveta Đorđević"
Smederevska Palanka

MATURSKI RAD

RAĐEN U JUNSKOM ISPITNOM ROKU ŠKOLSKE 2003/2004
NASTAVNA OBLAST: RAČUNARSTVO I INFORMATIKA



JUNA 2004. GODINE
U SMEDEREVSKOJ PALANCI

KANDIDAT
GORAN RAKIĆ

Sadržaj

Sadržaj	2
Uvodni tekst	3
SEKCIJA 1	
Pojam operativnog sistema	5
Istorijski razvoj i klasifikacija	6
Pokretanje operativnog sistema	8
Logičke komponente sistema	10
• Organizacija memorije	10
• Organizacija procesa	11
• Organizacija resura	12
SEKCIJA 2	
Principi distribuiranih sistema	15
Web kao najveći distribuirani sistem	17
Distribuirani sistemi objekata	18
Primeri distribuiranih sistema	19
SEKCIJA 3	
Definicija distribuiranog operativnog sistema	22
Komponente DOS-a	23
• Model deljenja memorije	23
• Model deljenja resursa	24
• Model deljenja procesorskog vremena	26
• Komunikacija	26
• Sinhronizacija vremena	30
• Kontrola grešaka	31
Distribuirani sistemi datoteka	33
Primeri distribuiranih sistema datoteka	37
• CODA sistem datoteka	37
• Lustre sistem datoteka	38
Sigurnost i višekorisnički rad	40
Dizajn Amoeba sistema	41
Dalji razvoj	44
Popis korišćene literature	45

1. Uvodni tekst

Operativni sistemi i generalno sistemski softver su jedno od retkih, možda i jedino polje softvera gde su inovacije i korenski nove ideje vrlo retke. Prvenstveno zbog konzervativnosti u prihvatanju novih ideja od strane programera i korisnika, evolucija sistemskog softvera drastično kasni u odnosu na razvoj hardverskih komponenti računarskog sistema. Iako smo svi svedoci povećanja „snage” sistema, pristup resursima računarskog sistema, komunikacija sa njima, kontrolisanje organizacije memorije i procesa i ostalo što spada u polje poslova koje obavlja operativni sistem su sa manjim izmenama isti kao i pre 20 godina.

Značajni napredak postoji jedino u grafičkom interfejsu operativnog sistema, što je bilo uslovljeno željama tržišta u prvenstveno komercijalnom razvoju poznatijih operativnih sistema. Međutim, negativna posledica tržišnog razvoja operativnih sistema jeste često evolutivno nazadovanje kroz istoriju razvoja ovog dela softvera.

Pojavom želje za moćnim višekorisničkim sistemima sa neblokirajućom organizacijom procesa započeo je i razvoj distribuiranih operativnih sistema kao sistema koji rade na homogenim i heterogenim klaster rešenjima i obezbeđuju korisnicima transparentnost pri korišćenju, odnosno skrivaju samu prirodu sistema, mrežnu komunikaciju, pojavu grešaka na različitim nivoima u različitim podsistemima itd. Kako je izrada računarskih sistema velike procesorsko-memorijske moći kroz klaster rešenja jeftino i otvoreno za nova proširenja, distribuirani sistemi se nameću kao optimalno rešenje za različite kritične operacije. Ovakvi sistemi najčešće imaju veliki broj korisnika, dinamički broj i vrstu resursa, a same komponente sistema se mogu nalaziti na geografski različitim lokacijama.

U ovom radu će se prvo govoriti o samim operativnim sistemima namenjenim jednoprocesorskim računarskim sistemima, karakterističnim delovima i operacijama. Zatim će se uvesti pojam distribuiranog sistema uz definicije slojeva distribuiranog sistema, karakteristika, i ciljeva i zadataka. Biće pomenuti i najveći danas prisutni distribuirani sistemi poput Web-a kao najvećeg distribuiranog sistema. Sledeća celina se bavi distribuiranim operativnim sistemima, i pokriva različite sekcije i probleme u realizaciji ovakvih sistema.

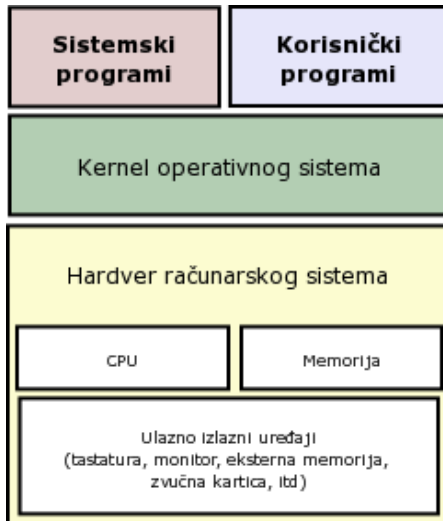
Kao ponuđena rešenja posebnih problema će se eventualno pojaviti i nove, do sada neviđene ideje u trenutno prisutnim distribuiranim operativnim sistemima. Posebna pažnja će biti obraćena na neke nove tek najavljene tehnologije prvenstveno u realizaciji uređenog sistema datoteka nad klaster rešenjem sa distribucijom i sinhronizacijom i samog sistema datoteka po jedinicama klastera u cilju ubrzavanja pristupa, organizacije i pretraživanja različitih tipova podataka.

SEKCIJA 1

OPERATIVNI SISTEMI

Operativni sistem nam olakšava komunikaciju i pristup resursima koje računarski sistem poseduje i, uređuje i organizuje njihovo korišćenje. Spada u grupu osnovnog sistemskog softvera i nemoguće je zamisliti funkcionalan računarski sistem koji ne poseduje neku implementaciju operativnog sistema.

2. Pojam operativnog sistema



Ilustracija 1 – Hardverski i softverski slojevi računarskog sistema

Operativni sistem predstavlja osnovni sistemski softver i prisutan je od samog nastanka računarskih sistema. Mnogi pojavu operativnih sistema vezuju za pojavu programiranja, jer je neophodno postojanje sistemskog softvera kako bi nastao aplikativni softver, odnosno kako bi sam računarski sklop imao ikakvu funkciju.

Kada je engleski matematičar Charles Babagge (1792-1871) pokušao da napravi mehaničku analitičku mašinu, shvatio je da mu je za ispravno funkcionisanje uređaja neophodno sredstvo za lakše sporazumevanje sa mašinom koje će obezbediti precizniji i uniformniji rad. Zaposlio je ćerku engleskog pesnika Bajrona, Ada-u Lovelace¹ koja je time postala prvi programer i ujedno i autor prvog operativnog sistema. Iako mašina nikada nije proradila zbog mehaničke nepreciznosti na osnovu početnog motiva za razvoj prvog operativnog sistema možemo operativni sistem definisati kao:

Operativni sistem predstavlja osnovni sistemski softver koji skriva detalje o hardveru računarskog sistema, olakšavajući rad korisnicima sa ciljem da obezbedi lepše i prijatnije okruženje.

Ovaj pristup definiciji se označava kao **princip ulepšavanja**. Princip ulepšavanja ne treba shvatiti bukvalno jer nije jedini cilj olakšati pristup resursima već je cilj i obezbediti unificiran pristup resursima na različitim računarskim sistemima. Takođe, pod principom ulepšavanja se podrazumeva i obezbeđivanje sigurnosti korišćenja, jer se kroz apstrakciju pristupa resursima zabranjuju nedozvoljene operacije nad pojedinim resursima.

Operativni sistem ima cilj da organizuje pristup resursima računarskog sistema. Da prati zauzimanja i oslobađanja resursa od strane izvršanih procesa, održava liste slobodnih resursa, dopušta ili zabranjuje korišćenje resursa itd. Pojavom modernih računarskih sistema (pedesetih godina dvadesetog veka) operativni sistemi prvenstveno služe za organizaciju resursa što je postao prekomplikovani posao za operatore, odnosno posao bi koštao previše vremena na skupim mainframe računarima i greške bi bile česte. Tako definišemo **princip resursa**:

Operativni sistem je sistemski softver čiji je osnovni cilj da organizuje resurse računarskog sistema i uređuje njihovo dodeljivanje procesima, odnosno obezbeđuje njihovo oslobađanje.

¹ U čast Ada-e Lovelace, prvog programera u logičkom smislu, programski jezik Ada nosi baš to ime.

Po obimu operativni sistem se takođe može posmatrati dvojako. U komercijalno-tržišnom pristupu pod operativnim sistemom se ne podrazumeva samo kernel operativnog sistema, već i raznovrsni programi koji obezbeđuju funkcionalan sistem, pa se čak podrazumeva i grafički interfejs (grafička školjka) sistema. Međutim, u ovom radu se pod operativnim sistemom podrazumeva samo jezgro (kernel) operativnog sistema, bez, za objašnjenje osnova funkcionisanja sistema, suvišnih detalja poput korisničkog interfejsa i možda neophodnih aplikativnih programa. Naravno, u produkcionom operativnom sistemu takvi delovi ne mogu da se izostave, ali izrada produkcionog operativnog sistema nije ni cilj ovog rada. Kod distribuiranih operativnih sistema, najčešće zbog neobičnosti platforme na kojoj se izvršava sam aplikativni softver ovakva, možda naizgled isuviše kruta i uska definicija onoga što operativni sistem obuhvata dobija svoj pravi smisao, iako je definicija potpuno u skladu i sa principom ulepšavanja i sa principom resursa koji se mogu koristiti za utvrđivanje pojma operativnog sistema.

3. Istorijski razvoj i klasifikacija

Kada se govori o operativnim sistemima bilo koje vrste nemoguće je izbeći barem skraćeni pregled istorijskog razvoja sistemskog softvera. Uvidom u istorijski pregled razvoja se može steći slika tendencija koje su se pojavljivale u razvoju i otkriti uzroci zašto su određeni koncepti u operativnim sistemima baš takvi kakvi jesu.

U povoju informatike, razvoj operativnih sistema je bio tesno povezan sa razvojem hardvera. Ukoliko odredimo operativni sistem kao sistemski softver, dovoljno apstraktan, fleksibilan i zamenjiv drugim softverom, i tako odbacimo hardversko-logičke operativne sisteme² koji su bili integrisani u sam računarski sistem, prvi operativni sistemi su se pojavili sa pojavom viših programskih jezika, koji su pojednostavili komunikaciju sa računarskim sistemom.

Sredinom pedesetih godina dvadesetog veka, prvi operativni sistemi su bili jednokorisnički sistemi koji su mogli da izvršavaju isključivo jedan proces u jednom trenutku. To je stvaralo velike gubitke vremena jer se program unosio, prevodio i izvršavao bez mogućnosti da se neki drugi program za to vreme prevodi ili izvršava. To su bili IBM-ovi sistemi **FMS** i **IBSYS**, napisani u FORTRAN-u namenjeni IBM-ovom 7094 računarskom sistemu. Mane sistema su rešavane uvođenjem podele rada, tj. koristili su se jeftini sistemi (na primer IBM 1401) za unošenje programa i štampanje rezultata, i skupi sistemi za prevođenje i izvršavanje programa. To je dovelo do postojanja dve potpuno nekompatibilne serije sistema.

Šezdesetih godina je IBM predstavio seriju sistema System/360 (sa operativnim sistemom **OS/360**), koje su bili softverski kompatibilne, tako da su programi za jedan sistem radili i na nekom snažnijem i skupljem sistemu. Iako je OS/360 bio izuzetno glomazan sistem zbog želje da se pokriju svi sistemi – kako oni najjeftiniji tako i oni najskuplji sa velikim brojem različitih periferija, ono što je OS/360 prvi put predstavio jeste **multiprogramiranje**, odnosno mogućnost da dok jedan proces radi sa periferijama (na primer štampačem), drugi proces koristi procesor. Kako bi se obezbedila separacija procesa u memoriji, radna memorija je bila podeljena i svaki proces se izvršavao u svojoj memoriji. Ova, za današnje standarde, vrlo primitivna tehnika multiprogramiranja označena kao spooling³ je omogućila bolje iskorišćenje procesora, ali je brzo otklanjanje grešaka i pravo izvršavanje više procesa istovremeno pomoću organizacije procesa kakvu danas poznajemo još uvek bilo relativno daleko. Sledeća faza

² Ukoliko su postojali ulazno-izlazni uređaji računarskog sistema, bez postojanja operativnog sistema logično se nameće pitanje kako su ti uređaji bili kontrolisani. Tako dolazimo do privobitnih hardversko-logičkih operativnih sistema koji su bili implementirani u samim uređajima i kontrolisali same uređaje. Stoga je ispravna tvrdnja da su operativni sistemi postojali od samog početka razvoja računarskih sistema.

³ Simultaneous Peripheral Operation On Line – istovremene operacije sa periferijama

multiprogramiranja jeste deljenje vremena, gde bi kratki aktivni procesi jednako delili slobodno vreme⁴.

U ovom trenutku treba pomenuti i **Multics** operativni sistem koji su napravili MIT, Bell Labs i General Electrics koji je imao ideju da omogući napredno deljenje vremena (time sharing) velikom broju korisnika. Nakon brojnih problema u realizaciji, kompanija Honeywell je otkupila sistem i dovršila njegov razvoj. Ideje prikazane u Multics operativnom sistemu su služile kao osnov za prve distribuirane sisteme. Veliki sistemi su služili kao serveri čije usluge su potraživali jeftini klijenti koji su bili laki za održavanje i korišćenje. Kao pouzdan sistem Multics je korišćen u velikim sistemima sve do sredine devedesetih godina dvadesetog veka.

Programeri iz belovih laboratorija, nakon prekida rada na Multics sistemu (Ken Thompson i Dennis Ritchie) počinju da razvijaju svoju verziju multikorisničkog, multiprogramskog sistema pod imenom **Unix**. Uporedo oni razvijaju i viši **programski jezik C**, u kome se piše i Unix. Javljaju se različite implementacije Unix operativnog sistema što dovodi do razbijanja kompatibilnosti. Kako bi se to sprečilo dolazi do usvajanja POSIX standarda koji definiše minimalan set sistemskih poziva koji jedan Unix operativni sistem mora da podrži. Usvajaju se i standardi za C biblioteku, osnovni aplikativni softver Unix operativnog sistema,...

U 1987. godini, pojavljuje se osiromašena verzija Unix-a pod imenom **Minix**. Bila je besplatno ustupana za edukacione potrebe. Finac, Linus Torvalds objavljuje otvoreni operativni sistem pod imenom **Linux**, kompatibilnim sa POSIX standardima izgrađenim po ugledu na Minix. Zbog ideje razvoja putem otvorenog koda, Linux je dobio veliku popularnost. U trenutku pisanja ovog rada aktuelna je verzija 2.6 Linux kernela.

Poseban podsticaj razvoju operativnih sistema daje pojava personalnih računara. Iako znatno slabiji u odnosu na tada prisutne gigante (prvenstveno u I/O moći) zbog svoje cene su obezbedili snažan prodor na tržište. Nakon što je Intel izdao svoj 8080 procesor, trebao mu je operativni sistem. Gary Kidall je napisao **CP/M** operativni sistem. Istovremeno, IBM konstruiše PC računarski sistem sa željom da otkupi prava na CP/M sistem, što im nije pošlo za rukom. Koincidencijom slučaja, IBM angažuje Bil Gates-a koji je otkupio **DOS** (Disk Operating System), proizvod lokalne kompanije iz Sijetla da uradi modifikacije u tom operativnom sistemu. On osniva kompaniju Microsoft i izdaje **MS-DOS**. To je bio jednokorisnički sistem bez podrške za multiprogramiranje, mnogo lošiji od ranije pomenutih sistema, ali zbog svoje cene i cene samih PC računara, vrlo brzo osvaja tržište.

Komercijalni razvoj uzima maha, pa se sredstva i inventivnost troše na pridobijanje tržišta. Nastaju grafički sistemi Xerox PARC, AppleOS i na kraju Microsoft Windows, dok stari sistemi dobijaju podršku za grafički interfejs – X Window System na Unix sistemima.

Takođe, u ovom periodu se intezivno razvijaju real-time operativni sistemi namenjeni za rad u embeded sistemima kao što su medicinski uređaji, kućni aparati, mobilni telefoni. Specifičnost ovih sistema umnogome određuju način razvoja operativnih sistema, koji poseduju tačno utvrđeno tajmiranje, bez ikakve varijabilnosti u vremenu izvršavanja neke operacije.

Za nas interesantniji period počinje polovinom osamdesetih kada se javljaju prvi distribuirani operativni sistemi. Razvoj se ipak nije glatko odvijao. Razni mrežni operativni sistemi nisu podržavali transparentnost distribuiranog sistema, već je svaki sistem imao svoj lokalni OS, dok je deljenje datoteka bilo moguće kroz operativni sistem. Kao i pre pojave mutliprogramiranja, uviđa se da je veliko vreme na brojnim jeftinim PC sistemima neiskorišćeno. Sa razvojem brzih mreža, pojavljuju se prvi distribuirani operativni sistemi koji zbog svoje izuzetne složenosti i danas predstavljaju poslednju fazu razvoja operativnih sistema. Ideja distribuiranih operativnih sistema, odnosno distribuiranog programiranja jeste distribucija procesa na veliki broj nezavisnih računarskih sistema (koji se povezuju mrežnom

⁴ Pod pojmom vremena se podrazumeva procesorsko vreme, odnosno vreme čekanja procesora da se neka I/O zahtevna operacija završi.

komunikacijom – klasteri⁵) čime se povećava iskorišćenost svakog procesora u sistemu. Ono što predstavlja probleme u razvoju ovog tipa sistema jeste problem sinhronizacije procesa, podele procesa i nezavisnot procesa od periferija koje postoje na jednoj, a ne i na drugoj jedinici heterogenog klastera. Distribuirani sistem mora da obezbedi i transparentnost za samog korisnika, tj. da sakrije karakterističnosti klastera, da obezbedi kontrolu grešaka mrežne komunikacije, da uspešno funkcioniše i uz latenciju komunikacije, itd. Parametri za rad rasporeda procesa u distribuiranim operativnim sistemima su najčešće nepravovremeni, nepotpuni ili netačni, što samo povećava složenost u odnosu na jednoprocorski sistem.

Vrlo složeni zadaci distribuiranog operativnog sistema još uvek nemaju pravog i jedinstvenog odgovora, tako da ovo polje i danas ostaje polje stalnih istraživanja i rađanja novih ideja, možda i jedini takav prostor u razvoju sistemskog softvera što je upravo ono što ovu oblast programiranja čini nadasve interesantnom.

Ukoliko dobro pogledamo istorijski pregled možemo da uvidimo da se kroz istorijski razvoj često išlo korak unazad. Kada su se pojavili jeftiniji računarski sistemi, njihovi prvobitni operativni sistemi su bili jednokorisnički, bez podrške za multiprogramiranje, napisani u assembleru, iako su tada na velikim sistemima već postojali višekorisnički, multiprogramski sistemi. Isto se ponovilo i sa pojavom PC računara kao i sa PDA sistemima.

Mnogi programeri veruju da je standardizacija Unix sistema (POSIX standardi) uništila razvoj operativnog sistema. Standardizacija je jednostavno zabranila bilo kakva odstupanja od standarda koji su svi hteli da barem teorijski poštuju. Ono što je rezultiralo takvom politikom razvoja je sa jedne strane velika kompatibilnost sistema, mogućnost umrežavanja različitih arhitektura i sistema, ali sa druge strane je sprečilo inovativnost, premeštajući kreativnost sa sistemskog na aplikativni nivo.

Zavisno od karakteristika operativnog sistema, operativne sisteme možemo karakterisati u sisteme sa podrškom za multiprogramiranje i one bez podrške. Dalje, možemo ih podeliti u jednokorisničke i višekorisničke i na kraju ih možemo podeliti u lokalne (klasične) i distribuirane operativne sisteme. Ovakva kategorizacija je dovoljno determinisana, a i dovoljno otvorena tako da u navedene kategorije možemo podeliti sve operativne sisteme.

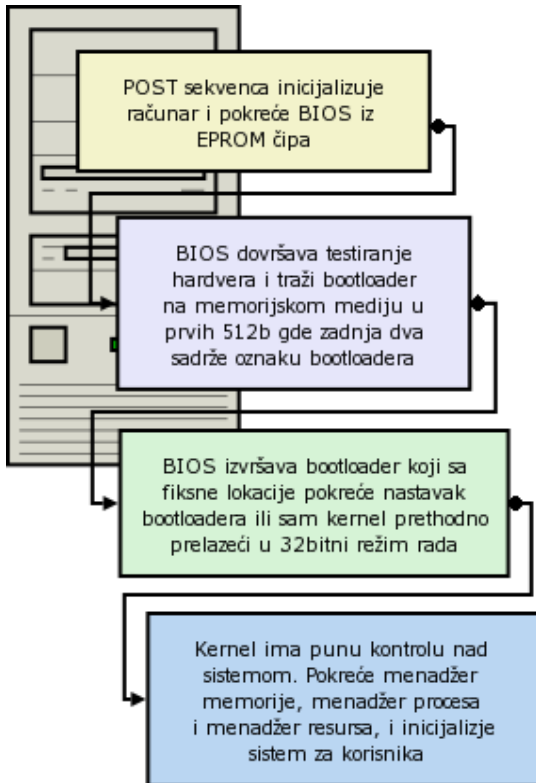
4. Pokretanje operativnog sistema

Startovanje računara predstavlja složeni proces u kome hardver i softver naizmenično preuzimaju glavne uloge. Nakon uključivanja računara, inicijalizuje se POST sekvenca (Power On Self Test) matične ploče koja proverava osnovne hardverske delove računara. Po završetku procedure, BIOS (Basic Input Output System) dobija punu kontrolu nad hardverom sa osnovnim ciljem pokretanja operativnog sistema. BIOS pretražuje memorijske uređaje kako bi učitao *image* veličine 512 bajta, odnosno veličine jednog sektora koji se označava kao *bootsector*. Redosled pretrage je najčešće korisnički definisan, a obično BIOS najpre pretražuje disketnu jedinicu, pa onda hard disk uređaje prisutne na sistemu. Po pronalasku validnog *image*-a, BIOS učitava kod u memoriju i izvršava ga.

Kako je praktično nemoguće napisati kernel operativnog sistema koji bi bio manji od 512b, u *bootsector*-u se nalazi program (označen kao *bootloader*) koji po startovanju sa predefinisane fizičke memorijske lokacije učitava kernel operativnog sistema koji onda nije ograničen veličinom sektora.

⁵ Klasteri mogu biti homogeni i heterogeni. Homogeni klasteri u kojima su sve jedinice klastera jednake – sa istim procesorima, resursima i periferijama, dok se jedinice klastera u heterogenim klasterima razlikuju.

Bootloader mora biti manji od 512b, što uvodi brojna ograničenja. Najčešće se piše u assembleru kako bi se što više dobilo na veličini. Takođe, kako bi BIOS prepoznao da se radi o validnom bootloader-u, zadnja dva bajta bootsector-a moraju biti oznaka 0x55 0xAA (pri tome treba voditi računa o razlici između Little Endian i Big Endian arhitektura jer na Intel x86 platformi koja je Little Endian 511 bajt je 0xAA, a 512 bajt 0x55). BIOS će bootloader učitati na fizičku memorijsku adresu 0x7C00 nakon čega bootloader treba sa zadate pozicije da učita kernel operativnog sistema i prepusti mu izvršavanje.



Ilustracija 2 – Šematski prikaz startovanja operativnog sistema

Ukoliko se kernel učitava sa hard diska, BIOS će prvo učitati MBR (Master Boot Record) koji se nalazi u prvom sektoru hard diska i sadrži tabelu primarnih particija. MBR pretražuje pristup bootloadera-a (koji završava sa specifičnom oznakom i nije veći od 512 bajta) na particiji koja ima boot oznaku, pa onda i na ostalim primarnim particijama.

Iako bootloader najčešće učitava kernel u memoriju, on mora prethodno da uradi dodatnu inicijalizaciju sistema, poput prebacivanja u 32-bitni način rada procesora sa aktiviranjem „protected mod” mogućnosti, uključivanje A20 linije (preko čipa za kontrolu tastature) i slično. Ukoliko bootloader podržava više kernela, često se može dogoditi da bootloader postane preglomazan pa ga je potrebno podeliti u dva dela – bootsector koji se nalazi u prvih 512 bajta memorijskog uređaja sa koga počinje učitavanje sistema i bootloader koji se nalazi na nekoj fiksnoj poziciji, koga poziva bootsector, a koji ulazi u 32-bitni način rada i zatim pokreće kernel sa druge lokacije na disku.

Po učitavanju kernel će inicijalizovati delove, prebrojati memoriju, podsisteme za upravljanje memorijom, raspoređivanje procesa i mehanizam za uređivanje pristupa resursima, a zatim učitati ostale korisničke programe kako bi korisniku podesio radno okruženje.

Listing 1 – Primer bootloadera koji štampa karakter 'A' na ekranu

```

01 ; kod je 16bitni jer nismo ukljucili 32bitni rezim
02 ; i bice ucitan na 0x7C00
03 [BITS 16]
04 [ORG 0x7C00]
05
06 main:
07
08 ; pozivamo BIOS funkciju da bismo prikazali tekst na ekranu
09 ; koja se nalazi na adresi 0x0E
10 mov ah,0x0E
11 mov bh,0x00
12
13 ; postavljamo nacin ispisa teksta – beli tekst na crnoj podlozi
14 mov bl,0x07
15
16 ; stavljamo karakter koji ce se ispisati
17 ; ASCII vrednost velikog slova A
18 mov al,65

```

Listing 1 – Primer bootloadera koji štampa karakter 'A' na ekranu

```
19
20     int 0x10
21
22     ; ne radi vise nista
23     jmp $
24
25     ; dopuni do 510 bajta i završi sa oznakom boot loadera
26     times 510-($-$$) db 0
27     dw 0xAA55
```

5. Logičke komponente sistema

Glavne funkcije operativnog sistema jesu organizacija procesa i resursa. Iako su funkcije naizgled jednostavne, realizacija ovih operacija nije ni malo trivijalna, jer implementacija mora biti dovoljno efikasna, sigurna i proširiva. Od ostalih funkcija treba napomenuti implementaciju sigurne (u smislu pouzdanosti) komunikacije procesa i interfejsa pristupa hardveru koji je moguće koristiti iz procesa (API operativnog sistema). Ne smemo gubiti iz vida i organizaciju radne memorije sistema koja se zbog specifičnog načina korišćenja od svih procesa pa i kernela ne može poistovetiti sa ostalim sistemskim resursima.

Proces predstavlja jednu nit izvršavanja programa, odnosno program u celini ukoliko on ne koristi više niti. Kako procesor može izvršavati samo jedan proces istovremeno⁶, on može biti u fazi izvršavanja ili u fazi čekanja na izvršavanje. Pošto je takvo izvršavanje krajnje neefikasno jer bi pojava procesa koji ima intenzivni rad sa I/O resursima potpuno zaustavilo neke procese koji bi se mnogo brže izvršili. Operativni sistem mora da definiše efikasniji način organizacije procesa od sistema čekanja u redu, pa možemo proces podeliti na najmanje deljive jedinice, označene kao atomske operacije ili atomski procesi koji se moraju izvršiti istovremeno. Zatim ćemo selektivno svakom procesu dodeljivati procesorsko vreme, a potom se prebaciti na drugi proces. Ukoliko količina procesorskog vremena bude isuviše mala dogodice se previše prebacivanja procesa što je operacija koja takođe ima vreme trajanja pa ćemo isuviše vremena izgubiti na prebacivanje umesto da ga posvetimo izvršavanju procesa. Kako operativni sistem treba da maksimum procesorskog vremena ostavi za izvršavanje samih procesa (inače OS postaje beskoristan) algoritam za organizovanje procesa, izbora dela vremena, načina za određivanje kritičnih procesa, i slično postaje složeniji.

U ovom delu ćemo prikazati način rada i funkcionisanja tri dela sistema – organizaciju memorije, organizaciju procesa i organizaciju resursa.

Organizacija memorije

Kao što je već objašnjeno, po uključivanju računara BIOS učitava boot loader sa predefinisane lokacije, koji učitava kernel operativnog sistema na unapred zadatu adresu u memoriji sistema. Jedan od prvih delova koju kernel poziva jeste deo za organizaciju memorije

Kernel operativnog sistema (i njegov deo za organizaciju memorije) će po startovanju imati punu kontrolu nad fizičkom radnom memorijom. Ovo nije samo RAM memorija, već ovaj adresni opseg obuhvata i BIOS memoriju, tabelu prekida (interrupt-a), videomemoriju, i RAM memoriju. Deo RAM memorije je u startu rezervisan, pa će BIOS, na primer, u 0x7C00 do 0x7DFF učitati bootsector memorijskog uređaja sa koga se obavlja startovanje sistema.

⁶ Intelova tehnologija HyperThreading neke može zavarati da procesor može da izvršava dve operacije istovremeno. To međutim nije slučaj, jer se takav procesor ponaša kao dvoprocesorski sistem gde je emulacija drugog procesora urađena na nivou hardvera.

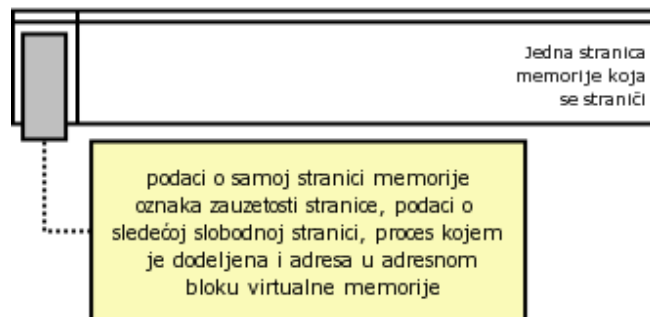
Pored fizičke postoji i virtualna memorija što je svojevrsna apstrakcija fizičke memorije koja je sprovedena na hardverskom nivou (MMU deo procesora) i nivou kernela operativnog sistema omogućava korišćenje većeg adresnog opsega nego što pruža fizička memorija, podršku za dodelu istog virtualnog adresnog opsega različitim procesima i zaštitu memorije od pristupa iz drugog procesa. Tako kernel može imati podršku za izvršavanje više procesa istovremeno, podršku za swap memoriju i sl. Uz pomoć MMU-a deo kernela za organizaciju memorije povezuje delove adresnog prostora virtualne memorije sa stvarnom fizičkom memorijom.

Funkcije koje memory management treba da obezbedi jesu alociranje (obezbeđivanje memorije procesu koji ju je zatražio), dealociranje (oslobađanje alocirane memorije), i realociranje (povećanje već alociranog bloka uz zadržavanje sadržaja prethodno alocirane memorije). Takođe, ne sme se zaboraviti ni pružanje usluge deljenja memorije između više procesa.

Postoje dva osnovna principa organizacije memorije – segmentacija i straničenje (paging). Što se tiče hardverske podrške, segmentacija je podržana samo na I32 arhitekturi pa to treba imati u vidu pri pisanju kenela. Operativni sistemi koji danas postoje primenjuju jedan ili drugi princip. Segmentacija obezbeđuje lakšu zabranu pristupa delovima memorije iz neautorizovanih procesa, dok se kod straničenja to mora implementirati u memory management podsklopu.

Straničenje se zasniva na podeli celokupnog adresnog prostora na blokove fiksne dužine (4kb) sa kojima će se operisati kao sa kvantima memorije. Dalje, potrebno je obezbediti način za obeležavanje slobodne memorije što se može izvesti prisustvom „zaglavlja“ svake stranice koja označava status stranice (globalna povezana lista gde svaki čvor ukazuje na sledeću slobodnu stranicu) ili smeštanjem adresa slobodnih stranica u globalni niz, što je najčešće lošije rešenje.

Stranica u memoriji koja se straniči i koja sadrži podatke o samoj memoriji



Ilustracija 3 – Struktura stranice memorije

Segmentacija pak podrazumeva podelu adresnog prostora na delove (segmente) koji imaju jasno označeno pravo pristupa koje uređuje MMU procesora. Unutar segmenta, procesi alociraju tačno onoliko memorije koliko im je potrebno, ali je problem memory management-a kako da obezbedi tako alociranje pri kome može da očuva dovoljno veliki kontinuirani blok memorije koji nekom porcesu može zatrebati.

Za potrebe distribuiranih sistema je straničenje bolje rešenje jer ne postoji MMU nadležan za određeni adresni prostor memorije, dok se stranice memorije mogu distribuirati transparentnije za same procese koji se izvršavaju u na taj način deljenoj memoriji.

Organizacija procesa

Druga bitna uloga kernela operativnog sistema jeste organizacija procesa, tj. dodeljivanje procesorskog vremena procesima kako bi se oni mogli izvršavati. Sve procese možemo podeliti na aktivne (one koji se izvršavaju – može biti jedan po procesoru), spremne (oni koji čekaju dodelu procesorskog vremena) i procese koji čekaju bilo završetak rada sa nekim resursom ili na završetak nekog drugog procesa i tek po ispunjenju uslova čekanja mogu postati spremni.

Najjednostavniji način organizovanja procesa jeste dodela procesorskog vremena prvom procesu, pa tek kada se on završi učitati tabelu virtualne memorije sledećeg spremnog procesa i preći na njegovo izvršavanje itd. Ovo je princip rada FCFS (First Come First Served) ili FIFO (First In First Out) algoritma i nije optimalni način realizacije, jer se pojavljuje preveliko vreme

čekanja kratkih procesa koji dolaze iza dugih procesa u odnosu na samo trajanje izvršavanja tih kratkih procesa.

Jedan od kvalitetnijih algoritama ja „Round Robin“. Koji određuje kvant procesorskog vremena koje dobijaju procesi iz grupe „spremnih“ procesa. Za mali kvant vremena, procesi će se izvršavati skoro simultano, ali će biti dosta izgubljenog vremena na prebacivanje procesa. Za veliki kvant vremena, Round Robin prelazi u FCFS algoritam jer će se proces izvršiti pre nego što istekne dodeljeno vreme.

Danas se najčešće koriste varijacije „Multiple Feedback“ algoritma koji favorizuje kraće procese. Svi procesi su osim podele na osnovu stanja podeljeni u kategorije. Proces koji pripadaju kategoriji sa manjim rednim brojem imaju veći prioritet. Svaki novi proces koji prelazi u listu spremnih postaje proces prve kategorije dok je nulta rezervisana za sam kernel sistema. Za samo izvršavanje se koristi Round Robin algoritam, stim što se kvant vremena češće dodeljuje procesima iz kategorije sa većim prioritetom. Nakon što procesor dobije određeni broj kvantova vremena, prelazi u kategoriju sa manjim prioritetom. Na ovaj način se kraći procesi izvršavaju brže, a algoritam pruža mogućnost za podešavanja i od strane korisnika odabirom kvanta vremena i razlike između količine vremena koje se dodeljuje kategorijama sa različitim prioritetom, pa i samim brojem kategorija.

Svaki od ovih algoritama se dalje može nadograđivati stvaranjem posebnih grupa procesa na osnovu okruženja procesa i sistema u kojima se procesi različito organizuju upotrebom drugih algoritama, prilagođenih datoj grupi procesa.

Organizacija resursa

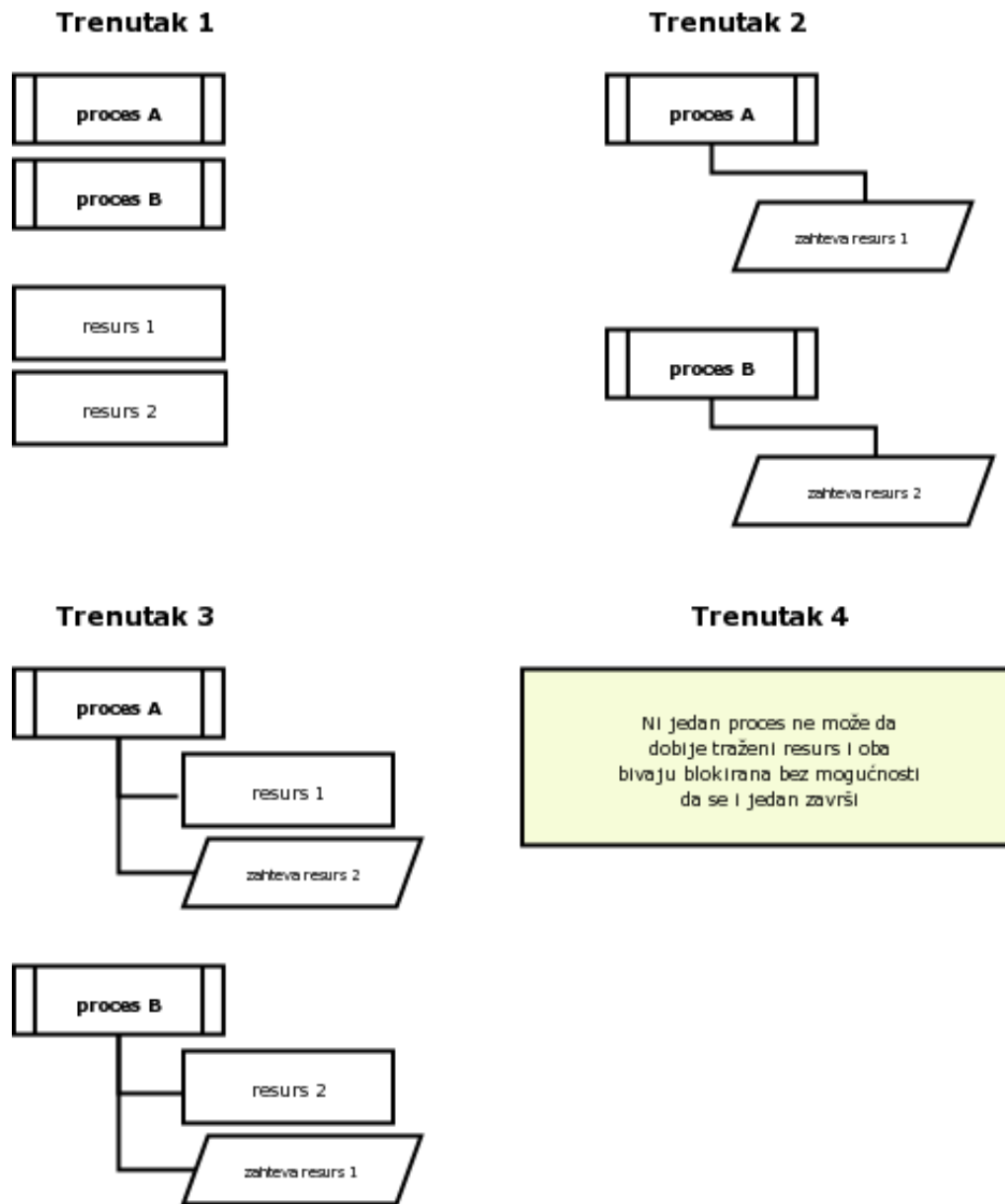
Treći neizostavni deo kernela operativnog sistema jeste menadžer resursa koji vodi računa koji resursi su dodeljeni pojedinim procesima i u zavisnosti od toga asistira u organizaciji procesa blokirajući pojedine procese. Posao ovog dela kernela je i da spreči upad u zamke kakva je prikazana na dijagramu na sledećoj stranici, koji bi bez postojanja menadžera resursa potpuno blokirao raspoređivanje procesa.

Rešavanje ovog problema uopšte nije trivijalno iako postoji više manje ili više uspešnih metoda. Najjednostavnije je zabraniti upotrebu bilo kog resursa dok je ijedan resurs zauzet. Međutim ovo nije nimalo efikasno. Drugi pristup jeste obaveza svih procesa da prijave sve resurse koji im mogu zatrebati što nije efikasno sa programerske strane jer procesi često ne znaju koji im resursi mogu trebati.

Menadžer resursa mora biti sposoban da organizuje kako deljive resurse (TCP na primer) tako i one koji se ne mogu deliti (štampač). Stoga se nameće kao logično rešenje implementacija ovog dela kao modularnog sistema gde bi sam menadžer resursa bio korišćen za svaki pojedini resurs ili za tip nekog resursa. Svaki pojedini menadžer može implementirati sopstveni model obezbeđivanja od zaglavljivanja, a dodatno kernel može pružati uslugu prekida čekanja na resurs ukoliko čekanje pređe kritičnu granicu. Na ovaj način, ukoliko na bilo kom nivou dođe do zaglavljivanja procesa, kernel će proces obavestiti u grešci pri dodeli resursa i prebaciti ga u grupu spremnih procesa kako bi mogao da nastavi izvršavanje.

Tako će na primer menadžer resursa za štampač implementirati spooler sistem gde će čuvati podatke prispele za štampanje obezbeđujući trenutni nastavak procesa koji je zatražio resurs, a kasnije gore opisanom metodom čekanja podatke preneti pravom resursu.

Ovakav decentralizovani način menadžera resursa najčešće ima implementiran i sam upravljački program za sam uređaj, a model je pogodan za implementaciju u distribuiranim operativnim sistemima. U modernim operativnim sistemima pod menadžerom resursa se najčešće podrazumeva samo deo za interakciju različitih menadžera za pojedine resurse i implementaciju prekida usled dugog čekanja, dok se sami menadžeri za pojedine resurse razvijaju kao korisnički aplikativni softver. Primer ovoga jesu sistemi za štampanje na Unix sistemima (cups), sistem za upravljanje zvukom na Linux sistemima (aRTSd, ESD) itd.



Ilustracija 4 – Scenario zaglavljivanja procesa usled nepostojanja kvalitetnog menadžera resursa

SEKCIJA 2

DISTRIBUIRANI SISTEMI

Pojava jeftinih PC računara male procesorsko-memorijske snage i brzih LAN i WAN mreža su omogućili pojavu distribuiranih sistema, odnosno udruživanje nezavisnih sistema koji se korisnicima prikazuju kao jedinstveni koherentni sistem sa velikom zbirnom procesorsko-memorijskom moći i niskom cenom

6. Principi distribuiranih sistema

Po definiciji, distribuirani sistem predstavlja kolekciju udruženih nezavisnih računarskih sistema koji se korisnicima, i procesima koji se na njima odvijaju predstavljaju kao jedinstveni koherentni sistem.⁷

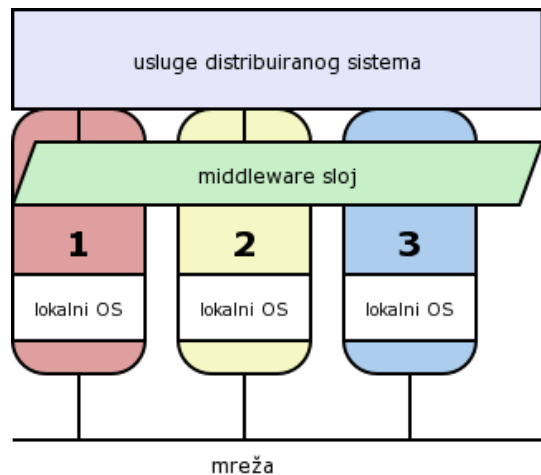
Distribuirano programiranje predstavlja evolutivni nastavak modela razvoja softvera označenog kao mrežno programiranje. Kao što i samo ime određuje, distribuirano programiranje kao osnovu sadrži ideju podele celokupnog sistema na više ili manje nezavisne delove koji će se izvršavati na nezavisnim kompjuterima u mrežnom okruženju. Distribuirani sistem može poštovati centralizovan server-klijent model dizajniranja informacionog sistema ili biti izgrađen od ravnopravnih čvorova mreže.

Na ovaj način izgrađen sistem pruža brojne prednosti kao što su laka proširivost, velika otpornost na greške usled padova pojedinih delova sistema i najčešće usled velikog broja čvorova velika zbirna procesorska i memorijska moć.

Postoji više različitih tipova distribuiranih sistema, a kategorizacija se vrši na osnovu tipa objekata koji su distribuirani. Danas se često pominju termini distribucije dokumenata, distribucije procedura, distribucije objekata i distribucije procesa (distribuirani operativni sistem).

Da bi distribuirani sistem mogao da poveže heterogene delove sistema i obezbedi unificirani pristup virtualnom jedinstvenom sistemu potrebno je da on implementira sloj koji će obezbediti distribuiranu komponentu koji se najčešće naziva „middleware“. Osnovni cilj ovog sloja jeste da ispuni povezivanje resursa bilo da su to dokumenti, deljene procedure ili objekti ili resursi poput štampača, TCP/IP veze i slično.

Ciljevi koji se obavezno moraju ostvariti programiranjem sistema u skladu sa distribuiranim modelom jesu: povezivanje resursa i korisnika, transparentnost, otvorenost i proširivost. Zapravo, distribuirani sistem treba da omogući lako povezivanje korisnika i resursa, da omogući transparentni pristup distribuiranim resursima kao da su oni dostupni lokalno, i da bude dovoljno otvoren i proširiv kako bi u potpunosti bio distribuiran.



Ilustracija 5 – pozicija middleware sloja

Prvi zadatak, povezivanje, predstavlja glavni cilj distribuiranog sistema. Resursi, zavisno od objekta distribucije u sistemu mogu biti bilo šta - štampač, slobodno procesorsko vreme, memorija, datoteke i dokumenti,... Povezivanjem se ostvaruju brojne prednosti od čisto ekonomskih do organizacionih - lakše saradnje više korisnika putem namenskog softvera, najčešće označenog kao „groupware“. Sa omogućavanjem deljenja resursa, javlja se i problem sigurnosti pa u današnjim distribuiranim sistemima delovi koji regulišu sigurnost koriste šifrovane komunikacione kanale, višestruke verifikacije pristupa na osnovu lokacije potraživanja resursa, koriste korisničke polise za pristup resursima itd.

⁷ Distribuirani sistemi – principi i paradigme, Andrew Tanenbaum i Maarten van Steen, strana 2

Transparentnost je očigledno neophodna kako bi korisnik ili procesi koje je korisnik aktivirao mogli neometano da pristupaju resursima. Transparentnost najčešće obezbeđuje middleware sloj distribuiranog sistema i može se ostvariti kao transparentnost pri pristupu, transparentnost lokacije, migracije, relokacije, replikacije, konkurentskog pristupa, i transparentnost pri greškama.

Transparentnost pristupa se zasniva na skrivanju načina pristupa resursima. Na primer, ukoliko postoji distribuirana procedura, proces bi joj morao pristupati na drugačiji način, sa specijalno obrađenim podacima za pristup nego da se ona nalazi lokalno. Upravo interfejs middleware-a rešava taj problem.

Transparentnost lokacije implementira takav pristup resursima da korisnik (ili proces) ne mora da zna tačnu fizičku lokaciju na kom čvoru distribuiranog sistema se resurs nalazi da bi ga koristio. Tipičan primer ovakve transparentnosti jeste URL adresa dokumenta (distribuiranog resursa) na danas najvećem distribuiranom sistemu - World Wide Web-u gde korisnik ili proces koji pristupa resursu ne mora da poznaje geografsku lokaciju servera gde je dokument sačuvan. Ukoliko resurs može da promeni lokaciju, a da se ne izmeni način pozivanja resursa onda sistem podržava transparentnost migracije. Korak dalje predstavlja mogućnost da resurs promeni fizičku lokaciju za vreme korišćenja i tada se govori o transparentnosti relokacije.

U distribuiranim sistemima se često formira lokalna kopija resursa (ukoliko se radi o resursima koji se mogu lokalno kopirati kao što su dokumenti, memorijska lokacija, procedure, objekti itd) kako bi se poboljšale performanse. Sam korisnik resursa (proces) ne treba da bude svestan postojanja kopije jer se onda gubi smisao postojanja iste, te ukoliko je ovo ispunjeno distribuirani sistem podržava i transparentnost replikacije. Prilikom pristupa resursima treba napomenuti i čestu potrebu da više procesa pristupa istim resursima ili da simultano pristupa različitim kopijama istog resursa. Ukoliko je ova mogućnost podržana, kažemo da postoji transparentnost konkurencije.

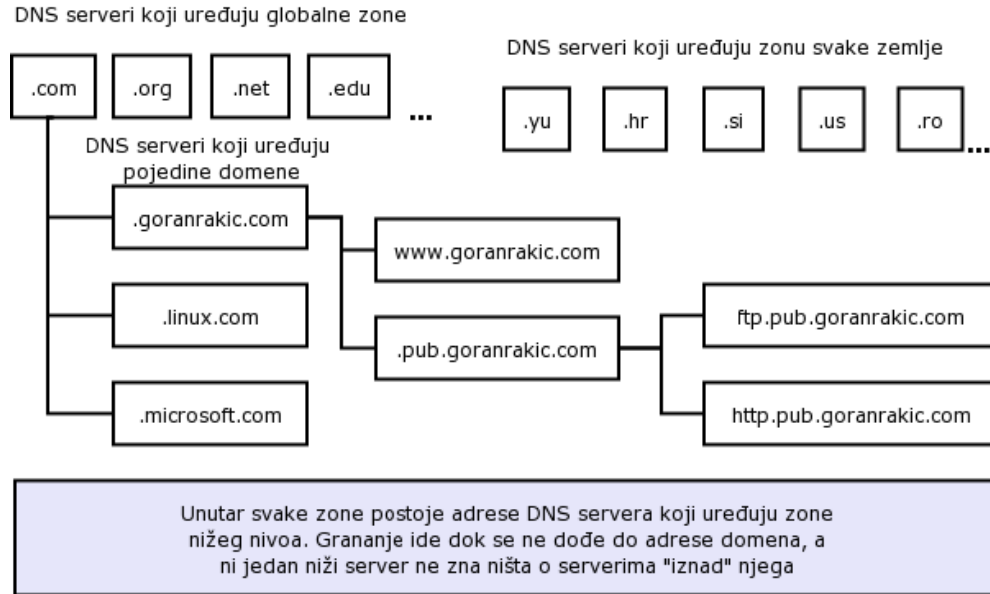
Transparentnost pri greškama ili otpornost na greške je jako bitna za distribuirane sisteme jer se oni sastoje od velikog broja nezavisnih računarskih sistema, njihov rad se odvija u mrežnom okruženju, najčešće imaju veliki broj korisnika i procesa koji se izvršavaju te lako može doći do neplaniranih situacija. Jedan od popularnih šaljivih odgovora na pitanje šta je distribuirani sistem lepo ocrta problem otpornosti na greške u današnjim distribuiranim sistemima, a glasi „Znate da imate jedan kada vas greška na kompjuteru za koji nikada niste čuli sprečava da obavite posao koji ste trebali da obavite”. Veliki problem predstavlja samo okruženje distribuiranog sistema. Naime, jako je teško razlikovati resurs koji je prestao da funkcioniše od resursa koji je preopterećen ili je komunikacija sa njim otežana usled sporog mrežnog okruženja.

Otvorenost distribuiranog sistema se postiže definisanjem interfejsa pristupa distribuiranim elementima koji ne zavisi od broja čvorova i strukture njihovog povezivanja. Pravila formiranog interfejsa za interakciju sa distribuiranim sistemom (deljenje lokalnih resursa i pristup udaljenim resursima) se specifiraju u protokolima koji se nalaze u sloju aplikativnog protokola OSI modela. (Jabber, RPC i sl.) Protokol definiše način komunikacije, procedure i funkcije koje su dostupne, listu parametara, povratne vrednosti itd. Potpuno otvoren distribuiran sistem poseduje portabilan i interoperabilan protokol (interfejs) tako da se procesi pod istim protokolom mogu izvršavati na različitim distribuiranim sistemima, kao i da se na sistemu mogu izvršavati različiti procesi na isti način.

Proširivost se nadovezuje na otvorenost dodajući podršku za zamenu i proširivanje hardvera čvorova distribuiranog sistema, menjanje broja čvorova i dodavanje novih distribuiranih resursa bez potrebe za promenom interfejsa i procesa koji će se na sistemu odvijati. Takav distribuirani sistem koristi decentralizovani model poput Jabber IM sistema⁸ ili DNS (Domain Name Server) sistema. Po ovom modelu ni jedan čvor ne poseduje kompletne

⁸ “Komunikacija procesa korišćenjem Jabber protokola”, Goran Rakić, Republička smotra mladih talenata, Kladovo 2003. godine

podatke o stanju sistema, sistem se odvija na svakom čvoru samo na osnovu lokalno dostupnih informacija, pad jednog čvora ne ruši ceo sistem i na kraju, ali ne i manje bitno, ne postoji pretpostavka da je tajmiranje na svakom čvoru isto.

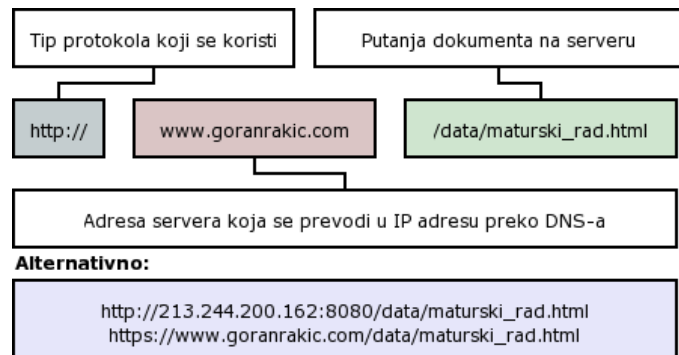


Ilustracija 6 – Prikaz otvorenog distribuiranog sistema na primeru DNS-a

7. Web kao najveći distribuirani sistem

Najveći, danas prisutan distribuirani sistem predstavlja World Wide Web (WWW ili skraćeno samo Web), sistem koji je zaslužan za veliku popularizaciju umrežavanja i distribuiranih sistema uopšte. Sama ideja Web-a je u predstavljanju svega kao dokument, te ovaj sistem pripada grupi sistema sa distribuiranim dokumentima. Web se danas sastoji od miliona klijenata i servera na kojima je deljeno više milijardi dokumenata. Ideja se rodila na CERN-u u Ženevi kao način za deljenje dokumenata i materijala između geografski udaljenih istraživačkih centara u svetu. Od 1994. godine razvojem Web-a, definisanjem standarda i protokola se bavi World Wide Web Consortium (W3C) osnovan od strane vodećih istraživačkih instituta, CERN-a i MIT-a.

Svaki dokument (resurs) na Web-u se može locirati preko URL-a (Uniform Resource Locator). Način adresiranja resursa preko URL-a podržava sve transparentnosti distribuiranih sistema - lokacije, relokacije, migracije i konkurencije. URL definiše i tip protokola koji treba koristiti, pa kako su protokoli tačno utvrđeni od strane W3C-a, postoji portabilnost i interoperabilnost samog sistema.



Mana URL-a jeste što podržava

Ilustracija 7 – Uniform Resource Locator

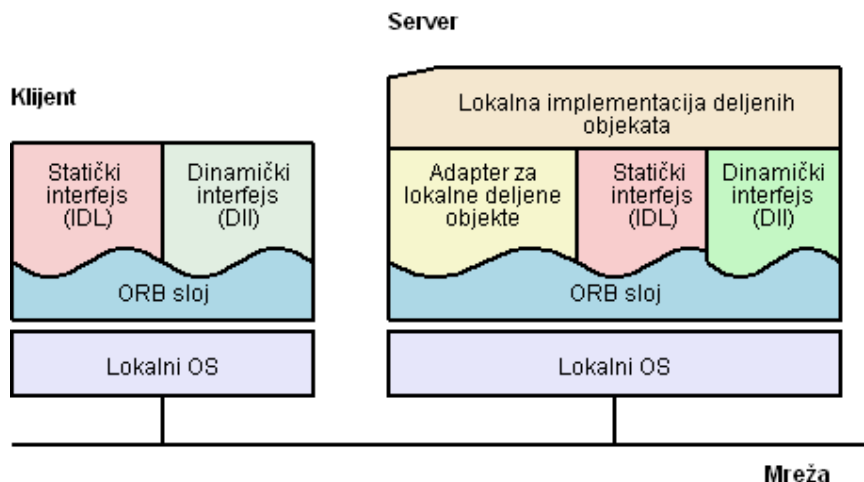
samo ASCII stranu karaktera. Međutim, karakteri iz drugih karakternih strana se mogu predstaviti enkodovani u ASCII entitete pomoću UrlEncode algoritma.

Sama razmena dokumenata (korišćenje resursa) između klijenta i servera se aktivira pristizanjem zahteva za određenim URL-om koji klijent upućuju na adresu servera. Server prihvata zahtev, nalazi dokument u memoriji i prosleđuje ga klijentu koristeći protokol iz URL-a. Naravno, neophodno je da i klijent i server podržavaju dati protokol. Kako su protokoli koji se koriste na Web-u standardizovani kroz rad W3C organizacije, ovo ne predstavlja problem.

8. Distribuirani sistemi objekata

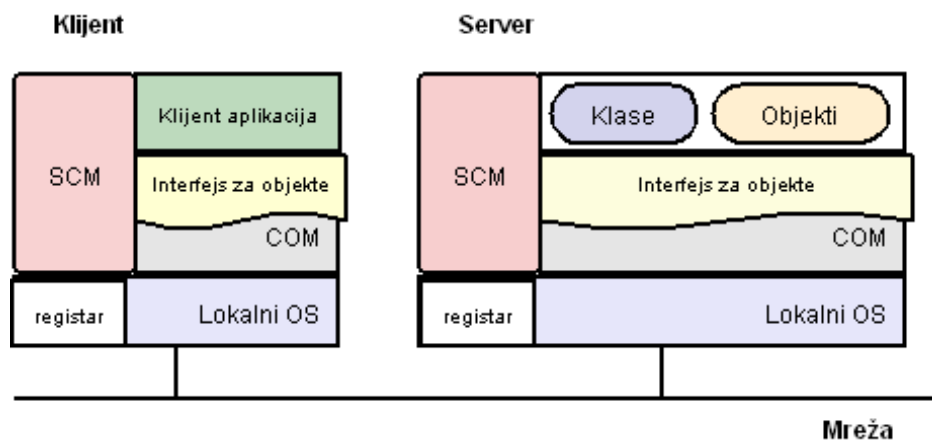
Iako jako raširen, Web kao sistem distribuiranih dokumenata ne predstavlja tehnički kompleksno rešenje kao kada se radi o distribuciji procedura, objekata ili procesorsko-memorijske moći kao kod distribuiranih operativnih sistema. Kada se govori o primerima sistema koji distribuiraju objekte nemoguće je ne pomenuti CORBA i Microsoft DCOM arhitekture kao modele pri izradi distribuiranih sistema. Objektni distribuirani sistemi svaki resurs tretiraju kao objekat, napram dokumenta kao osnovne paradigme kod Web-a.

CORBA (Common Object Request Broker Architecture) je više specifikacija distribuiranog sistema distribuiranih objekata nego sam sistem. Predstavlja industrijski standard iza koga stoji preko 800 različitih konzorcijuma i industrijskih lidera. Prva CORBA specifikacija se pojavila početkom devedesetih godina dvadesetog veka i od tada se primenjuje u velikom broju distribuiranih sistema bilo da su to projekti zajedničke kolaboracije, deljenja informacija, distribuirane relacije baze podataka itd. CORBA se zasniva na postojanju ORB (Object Request Broker) sloja kao middleware-a u sistemu koji služi za uspostavljanje komunikacije između objekata i klijenata koji im pristupaju. Iznad ORB sloja se nalaze delovi koji obezbeđuju uspešnu interakciju deljenih objekata i samog sistema. Sami deljeni objekti su specifikirani preko jezika za definiciju interfejsa (IDL) kojim se određuju metode objekata, liste parametara poziva i povratne vrednosti metoda. Specifikacija objekata može biti dvojaka – u binarnoj formi kada se data specifikacija linkuje sa samim procesom koji će je koristiti i u vidu biblioteka koje koriste IDL specifikaciju da bi koristili objekat. Da bismo koristili dinamički interfejs moramo imati biblioteku koja barata sa IDL specifikacijama napisanu za programski jezik koji koristimo. Poziv objekta može biti statički (IDL) ili dinamički (DII). Proces može kombinovati oba pristupa i na taj način ostvarujući maksimalnu slobodu u pristupu distribuiranim objektima.



Ilustracija 8 – CORBA specifikacija

Na drugoj strani **DCOM** je Microsoft-ova tehnologija iza koje ne stoji ni jedna otvorena organizacija te ista ne može predstavljati industrijski standard kao što je to CORBA. Nastao kao evolucija COM interfejsa, popularne komponente za povezivanje aplikacija. DCOM proširuje COM omogućavajući korišćenje udaljenih objekata. DCOM takođe opisuje objekte pomoću nestandardne specifikacije nalik IDL-u (Microsoft IDL) ali se ona može koristiti samo u binarnoj formi. Svaki IDL poseduje jedinstvenu indentifikaciju (IID) preko koga se uspostavlja instanca objekta i dobija indentifikacija klase/objekta (CLSID). Zbog ovakvog definisanja, DCOM vodi računa o referencama poziva objekta pa sam oslobađa memoriju kada ne postoji ni jedan proces koji koristi dati objekat. Umesto ORB sloja, postoji Service Control Manager (SCM) koji se takođe bavi povezivanjem i pružanjem usluga middleware-a, ali i samim aktiviranjem objekata. Unutar Windows registra (Windows Registry) se formira veza između CLSID-a i fizičke implementacije klase. Ukoliko proces poziva lokalno dostupni objekat, SCM će pročitati lokaciju implementacije iz lokalnog registra, ako se pak radi o udaljenom objektu SCM će kontaktirati middleware čvora distribuiranog sistema koji ima definisan dati objekat. SCM udaljenog sistema pronalazi na osnovu svog lokalnog Windows registra implementaciju klase, inicijalizuje sam objekat i prosleđuje IID deljenog objekta SCM-u koji je zahtevao objekat. Za razliku od CORBA sistema, DCOM ne implementira celu arhitekturu distribuiranog sistema već je za pojedine delove zadužen operativni sistem (Aktivni direktorijum, LDAP,...) što smanjuje portabilnost DCOM-a. Kao zatvoreni standard, koga razvija Microsoft i postoji samo na Microsoft Windows familiji sistema, DCOM ne predstavlja dobro rešenje za bilo koje sisteme gde se može dogoditi potreba za podrškom drugih arhitektura.



Ilustracija 9 – Microsoft DCOM specifikacija

Postoje takođe i distribuirani sistemi koji definišu svoje interfejse, arhitekture i modele koji su prilagođeniji cilju nego generički modeli poput CORBA i DCOM arhitektura. Komunikacija u ovakvim modelima se najčešće obavlja razmenom strukturiranih informacija, poput XML-a, a sama razmena se obavlja putem RPC-a, razmenom kanala, pomoću nekog drugog protokola višeg nivoa i sl. Mana ovakvog pristupa dizajnu je potpuna nekompatibilnost objekata koji su pravljeni za gore navedene popularne arhitekture.

9. Primeri distribuiranih sistema

Kada želimo da pomenemo neke poznate sisteme u praksi, pored Web-a, treba izdvojiti projekte **SETI@Home** i **Folding@Home** koji obrazuju veliku mrežu čvorova distribuiranog sistema kako bi koristili resurse svih čvorova za obradu naučnih rezultata prikupljenih podataka sa radio teleskopa, odnosno za obradu rezultata proučavanja sinteze i uklapanja proteina. SETI@Home projekat postoji duže vremena i može se pohvaliti

impresivnom činjenicom da trenutna zbirna procesorska snaga iznosi oko 15 teraflopsa ($15 \cdot 10^{12}$ operacija sa pokretnim zarezom u sekundi). Treba navesti i projekat distributed.net, osnovan 1997. godine sa ciljem pružanja kako resursa projektima u razvoju tako i pružanja pomoći u razvoju distribuiranih sistema.



Ilustracija 9 – Najpopularniji distribuirani sistemi

Popularno korisničko okruženje i razvojna platforma za Unix i Linux radne stanice – **GNOME** (Gnom, Gnome engl.) implementira CORBA arhitekturu kao osnovu za kako lokalno distribuirane objekata tako i za pristup i korišćenje udaljeno distribuiranih objekata što obezbeđuje laku saradnju različitih procesa na više nivoa u cilju formiranja kvalitetnog i dobro uklopljenog radnog okruženja.

U popularni distribuirani softver se ubraja i **distcc**, distribuirani prevodilac izvornog koda (kompajler) koji značajno skraćuje vreme potrebno za prevođenje velikih projekata koristeći procesorsku snagu više kompjutera – čvorova distribuiranog sistema. Odlikuje se izuzetno lakom integracijom zbog potpunome kompatibilnosti sa standardnim lancem alata koji se koriste pri prevođenju izvornog koda (GNU make, GNU C Compiler, automake i autoconf,...).

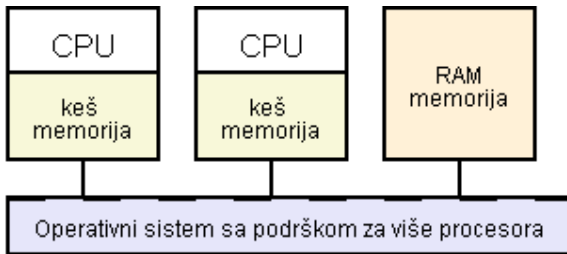
SEKCIJA 3

DISTRIBUIRANI OPERATIVNI SISTEMI

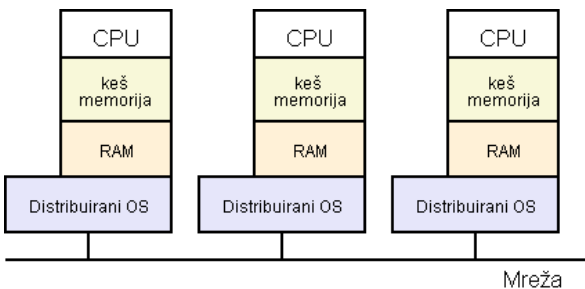
I pored velikih istraživanja u ovoj oblasti, naročito u poslednjoj dekadi, distribuirani operativni sistemi koji su danas prisutni ne predstavljaju sistem opšte namene koji uspeva da ispoštuje sve principe distribuiranih sistema

10. Definicija distribuiranog operativnog sistema

Krajem prošle dekade doživljavamo veliku ekspanziju u broju i veličini kompjuterskih mreža kao i razvoj same kako hardverske arhitekture tako i softverskih modela. Opšte prisuran je trend napredovanja mobilnosti, i portabilnosti u softverskim modelima kao i želja za povezivanjem uređaja i resursa na globalne kompjuterske mreže.



Ilustracija 10 – Višeprocesorski operativni sistem



Ilustracija 11 – Distribuirani operativni sistem

ispunjavaju sve zadatke distribuiranih sistema. Proširivost i transparentnost konkurencije su veliki problemi upravo zbog nepostojanja centralizovanog upravljanja resursima kao u klasičnim operativnim sistemima.

Svaki čvor distribuiranog sistema mora imati implementaciju kernela koji će upravljati lokalnim resursima obezbeđujući njihovu distribuciju na nivou celokupnog sistema svih čvorova – najčešće obeleženog kao klaster. Čvorovi mogu biti isti, kada se govori o homogenom, ali i različiti kada je reč o heterogenom klasteru.

Za potrebe distribuiranog operativnog sistema koriste se mikrokerneli koji sadrže minimum koda koje je potrebno izvršiti u kernel modu (postavljanja podešavanja nad hardverom, prebacivanje procesora sa procesa na proces, upravljanje MMU procesora, prihvatanje hardverskih prekida itd.) dok se sve ostale celine (upravljanje procesima, memorijom, resursima, komunikacija itd) obavljaju iz procesa pokrenutih u korisničkom modu iz procesa kernela. Prednost mikrokernela je u fleksibilnosti, sami delovi sistema iz korisničkog prostora mogu imati potpuno drugačiju realizaciju na različitim čvorovima, uz zadržavanje istog interfejsa komunikacije sa kernelom. Mana mikrokernela je pojava usporavanja usled povećanja komunikacije između delova sistema, što za distribuirane operativne sisteme ne predstavlja veliki gubitak pošto se odvijaju u mrežnom okruženju gde nesumnjivo postoje mnogo uža uska grla. Druga mana je razbijanje tradicije razvoja sistemskog softvera koji nije u skaldu sa monolitičkim kernelima koji se koriste u današnjim

Preteče distribuiranih operativnih sistema su bili sistemi sa podrškom za više procesora sa zajedničkom memorijom nakon čega su došli mrežni operativni sistemi na kojima se izvršavaju složeni distribuirani sistemi uz pomoć middleware sloja jer sam OS nema podršku za transparentciju distribucije. Uključivanjem middleware sloja u sam OS dolazimo do distribuiranog operativnog sistema koji implementira transparentnost distribucije i manje ili više omogućava izvršavanje procesa kao na jednoprocorskom sistemu sa zajedničkom memorijom i resursima.

Distribuirani operativni sistem je sistem koji upravlja kolekcijom nezavisnih kompjutera i njihovim resursima čineći da se oni korisniku prikazuju kao jedan računarski sistem.

Iako istraživanja u ovoj oblasti do sada imaju velikog uspeha, distribuirani operativni sistemi koji danas postoje ne

Unix sistemima. Distribuirani operativni sistemi pak inače razbijaju stege konzervativizma razvoja te mikrokerneli definitivno predstavljaju najbolje rešenje za korišćenje.

Distribuirani operativni sistem treba da obavlja sve poslove klasičnog operativnog sistema (upravljanje memorijom, procesima i resursima) samo u drugačijem okruženju. Osnova svakog distribuiranog sistema je komunikacija sa ostalim čvorovima koja se odvija po određenom aplikativnom protokolu. Uvek treba znati da se distribuirani operativni sistemi izvršavaju u mrežnom okruženju te je komunikacija često nepouzdana i prespora. Postoje brojna rešenja kako se sigurnost i brzina komunikacije mogu povećati korišćenjem kvalitetnih aplikativnih protokola koji imaju brzo vreme odziva i dobru kontrolu grešaka, koji dobro povezuju i dobro prosleđuju poruke čvorovima klastera,

Krajnja reprezentacija distribuiranog operativnog sistema je virtualni sistem (VM computer) koji na raspolaganju ima sve resurse klastera i u svakom trenutku može obezbediti korisniku njihovo korišćenje

11. Komponente DOS-a

Pod komponentama distribuiranog operativnog sistema se pored komponenta klasičnog operativnog sistema (upravljanje memorijom, resursima i procesima) ubrajaju i komponente koje se nalaze u slojevima iznad ili ispod nabrojanih. U ove dodatne, za DOS specifične komponente se ubrajaju model deljenja memorije, model deljenja resursa i procesorskog vremena, kao i komunikacija i sinhronizacija.

Distribuirani operativni sistem takođe mora implementirati kontrolu grešaka u sistemu, sigurnosnu zaštitu pri pristupu resursima kako na korisničkom tako i na sistemskom nivou, distribuirani sistem datoteka,...

Da bismo dobili distribuirani operativni sistem opšte namene, mora postojati, pored već u sistemu obezbeđene transparentije, i dodatni nivo emulacije sistema koji usklađuje API sistema (interfejs koji procesi koriste za komunikaciju sa sistemom, potraživanje i oslobađanje memorije i resursa itd) sa POSIX i SysV specifikacijama bilo da se to radi na binarnom nivou (podrška za pokretanje programa kompajliranih za druge sisteme) ili putem specijalnih biblioteka koje ostvaruju kompatibilnost i sa programom se vezuju za vreme prevođenja stvarajući verziju programa za konkretni distribuirani operativni sistem.

Model deljenja memorije

Jedan od razloga zašto je mnogo teže napraviti distribuirani operativni sistem jeste iz razloga što nije moguće kao kod klasičnih operativnih sistema vrlo jednostavno napraviti deljenje memorije gde će biti smešteni podaci o stanju sistema kako softvera tako i hardvera i njegovih kompozitnih delova, potrebni za rad operativnog sistema. Za razliku od toga posedujemo samo razmenu poruka kroz komunikacione mehanizme. Kašnjenje takvih poruka u komunikaciji usled zagušenja mreže, a pak neophodna komunikacija u realnom vremenu za, na primer blokiranje pojedinih resursa, samo otežava stvar.

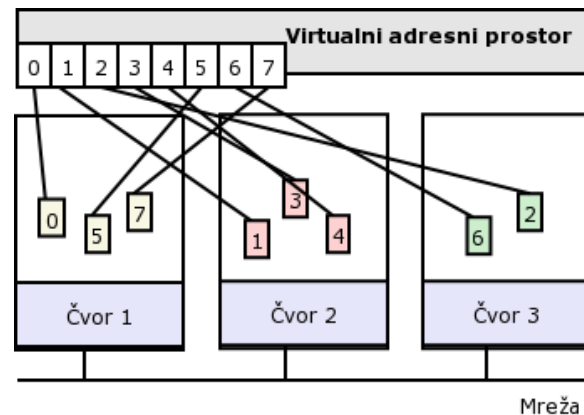
Upravo zbog navedenog problema, pojavili su se različiti modeli koji pružaju emuliranje deljene memorije unutar klastera. Uz pomoć ovakvog dizajna pruža se podrška za rad programima koji su pisani za klasične sisteme, a omogućena je i podrška POSIX stadarda.

Jedan od modela deljenja memorije u DOS okruženju jeste DSM (Distributed shared memory) i zasniva se na postojećoj implementaciji imaginarnog adresnog prostora u klasičnim operativnim sistemima po principima straničenja i segmentacije koji su opisani u odgovarajućem poglavlju ovog rada. DSM prikuplja stvarni fizički adresni prostor memorije

svakog čvora klastera i organizuje virtualni adresni prostor nad kojim postavlja organizaciju stranicenjem. Kada proces koji se izvršava zahteva adresu koja nije prisutna lokalno, lokalni kernel distribuiranog operativnog sistema suspenduje proces, prihvata sadržaj stranice i dostavlja je procesu nakon čega ponovo nastavlja sa izvršavanjem procesa.

Usko grlo ovakvog sistema jeste brzina mreže. Pristojna ubrzanja je pak vrlo lako postići prostom replikacijom stranica memorije. Naime, nakon što proces zahteva određenu adresu memorije koja se nalazi u udaljenoj stranici u koju je moguće samo vršiti upis, kernel može izvršiti replikaciju te stranice, odnosno lokalno napraviti njenu kopiju. Kako je nad stranicom memorije omogućeno jedino čitanje, ne dolazimo do problema konkurencije i sinhronizacije, tj. ne moramo pratiti promene nad stranicom. Postoje i modeli kod kojih postoji replikacija svih stranica, kao i heurističko pronalaženje stranice koja će sledeća trebati kako bi se izvršila i njena replikacija pre nego što proces uopšte i zahteva učitavanje neke adrese iz te stranice. Kada se radi replikacija svih stranica, na sistemu mora biti definisan efikasan i vrlo brz način obaveštavanja svih čvorova da je stranica promenjena kako bi se izbeglo postojanje više različitih kopija istog adresnog prostora. Kako bi informacija sigurno stigla, ovo obaveštavanje svih čvorova mora biti izvršeno pre samog upisa. Tada će, u trenutku upisa postojati samo jedna kopija stranice, i to ona u koju se trenutno upisuje promena.

Određivanje veličine same stranice, kao i kod klasičnih operativnih sistema, puno menja ponašanje sistema, ali se ovde to dešava na drugačiji način. Stranice su svakako dovoljno male da su podaci za uspostavljanje komunikacije (zahtevanje i prihvatanje stranice) kao i zaglavlje same poruke veći od stranice. Ukoliko bi stranice bile veće (8, 16 ili 32Kb), a proces pristupao kontinuiranom bloku adresnog prostora (nizovi podataka) bilo bi manje transfera stranica. Međutim, tada postoji veća verovatnoća da će se u adresnom prostoru iste stranice nalaziti podaci više procesa, pa distribuirani operativni sistem mora vršiti više replikacija i češće se boriti sa višestrukim upisima i problemima konkurencije.



Ilustracija 12 – Distribuirano straničena memorija

Iako istraživanja traju duže od 15 godina, još uvek se bje bitka između efikasnosti distribuiranja memorije, lakoće implementacije takvog modela ali i brzine izvršavanja takvog sistema.

Model deljenja resursa

Osnovna uloga distribuiranog operativnog sistema jeste da klaster, bilo da je on otvoren ili zatvoren, predstavi kao koherentni jedinstveni sistem i da procesima kao korisnicima sistema omogući transparentno korišćenje resursa bez potrebe da procesi poznaju lokaciju resursa koji koriste. Delovi distribuiranog operativnog sistema koji implementiraju pronalaženje resursa se označavaju kao podsistemi za imenovanje i lociranje.

Kako se ne možemo osloniti na fizičku adresu ili na neki fizički adresni opseg, resursu moramo dodeliti virtualno ime kroz sistem imenovanja (engl. naming). Samo ime resursa mora biti jedinstveno unutar distribuiranog operativnog sistema, jer se mora omogućiti jednoznačno lociranje fizičkog resursa na osnovu imena čime se može korišćenje fizičke adrese u potpunosti zameniti imenom i time postići transparentciju lokacije.

Resurs koji se imenuje, unutar distribuiranih operativnih sistema, može biti čvor klastera, štampač, datoteka u distribuiranom sistemu datoteka, proces, prijavljeni korisnik, dokument, objekat, grafički prozor na jednom od ekrana prikaza, poruka u mrežnoj komunikaciji, mrežna konekcija itd... Imenovanje postoji i u klasičnim operativnim sistemima, kada se imenuje komunikacioni port, putanja datoteke u sistemu datoteka itd.

Većini resursa se može i pristupiti, tj. mogu se koristiti. Za ove potrebe definišemo adresu tačke pristupa imenovanog činioca (engl. Access point address) ili kraće samo adresu imenovanog činioca. Na osnovu adrese imenovanog činioca, a preko sistema imenovanja i lociranja procesi i korisnici distribuiranog operativnog sistema mogu koristiti interfejs datog distribuiranog resursa, odnosno pristupiti i koristiti sam resurs. Imenovani resurs, često za ime vezuje i više tačaka pristupa, međusobno potupno ravnopravnih.

Logično je sada postaviti pitanje zašto razlikovati ime resursa i adresu tačke pristupa. Odgovor je da se upravo vezivanje resursa (kakvog ga mi vidimo fizički, štampač) sa posebnim jedinstvenim imenom koje obavlja dodeljenu adresu tačke pristupa preko koje se obavlja komunikacija sa resursom kroz namenski interfejs model koji obezbeđuje transparentnost lokacije. Resurs će imati stalno ime, uz promenljivu adresu tačke pristupa koju je uvek moguće dobiti na osnovu imena resursa i zatim je koristiti.

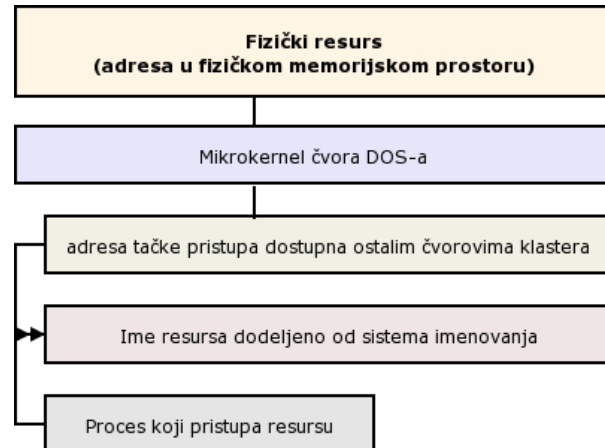
Treba imati u vidu da su svi elementi sistema imenovanja virtualne adrese i imena koje se tek na nivou mikrokernels čvora klastera povezuju sa fizičkim memorijskim opsegom koji je rezervisan za komunikaciju sa datim fizičkim resursom.

Samo ime resursa može biti sastavljeno od niza bitova ili karaktera. Često se ime gradi u čitljivom i razumljivom obliku kako bi se olakšao pristup resursima. Na primer, ime datoteke u sistemu datoteka je tipičan primer takvog imena.

Radi bržeg lociranja fizičke adrese imenovanog resursa koriste se različite tehnike grupisanja u logičke celine – prostore imena, koji se mogu pretraživati efikasno korišćenjem povezanih lista. Drugi način ubrzavanja lociranja se zasniva na pamćenju lokacije pristupne tačke jednom već lociranog resursa, kada se narušava transparentnost, pa je neophodan mehanizam obaveštavanja kada se dogodi migracija ili relokacija resursa.

Postoje situacije kada nije neophodno uopšte poznavati adresu tačke pristupa već na više tačaka pristupa koje odgovaraju različitim resursima postoji jedno ime, a sam distribuirani operativni sistem na osnovu parametara (opterećenost resursa, korisnička ovlašćenja procesa i dr.) određuje sa kojom pristupnom tačkom će povezati proces koji je zatražio lociranje imenovanog resursa.

Veliki problem u izradi modela deljenja resursa predstavlja i imenovanje i lociranje privremeno dostupnih resursa kao i mobilnih resursa koji često menjaju fizičku lokaciju pa su relokacije i migracije česte. Opisan koncept modela imenovanja i lociranja može efikasno da se izbori sa ovim problemom uz dodatnu proveru da li je resurs i dalje dostupan i uklanjanja imena resursa u slučaju nedostupnosti. Međutim, ovaj koncept ne može da postigne efikasnost modela u kome je prisutno keširanje lokacija, hijerarhijsko povezivanje, grupisanje resursa u prostore imena itd.



Ilustracija 13 – Sistem imenovanja

Model deljenja procesorskog vremena

Distribuirani operativni sistem opšte namene mora da ima podršku i za deljenje procesorskog vremena. Za implementaciju ove, najverovatnije najsloženije distribucije, potrebno je izraditi poseban model distribucije na nivou sistema, ali i na nivou upravljanja procesima lokalnog kernela svakog čvora.

Procese koji se brzo izvršavaju najčešće je bolje izvršiti lokalno na čvoru, osim ukoliko je potreban intezivni pristup udaljenom resursu kada je proces bolje migrirati na čvor koji poseduje resurs koji se koristi jer je lokalna unutar procesorska komunikacija za nekoliko redova veličine brža od komunikacije između čvorova, čak i u danas prisutnim veoma brzim LAN okruženjima.

Korisnik će proces svakako pokrenuti na jednom čvoru sistema. Zavisno od dizajna distribuiranog operativnog sistema, čvor će ili krenuti sa izvršavanjem procesa ili će proces uvrstiti u centralizovani sistem upravljanja procesa koji će ga rasporediti veoma slično kao i u klasičnim operativnim sistemima uz pomoć određenog modela (RoundRobin, Multiple Feedback,...). U drugom metodu, korisnik sa sistemom komunicira preko terminala, unoseći naredbe koje sistem izvršava poput mainframe višekorisničkih sistema. Međutim, kako je cilj DOS-a što veća decentralizovanost čime se dobija laka proširivost, drugi metod ne predstavlja dobar izbor, iako je jednostavniji za realizaciju.

Za koncept distribucije je efektivniji pristup pokretanja procesa lokalno kroz modifikovani Multiple Feedback algoritam koji forsira kraće procese. Ukoliko je trajanje izvršavanja duže, proces će prelaziti u sledeći nivo. Kada dođe do kritičnog nivoa, proces će prestati sa lokalnim izvršavanjem i biti dodeljen upravljanju procesa na nivou distribuiranog operativnog sistema. Čvor sa najviše neiskorišćenosti procesorskog vremena će preuzeti proces, odnosno proces će migrirati sa sobom prevlačeći i delove deljene memorije koja mu je neophodna za izvršavanje. Ukoliko je model distribucije memorije dobro dizajniran, migracija procesa će se svesti na prebacivanje stanja procesora i izmenu liste procesa na jednom i na drugom čvoru. Pri početku izvršavanja, proces će zahtevati memoriju iz virtuelnog adresnog prostora drugog čvora pa će i stranice biti replicirane na novom mestu izvršavanja procesa. Proces će se ponovo izvršavati na isti način primenom Multiple Feedback algoritma. Odabir kritičnog nivoa za migraciju procesa je jako bitan jer će uz malu vrednost biti previše prebacivanja što je veoma vremenski zahtevna operacija naročito ako postoji veliki memorijski blok koji proces koristi, dok će se uz veliku vrednost izgubiti distribucija – tj. neće ni dolaziti do migracije.

Ako bismo želeli da dodatno unapredimo ovakav model distribucije procesorskog vremena, morali bismo uračunati i ostale faktore isplativosti migracije procesa – veličinu i lokaciju memorijskog bloka koji proces koristi i resurse kojima proces pristupa ili može pristupati. Ispitivanje ovih faktora često zahteva i određenu heuristiku i pretpostavljanje šta proces može raditi u narednom ciklusu dodele procesorskog vremena. Tako dobijamo kvalitetan model distribucije gde proces migrira na najrasterećeniji čvor klastera, najbliže korišćenom resursu, uz lokalno dostupan adresni prostor memorije koju proces koristi pri izvršavanju. Međutim, situacija najčešće nije tako idealna pa je potrebno napraviti kompromis između izvršavanja na najrasterećenijem procesoru i, na primer, čvoru gde je lociran resurs koji se koristi što se vrši pravilnim odabirom nivoa migracije, nivoa uticaja svakog pojedinačnog faktora, predhodnim istorijatom migriranja procesa i drugo.

Komunikacija

Radno okruženje distribuiranog operativnog sistema isključuje postojanje zajedničkog deljenog memorijskog prostora koji se može iskoristiti za komunikaciju između procesa. Sve što se odvija u distribuiranom operativnom sistemu je praćeno veoma intenzivnom komunikacijom između sistemskih procesa na čvorovima klastera. Očigledno je da je kvalitet implementirane komunikacije jako bitan za brzinu izvršavanja distribuiranog operativnog sistema. Takođe, mrežno okruženje u kome se komunikacija odvija, zavisno od

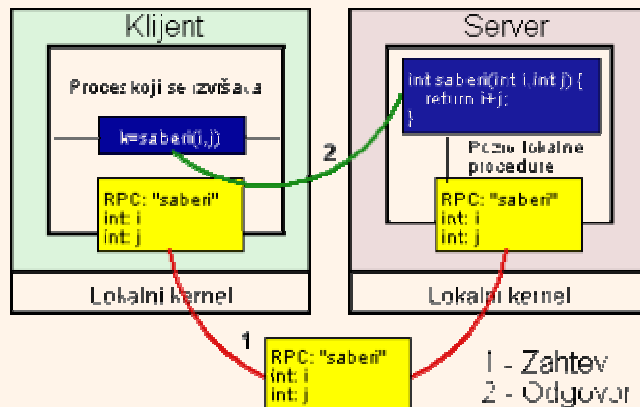
upotrebljene infrastrukture često nije potpuno pouzdana pa distribuirani operativni sistem mora imati metode za verifikaciju informacija.

Prvo što treba odlučiti u dizajnu komunikacije distribuiranog operativnog sistema jeste koji protokol će se koristiti. Protokol treba biti dovoljno portabilan (OSI kompatibilnost), dovoljno visokog nivoa sa mogućnostima ispunjenja osobina koje su gore navedene, ali u isto vreme dovoljno pouzdan i, zbog pitanja performansi, redukovan do najmanje moguće mere.

Vrlo često korišćen, a jako efikasan niži protokol komunikacije jeste TCP. On skriva omaške, izgubljene poruke, duplirane poruke, zakasnele poruke i sl. Specijalnim načinom grupisanja i transporta poruka. TCP stack jeste OSI kompatibilan ali odstranjuje deo OSI slojeva, tako da se sastoji od sloja mrežne infrastrukture, mrežnog sloja (IP, ICMP, IGMP), transportnog sloja gde je sam TCP protokol i aplikacionog sloja gde se nalaze viši protokoli. Viši protokoli mogu potpuno zanemariti smetnje u komunikaciji. Ipak u slučaju prekida komunikacije, TCP protokol ne može ponovo pokrenuti komunikaciju ili pronaći čvor klastera koji poseduje replikaciju stanja sistema i podataka koje je trebalo preneti već je to zadatak viših protokola kao što je RPC (Remote Procuders Calls).

RPC protokol komunikacije

Osnova RPC protokola jeste obezbeđivanje lakog pozivanja deljenog udaljenog servisa, na isti način kao što se obavlja poziv lokalne systemske procedure (systemskeg poziva). Ulazni podaci na osnovu kojih se izvršava servis se prenose kao parametri poziva udaljene procedure. U lokalnom pozivu systemskih procedura parametri mogu biti preneti po vrednosti i po referenci, a pored direktnih parametara, koriste se i globalne varijable. Očigledno je da globalne varijable nisu moguće u pozivima udaljenih procedura. Prenos po referenci je, zavisno od modela deljenja radne memorije obično sporiji od prenosa po vrednosti jer je potrebno izvršiti dodatne upite kako bi se saznao sadržaj referenciranog virtualnog adresnog prostora sa drugog čvora.



Ilustracija 14 – Model komunikacije u RPC protokolu

RPC se ni po čemu ne sme razlikovati od običnog systemskog poziva. Kernel presreće poziv i usmerava ga na RPC modul koji od sistema imenovanja (takode predefinisana udaljena deljena procedura) saznaje adresu na koju treba uputiti poziv, pakuje parametre poziva na osnovu IDL specifikacije deljene procedure i emituje ih ka njoj u klijent-server komunikaciji. Serverski RPC modul prima zahtev, na osnovu IDL-a obrađuje zapakovane parametre i poziva ovaj put lokalnu proceduru i hvata rezultat. Ponovo na osnovu IDL-a deljene procedure formira odgovor i povratnu vrednost procedure i vraća je klijentu koji prima odgovor i procesu koji je zahtevao poziv deljene procedure dostavlja povratnu vrednost kao da je procedura izvršena lokalno.

O RPC-u (i objektnoj varijanti – RMI-u) je već bilo reči u prikazu distribuiranih sistema. Princip rada RPC-a jeste u skrivanju eksplicitne komunikacije, odnosno pružanja mogućnosti da pomoću posebnih podsistema smeštenih unutar kernela i proširenog TCP stack-a procesi mogu pozivati procedure sa drugih čvorova klastera na isti način kao i

lokalno dostupne systemske pozive. RMI za razliku od RPC-a umesto sa procedurama interaguje sa objektima i njihovim metodama. RPC i RMI protokoli se koriste za obavljanje i komunikacije niskog nivoa (u klasičnim operativnim sistemima realizovana preko deljene memorije) i međuprocesnom komunikaciju višeg nivoa (eksplicitna takozvana IPC komunikacija).

Svu komunikaciju u DOS-u možemo podeliti u nekoliko kategorija. To je najpre komunikacija između dva čvora, uređena po **klijent-server** modelu. Komunikacija ove vrste započinje slanjem zahteva od klijenta ka serveru, a završava vraćanjem odgovora nakon što server izvrši traženu uslugu. Za formiranje zahteva klijent mora znati adresu servera. Adresa može biti unapred predodređena, ili može biti promenljiva. U drugom slučaju klijent najpre emituje broadcast paket sa zahtevom usluge koju traži, a server koji tu uslugu pruža odgovara sa svojom adresom. Alternativno postoje i modeli sa posebnim serverima zaduženima za adresiranje koji čuvaju listu usluga koje serveri pružaju. Sama komunikacija se obavlja u protokolu višeg nivoa, a ukoliko je to RPC, onda klijent poziva proceduru servera čime je obuhvaćen celokupni proces.

Druga vrsta komunikacije je **komunikacija sa grupom**. Obično je izrađena u jedan ka ostalima modelu gde se jedan član grupe obraća svim ostalim članovima grupe, odnosno obraća se grupi kao virtualnom serveru, dok on preuzima ulogu klijenta. Grupa može biti dinamična, tj. promenljiva kada je virtualizacija grupe neophodna. Adresiranje grupe se može vršiti adresiranjem svakog njenog člana što je relativno sporo i nedovoljno proširivo. Modelu dinamične grupe bolje odgovara broadcast komunikacija kada samo članovi grupe primaju poruku upućenu svim čvorovima klastera. Iako efikasna za realizaciju, ovakav princip komunikacije nepotrebno zagušuje mrežu. Treći metod adresiranja, obeležen kao *unicast*, predstavlja varijaciju broadcast-a kada se poruka na nivou protokola komunikacije upućuje samo članovima grupe.

Čest je slučaj kada u grupi čvorova treba izabrati lidera, koji će voditi bilo oporavak od havarije ili biti zadužen za koordinaciju grupe i njenu interakciju sa spoljnim članovima. Termin lider ne treba shvatiti bukvalno, jer u jednom trenutku u grupi može postojati više lidera, zaduženih za različite operacije. Algoritam korišćen u Amoeba distribuiranom operativnom sistemu je poznat kao pozivni algoritam (engl. Invitation Algorithm), primenjuje se u izboru lidera u grupi čvorova koji komuniciraju asihrono. Na početku izbora svi čvorovi imaju jednake šanse da postanu lider, što je jedan od osnovnih zahteva potpune distribucije i samim tim i velike otpornosti na greške jer ne postoji predodređeni ili favorizovani lider. Svaki čvor pamti informacije o grupi kojoj pripada i lideru te grupe. Najpre svaki čvor kreira samostalnu grupu i sebe proglašava liderom. Periodično svaki čvor šalje poruku ostalim čvorovima koji zajedno sa njim treba da oforme grupu proveravajući da li su oni lider svoje pod-grupe. Ukoliko je odgovor potvrđan, čvor pauzira aktivnost određeno vreme koje je proporcionalno veličini njegove grupe, a zatim pokreće proceduru sjedinjavanja grupa. U ovoj proceduri, čvor poziva ostale lidere da mu se priključe. Drugi lider po prijemu poziva prosleđuje poziv svim svojim članovima. Svaki čvor koji bilo direktno ili indirektno dobija poruku da se priključi odgovara svojim pristankom. Svaki čvor koji je odgovorio da pristupa grupi ne dobije potvrdu o članstvu u zadatom periodu čekanja (koji zavisi od toga da li je on trenutno lider neke grupe i kolika je ta grupa) on pokreće operaciju slanja poziva. Postoji nekoliko uslova kojima se određuje da navedena procedura uvek rezultuje svim članovima u grupi i jednim odabranim liderom, odnosno razbijanjem do sada sačinjenih pod-grupa i ponovnim pokretanjem procedure.

Pored izbora lidera u grupi procesa i formiranje samih grupa procesa, u grupi je potrebno obezbediti i decentralizovano glasanje kojim procesi mogu doneti zajedničku odluku onda kada je to potrebno. Najčešće se koristi **dvofazni protokol** odlučivanja, u kome se u prvoj fazi najpre održava glasanje, a u drugoj izvršava akcija na osnovu rezultata glasanja. Klasična implementacija ovakvog dvofaznog algoritma ide sledećim tokom: u grupi se izborom određuje lider koji organizuje glasanje, on multicast-uje zahtev za glasanje svim ostalim članovima grupe. Zatim članovi obrađuju zahtev i odgovaraju organizatoru izbora koji prebrojava glasove i određuje konačnu odluku. U drugoj fazi grupa se obaveštava o rezultatima i akcija se preduzima.

Problem dvofaznog protokola glasanja je očigledan – ne postoji kontrola prebrojavanja glasačkih listića i **krađa na izborima** je time omogućena. Ovakvo ponašanje je doduše često u stvarnosti, ali u distribuiranim operativnim sistemima nije dozvoljeno jer narušava sigurnost sistema. Zato se uvode posmatrači - kontrolori prebrojavanja, ili u boljem slučaju – vrši se više prebrojavanja, možda čak i ekstremno kada svaki birač učestvuje u prebrojavanju glasova. Sada, umesto organizatora glasanja postoji distribuirani interfejs glasanja. Proces glasanja se započinje i interfejs sistema glasanja formira prostor za čuvanje rezultata, i otvara distribuirane procedure za pregled rezultata i slanje rezultata. Nakon toga on pokreće vremensko ograničenje u kome se birači moraju javiti. Svaki birač pojedinačno proverava da li do sada postoji neki glas prijavljen interfejsu glasanja i ako ne postoji šalje glas. Ukoliko već postoji glas prijavljen interfejsu, birač najpre upoređuje vrednost glasanja sa svojim glasom. Ako se slažu, birač ne radi ništa, u protivnom oglašava svoj glas ostalim biračima. Kada svi birači dobiju priliku da uporede svoj glas sa prvim sačuvanim u interfejsu glasanja (pre isteka vremenskog ograničenja), birači analiziraju sve oglašene vrednosti i utvrđuju da li postoji kritična većina koja se nije složila sa prvim glasom. Ako kritična većina postoji, u saradnji sa interfejsom se po utvrđenim kriterijumima ponavlja glasanje kako bi se eliminali ubačeni ili pogrešni glasovi pojedinih birača. Jedini preduslov za uspešno glasanje u ovom algoritmu jeste da većina birača budu ispravni, što je i logično. Sam algoritam je vrlo uspešan i jako dobro se skalira sa porastom broja birača (približno ostvaruje idealnu zavisnost $O(1)$).

Kada se govori o greškama u komunikaciji mora se imati u vidu da protokoli nižeg nivoa (TCP na primer) imaju zadatak da obezbede pouzdan transfer poruke međutim u slučaju prekida veze ili havarije transmisija poruka se nikako ne može obezbediti. Zato protokol višeg nivoa (RPC) i sam model komunikacije u distribuiranom operativnom sistemu mora imati načina da premosti greške nastale usled havarije mreže, usled havarije servera i usled havarije klijenta. Kod havarije servera, server može biti onesposobljen pre primanja zahteva kada klijent (i sistem imenovanja) ne može da pronade server koji nudi datu proceduru. U takvom slučaju se korisniku i procesu prijavljuje greška. Havarija servera može nastati i nakon primanja zahteva i tada nastaje problem jer klijent ne može znati da li je server izvršio operaciju pre havarije ili ne. Ovo postaje naročito opasno u slučaju da operacija poziva druge udaljene procedure ili menja stanje distribuiranog operativnog sistema. Zato se ovakve bitne operacije rade pomoću transakcija.

Transakcije mogu potpuno da zamene princip delimične sinhronizacije, ili da ga u blažoj varijanti korišćenja kvalitativno dopunjuju. Promene koje se vrše transakcijama imaju osobinu sve-ili-ništa, odnosno ukoliko dođe do greške u komunikaciji ni jedna operacija iz tekuće sesije neće biti izvršena, odnosno biće sve izvršene ukoliko se transakcija završi uspešno.

Kontrola grešaka u komunikaciji u distribuiranim operativnim sistemima i distribuiranim sistemima uopšte postoji na više različitih nivoa. Sam protokol prenosa (TCP, UDP i sl.) ima kontrolu grešaka kojom se ostvaruje pouzdanost prenetih poruka i podataka između dva čvora. Ukoliko je protokol konekcijski (sesijski) orijentisan, kao što je TCP, onda je zagarantovan i redosled poruka kao i informacija o prekidu konekcije.

Ipak, na nekim višim nivoima je potrebno ostvariti potvrdu da li je poruka validno obrađena. Ovde se dizajneri DOS-a susreću sa klasičnim **problemom dve armije**. Naime, dva generala se moraju dogovoriti o napadu na trećeg. Ukoliko napadnu istovremeno, pobiće dok će u protivnom izgubiti. Problem je što je njihova jedina komunikacija nepouzdan glasnik koji se mora provući kroz teritoriju trećeg generala. Prvi general je poslao glasnika da obavesti drugog da napad počinje sutradan izjutra i glasnik je stigao do drugog generala. Međutim, drugi general se bojao da prvi neće znati da li je glasnik stigao i da će odustati od napada pa je poslao glasnika nazad da potvrdi da je poruka stigla. Glasnik je ponovo uspešno stigao, ali se sada prvi general bojao da drugi neće znati da je glasnik stigao pa da će odustati od napada. Kao što se vidi, ma koliko puta generali slali glasnika, nikada neće doći u situaciju da su sigurni da će jedan i drugi napasti istovremeno, odnosno nisu uspeli da razmene jednu veoma jednostavnu informaciju. Logično je zapitati se kako će se onda u distribuiranim operativnim sistemima prenositi pokazivači na memorijske strukture, informacije o distribuciji procesa i mnoge druge znatno složene informacije.

Upravo je kontrola grešaka u komunikaciji, komponenta protokola višeg nivoa zadužena za uspostavljanje uspešne verifikacije primljene i obrađene informacije.

Malo drugačiji je i problem **vizantijskih generala**. Umesto o dogovoru o vremenu napada, n generala moraju razmeniti informacije o broju svojih trupa kako bi se dogovorili o napadu. Međutim, među njima postoje generali koji ili nemaju sigurnu komunikaciju (glasnik može biti zarobljen i zamenjen prevarantom koji će lažirati informaciju) ili sami generali žele da zbune ostale (neprijateljski proces, ili faličan proces). Rešenje problema spada u klasične algoritme i može se pronaći obrađen na različite načine. Kao krajnju informaciju svi generali moraju imati podatke o validnom stanju ostalih, odnosno obeležene neispravne podatke. Rešenje se svodi na prenos informacija o svim generalima kroz sve generale, i generalno rešenje se može dobiti samo u slučaju da postoji više od dve trećine pouzdanih generala. Konkretno, u distribuiranim operativnim sistemima algoritam rešavanja problema vizantijskih generala se često koristi u sistemima za glasanje, dogovor, komunikaciju unutar grupe i sl.

Različiti su scenariji prekida komunikacije. Klijent može neuspevati da locira server, poruka sa zahtevom može biti izgubljena, server može napraviti grešku prilikom obrade zahteva, poruka sa odgovorom može biti izgubljena, ili klijent može naleteti na grešku nakon slanja zahteva. Svaki od ovih problema traži različita rešenja. Ponekad je moguće izvršiti oporavak od greške ponovnim slanjem zahteva, dok to ponekad nije moguće. Takođe, u nekim situacijama vremensko ograničenje nakon čega se emituje poruka o grešci je jedino rešenje. Ipak, ponekad je to neprihvatljiva solucija.

Greške u komunikaciji sa grupom se moraju razložiti na dva dela. Prvi su greške koje nastaju u komunikaciji grupe sa spoljnim činiocima, kada se one posmatraju kao greške u klijent-server komunikaciji, a drugo su greške koje nastaju u komunikaciji unutar same grupe bilo da su one vezane za odlučivanje i glasanje ili za razmenu informacija. Takođe, unutar same grupe mora u svakom trenutku postojati sinhronizacija, i svaka pojava nesinhronizovanosti grupe pre ili kasnije vodi ka pojavi greške u radu te grupe.

Sinhronizacija vremena

Čvorovi klastera su, kao što je već nekoliko puta naglašeno nezavisni računarski sistemi. Kao takvi, svaki čvor operiše samostalno, pod upravljačkom palicom distribuiranog operativnog sistema. Na nivou sistema međutim većina atomskih operacija mora biti izvršena tačno određenim redosledom, u tačno određenom vremenskom trenutku na nivou celokupnog sistema. Upravo obezbeđivanje usklađenosti toka vremena na svim čvorovima klastera kao i toka komunikacije na nivou distribuiranog operativnog sistema uopšte je zadatak sistema za sinhronizaciju.

U klasičnim operativnim sistemima pojam sinhronizacije vremena je potpuno nepoznat jer se svaki proces u svakom trenutku može obratiti kernelu odgovarajućim sistemskim pozivom kako bi dobio informaciju o tačnom vremenu. Ukoliko jedan proces zahteva informaciju o vremenu, i odmah zatim drugi proces zahteva informaciju o vremenu, drugo vreme će biti veće od prvog, baš kao što je i realno stanje. U distribuiranim operativnim sistemima ovo nije slučaj jer svaki čvor klastera meri sopstveno vreme na nivou hardvera. Kada se prvi proces pokrene na jednom čvoru i zahteva informaciju o vremenu, nepostoji nikakva garancija da će to vreme biti manje od vremena koje kasnije dobije drugi proces koji je pokrenut na drugom čvoru klastera.

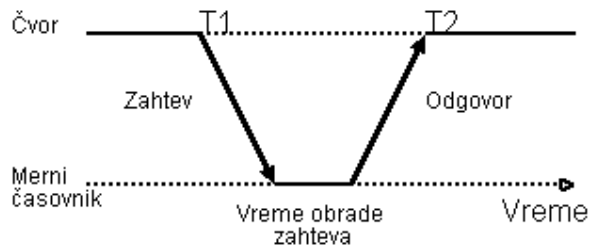
Najpre moramo utvrditi kako može doći do odstupanja u vremenu između različitih čvorova klastera. Časovnik u svakom čvoru predstavlja oscilujući kristal sa relativno stalnom frekvencijom (brojem otkucaja). Nakon definisanog broja perioda oscilacije, stalni brojač u posebnoj memoriskoj oblasti pod kontrolom BIOS-a se uvećava za jedan kvant vrednosti. Unutar BIOS-a se povezuje vrednost memorijske lokacije sa stvarnim datumom i vremenom nakon čega dobijamo vreme izraženo u klasičnim jedinicima (godina, mesec, dan, čas, minut, sekunda,...). Ipak, broj oscilacija kristala nije isti na svim čvorovima

klastera. Iako naizgled neznatna, nakon kritičnog perioda razlika počinje da se intenzivno primećuje ometajući rad distribuiranog operativnog sistema.

Kao rešenje ne možemo koristiti jedan časovnik kome verujemo i to iz dva razloga. Jedan je narušavanje autonomnosti svakog čvora, a drugi praktična neupotrebljivost takvog pristupa jer bi centralizovani merni časovnik imao preveliko opterećenje iz celog sistema, odnosno postojalo bi kašnjenje između realnog vremena i informacije koju časovnik prosleđuje kao odgovor usled perioda potrebnog za obradu zahteva, pripremanje odgovora i njegovo uspešno slanje do destinacije.

Ako sinhronizujemo sve časovnike u realnom trenutku T_1 na broj otkucaja N_1 , u realnom trenutku T_2 svi časovnici čvorova bi trebali da pokazuju $N_2 = N_1 + (T_2 - T_1) \cdot C$, gde je C odnos broja otkucaja po jedinici merenja vremena. Praktično, današnji časovnici stvaraju grešku tako da N_2 neće biti isto na svim čvorovima klastera, odnosno broj može biti manji ili veći zavisno od načina devijacije merenja vremena u samom časovniku. Da bismo ispravili grešku, moramo periodično sinhronizovati časovnike sa mernim časovnikom i pri tome formirati takav sistem koji ispravlja dve greške. Prvo, da je od trenutka kada čvor pošalje informaciju o trenutnom broju otkucaja, do trenutka kada drugi čvor informaciju primi, proteklo vreme i drugo da ni po koju cenu ne smemo na bilo kom čvoru vreme vratiti unazad jer to može izazvati probleme sa objektima, datotekama i procesima na samom čvoru i sistemu uopšte.

Kao što se vidi na slici, pretpostavka je da će vreme slanja zahteva i vreme slanja odgovora biti isto, te da možemo smatrati da ono stoga iznosi $(T_2 - T_1)/2$. Kako i T_2 i T_1 merimo na istom časovniku, iako on nije tačan, možemo tačno izračunati interval. U ovakvoj postavci nije uračunato vreme obrade zahteva pa dodatno možemo poboljšati sistem šaljući i vreme obrade uz informaciju o trenutnom broju otkucaja.



Ilustracija 15 – Kašnjenje broja otkucaja u odgovoru

Opisani pristup je vrlo centralizovan, ali veoma pouzdan ukoliko možemo da garantujemo tačnost poredbenog časovnika što je lako izvesti sinhronizujući časovnik sa radio signalom sa satelita ili sl. preko komercijalno dostupnih uređaja.

Alternativno, možemo izbeći prisustvo poredbenog časovnika koristeći se prosečnom vrednošću. Periodično svaki čvor može upitati ostale čvorove klastera trenutno vreme, šaljući im svoje trenutno vreme. Na osnovu prikupljenih odgovora čvor može odlučiti kako treba da podesi svoj časovnik.

Treće, u modernim distribuiranim operativnim sistemima takođe prisutno rešenje jeste korišćenje logičkih časovnika. Pre početka vremenski kritičnog procesa, za potrebe tog procesa čvorovi se dogovore o početku inicijalizacije logičkog časovnika za dati proces. Nakon toga, logički časovnik se ažurira na osnovu razlike broja otkucaja između vremena formiranja časovnika i sadašnjeg vremena. Usled različite frekvencije oscilovanja, logički časovnik se takođe mora ažurirati ali je jednostavnije dobiti tačniju sinhronizaciju jer je časovnik vezan samo za jedan proces, a ne za ceo čvor ili ceo distribuirani operativni sistem.

Kontrola grešaka

Do sada je u kontekstu pojedinih komponenti distribuiranog operativnog sistema bilo reči o sistemu za kontrolu i prikriivanje grešaka, odnosno toleranciji grešaka. Specifičnost grešaka u distribuiranim sistemima jeste parcijalna greška, greška samo na jednom delu ili u jednom sloju sistema. Greška može biti izolovana ili može voditi krahu

sistema procesa ili gubitku podataka. Za zadovoljavanje osnovnih principa u dizajnu distribuiranog operativnog sistema, sistem mora imati mogućnost da nastavi funkcionisanje i pri takvim neočekivanim situacijama.

Kao merilo kvaliteta distribuiranog operativnog sistema se uzima dostupnost sistema, odnosno verovatnoća da se sistem može odmah koristiti. Sa druge strane pouzdanost je verovatnoća da će sistem raditi neprekidno. Iako slični, ova dva indikatora nisu isti. Sistem koji svakog sata ne radi jednu milisekundu ima dostupnost od preko 99,999%, ali je jako nepouzdan. Sistem koji svakog januara ne radi dve nedelje ima dostupnost od skromnih 96%, ali visoku pouzdanost. Sigurnost i mogućnost popravke su takođe bitni faktori u oceni kvaliteta. Sigurnost obezbeđuje da se ništa katastrofalno neće dogoditi pri pojavi greške, iako to nikako ne može biti zagarantovano jer greške u sistemu nisu predvidive.

Sve moguće greške možemo klasifikovati u prolazne (privremeni gubitak kvaliteta transmisije podataka), periodične (loš kontakt u delu sistema) i stalne (havarija koja se manifestuje sve do popravke oštećenog dela). Po tipu nastanka greške mogu biti hardverske i softverske. Dalje, po posledicama greške možemo podeliti na one koje dovode do prestanka rada sistema, one koje dovode do neispunjavanja zahteva, greške koje dovode do vremenskih prekoračenja, ali i greške mnogo teže za prepoznavanje kao greške koje dovode do grešaka u odgovoru, greške koje dovode do promene stanja sistema (naročito opasne kod distribuiranih sistema datoteka), i na vrhu – greške koje dovode do slučajnih posledica, najčešće nastale usled prepisivanja memorije, pogrešne komunikacije, zakazivanjem sistema sigurnosti i slično kada su posledice najgore.

Klasična kontrola grešaka se uglavnom svodi na redundantnost na svim nivoima gde može doći do greške. Na primer, u komunikaciji se pored same poruke dodaje i informacija koja pomaže verifikaciji ispravnosti poruke. Ukoliko verifikacija neuspe, zahtevaće se ponovno slanje poruke, ili preusmeravanje poruke drugom rutom kako bi se zaobišle smetnje. Tako dolazimo do drugog uslova, postojanja fizičke redundantnosti. Kao što čovek ima dva oka iako može videti i sa jednim, Boeing 747 ima četiri motora iako može da leti sa tri, tako se i u hardveru distribuiranog sistema koriste iste metode.

Kontrola grešaka hardvera se može relativno lako implementirati uz pomoć dobrih protokola komunikacije i dobre postavke sistema sa kvalitetnom redundantcijom na različitim hardverskim nivoima. Međutim, kontrola grešaka procesa koji se izvršavaju na distribuiranom operativnom sistemu je izuzetno složena i zahteva velike izmene načina pogleda na rad procesa u distribuiranom operativnom sistemu. Najpre treba utvrditi da li se moraju svi procesi kontrolisati. Ukoliko proces nije sistemski, niti sistemski proces zavisi od njegovog rezultata, pod pretpostavkom da mehanizam sigurnosti sistema štiti sistem od nepropisne interakcije sa procesom, nije potrebno procesu obezbeđivati kontrolu grešaka. Osnovna kontrola grešaka procesa se svodi na redundantnost. Umesto da imamo jedan proces koristimo **grupu procesa** sastavljenu od više nezavisnih indentičnih procesa. Grupa se spoljnim procesima predstavlja kao jedinstveni proces, kao što se i klaster kroz distribuirani operativni sistem predstavlja kao homogeni računarski sistem. Grupa ne sme biti hijerarhijski uređena jer onda postoji jedinstvena tačka oslonca – koordinator grupe čijim krahom grupa prestaje da funkcioniše. Sve odluke koje je potrebno doneti u grupi se donose sistemom glasanja gde svaki proces grupe donosi odluku, a preovlađujuća odluka se smatra odlukom grupe – tj. odlukom procesa ukoliko grupu posmatramo spolja. Pri grešci na nekom od članova grupe, taj član prestaje da učestvuje u komunikaciji unutar grupe, a ostali članovi moraju proveriti da li se ne radi o privremenom kašnjenju u komunikaciji. Postoji opasnost da deo članova grupe izgubi komunikaciju sa ostalim članovima čime se zapravo uspostavljaju dve grupe. Zato distribuirani operativni sistem mora imati načina da ili grupu eliminiše ili da procese poveže u novu grupu, prethodno ustanovivši šta se dogodilo. Iako na prvi pogled jednostavno, ovo je izuzetno složeno za realizaciju i danas postoji više aktuelnih istraživanja koja utvrđuju najbolje algoritme kojima se može postići sigurnost u odlučivanju nad operacijama sa grupom. Ne treba ni pominjati da je i sam proces koji odlučuje šta treba uraditi sa grupom (deo raspodele procesa) takođe grupa procesa i da se svaki put odvija glasanje unutar te grupe. Takođe, radi poboljšavanja performansi odabir čvorova na kojima će se izvršavati u svakoj svojoj atomskoj jedinici je

vrlo važan, a procesi grupe moraju imati težnju da migriraju na one čvorove između kojih je komunikacija najbrža.

Broj procesa koji trebaju biti u grupi zavisi od kritičnosti procesa koji se izvršava (odnosno od kritičnosti po sistem ako se dogodi greška pri izvršavanju procesa) ali i od statističke analize verovatnoće događanja greške pri izvršavanju procesa. Metod je naročito efikasan ukoliko se proces često izvršava što i jeste slučaj sa sistemskim procesima.

Pored kontrole grešaka koja će osigurati sistem od potpunog kolapsa, sistem mora imati i mogućnost da stanje sistema u kome se dogodila greška zameni sa prethodno ispravnim stanjem sistema. Dva su pristupa na koji se ovo može izvesti. U prvom se sistem iz sadašnjeg stanja sa greškom vraća u prethodno zabeleženo stanje. Drugi pristup je da se sistem umesto vraćanja u ranije stanje pokuša prebaciti u novo ispravno stanje. Prvi pristup je univerzalniji ali često nedovoljno poštuje principe distribuiranog sistema, dok drugi možemo izvesti jedino onda kada unapred znamo koja se greška može dogoditi jer samo tada sistem može znati na koji način da ispravi grešku i krene dalje. Razlike ova dva pristupa se najlakše vide na primeru greške u komunikaciji. Ukoliko trenutno emitovani paket ne dopre do svog cilja, ispravkom greške vraćenjem unazad će biti zahtevano ponovno slanje paketa. U drugom pristupu, primalac paketa će pokušati da na osnovu primljenih paketa otkrije paket koji zbog greške nije stigao, a ukoliko u tome ne uspe zatražiće, po prvom pristupu, ponovno slanje paketa. Evidentno je da u drugom slučaju mora doći do preklapanja paketa, tj. da se u svakom slanju paketa šalje deo prethodnog i deo sledećeg, tako da se gubitkom jednog paketa ne narušava komunikacija.

Postoje situacije kada ipak nije moguće uraditi svojevrstu popravku nastale štete kada sistem nad kojim je izvedena operacija ne možemo vratiti u stanje pre te operacije. Drugi problem je što ne postoje nikakve garancije da neće doći do iste greške i nakon vraćanja stanja sistema.

12. Distribuirani sistemi datoteka

Sistem datoteka u operativnom sistemu uređuje način na koji će se datoteke i direktorijumi organizovati i čuvati na memorijskom medijumu. Analogno tome, distribuirani sistem datoteka (distributed file systems – DFS) ima istu ulogu, koju međutim igra u znatno težem okruženju, kao i sami distribuirani operativni sistemi. Cilj DFS-a je da omogući korisnicima distribuiranog operativnog sistema deljenje podataka i deljenje resursa stalne memorije kroz jedinstveni sistem datoteka. Postojanje jedinstvenog sistema datoteka pruža **mobilnost korisnika**, odnosno pruža mogućnost da se korisnik prijavi na bilo kom čvoru sistema i normalno pristupa svojim podacima.

Idealan DFS eliminiše **probleme performansi, dostupnosti, i proširivosti** ovakvih sistema, i poštuje principe transparentnosti na način na koji to radi i distribuirani operativni sistem čiji je DFS deo. Performanse DFS-a se mogu posmatrati i kao još jedna dimenzija transparentnosti jer se DFS mora ponašati, ako ne isto, onda barem slično kao i klasičan sistem datoteka. Očigledna razlika u performansama koju je potrebno premostiti jeste vreme da zahtev korisnika stigne do fizičke lokacije gde je traženi podatak smešten kao i vreme potrebno da korisnik dobije odgovor. Različitim metodama keširanja informacija o podacima, putanja do podataka, i samih podataka ali i razvojem brzina internih mreža se u današnjim sistemima problem performansi potpuno isključuje.

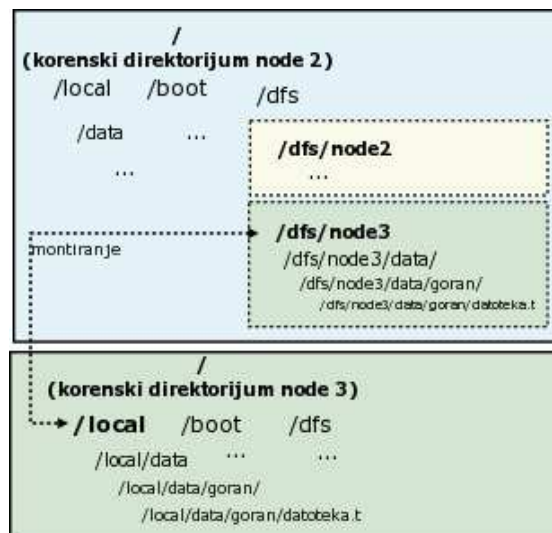
Otpornost na greške je bitna karakteristika DFS-a. Greške mogu nastati u komunikaciji, u hardveru čvora, bilo da je to čvor koji treba da primi ili čvor koji treba da pošalje informaciju u nekom trenutku, ali greške mogu nastati i pojavom nepredviđenog stanja u DFS-u te sam sistem mora biti dovoljno inteligentan da izađe iz zastoja koji na taj način mogu nastati. Idealan DFS omogućava neometani nastavak rada do neke granice, a

svakako ne dozvoljava gubitak podataka, nemogućnost kratkoročnog premošćavanja problema i sl. Za razliku od klasičnih sistema datoteka, distribuirani sistemi datoteka imaju veliku prednost u rešavanju problema otpornosti na greške jer je zbog velikih memorijskih resursa moguće čuvati kopije podataka i informacija o stanju sistema. Neodgovarajući dizajn sistema može pak potpuno sakriti ovu prednost do krajnjih granica.

Sistem datoteka mora da obezbedi interfejs za rad sa datotekama kao osnovnim jedinicama zapisanih podataka. Datoteka (file) sadrži same podatke i attribute koji određuju osobine datoteke i način na koji će joj korisnik preko operativnog sistema i sistema datoteka pristupiti. Osnovne operacije nad datotekom su kreiranje nove datoteke, pisanje u datoteku, čitanje datoteke, i brisanje datoteke. Dodatna operacija predstavlja premeštanje datoteke i ona može biti realizovana na nivou sistema datoteka ili na nivou programskog interfejsa operativnog sistema preko osnovnih operacija. Atributi datoteke (sigurnosne postavke, režimi rada, trenutno stanje) predstavljaju podatke o podacima ili **metapodatke**. Moderni sistemi datoteka ili dodaci za klasične sisteme datoteka koji još uvek nisu u upotrebi (GNOME Storage, Microsoft Yukon) proširuju metapodatke različitim informacijama koje omogućavaju indeksiranje, sortiranje i pretraživanje podataka.

Pojavi DFS-a je prethodila pojava **mrežnih sistema datoteka** koji su se koristili u mrežnim operativnim sistemima i pružali uslugu pristupa sistemu datoteka udaljenog računara, ali su to radili na za korisnika netransparentan način. Prvi distribuirani sistemi datoteka su zadržavali centralizaciju podataka koji su bili smešteni na posebnoj grupi čvorova označenih kao „data serveri“. Ovakav koncept nije omogućavao transparentnost distribuiranog sistema pa se u modernim distribuiranim sistemima pristup uređajima za čuvanje podataka poistovećuje sa ostalim resursima, odnosno postoji potpuna distribucija podataka. Ovo je naročito izraženo kod, još uvek teorijskih sistema datoteka koji rade na otvorenom klasteru mobilnih uređaja.

Na korisničkom nivou (u finalnoj reprezentaciji sistema) postoje dva modela DFS-a. U prvom, istorijski starijem modelu DFS nema jedinstvenu strukturu direktorijuma. Svaki čvor poseduje svoj lokalni sistem datoteka koji je vidljiv procesu koji se na njemu trenutno, u atomskim jedinicama izvršavaju. **Procesom montiranja** (engl. mount u Unix i map u DOS/Windows terminologiji) se vezuje korenski direktorijum sistema nekog drugog čvora sa direktorijumom u strukturi lokalnog sistema na čvoru koga posmatramo. Direktorijum se označava kao tačka montiranja, a veza između tog direktorijuma i posebnog sistema datoteka (najčešće modul u mikrokernelu čvora) koji obezbeđuje pristup korenskom direktorijumu drugog sistema se pamti u tabeli montiranih jedinica koja je smeštena u memoriji. Ovakav pristup koriste mnogi DFS-i kao što su Locus, NFS, AFS, Sprite, Microsoft CIFS i drugi. Sama operacija montiranja može biti automatizovana i sistemski uređena tako da se obezbeđuje transparentnost lokacije, relokacije i migracije podataka, ali to ipak najčešće nije slučaj.



Ilustracija 16 – Proces montiranja sistema datoteka sa node3 unutar sistema datoteka node2

Drugi, moderniji pristup, se zasniva na postojanju **globalnog stabla direktorijuma** (prostora imena), jedinstvenog na svim sistemima. Ovakav pristup obezbeđuje potpunu transparentnost, omogućava mobilnost korisnika, a daje i izvesne prednosti u mogućnostima odvajanja metapodataka od samih podataka i podršku za lakšu realizaciju keširanja podataka. Ostvarivanje veze između logičke reprezentacije lokacije datoteke (tekstualno ime datoteke) i fizičke lokacije datoteke na uređaju za čuvanje podataka se

odvija kroz **sistem imenovanja** (naming). Najčešće se tekstualno ime vezuje sa numeričkom oznakom (analogno memorijskoj adresi u deljenju memorije) koja se onda mapira na stvarne podatke. Ako poštujemo transparentnosti lokacije, migracije i relokacije preslikavanje imena datoteke na numeričku oznaku neće biti jednoznačno. Zapravo, kako će zbog poboljšanja performansi čvorovi koji su pristupali datoteci (i gde distribuirani operativni sistem ima razloga da sumnja da će procesi sa tog čvora ponovo pristupati datoteci) na čvorovima biti zadržana replikacija (kopija) datoteke, povratna informacija upita pretvaranja tekstualnog imena u numerički reprezent će biti set različitih odgovora sa svih čvorova koji poseduju replikaciju datoteke. Među njima ne sme postojati glavni čvor, čvor koji je vlasnik datoteke jer bi greškom u interakciji sa njim bila narušena funkcionalnost celog distribuiranog sistema datoteka i distribuiranog operativnog sistema u celini. Nakon dobijanja liste čvorova, zavisno od karakteristika (omiljeni čvor, čvor sa kojim je ostvareno najviše uspešne komunikacije na nivou DFS-a, čvor sa najbržim odgovorom i slično) proces mora odlučiti koji odgovor će koristiti. Distribuirani operativni sistem se brine o problemima replikacije te proces može smatrati da su sve kopije iste i jednake.

Kvalitetan sistem imenovanja i lociranja obezbeđuje i bolju proširivost. Ukoliko sistem može da istu datoteku dobije od više izvora to predstavlja raspoređivanje opterećenja u sistemu i sprečava kašnjenje usled preuzetosti jednog čvora.

Mnogi, danas korišćeni, distirbuirani sistemi datoteka ipak ne poštuju potpuno transparentnost lokacije (ime datoteke ne otkriva njenu fizičku lokaciju) i nezavisnost lokacije ili transparentnost migracije i relokacije (ime datoteke se ne mora promeniti kada se fizička lokacija smeštaja datoteke izmeni). Takvi sistemi su Locus, NFS⁹, Sprite, Microsoft CIFS¹⁰ i drugi. AFS (Andrew File System) pak na primer ima primitivan način ostvarivanja transparentnosti relokacije i migracije, ali ne i transparentnosti lokacije što izaziva veliku konfuziju. Naime u tekstualom imenu datoteke (sa kojim korisnik interaguje) je prisutna indentifikacija fizičke lokacije, ali se takva indentifikacija kroz svojevrsnu bazu podataka (volume manager) koja se replicira na sve klijente prevodi u stvarnu indentifikaciju čvora na kome se datoteka nalazi. Izmenom sloga u bazi, moguće je uraditi relokaciju na transparentan način, ali kako je korisniku i dalje vidljiva stara lokacija u imenu datoteke nakon većih izmena sve postaje isuviše komplikovano za održavanje. Drugi DFS-i kao što su Lustre i CODA su potpuno uklopljeni u distribuirani koncept.

Radi poboljšanja performansi, kao i kod deljenja radne memorije, distribuirani sistem datoteka vrši **keširanje** na različitim nivoima. Da li će keširanje biti u radnoj memoriji ili na disku, kako se rešava problem sinhronizacije svih keširanih datoteka, koje su jedinice keširanja (deo podataka, cela datoteka, sektor memorijskog uređaja, ceo memorijski uređaj), sve su to pitanja čiji odgovori određuju dizajn sistema za keširanje i replikaciju. Način implementacije keširanja direktno zavisi od **polise deljenja** podataka koja specifikira kako sistem reaguje na promenu datoteke koju trenutno koristi više procesa. Tradicionalna, **UNIX polisa deljenja** zahteva da se svaki upis u datoteku odmah prikaže i u svim trenutno otvorenim datotekama. Vrlo efikasna, ali i gruba i često neupotrebljiva polisa deljenja jeste da se deliti mogu samo datoteke kojima je upis zabranjen. Distribuciji prilagođenija, **sesijska polisa** zahteva da se upis u otvorenu datoteku tretira kao u UNIX polisi unutar lokalnog sistema datoteka, ali ne i na datotekama koje su otvorene preko DFS-a. Po zatvaranju datoteke u koju je vršen upis završava se sesija i keširane kopije datoteke se sinhronizuju. Ovo je u skladu sa POSIX standardima, ali u različitim trenucima postoji više različitih verzija iste datoteke, a uzastopnim upisima u njih dolazi do još većeg šarenja i potrebe za rešavanjem konflikata. Na kraju, postoji i **transakcijska polisa deljenja** koja rešava probleme sesijske polise, ali kreira i nove (prvenstveno u pitanju brzine). Ona

⁹ Network File System – standard za deljenje podataka na Unix sistemima datoteka. Najnovija je četvrta generacija opisana u RFC 3530 dokumentu (jun 2003.). Samu interakciju sa hardverom obavlja klasičan sistem datoteka (ext3, reiserFS,...), a pristup sa drugih klijenata se obavlja montiranjem (mapiranjem) NFS sistema unutar jednog direktorijuma u stablu lokalnog sistema te transparentnost nije podržana.

¹⁰ Microsoft CIFS, poznatiji kao SMB ili Samba sistem datoteka je standard za deljenje podataka na Microsoft Windows operativnim sistemima, a ima i implementaciju za Unix-olike sisteme. Ponaša se slično NFS-u, ali koristi drugačije protokole i sistem imenovanja.

zahteva da se upis u datoteku razmatra kao transakcija u bazama podataka. Pre upisa datoteka se zaključava tako da nijedan drugi proces ne može pristupiti toj datoteci za upis, a po završetku upisa sinhronizuju se replikacije datoteke i datoteka se otključava. Potrebno je obezbediti mehanizme odbrane u slučaju havarije u fazama dok je datoteka zaključana, kao i obezbediti da se zaključavanje datoteke odigra atomski nad svim njenim replikacijama.

Keširanje može biti na različitim nivoima. Mogu se keširati odgovori dobijeni od sistema imenovanja (takozvani hint). Keširani odgovor se ipak mora proveriti pre korišćenja (naravno, ne novim pozivom sistema imenovanja jer time ništa nismo dobili) jer je moguće da se dogodila relokacija u međuvremenu za koju ne znamo. Dalje, moguće je keširati metapodatke, same podatke, ali i takozvane **prljave podatke**, odnosno datoteka u koju se trenutno vrši upis zajedno sa novim podacima koji su upisani.

Posmatrajući polise deljenja, postoje različiti načini da se sistem keširanja odbrani od postojanja nesinhronizovanih podataka na sistemu. Postoje situacije kada je moguće dozvoliti postojanje neusklađenih datoteka, ali to onda nije distribuirani operativni sistem opšte namene jer sami procesi moraju voditi računa o keširanju datotekama. NFS ne garantuje sinhronizaciju podataka. Primitivni metod koji on koristi jeste vremensko ograničenje, odnosno svaka replicirana kopija će biti obrisana kroz 60 sekundi, dok će prljavi podaci biti obrisani za 30 sekundi. Generalno može se smatrati da keširanje ne postoji u NFS-u.

Za razliku od NFS-a koji implementira UNIX polisu deljenja, u AFS-u postoji keširanje datoteka na distribuirani način, a sinhronizacija se vrši pomoću callback poziva, odnosno koristi se sesijska polisa. Kada čvor zahteva deo datoteke, umesto prenosa traženog dela prenosi se cela datoteka kao osnovna jedinica keširanja i čuva na lokalnom disku ali i ostvaruje callback veza sa čvorom od koga je preuzeta datoteka. Prilikom upisa u neku drugu replikaciju datoteke, nakon zatvaranja sesije, lancem callback poziva se osigurava sinhronizacija na celom sistemu. Problemi koji se javljaju jesu vezani za prekid mrežne komunikacije, otkazivanje čvora koji treba da incijalno prosledi callback ili slično. Takve situacije se u AFS-u rešavaju vremenskim ograničenjem. Nova, treća verzija AFS-a umesto callback mehanizma koristi token mehanizam koji kombinuje proveru verzije datoteke sa sesijskom polisom deljenja.

Sprite sistem datoteka se na treći način sukobljava sa sinhronizacijom, koristeći transakcijsku polisu deljenja. Pri otvaranju datoteke za upis, blokira se keširanje te datoteke, a pre završetka transakcije osigurava se da na sistemu ne postoji zaostala keširana verzija datoteke. Svi procesi nakon toga moraju napraviti novu keširanu verziju datoteke koja se koristi do sledećeg upisa. Ukoliko je broj upisa prema broju čitanja mali, keširanje je vrlo efikasno, u protivnom, keširanje se minimalizuje jer je često neophodno ponovo dobiti keširanu datoteku od čvora na kome je u nju izvršen upis. Drugi problem je što ukoliko dođe do prekida komunikacije ili do kraha čvora na kome je izvršen upis, sve promene će biti izgubljene.

Svako keširanje se smatra i repliciranjem i samim tim pomaže u otpornosti na greške jer postoji više sinhronizovanih kopija podataka. Postoji mogućnost da se u trenutku sinhronizacije dogodi havarija i time unište sve replikacije, te se u praksi i sama sinhronizacija datoteka nakon završenog upisa odvija po modelu transakcije.

Da bismo imali stabilniji sistem sa boljom transparentnošću u slučaju pojave grešaka na različitim nivoima ne sme postojati kritična tačka sistema u kojoj se pamti stanje

Standardan pristup datoteci

```
01: FILE *fp = NULL;
02: char *ime = "datname.t";
03: int i;
04:
05: fp = fopen(ime, "w");
06: fscanf(fp, "%i\n", &i);
07: fclose(fp);
```

Pri sistemskom pozivu fopen, navodimo način otvaranja datoteke (u primeru je otvorena datoteka za upis) tako da DFS zna način pristupa datoteci. U šestoj liniji se radi sam upis, dok se u sedmoj datoteka zatvara novim sistemskim pozivom.

Očigledno je da baš sistemski pozivi fopen i fclose trebaju implementirati zaštitu od simultanog upisa kao i sinhronizaciju replikacija datoteke na sistemu.

sistema. Na primer, kada proces otvori datoteku sistemskim pozivom `fopen`, klasični sistem datoteka kao povratnu informaciju vraća pokazivač na datoteku koji služi kao indentifikacija zahteva od procesa ka datoteci i sve dalje interakcije su na taj način obeležene. Sistem datoteka u internoj memoriji pamti metapodatke o procesu i o datoteci (kao na primer pozicija u datoteci) koje koristi u procesiranju daljih zahteva. Ukoliko isto primenimo i na distribuirani sistem datoteka susrešćemo se sa problemom distribuirane greške. Do greške može doći na strani čvora koji prima zahtev i čuva podatke o zahtevu u memoriji kada se gubi svaka informacija o zahtevima koji će uslediti. Nemoguće je u tom trenutku konekciju migrirati ka replikaciji datoteke na drugom čvoru jer on nema informacije koje je imao prethodni čvor klastera. U povoljnijem ishodu, sistem za oporavak na strani klijenta će ponoviti zahtev, dok bi u drugačijem ishodu deo podataka bio izgubljen, a izvršavanje procesa bi bilo prekinuto. Obrnuto, ako bi se havarija desila na strani čvora koji postavlja zahtev, drugi čvor bi imao ostatke memorije koje ne bi mogao da ukloni (jer datoteka nije zatvorena) već bi to bio dodatni posao za takozvani sistem uklanjanja siročadi (engl. orphan detection and elimination). Stoga, ukoliko želimo stabilan, na greške otporan distribuirani sistem datoteka, čvor koji postavlja zahtev je taj koji mora čuvati stanje (jer njegovim krahom se prekida i sam proces) i prosleđivati ih prilikom svakog zahteva (kako početnog tako i svakog narednog) čvoru koji poseduje jednu od replikacija zahtevane datoteke. Greška u interakciji sa čvorom koji poseduje datoteku se lako rešava upućivanjem zahteva nekom drugom čvoru koji poseduje replikaciju datoteke. Tek ukoliko takvog čvora nema, aktivira se mehanizam oporavka koji ponavljanjem zahteva, uz vremenski uslov prekida može efikasnije da reši problem.

13. Primeri distribuiranih sistema datoteka

CODA sistem datoteka

Ovaj sistem datoteka implementira jedinstveno stablo direktorijuma koje se montira unutar lokalnog stabla svakog čvora, kao što je slučaj i kod AFS-a. Sistem je originalno kreiran za rad na GNU Mach mikrokernelu, ali sada ima zvaničnu podršku za Linux, NetBSD, FreeBSD, a podržava i druge Unix-olike sisteme. CODA sistem je sačinjen od nekoliko komponenti koje se nalaze na svakom čvoru - upravljač replikacija (Venus), server (Vice) i modula za kernel. Sistemski poziv za neku od operacija nad datotekom ili direktorijumom koji se nalazi unutar tačke montiranja sistema datoteka se upućuje CODA kernel modulu. On otvara standardni teletype uređaj za komunikaciju ka Venus-u koji se nezavisno izvršava u korisničkom prostoru.

Venus najpre proverava da li poseduje traženi objekat interakcije u lokalnom skladištu replikacija. Ukoliko ga ne poseduje, Venus kontaktira sistem imenovanja od koga saznaje listu čvorova koji poseduju replikaciju objekta. Na odabranom čvoru koristeći RPC2 protokol komunikacije uspostavlja vezu sa serverskom komponentom nazvanom „Vice“. Objekti se preuzimaju kao atomske jedinice (cela datoteka, odnosno sve informacije o direktorijumu) i smeštaju u lokalno skladište replikacija. Kada je objekat interakcije dostupan u skladištu replikacija, Venus o tome obaveštava CODA modul kernela koji novim sistemskim pozivom izvršava stvarnu akciju nad objektom.

Pored podele u datoteke i direktorijume, unutar jednog čvora distribuirane datoteke se dele u logičke celine - delove (engl. volume). Svaki „deo“ sadrži nekoliko direktorijuma i datoteka povezanih u logičku celinu (datoteke vezane za jedan projekat i sl.) koju određuje administrator. Oznaka „dela“ je osnovna jedinica montiranja preko koje se obezbeđuje od upada u ciklični graf putanje koja u sebi sadrži montirane direktorijume.

„Deo“ je osnovna jedinica replikacije datoteka. Prilikom replikacije „dela“, indentifikacija čvora se dodaje u listu čvorova koji poseduju replikaciju tog „dela“. Listom je

obeležena grupa čvorova označena kao VSG (Volume Storage Group). Prilikom izmena nad objektom, podaci se čitaju sa jednog čvora, a dostavljaju svim čvorovima iz VSG grupe „dela“ u kome se nalazi objekat koji se menja. CODA sistem datoteka podržava *multicast* (*unicast*) RPC komunikaciju i ovakva polisa upisa ne usporava rad sistema.

Venus može da detektuje i prekid veze kada se prebacuje u *offline* režim rada što je jedna od naprednih mogućnosti CODA sistema datoteka, izuzetno korisna u mobilnom okruženju, ali i u okruženju podložno havarijama na mreži kada se ova mogućnost ponaša kao komponenta sistema zaduženog za kontrolu grešaka i transparentnost pri greškama uopšte.

Mogućnost *offline* načina rada i agresivni model čuvanja replikacija dovodi do problema postojanja više verzija iste datoteke na sistemu (izmena datoteke u lokalnom skladištu na čvoru koji se privremeno nalazi van mreže), pa CODA sistem datoteka poseduje mehanizme za njegovo rešavanje. Kada je u normalnom režimu rada, po izmeni podataka u lokalnom skladištu replikacija Venus obavestava odgovarajući VSG skup. U *offline* režimu to nije moguće raditi pa se izmene čuvaju lokalno u bazi označenoj kao CML (Client Modification Log). Po uspostavljanju konekcije, Venus će poslati sadržaj CML-a VSG-ovima na čije datoteke se izmene odnose. Pri tome se neće izvršiti poništene izmene (datoteka koja je kreirana pa kasnije obrisana neće izazvati nikakvu akciju). Problem koji se dešava jeste što su se mogle dogoditi i druge izmene nad istim datotekama pa se promene iz CML-a ne mogu izvršiti. Na primer, čvor u *offline* modu kreira novu datoteku u jednom od direktorijuma čiju replikaciju poseduje. Operacija se upisuje u CML. Međutim, u vremenu *offline* rada drugi čvor kreira datoteku sa istim imenom. Po ponovnom uspostavljanju veze, akciju iz CML baze nije moguće izvršiti. Konflikta mogu biti i usled promene strukture direktorijuma, uklanjanja datoteke, ali i konkurentnih upisa u istu datoteku ili izmena istog direktorijuma. Neke situacije rešavaju Vice i Venus komponente odgovarajućih čvorova u direktnoj komunikaciji, dok je za neke složenije konflikte neophodna akcija korisnika ili procesa. Sam mehanizam na krajnjem nivou je vrlo sličan popularnom sistemu kontrole konkurentnih verzija datoteka - CVS-u.

CODA sistem datoteka je namenjen velikim distribuiranim operativnim sistemima sa nekoliko stotina do nekoliko hiljada klijenata u klasteru. Aktivno se koristi i razvija na Carnegie Mellon univerzitetu, a pomoć pruža i veliko udruženje korisnika. CODA sistem se može efikasno iskoristiti i na drugim poljima, za FTP i WWW *mirror*, kao rešenje za replikaciju bitnih podataka na klasičnim operativnim sistemima i sl.

Lustre sistem datoteka

Ovo je primer distribuiranog sistema datoteka nove generacije, namenjenog velikim distribuiranim sistemima, sa podrškom za desetine hiljada čvorova i memorijom koja se meri u petabajtima (*jedan petabajt ima 2⁵⁰ bajtova*), velikim prenosnim brzinama, sistem datoteka koji svojim dizajnom eliminiše probleme performansi, dostupnosti, proširivosti. Lustre sistem datoteka potpuno ostvaruje transparentnost lokacije, ne sadrži jedinstvenu tačku oslonca sistema, i ima visoku otpornost na greške.

Razvija se kao softver otvorenog koda (GNU GPL licenca), a pod pokroviteljstvom kompanije "Cluster File Systems, Inc." koja kao strateške partnere ima najbolje predstavnike kompjuterske industrije (Dell, Cray, HP i drugi).

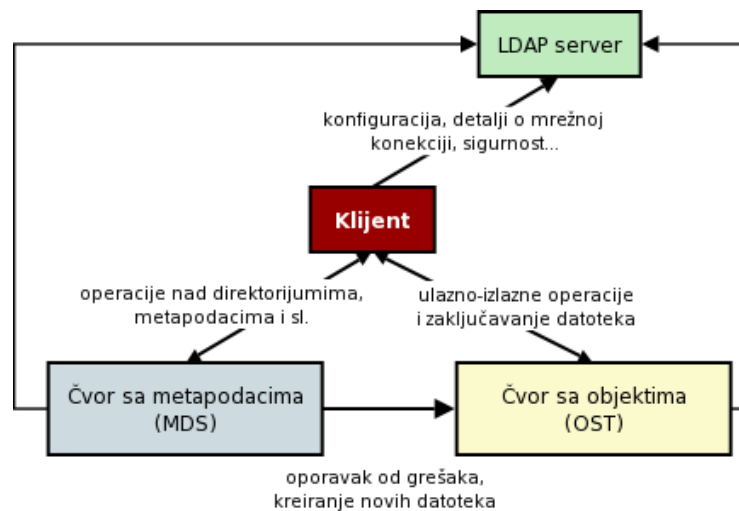
U novembru 2003. godine je izvršeno testiranje Lustre sistema datoteka u jednom od produkcionih okruženja gde se on danas koristi (Američka nacionalna agencija za razvoj superkompjutera) i ostvareni rezultati su impresivni. Operišući nad 1000. čvorova klastera, ostvaren je paralelni ulaz/izlaz pri brzini od 11,1 GB podataka u sekundi, uz ostvareno 90% konstantno opterećenje veze između svih čvorova!

Lustre sistem datoteka razdvaja fizičku lokaciju čuvanja podataka i metapodataka. Ulogu čuvanja podataka i metapodataka preuzimaju grupe čvorova, a generalizacijom je

moguće postići i potpunu distribuciju. Informacije o stanju datoteka na sistemu (pozicija pokazivača u datoteci, status zaključavanja datoteke i sl.) se čuvaju zajedno sa podacima i njihovim replikacijama čime se obezbeđuje bolje snalaženje pri greškama.

Sistem se ne oslanja samo na Ethernet mrežne slojeve, već postoji sloj za apstrakciju mreže iznad koga stoji Portals API, specijalni mrežni protokol. Na vrhu slojevitog modela se nalazi Lustre-ov protokol komunikacije. Kao i ostale komponente Lustre sistema i protokoli komunikacije su laki za integraciju u različita postojeća okruženja i standarde. Već pomenute informacije o stanju datoteka ali i informacije o samom Lustre DFS-u se čuvaju u standardizovanim formatima kao što su XML i LDAP što olakšava prelazak i dozvoljava korišćenje postojećih administracionih alata i po prelasku na Lustre sistem datoteka.

Osnova predstavljanja datoteke je objekat koji se beleži (kroz apstrakciju) na magnetne uređaje sa stalnom memorijeom. Objekat se locira i objektu se pristupa na osnovu informacija o opisu objekta sačuvanih na serverima koji čuvaju metapodatke (MDS). Ovi serveri podržavaju sve operacije nad prostorom imena datoteka, odnosno više operacije nad objektima kao što su kreiranje objekta, imenovanje i lociranje objekta, operacije nad atributima objekata i direktorijuma, itd., dok se same ulazno izlazne operacije obavljaju nad objektno-baziranim diskovima, odnosno danas kroz objektnu apstrakciju sa uređajima za spremanje podataka.



Ilustracija 17 – Tokovi komunikacije između komponenti Lustre sistema datoteka

Pojavom diskova koji na nivou hardvera podržavaju osnovne operacije sa objektima, takozvanih objektno-baziranih diskova (OBD) ubrzale bi se performanse Lustre sistema datoteka. Kompanija Cluster File Systems pregovara sa proizvođačima i pojava ovakvih uređaja bi se mogla očekivati u budućnosti. Trenutno Lustre emulira ovakve uređaje kroz specijalne kernel module u OST kernelima koji prave vezu između OBD interfejsa i sistema datoteka na klasičnim diskovima kao što su ex3, JFS, ReiserFS i XFS.

Razdvajanjem metapodataka i samih podataka (objekata) se pojavljuju novi slučajevi prestanka funkcionalnosti pojedinih delova Lustre distribuiranog sistema datoteka, pa se distribuirana LDAP baza sa listom dostupnih MDS ali i OST čvorova mora stalno održavati. Takođe razdvajanjem se mogu dobiti prednosti u dostupnosti sistema jer se mnoge operacije mogu obaviti interakcijom samo sa jednom grupom čvorova, a ulazno-izlazna interakcija se obavlja bez posrednika.

Poput AFS-a i CODA sistema datoteka, i Lustre koristi jedinstveni prostor imena, ali tačka montiranja virtualne hijerarhije datoteka nije predefinisana već se ona određuje pomoću konfiguracionih datoteka u samim direktorijumima koji predstavljaju tačku

montiranja sistema datoteka. Tako je moguće u Lustre sistem datoteka uklopiti postojeće tačke montiranja drugih sistema pa i drugih distribuiranih sistema datoteka, prepisati deo lokalnog stabla distribuiranim stablom i slično.

U ubrzanja koja će se dodatno postići daljim razvojem Lustre sistema datoteka spada podrška za asihroni upis u datoteke, naspram sadašnjeg zahtev-odgovor modela. Dalje, u planu je podrška za bolju distribuciju servera sa metapodacima koji su trenutno u redundantnoj grupi čvorova, umesto u ravnopravnoj distribuiranoj grupi.

Pristup objektima se uređuje POSIX ACL semantikom pristupnih lista koja je jako efikasna u okruženjima sa velikim brojem korisnika. Lustre bi u budućnosti trebao da potpuno podržava Kerberos5 i PKI mehanizme kao pozadinu autentifikacije. Transmisija podataka je obezbeđena automatskom enkripcijom/dekripcijom i metodom deljenih ključeva.

Dalji razvoj Lustre-a se kreće ka većoj distribuciji, ka povećanju količine keširanih i repliciranih vrsta podataka i količine samih takvih podataka. Takođe, za novu verziju je predviđena pojava prelomnih tačaka sistema (engl. breakpoint ili snapshot) na koje će se sistem moći vratiti u svakom trenutku. Delovi podataka za kreiranje prelomne tačke će biti ravnopravno raspodeljeni na klijente, OST i MDS servere.

14. Sigurnost i višekorisnički rad

Distribuirani operativni sistemi najčešće imaju veliki broj korisnika, veliku količinu važnih i poverljivih podataka, komplikovanu organizaciju pristupa velikom broju uređaja itd. Pod pojmom sigurnosti se podrazumeva nekoliko različitih stvari. Najpre, sigurnost podataka u distribuiranom sistemu datoteka, odnosno obezbeđenje da će podatke moći da vidi, koristi, izmeni i ukloni samo onaj ko za to ima dozvolu. Dalje, sigurnost podrazumeva i zaštitu radne memorije kako bi se sprečila nasilna promena stanja distribuiranog operativnog sistema ili procesa koji se na njemu izvršavaju od strane trećeg procesa. Pod istim pojmom se podrazumeva i uređenje korišćenja resursa i uređaja (prostor u memoriji, adresni opseg, procesorsko vreme, ulazno izlazne jedinice i sl.) od strane procesa i korisnika. Na kraju, sigurnost podrazumeva i zaštitu od umetanja stranog čvora u klaster kako bi se sprečila krađa informacija.

Osnovno pravilo u koncepciji bilo kog sigurnosnog sistema jeste ne verovati nikome i ničemu. Mehanizmi sigurnosti su enkripcija, autentifikacija, autorizacija i zapisivanje i ispitivanje (engl. auditing). Enkripcija je fundamentalni mehanizam sigurnosti nastao i pre pojave računarskih sistema. Pored onemogućavanja lakog pristupa samim podacima, može poslužiti i za proveru integriteta i verodostojnosti samih podataka. Autentifikacija je prvi preduslov za izvršavanje usluge. Može se posmatrati na nivou korisnika (prijavlivanje na sistem) i na nivou distribuiranog operativnog sistema kada postoji autentifikacija procesa i samih čvorova klastera. Nakon autentifikacije, potrebno je izvršiti proveru da li proces/korisnik/čvor ima pravo korišćenja usluge koju je zatražio što je proces sistema za autorizaciju. Na kraju, zapisivanje i ispitivanje se obavlja kroz analizu zapisa sistema (engl. log). Iako ne predstavlja nikakvu stvarnu sigurnosnu zaštitu ovi zapisi (i njihova analiza) mogu pomoći za spoznaju načina upada u sistem i pružaju eventualnu mogućnost za otkrivanje počinilaca.

Za obezbeđivanje sigurne komunikacije često nije dovoljna samo enkripcija, već i autorizacija puta poruke pre slanja podataka. Postoje situacije kada je dovoljno imati informaciju o količini podataka, bez potrebe uvida u podatke. Karakterističan primer koji se često navodi u literaturi jeste sledeći slučaj: Za vreme nove svetske krize, količina informacija ka Beloj Kući se smanjuje, dok se istovremeno uvećava količina informacija

upućena na lokaciju u Koloradu. Iako su poruke pravilno enkriptovane, sama činjenica o postojanju komunikacije u ovom primeru isuviše mnogo govori.

Sama enkripcija može biti simetrična, kada je ključ za šifrovanje i dešifrovanje poruke isti ili asimetrična kada se oni razlikuju, ali zajedno čine jedinstveni par. Vrlo koristan je i metod jednosmerne enkripcije (kada ne postoji mogućnost mogućnosti dekripcije). Koristi se prilikom verifikacije samih podataka. Odlika funkcije za ovakvo šifrovanje jeste mala podudarnost, tj. zanemariva verovatnoća da će $f(m)$ biti isto što i $f(n)$, za $m \neq n$.

Van okvira ovog rada je obrada konkretnih algoritama šifrovanja i dešifrovanja jer bi to zahtevalo preveliki odlazak u širinu. Napomenimo samo poznate algoritme kao što je DES za simetričnu enkripciju, RSA za asimetričnu i izuzetno često korišćeni MD5 algoritam jednosmerne enkripcije. Svi pomenuti algoritmi su definisani RFC dokumentima, a njihove implementacije postoje na svim programskim jezicima. Često se praktičnosti radi autentifikacijom najpre otvara sigurni kanal komunikacije, a zatim kroz njega šalju informacije bez potrebe za složenijim algoritmima enkripcije.

Spoljne karakteristike sistema sigurnosti (dozvola korišćenja podataka i datoteka) je lako implementirati kada je prisutna sigurna komunikacija na nivou distribuiranog operativnog sistema kao i načini za utvrđivanje indentiteta čvora i procesa. U toku implementacije najpre se deskriptor procesa proširuje za oznaku korisnika i grupe kojoj korisnik pripada. Dalje, deskriptor deljenog resursa (ne uključujući i distribuirane stranice memorije) se proširuje kako bi obuhvatio povezanu listu korisnika koji mogu pristupiti resursu, kao i spisak grupa korisnika. U ovoj informaciji se sadrži i pravo pristupa koje se upoređuje sa zahtevanom operacijom i onda tek ako je zahtev validan sistem prelazi na izvršavanje operacije.

Zaštita memorije se ne vezuje za korisnika već za proces. U deskriptoru stranice memorije se dopisuje informacija – veza sa deksriptorom procesa koji je memoriju alocirao. Ukoliko DOS podržava deljenu memoriju, zapisuje se povezana lista deskriptora i niti koji joj mogu pristupati. Na osnovu ove informacije procesima se ne dozvoljava pristup memoriji koju nisu oni alocirali ili im nije data eksplicitna dozvola da mogu da pristupe toj memoriji.

Same informacije o stanju prijave korisnika, pomenute proširene informacije deskriptora i sl. se takođe repliciraju i keširaju na sistemu, što je slučaj i sa samim i javnim i tajnim ključevima čvorova. Ove informacije i pored validne autentifikacije čvorova ipak ne mogu biti distribuirane potpuno slobodno već se poverljive informacije repliciraju samo na pojedine grupe čvorova, a sam način tretmana ovih informacija je različit od sistema do sistema.

Kada se govori o industrijski priznatim i podržanim sistemima (i standardima) sigurnosti koji uključuju sve pomenute komponente treba svakako pomenuti Kerbreos i SESAME. Ipak, upuštanje u dizajn ovih sistema je isuviše nezahvalno zbog glomaznosti samih sistema. O sistemima postoji velika količina literature i moguće je detaljno proučiti njihov rad bez potrebe za preterani odlazak u širinu u ovom radu.

15. Dizajn Amoeba sistema

Prvi distribuirani operativni sistemi su se pojavljivali kao prilagođene verzije već postojećih mrežnih operativnih sistema napravljeni za potrebe konkretnih ciljeva, za rad na konkretnim klasterima. Takvi sistemi nisu imali dovoljnu otvorenost, skalabilnost niti transparentnost.



Pojavom distribuiranih sistema široke namene i razvojem modela distribuiranog programiranja, pojavom prvih specifikacija standarda komunikacije, imenovanja i distribucije pojavljuju se i distribuirani operativni sistemi nove generacije.

Nakon Multix-a i njemu sličnih sistema, u drugoj polovini osamdesetih godina se pojavljuje Amoeba – distribuirani sistem nove generacije koji se aktivno razvija do današnjih dana. Sistem poseduje biblioteke za emulaciju POSIX specifikacije tako da se na njemu mogu ponovo prevesti programi kompatibilni sa njom i normalno izvršavati kao na ostalim Unix sistemima.

Amoeba sistem, po rečima autora distribuirani operativni sistem nove generacije koji se korisnicima uspešno predstavlja kao klasični operativni sistem, potpuno skrivajući samu distribuciju, je baziran na distribuiranim objektima. Za komunikaciju koristi RPC (Remote procedure calls). Sistem datoteka (FileSystem) je odvojen i posebno distribuiran, tako da može doći do uskog grla u komunikaciji između čvorova i sistema datoteka. Međutim, procesi na distribuiranim operativnim sistemima nemaju veliko I/O opterećenje, a Amoeba poseduje mehanizme koji takve procese izvršavaju lokalno.

Amoeba sistem je nastao pre pojave CORBA i IDL specifikacija, ali on koristi sličnu specifikaciju za definiciju distribuiranih objekata – AIL (Amoeba interface language).

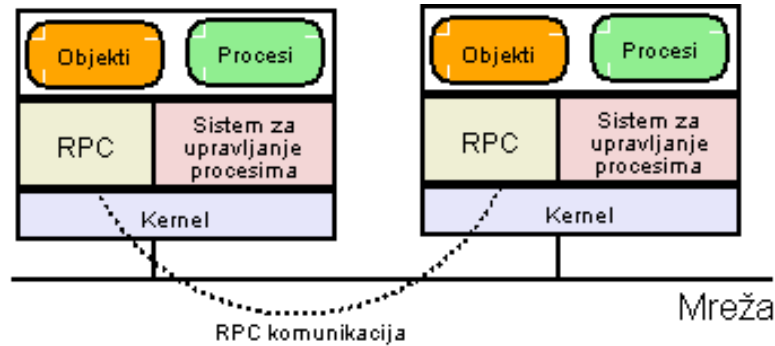
Sistem koristi mikrokernel dizajn kernela, gde kernel poseduje samo minimalne sisteme – upravljanje memorijom, kreiranje procesa i niti, komunikaciju procesa i upravljanje lokalnim resursima. Kreiranje udaljenih procesa se vrši preko sistema za upravljanje procesima koji obezbeđuje kreiranje udaljenih procesa, ispravljanje grešaka (debugging), postavljanje prelomnih tačaka izvršavanja (breakpointing), migraciju procesa i sl. Svi ostali delovi distribuiranog sistema se izvršavaju kao korisnički procesi što pojednostavljuje dizajn sistema, bez dodatnih gubitaka u performansama.

Izuzev emulacije POSIX specifikacije, Amoeba sistem ne kopira ni jedan drugi operativni sistem i time izbegava da zbog pokušaja implementacije dizajna sistema druge namene ne uspe u svom osnovno cilju – stvaranja kvalitetnog i dovoljno brzog distribuiranog operativnog sistema opšte namene.

Svaki poziv za izvršavanje određene metode objekta se obavlja preko RPC toka komunikacije. RPC model kod Amoeba sistema je vrlo jednostavan i sastoji se od tri sistemski poziva: `do_operation`, `get_request` i `send_reply`. Prvi sistemski poziv koristi proces koji zahteva korišćenje udaljene procedure. Implementacijom se poziva sistem imenovanja, šalje zahtev, verifikuje potvrda prijema zahteva, i po prispeću odgovora isti dostavlja procesu kao povratna vrednost. Drugi koristi proces koji želi da primi zahtev, a treći služi za slanje odgovora nakon obavljene operacije.

Dakle, klijentski proces ili nit procesa koji poziva distribuirani proces šalje zahtev lokalnom sistemu procesa (sistemski poziv `do_operation`) koji pokušava da locira čvor na kome se nalazi zahtevani proces (proces koji je pozvao poziv `get_request`). Ukoliko sistem ne zna tu informaciju, on će poslati broadcast paket na koji će odgovoriti sistem procesa koji poseduje dati objekat ili više njih kada će se izbornim algoritmom odabrati jedan. Informacija dobijena od sistema imenovanja se pamti u kešu u memoriji. Sada, kada je poznata lokacija objekta, šalje se RPC upit sačinjen od zaglavlja fiksne dužine i samih parametara za poziv metode. Udaljeni čvor prima upit, poziva proces sa dobijenim parametrima i nakon izvršavanja vraća odgovor sa povratnim vrednostima, eventualno izmenjenim prvobitno prenetim parametrima (sistemski poziv `send_reply`). Proces ili nit procesa koja je pozvala distribuirani proces dobija povratne vrednosti kao da se metoda nalazila lokalno.

Da bi se dodatno olakšalo korišćenje razvijen je i poseban više RMI orijentisan interfejs koji dozvoljava interakcije sa objektima.



Ilustracija 18 – Arhitektura Amoeba sistema

Prilikom poziva `get_request` proces navodi priključak (port) na koji treba uputiti `do_operation` sistemski poziv. Port je 48-bitna vrednost i poznat je celom sistemu samo za neke sistemske procedure (kao što je interakcija sa DFS-om) dok je za ostale procedure nepoznat. Poznavanje porta je prvi uslov verifikacije prava poziva procedure. Drugi nivo sigurnosti u Amoeba sistemu dodatno zabranjuje interakcije sa pojedinim objektima, bilo da je ona direktna ili preko distribuirane procedure. Port ipak ne može biti jedini mehanizam autentifikacije procesa jer proces može pokrenuti svoju distribuciju na portu koji je dat DFS-u pokušavajući da se predstavi kao DFS ostatku sistema. Zato Amoeba implementira specijalno hardversko rešenje za enkripciju portova po principu razmene javnog i tajnog ključa (F-box) kojim se uređuje koji proces može koristiti određeni port.

Ovo rešenje enkripcije porta se svodi na postojanje dva reprezentata samog porta, gde se drugi (P) izračunava preko javno poznatog preslikavanja prvog reprezentata (G) koje ima osobinu da nije moguće napraviti inverzno preslikavanje (hash algoritam enkripcije, poput popularnog RSA MD5 algoritma). Čvor koji pruža neku proceduru prilikom prijave dobija vrednost G i izračunava vrednost P. Vrednost G se čuva skriveno od ostatka sistema, dok se potencijalni klijenti obaveštavaju slanjem vrednosti P. Parametar poziva `get_request` je tajni ključ G, gde F-box izračunava javni ključ P i čeka poruku za port P. Kada poruka stigne, preusmerava se na proces koji pruža uslugu deljenja procedure. Kako je G strogo čuvan podatak, ni jedan zlonamerni proces ne može koristiti isti port, niti ga na osnovu javnog porta može saznati. Svaki pokušaj zlonamernog procesa da osluškuje poruke za javni ključ P će rezultirati osluškivanjem za vrednošću preslikavanja vrednosti P u P', što je beskorisno. U daljem razvoju će se ovaj metod autentifikacije čvorova i procesa proširiti na podršku za digitalne potpise, odnosno javni i tajni ključ za klijentske procese Amoeba distribuiranog sistema.

Sistem datoteka Amoeba sistema koristi strukturu usmerenog grafa za prikaz direktorijuma u globalnom prostoru imena koji u hijerarhiji sadrži i lokalni sistem datoteka montiran u jednom direktorijumu. Direktorijum je reprezentovan kao objekat, a pristupa mu se RPC pozivima za ulazak, pregled i brisanje. Uređivanje pristupa se obavlja pristupnim listama u korisnik/grupa notaciji. Objekat direktorijuma može sadržati reference na druge objekte čime se formira pomenuti usmereni graf bilo cikličnog ili razgranatog tipa. Pored referenci na direktorijume, objekat direktorijuma sadrži i referencu ka povezanoj listi referenci na datoteke. Amoeba-in sistem datoteka ne podržava upis u postojeće datoteke tako da ne postoje problemi prilikom replikacije. Kada je potrebno kreirati novu datoteku, proces zahteva kreiranje novog objekta, zatim dostavlja podatke i dodeljuje referencu novog objekta u povezanu listu nekog direktorijuma. Metapodaci su smešteni unutar objekta i kreiraju se automatski.

Proces u Amoeba sistemu se ne može simultano izvršavati na više od jednom čvoru. Deskriptor procesa sadrži indentifikaciju domaćina, interfejs ka sistemu imenovanja ako je proces distribuiran, deskriptore svih niti koje je proces pokrenuo kao i povezanu listu segmenata memorije koje proces trenutno drži alocirane. Distribucija memorije jednog procesa nije podržana. Kernel koji se ne nalazi u indentifikaciji domaćina u deskriptoru

procesa ili nije u grupi koja je navedena ne može izvršiti taj proces niti može izvršiti njegovu lokalnu replikaciju. Deskriptor niti dalje ukazuje na pokazivač na stek, vrednosti registara, stanje sistemskih poziva i sl. Iako memorija nije distribuirana, alocirani segment memorije naveden u deskriptoru procesa može se replicirati na drugi čvor koji je naveden u indentifikaciji domaćina procesa ili na njega premestiti ako se vrši migracija procesa.

Što se stanja procesa tiče, proces može biti zaglavljen ili pokrenut. Proces može biti zaglavljen spoljnim ili ličnim zahtevom. Pri migraciji procesa on se prvo zaglavljuje, zatim kreira njegova replikacija kao i replikacija svih niti, memorijskih segmenata, stanja i steka. Na kraju originalni proces biva ubijen čime se završava migracija.

Iako po mnogim stvarima neobičan sistem, sa manjkom podrške za na, primer virtualnu memoriju, mogućnost parcijalnog upisa u datoteke na DFS-u, podršku za UNIX emulaciju na binarnom nivou, pristupne liste naspram objektno opisanih mogućnosti i sl., Amoeba kernel je jednostavan i mali, jer ne prilagođava distribuirano okruženje klasičnim operativnim sistemima već funkcije OS-a prilagođava okruženju.

16. Dalji razvoj

Usko grlo današnjih distribuiranih operativnih sistema opšte namene jeste samo njihovo okruženje koje po brzini prenošenja informacija između komponenti ne može da se meri sa unutarprocesorskom komunikacijom. Stoga se razvoj distribuiranih operativnih sistema danas odvija na dva polja – hardverskom gde se stvara nova, brža mrežna infrastruktura i softverskom gde se stvaraju bolji metodi komunikacije, kvalitetniji i efikasniji protokoli sa boljom kontrolom grešaka i sa manjom veličinom gubitaka po svakoj poruci kroz zaglavlja, kontrolne pakete i sl.

Razvoj veštačke inteligencije omogućava bolju primenu heuristike u odlučivanju vezanom za migraciju procesa pri distribuciji procesorskog vremena dok praksa koja se do sada stekla i stiče u distribuiranim operativnim sistemima sa praktičnom primenom omogućava bolji odabir karakterističnih vrednosti. Primenom dinamičkog odabira vrednosti pojedinih faktora kroz genetske algoritme i automatsko učenje dodatno obezbeđuje optimalnije performanse.

Iako veoma mlada grana u razvoju sistemskog softvera, ovakvi sistemi se mogu već sada pohvaliti dobrom upotrebom vrednošću, a uspešno se primenjuju u brojnim naučnim istraživanjima za komplikovana i procesorsko-memorijski izuzetno zahtevna izračunavanja, analize i obrade rezultata.

Distribuirani operativni sistemi nisu opterećeni kočnicom kompatibilnosti jer predstavljaju potpuno novi način razmišljanja pri dizajniranju sistemskog softvera te se u narednom periodu mogu očekivati izvanredni, prvo teorijski, a onda i praktični prodori u ovoj oblasti.

Mišljenje autora ovog rada je da će zbog kvaliteta u dizajnu (proširivost, otpornost, geografska nezavisnost delova i modularnost, a ipak visoka transparentnost) i samoj ideji distribucije resursa unutar virtualnog sistema, distribuirani operativni sistemi svakako predstavljati još dugo vremena vrh razvoja sistemskog softvera kako na klasičnim arhitekturama tako i na mobilnim uređajima koji bi činili heterogeni otvoreni klaster uz obezbeđivanje resursa na zahtev omogućavajući njihovo korišćenje svima u svakom trenutku.

Popis korišćene literature

[1] Andrew Tanenbaum i Maarten van Steen:

Distributed systems – principes and paradigms, Prentice Hall, 2002.

[2] Raphael Finkel:

An Operating System vade mecum, University of Wisconsin at Madison i Prentice Hall, 1988.

[3] David Hulse i Alan Dearle:

Trends in operating systems design, University of Stirling, 1998.

[4] Andrew Tanenbrawn:

Modern Operating Systems, Prentice Hall, 2002.

[5] OSKIT – The Flux Operating system toolkit 0.97, University of Utah, 1997-2002.

[6] Cornelius Frank:

Compiling kernel using C compiler, 2000.

[7] The Amoeba Distributed operating System User Guide, Amoeba project, 1996.

[8] Intel Architecture Software Developer's Manual Volume 1 (Basic architecture), Intel, 2003.

[9] Pei Cao:

Distributed File System: Design Comparisons, transkript predavanja

[10] Eliezer Levy i Abraham Silberschatz:

Distributed File Systems: Concepts and Examples, University of Texas at Austin, ACM Computing Surveys, Decembar 1990.

[11] Lustre whitepaper, Cluster File Systems Inc., 2003.

Potpis kandidata-učenika
