Izvršio:
SRCE, Sektor za operacijske sustave, Dinko Korunić

Predmet:
Komparativno testiranje Sun Solaris 8 i Sparc Linux 2.4.20 platforme

Uvod:
Sun Solaris se, na produkcijskim strojevima u CARNet ustanovama posljednjih godina, pokazao proizvodom sa raznim nedostacima. Sporosti pridolaženja sigurnosnih zakrpa predstavlja sigurnosni problem, ne samo za odredišnu ustanovu, već i za ostatak mreže. Nadalje, Solaris za ustanove se već godinama softverski prilagođava CARNetovim potrebama kroz Debianizirane pakete i sl. Postavlja se pitanje da li je Linux na Sparc (preciznije, UltraSparc) arhitekturi sposoban zamijeniti dosadašnje instalacije Solarisa. Uz same jasne prednosti Linuxa poput slobodnog i izmjenjivog GNU koda, integriranog paketnog filtera (Netfilter), sigurnosnih jezgrinih ojačanja (SELinux, Grsecurity, LIDS), itd. neizostavno je spomenuti i činjenicu o ogromnoj arhivi unaprijed pripremljenog softvera koji Debian kao distribucija ima za Sparc Linux arhitekturu i što sistemcima pojednostavljuje posao u slučaju potrebe za nepodržanim paketima. Jasno, predstavlja pojednostavljenje i Grupi za izradu paketa budući da je razlika između već standardno podržanog Debiana za i386 i onog za Sparc minimalna i praktički jedina u jezgri sustava. Samim time se smanjuje potrebno vrijeme za razvoj i testiranje paketa i maksimira produktivnost, uz već spomenute sigurnosne i ine prednosti Linuxa. No, da li nam performanse Linuxa to omogućavaju i kakvo je ponašanje sustava po opterećenjima će pokazati idući set ekstenzivnih testova.

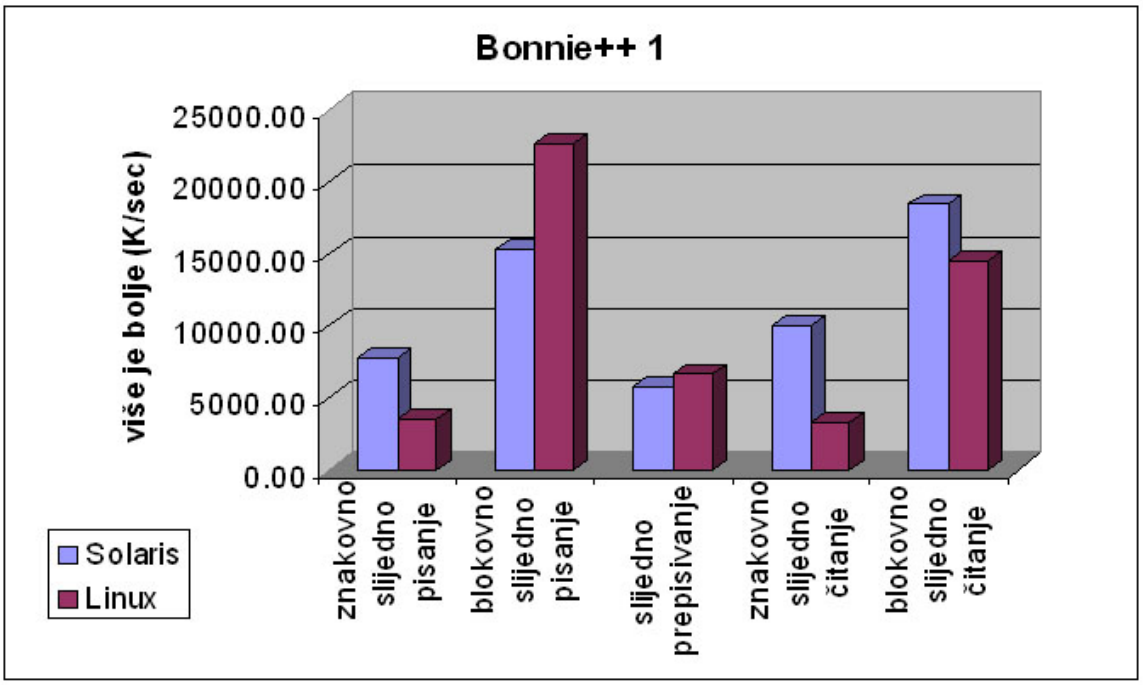
O testovima:
Izvršeno je 2200 testova na svakoj pojedinačnoj platformi (Solaris 8 i Sparc Linux 2.4.20 platformi), uz dodatnih 520 testova na samoj Sparc Linux platformi u svrhu testiranja performansi Solaris emulacije pod Sparc Linuxom, što je u ukupnosti 4920 testova. Svi rezultati su statistički obrađeni i tražilo se minimalno statističko odstupanje testova ili su oni bivali odbačeni pojedinačno ili u potpunosti. Redom, testove možemo podijeliti u slijedeće skupine:

- Testovi procesorskih performansi i performansi memorijskog podsustava,
- Testovi mrežnih performansi,
- Testovi performansi standardnog datotečnog podsustava,
- Testovi ponašanja jezgrinih sustava i operacijskog sustava općenito,
- Aplikativni testovi (SMTP i baza podataka).

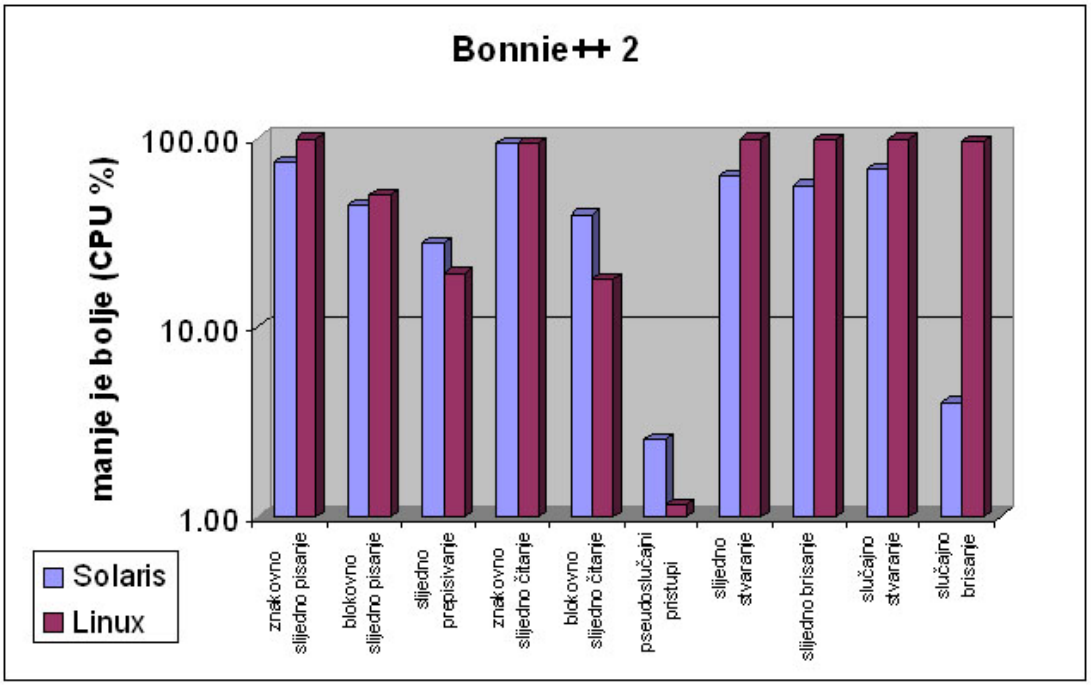
Testovi su odabrani na način da pokrivaju standardni set ponašanja sustava i da usporedo pokažu potencijalne prednosti i mane pojedinog dijela sustava. Korištene su posljednje inačice u Unix svijetu renomiranih i provjerenih testova, redom: LMBench (jezgra, CPU, RAM, I/O, itd.), Postal (SMTP), DBHammer (baze podataka), NBench-Byte (CPU, RAM), Netperf (TCP, UDP), Bonnie++ (disk I/O).
Testirano računalo bilo je Sun4U arhitekture, specifično Sun Workstation Blade 100 sa slijedećim komponentama: UltraSparc IIi procesor na 500MHz, 192MB RAM te IDE diskom od 20GB. Jasno, to računalo predstavlja low-end radnu stanicu odnosno poslužitelj u nižim kategorijama i svojim performansama zasigurno obuhvaća i Ultra5 (i možebitno zaostale Ultra1) računala po ustanovama.

Rezultati:

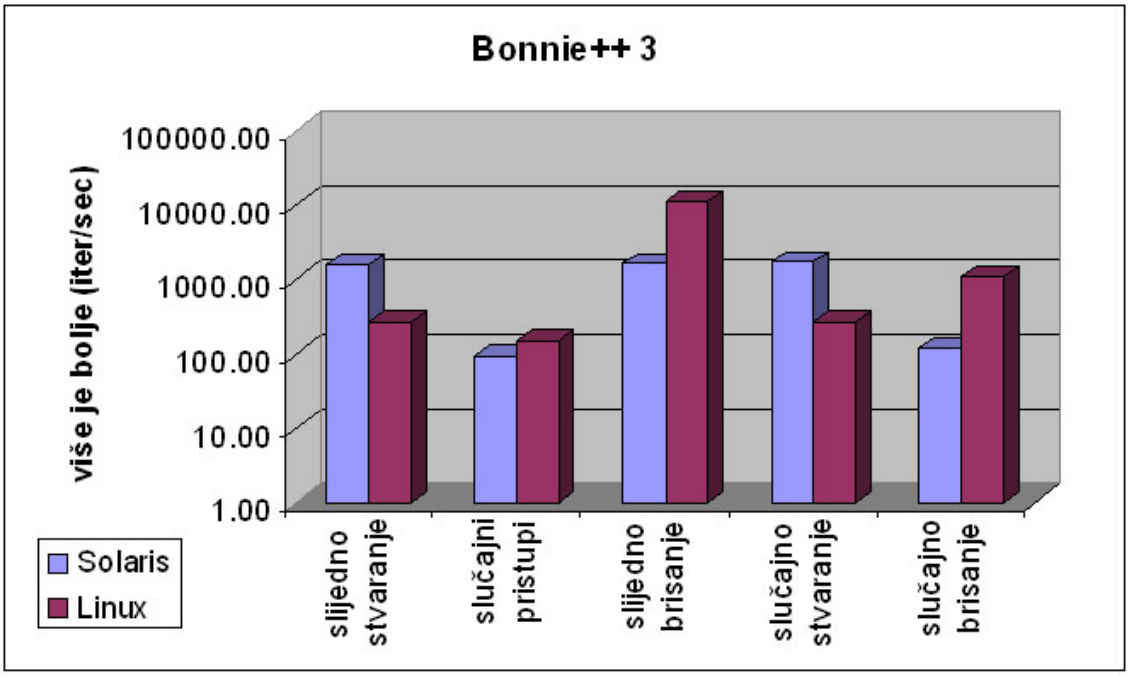
1) Krenimo prvo sa diskovnim podsustavom: Za njega je korišten Bonnie++ program u verziji 1.03a. Njegov autor je Russell Coker, a program je C++ nadograđena verzija starog Bonnie programa autora Tima Braya. Njegova primarna funkcionalnost je, kao što je već rečeno, testiranje diskovnog podustava korištenjem različitih datotečnih testova. To su, redom:
   a) Slijedno pisanje u datoteke
      i) Znakovno orijentirano:
         U datoteku se piše korištenjem putc() makroa iz stdio grupe poziva. Riječ je o vrlo čestom i praktički standardnom načinu za ispis znaka bilo na konzolu bilo na proizvoljan tok. Ovdje se mjeri vrijeme potrebno operacijskom sustavu da izvede stdio kod zajedno sa vremenom koje je potrebno jezgri da stvori prostor za datoteku.
      ii) Blokovski orijentirano:
         Koristi se write() poziv, standardni poziv za binarni ispis bloka podataka iz memorije. Vrijeme potrebno za ovaj proces je obično samo vrijeme koje jezgra provede alocirajući prostor za datoteku.
      iii) Prepisivanje:
         Svaki blok podataka se čita sa read() naredbom, obriše i prepiše ponovo sa write() naredbom. Naravno, koristi se repozicioniranje u datoteci pomoću lseek(). Budući da se ovdje generalno ne koristi nikakvo dodatno alociranje prostora i kako je I/O potpuno lokaliziran, u stvari se mjeri efektivnost međuspremničkih algoritama kao i podatkovni prijenos.
   b) Slijedno čitanje iz datoteka
      i) Znakovski orijentirano:
         Datoteke se čitaju koristeći samo getc() makro. Očito je da se koristi samo stdio kod i jednostavni slijedni ulaz podataka.
      ii) Blokovski orijentirano:
         Datoteke se ovdje čitaju koristeći read() poziv, pa testiranje predstavlja jednostavan test performanse unosa/čitanja podataka.
   c) Pseudoslučajni pristupi datotekama:
      Ovaj test pokreće standardno 3 paralelna lseek() procesa koji svi ukupno učine 8000 iteracija na datoteke koje specificira random() poziv. Nadalje, svaka datoteka se čita sa read() i nanovo piše sa write(). Ovdje se mjeri i ponašanje međuspremničkih algoritama kao i ponašanje diskovnog sustava pod većim opterećenjima kakva se sreću u višekorisničkim okolinama.



U ovom prvom setu testova su rezultati u znakovnom slijedom čitanju i pisanju na strani Solarisa, dok je kod blokovnog situacija potpuno na strani Linuxa. Primijetite razliku kod blokovnog slijednog pisanja koja je uvjetovana najvjerojatnije inteligentnijim međuspremničkim algoritmima kod Linuxa. Dapače, kako se čitanje individualnih znakova iz datoteka u prosjeku rjeđe dešava (podaci se danas skoro svi čitaju i pišu blokovski i međuspremnički), jasno je da ovi testovi daju donekle izjednačen rezultat – čak i sa efektivnom prednošću Linuxa u praksi.



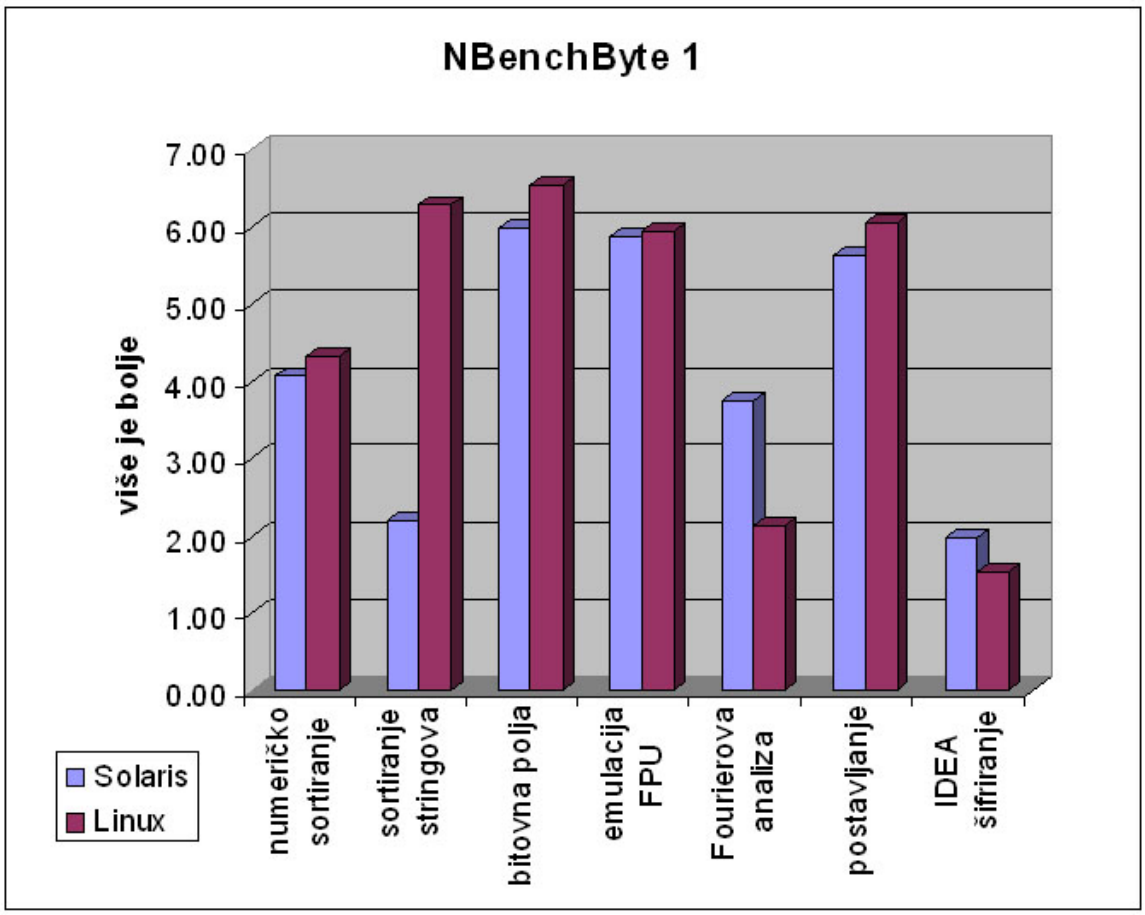
Prosječno gledano, Solaris i Linux su procesorski vrlo slično zahtjevni u baratanju datotekama. To naizgled nije očito iz gornjeg grafa – no njega treba promatrati u svjetlu prvog i zadnjeg grafa u kojima se vidi da brži rad datotečnog podsustava ujedno znači i nešto veće procesorsko opterećenje u jedinici vremena. Ogromna razlika u pseudoslučajnom brisanju nažalost reflektira vrlo velike razlike u brzini obavljanja iste zadaće u korist Linuxa (donji graf je u logaritamskoj skali, pa veće promjene u gornjem području znače prilične razlike). Osim toga, impresivno je i vidjeti da pseudoslučajni pristupi kod Linuxa ne samo da su brži, već i bitno manje opterećuju procesor. Kako je ovdje testiran ext3 datotečni sustav koji od svih Linuxovih datotečnih sustava slovi za najsporiji, možemo samo konstatirati da bi se vrlo velika dodatna ubrzanja pod Linuxom mogla dobiti biranjem nekog od novijih i bržih journaling datotečnih sustava poput SGI-jevog XFS-a i inih. U svakodnevici se Linux pokazuje boljim zbog kvalitetnijeg ponašanja u najgorim slučajevima – kad su datoteke raspršene, a mnogo korisnika/procesa paralelno pristupa različitim datotekama.



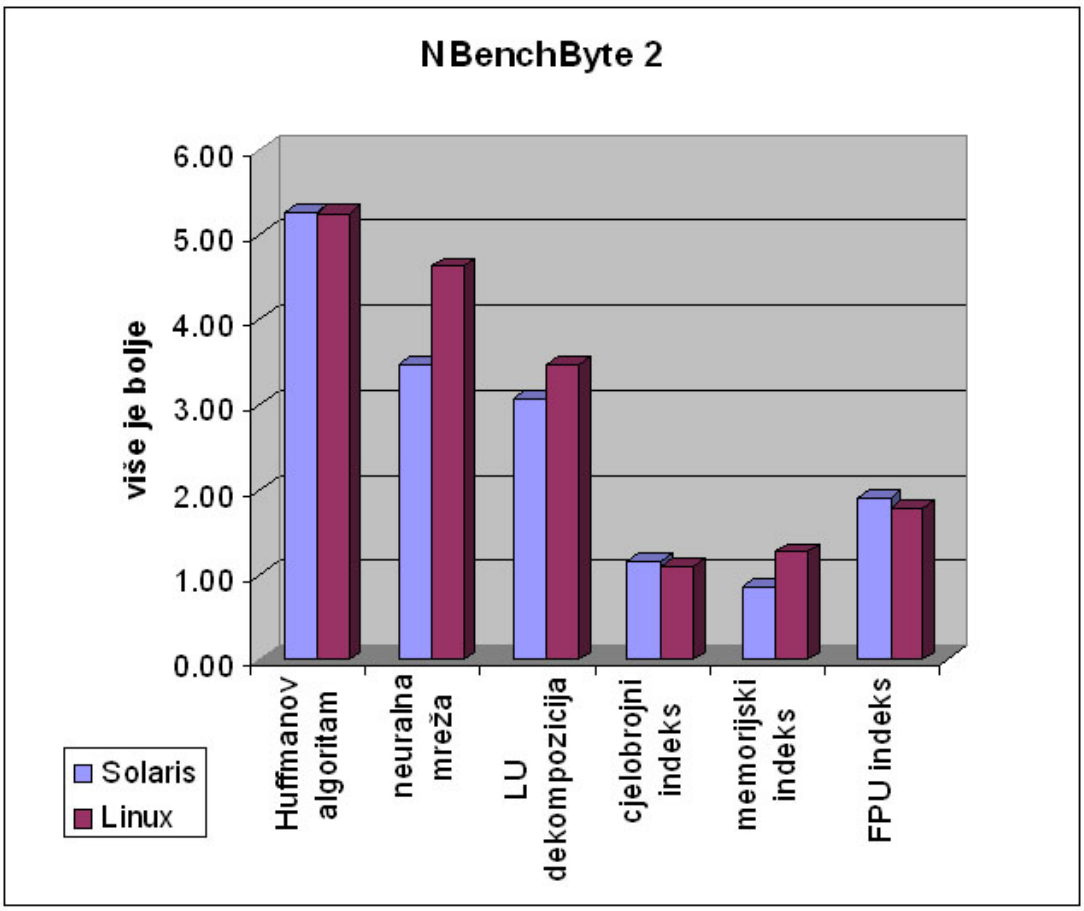
2) Slijedeći test je NBench (BYTEmark), test performansi procesora. Riječ je o verziji 2 (2.2.1) testa nekadašnjeg BYTE informatičkog časopisa koja je prenesena na Unixoide, pa i Linux. Sami testovi su dizajnirani tako da omogućavaju ekstenzivno testiranje CPU i FPU-a kao i memorijskog sustava. Testovi su sastavljeni tako da pokrivaju vrlo poznate i prokušane algoritme, stoga ih navedimo redom:
   a) Numeričko sortiranje:
      Sortira se polje 32-bitnih cijelih brojeva – što mjeri generičke performanse procesora u radu sa cijelim brojevima. Ovakva vrsta testa bi trebala izmjeriti i ne-slijedne performanse procesorskog međuspremnika (ne memorije, budući da su L1 i L2 međuspremnici veći od 8K).

b) Sortiranje niza znakova:
   Sortira se polje nizova znakova različitih dužina, što je u praksi mjerenje performansi premještanja blokova memorije. I ovaj algoritam mjeri ne-sekvencijalne performanse međuspremnika. Jasno, algoritam je kompliciraniji zbog problema da su pomaci (selidbe u memoriji) bajtovne širine i da se mogu desiti na neparnim adresnim granicama.
c) Bitovno polje:
   Izvršava se niz bitovnih operacija nad poljima bitova, što je standardno mjerenje bitovnih performansi procesora. Algoritmi se kreću memorijom ponešto linearnim načinom, no podaci se ne premještaju već samo mijenjaju na mjestu.
d) Emulirani FPU:
   Softverski se emulira FPU korištenjem cjelobrojnih rutina. Ovakav test je dobar za općenitu generičku analizu performansi.
e) Fourierova analiza: Numerička analiza za računanje nizova aproksimacija valnih oblika. Algoritam intenzivno koristi brojeve sa pomičnim zarezom, stoga je dobar test FPU performansi uz minimalno baratanje matricama. Memorijske i međuspremničke performanse ne utiču na opći rezultat.
f) Pridruživanje: Simulira se algoritam za raspodjeljivanje zadaća, što je u praksi pomicanje kroz vrlo velika cjelobrojna polja i kroz retke i kroz stupce. Međuspremnici i memorije sa dobrim sekvencijalnim performansama bi ovdje mogli imati bitna poboljšanja, budući da se ne dešava nikakvo sortiranje ili pomicanje blokova, već isključivo promjena podataka na mjestu.
g) Huffman: Koristi se vrlo poznati algoritam za kompresiju teksta i grafičkih sadržaja (cjelobrojno bazirani algoritam). Dotični je kombinacija bajtovno orijentiranih operacija, bitovnih operacija kao i standardnih manipulacija cijelim brojevima.
h) IDEA: Izvršava se blokovski orijentiran algoritam za enkripciju (cjelobrojno bazirani algoritam). Riječ je o algoritmu koji se kreće kroz podatke sekvencijalno i to u 16-bitnim blokovima.
i) Neuralna mreža: Simulira se mreža koristeći vrlo mali unazadni-propagacijski algoritam. To je zapravo FPU test sa malim poljima brojeva pomičnog zareza, a koristi se eksponencijalna funkcija.
j) LU dekompozicija: Riječ je o vrlo kratkom i robusnom algoritmu za rješavanje linearnih jednadžbi uz minimalnu potrošnju memorije. I ovdje je riječ o FPU testu koji se u algoritmu kreće kroz polja i recima i stupcima te izvršava standardne i osnovne matematičke funkcije zbrajanja, oduzimanja, množenja i dijeljenja.

Zajednička osobina svih ovih testova je standardnost algoritama i njihova pojavnost u «stvarnom» svijetu. Budući da su svi ovi testovi bazirani na jednom «zadatku», odnosno jednoj radnoj dretvi, oni mjere ponašanje samo jednog procesora – odnosno njegovo ponašanje pod određenom jezgrom.



U gornjem grafu je zanimljiva nevjerojatna razlika u performansama Linuxa u baratanju memorijom kao što je npr. sortiranje nizova znakova. Jedini test u kojem se vidi značajna prednost u korist Solarisa je Fourierova analiza što bi dalo naslutiti da je za proračunske probleme bolji Solaris, dok je za opći rad (npr. radna stanica ili poslužitelj) bolji Linux zbog boljeg memorijskog i cjelobrojnog baratanja. No, iz daljnjih testova se vidi da to nije posve tako.

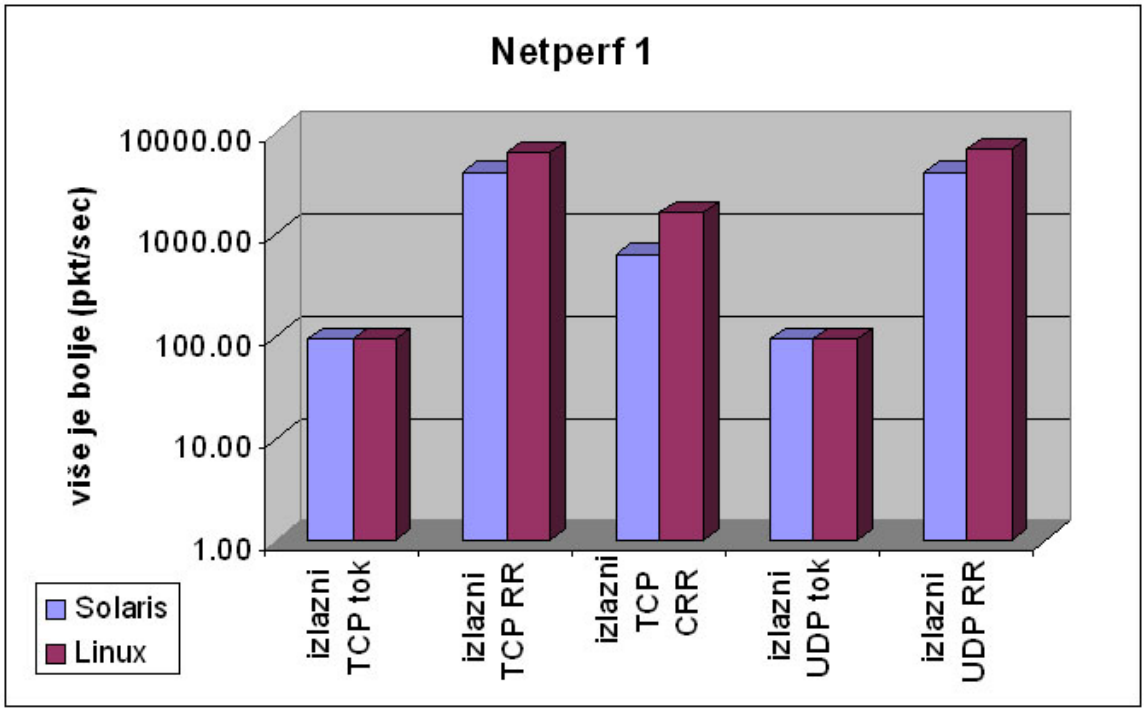


Sa druge strane, zanimljivo je da u testu neuralne mreže (također FPU test) Linux daleko prednjači, pa je vjerojatno razlog različit način baratanjem memorijom i podacima pri proračunima. Ukupno gledano koristeći statističku analizu (posljednja tri dvostruka stupca), u cjelobrojnim i FPU proračunima je Solaris tek zanemarivo iznad Linuxa po performansama – dok je Linux u baratanju memorijom značajno brži što u poslužiteljskom radu može značiti kvalitetan dobitak u peformansama.

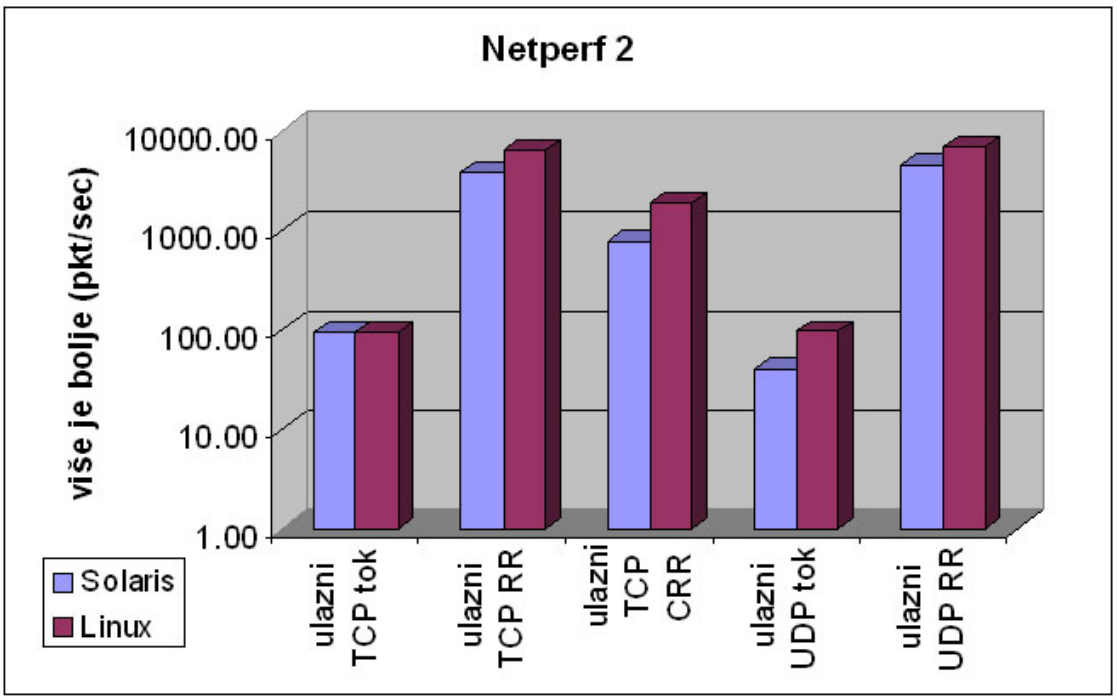
3) Sljedeći je Netperf u verziji 2.2pl3, jedan od poznatijih testova TCP/IP stoga u Unixoidima kojeg je razvio Hewlett-Packard. Služi mjerenju različitih aspekata mrežnih performansi pri čemu najveću pažnju posvećuje standardnom prijenosu podataka i performansama tzv. zahtjev/odgovor načina komunikacije koristeći TCP, UDP i Berkeley sockete. Sam program je građen po klijent-poslužitelj modelu, pri čemu se klijent spaja na poslužitelj, ostvaruje vezu i onda započinje mrežno testiranje. Naravno, mjeri se i korištenje procesorskih resursa različitim algoritmima ovisno o platformi – od «spavajućih» petlji, pa do preciznih sistemskih poziva tipa pstat() i sličnih. U našem slučaju mjerena je propusnost po lokalnoj (UTP CAT5e, preko gigabitnog preklopnika) mreži do kontrolnog računala koje je bilo usredotočeno samo na obavljanje Netperf poslužiteljskih zadaća i prikupljanje rezultata. Provedeni UDP i TCP testovi se mogu podijeliti na:
   a) Testove «sirove» brzine:
      Dotični mjere protočnosti toka u oba smjera ili samo jednom smjeru. Takvi testovi

pokazuju kako brzo može jedan sustav poslati drugom sustavu podatke, odnosno kako brzo drugi sustav može primiti takve podatke.

b) Testovi tipa zahtjev/odgovor:
Riječ je o testovima brzine obavljanja «transakcija», odnosno brzine potrebne za izmjenu jednog zahtjeva i jednog odgovora. Ovakav način mjerenja predstavlja dobar način kako izmjeriti performanse načina komunikacije kakav se obično pojavljuje između poslužitelja na mreži i klijenata, npr. Web servera.



Pogledamo li slijedeći graf, vidimo da u TCP i UDP, RR i CRR načinu rada Linux vodi Solaris. Ako spomenemo i detalj da je TCP CRR način testiranja najbliži standardnom Web poslužitelju, jasan je detalj da je Linux u značajnom vodstvu – posebice znajući da je riječ o logaritamskoj skali.



Još jednom, pokazuje se da su u sirovim TCP/UDP performansama protoka Solaris i Linux identični, što je i za očekivati budući da takav prijenos ne opterećuje značajnije TCP/IP stog, već prvenstveno mrežnu karticu koja je identična u oba slučaja. No, opet i u CRR i RR slučaju koji uvjetuje i određena parsiranja i reakcije jezgre, Linux značajno vodi pa se daje zaključiti da su mrežne performanse općenito gledano bolje kod Linuxa, odnosno može se očekivati brže odgovore i interaktivniji rad.

4) LMbench je jedan od najpoznatijih novijih Unixoidnih testova. Autori su Larry McVoy (Sun Microsystems, Silicon Graphics) i Carl Staelin (Hewlett-Packard Laboratories). Riječ je o setu kratkih i laganih testova koji mjere čistu brzinu, sporosti i propusnosti različitih dijelova sustava i jezgre, odnosno sistema općenito. Dakle, fokus je upravo na performanse koje dobiva aplikacija kao takva od sustava. Brojke koje ovakvi testovi daju su u praksi «cijena operacija», odnosno pokazatelj koliko je takvih kontrolnih i inih operacija moguće postići u jednoj vremenskoj jedinici, odnosno kakvo ponašanje imaju slični ali različito implementirani pozivi. Nadalje, ovdje se vide i mogući problemi u performansama – problemi sa propusnošću ili pak sporošću sustava ili određenih dijelova sustava. Što se sporosti tiče, mjere se brzine procesa, lokalnih mrežnih situacija, datotečnog sustava i memorije. Sa druge strane, mjeri se i propusnost lokalnih mrežnih poziva, datotečnog sustava i memorije. Ugrubo testove možemo podijeliti na:
   a) Memorijske testove:
      i) Mali i pseudoslučajni prijenosi podataka po memoriji – sporost učitavanja,
      ii) Veliki i slijedni prijenosi podataka – mjeri se bcopy() propusnost;
   b) Procesne testove:
      i) Vrijeme izvršavanja «praznog» procesa,
      ii) Vrijeme sistemske izmjene konteksta;
   c) Razne testove;
      i) Prazni unosi u sustavu;
   d) Mrežne testove:

   i) Mali pseudoslučajni prijenosi,
      (1) Prijenosi u sekundi,
      (2) CPU ciklusa po prijenosu,
      (3) Socket(), bind() i close() pozivi u sekundi;
   ii) Veliki i sekvencijalni prijenosi:
      (1) MB u sekundi,
      (2) CPU ciklusa po MB;
e) Diskovne testove:
   i) Mali pseudoslučajni prijenosi:
      (1) Prijenosa u sekundi,
      (2) CPU ciklusa po prijenosu;
f) Testove datotečnog sustava:
   i) Mali pseudoslučajni prijenosi:
      (1) Stvaranja datoteka u sekundi,
      (2) Brisanja datoteka u sekundi,
      (3) Pseudoslučajni I/O po sekundi u velikim datotekama,
      (4) CPU ciklusa po prijenosu,
      (5) MB/s pri čitanju mnogo malih datoteka na istim mjestima;
   ii) Velike datoteke:
      (1) MB/s pri čitanju i pisanju,
      (2) CPU ciklusa po prijenosu;
g) VM testove:
   i) Stvaranje:
      (1) Mmap() poziva po sekundi,
      (2) Munmap() poziva po sekundi,
      (3) Varijacija istih sa različitim veličinama mmap()iranog područja;
   ii) Mali pseudoslučajni prijenosi:
      (1) Pseudoslučajna čitanja po sekundi velike mmap()irane datoteke,
      (2) CPU ciklusa po čitanju;
   iii) Veliki slijedni prijenosi koristeći međuspremnike:
      (1) MB/s po čitanju,
      (2) CPU ciklusa po MB.

Opisi testova su redom:

1) Propusnost:
   a) Datoteke: Čita se neka (pseudoslučajno stvorena) datoteka u 64KB blokovima, svaki blok se sumira kao serija od 4-bajtnih cijelih brojeva u potpunoj petlji. Ovaj test se izvršava na datotekama koje su u memoriji, a u većini Unixoida je read() implementiran kao kopiranje iz jezgrinog prostora u aplikativni prostor. Alternativa ovom testu je stvaranje mmap()iranog prostora u memoriji i asociranog sa datotekom, te čitanje datoteke u petlji tako da se iz memorije čita po 4-bajtni cijeli broj i zbraja na sadržaj akumulatora.
   b) Memorija: Alocira se dvostruka količina potrebne memorije, izbriše (upišu nule) i onda se mjeri vrijeme kopiranja prve polovine u drugu polovinu. Postoje i druge varijante ovog testa u kojem se iz memorije čita po 4-bajtni cijeli broj i sprema u akumulator zbrajajući sa postojećim sadržajem.
   c) Neimenovani cjevovod: Stvara se Unix neimenovani cjevovod između dva procesa i mjeri se propusnost podataka među njima puštanjem blokova podataka, obično po 10MB.

   d) TCP: Standardno, stvara se klijent i poslužitelj i podaci se preko TCP/IP socketa šalju u blokovima po 10MB. Jasno, mjeri se propusnost prema 127.0.0.1, odnosno lokalnom računalu.
   e) Unix: Stvara se neimenovani cjevovod i dijete proces koje piše u cjevovod što brže može. Ovaj test mjeri koliko brzo može dijete roditelj čitati podatke iz cjevovoda u blokovima.
2) Sporost:
   a) Spajanje: Riječ je o klijent/poslužitelj programu koji mjeri sporosti međuprocesne komunikacije na takav način da se stvara AF_INET (TCP/IP) socket na udaljeni (lokalno računalo) poslužitelj, pa se mjeri vrijeme potrebno za stvaranje i spajanje na takav socket.
   b) Izmjena konteksta: Stvara se nekoliko procesa koji se spajaju u prsten Unix neimenovanih cjevovoda. Svaki proces čita određeni blok podataka iz cjevovoda, obavlja kakav posao (zbraja cjelobrojno polje određene veličine) i piše dotični blok slijedećem procesu. Mjeri se ukupno vrijeme za restauraciju cjelokupnog stanja procesa dakle i međuspremničkog stanja.
   c) Fcntl: Riječ je također o klijent/poslužitelj načinu testiranja kod kojeg se mjere sporosti zaključavanja datoteka. Proces izmjenjuje zaključavanje i otključavanje datoteke tako da se samo po jedan klijent ili poslužitelj mogu izvršavati, slično kao načinu predaje poruka tipa «vrući krumpir».
   d) Datoteka: Stvara se niz malih datoteka u jednom direktoriju i mjeri se vrijeme kako za stvaranje tako i za brisanje datoteka.
   e) HTTP: Stvara se klijent/poslužitelj program koji mjeri sporosti jednostavnog HTTP prijenosa (HTTP GET).
   f) Čitanje memorije: Mjeri se sporost čitanja memorije tako da se stvara polje pokazivača koji pokazuju na matricu i prolazi se kroz matricu uzduž i poprijeko. Jasno, kako veličine matrica variraju od 512 byteova do 8MB, procesorski i ini međuspremnici mogu bitno utjecati na rezultate.
   g) Mmap: Računa se koliko brzo radi uzastopni mmap() i munmap() poziv. Takvi pozivi su inače u većini Unixoida iznimno bitni jer se na njega oslanjaju različiti dijelovi sistemskih biblioteka. Osim toga, pri pokretanju kojeg procesa, sistemske biblioteke se mapiraju, te pri završetku odmapiraju na upravo navedeni način.
   h) Matematičke operacije: Mjeri se vrijeme za cjelobrojne standarne operacije (bitovne, zbrajanje, množenje, dijeljenje, mod operacije, paralelizam). Jasno, to isto se radi i za 64-bitne brojeve kao i za brojeve sa pomičnim zarezom, te brojeve sa pomičnim zarezom dvostruke točnosti.
   i) Nepostojeća stranica: Mjeri se koliko brzo stranica neke datoteke može biti izbačena iz lokalne memorije koristeći msync() i dužinu polja od 256K.
   j) Neimenovani cjevovod: Stvaraju se dva procesa koja komuniciraju kroz Unix neimenovani cjevovod i mjeri se vrijeme međuprocesne komunikacije. Kroz dotični cjevovod se prosljeđuje određeni blok podataka naprijed i nazad između procesa na način koji se inače naziva tehnika «vrućeg krumpira».
   k) Procesi: Stvaraju se procesi na više različitih načina da bi se izmjerilo koliko je potrebno vrijeme za kontrolu u osnovnoj dretvi: radi se fork() + exit() pri čemu se stvaraju dvije identične kopije i jedna se terminira; radi se fork() + execve() pri čemu se mjeri koliko treba da se stvori novi proces i da taj novi proces izvrši neki posao; radi se fork() + /bin/sh –c, što je u praksi identično kao i

prošla naredba s time da se još dodaje vrijeme potrebno sistemu da pronađe ljusku i izvrši je.

l) RPC, TCP, UDP, Unix: Stvara se klijent/poslužitelj test za mjerenje sporosti komunikacije između procesa. U testu se šalju podaci naprijed i nazad između dva procesa, ekvivalentno «vruć krumpir» tehnici. No, u ovom slučaju se za prijenos koriste udaljeni pozivi procedura (RPC), TCP, UDP ili Unix socketa kao prijenosnog medija.

m) Selektiranje: Mjeri se vrijeme za izvršavanje sinkronog multipleksiranja n datotečnih opisnika, odnosno za izvršavanje select() poziva koji je standardni način za nadziranje događanja na n datotečnih opisnika.

n) Signali: Mjeri se vrijeme potrebno za instaliranje prekidnih vektora (signal() poziv) i hvatanje signala, npr. grešku zaštite.

o) Sistemski poziv: Mjeri se prazni sistemski poziv (poziva se getppid() koji predstavlja jednostavni prijenos podataka između jezgrinog prostora i korisničkog prostora), čitanje (čita se po 1 bajt iz /dev/null), pisanje (piše se po 1 bajt u /dev/null), datotečni (mjeri se potrebno vrijeme za stat() poziv na datoteku u međuspremniku, vrijeme za open() i vrijeme za close() otvaranja datotečnih opisnika),

3) Razno:

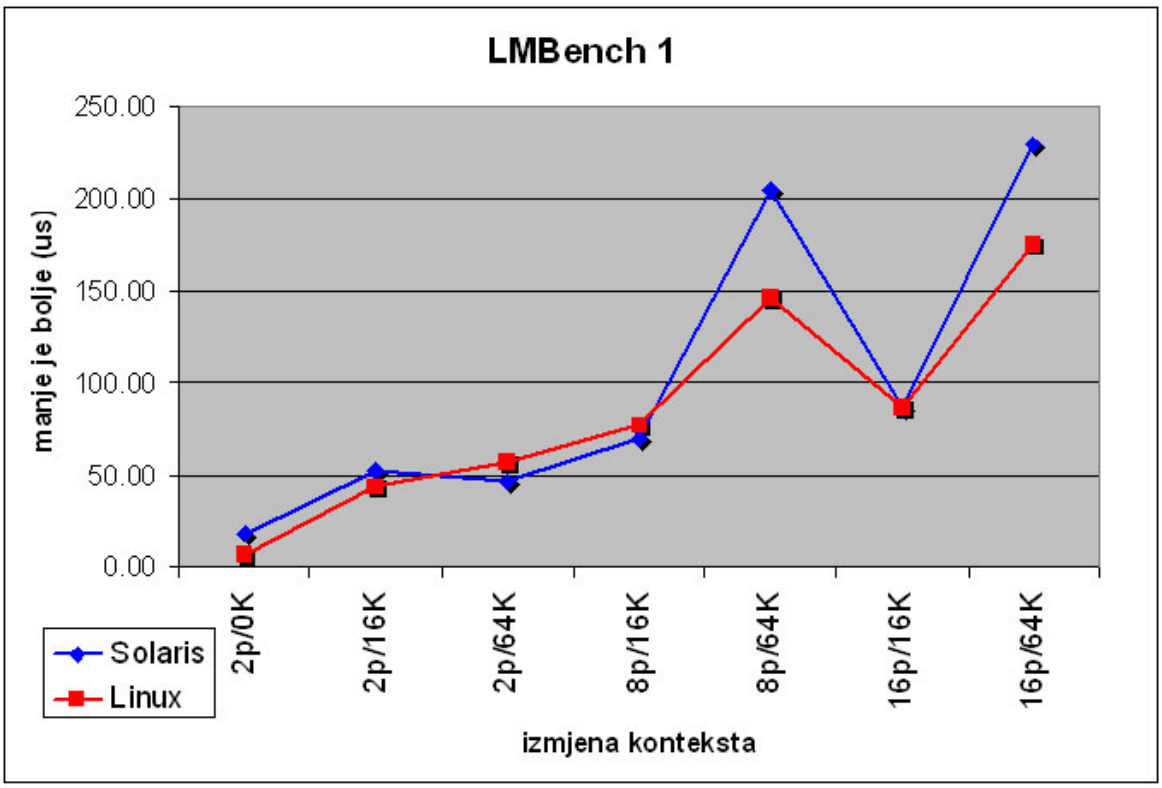
a) Međuspremnici: Test pokušava izmjeriti i utvrditi karakteristike memorijske hijerarhije i arhitekture. Provjerava se broj međuspremnika, veličina svakog međuspremnika, veličina linije svakog međuspremnika kao i to da se vrše standardni testovi memorijskog paralelizma u svakom nivou memorijske hijerarhije.

b) Linije: Pomoću «rešetki» pokazivača se pokušava saznati koja je veličina linije sistemskog međuspremnika u najvećem međuspremniku. Stvara se rešetka pokazivača od kojih svaki pokazuje na prvu riječ na svakoj međuspremničkoj liniji na stranici (i pseudoslučajno se premještaju po svim linijama u stranici prije skoka na iduću stranicu). Mjeri se srednje vrijeme sporosti za različite veličine linija sve dok se ne pronađe skok u srednjoj sporosti (odnosno bitno smanjenje) koje bi trebalo pokazati da je pronađena odgovarajuća veličina linije.

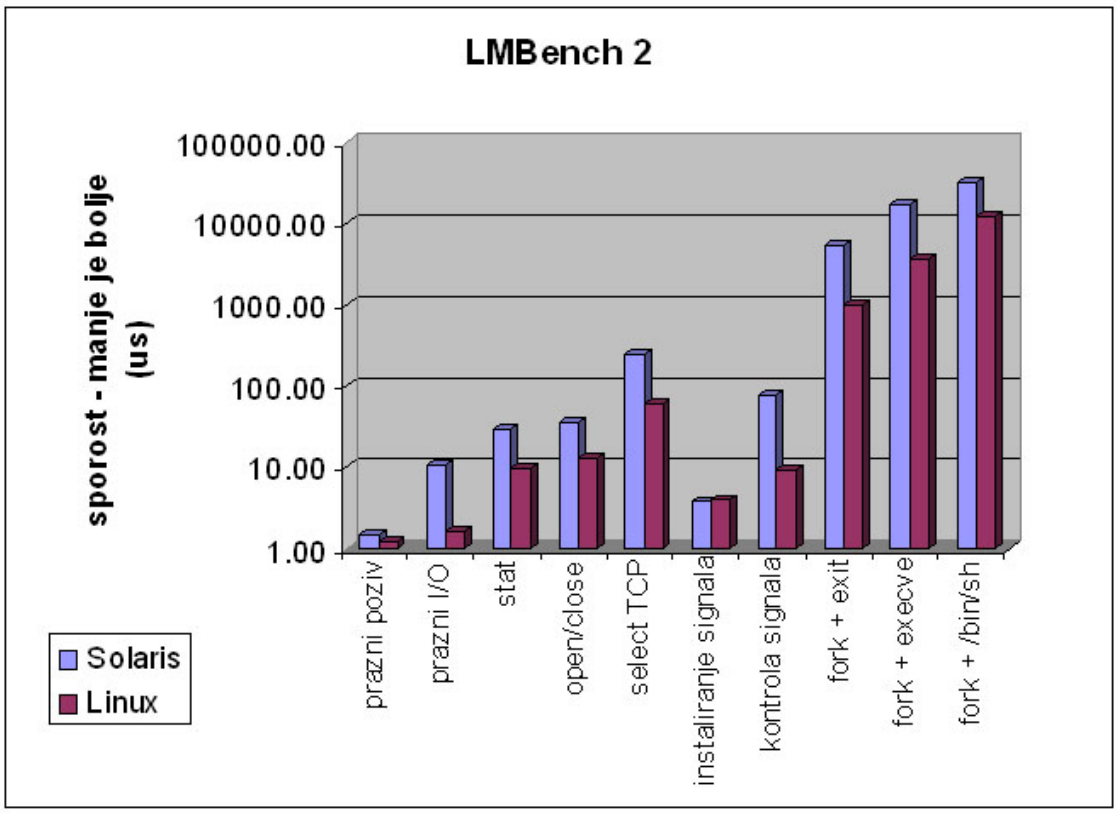
c) Lmdd: Mjeri se potrebno vrijeme za kopiranje ulazne datoteke u izlaznu datoteku uz određene izmjene, a riječ je primarno o ulazno/izlaznom testu.

d) Memorijski paralelizam: mjeri se dostupni paralelizam u trenutnoj memorijskoj hijerarhiji. Današnji moderni procesori omogućavaju višestruke paralelne memorijske zahtjeve, te ovaj test mjeri upravo to u različitim mjestima u memoriji. Stvara se rešetka pokazivača određene veličine, a zatim se stvara polje pokazivača koje pokazuje na određene elemente u rešetci koji su jednoliko udaljeni jedni od drugih. Zatim se prolazi (koristeći paralelne pozive) kroz tu rešetku i mjeri se potrebno vrijeme za svaki nivo paralelizma.

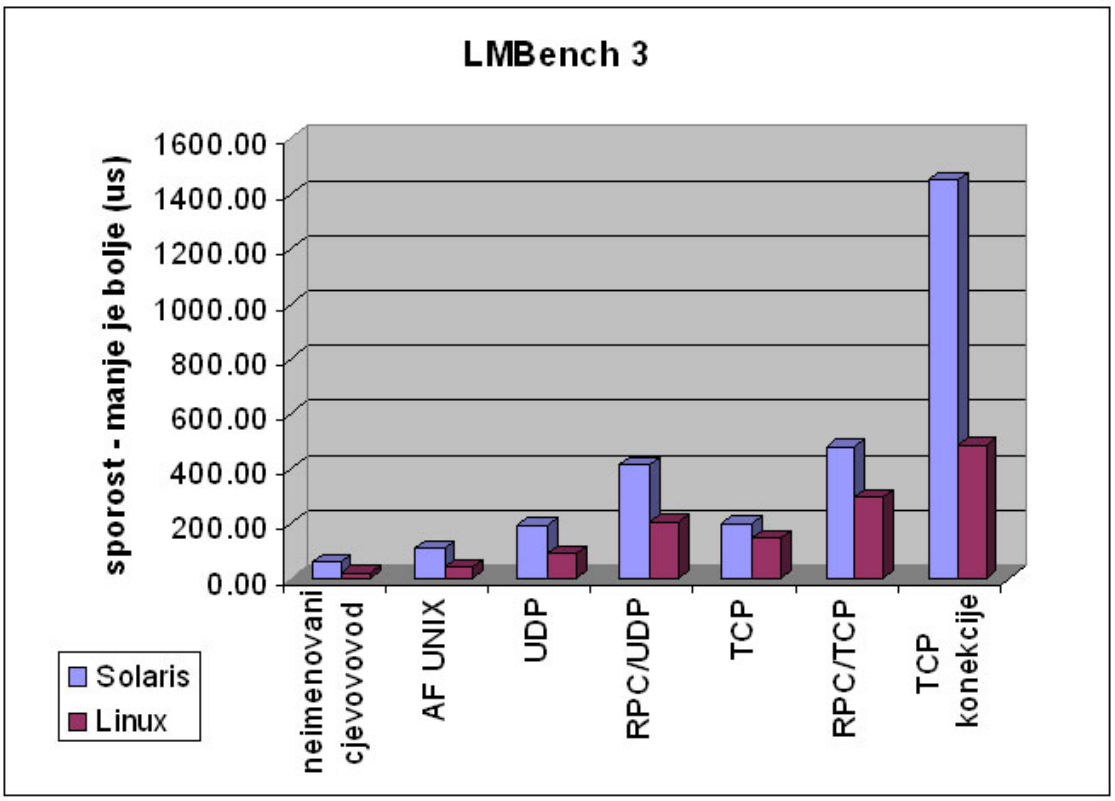
e) Paralelizam operacija: Mjeri se mogući paralelizam za standardne matematičke operacije (bitovne, zbrajanje, množenje, dijeljenje, mod operacije, paralelizam). To isto se radi i za 64-bitne brojeve kao i za brojeve sa pomičnim zarezom, te brojeve sa pomičnim zarezom dvostruke točnosti.



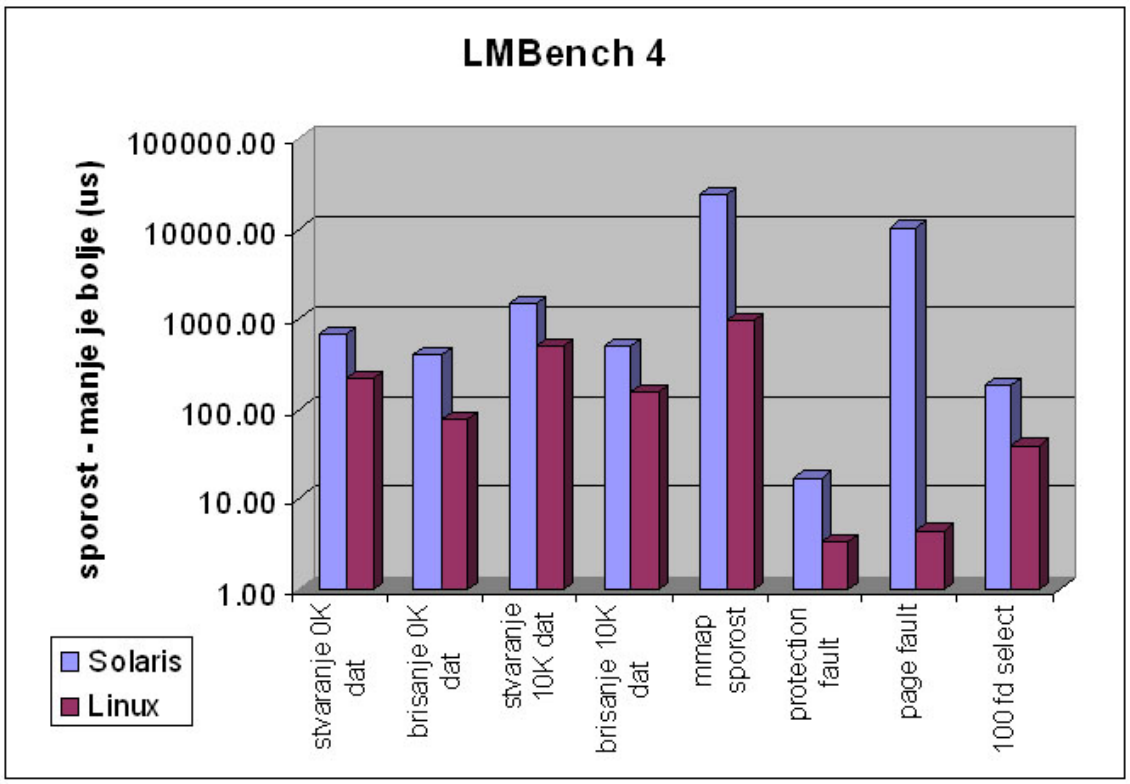
Ovaj graf pokazuje nedvojbeno manje potrebno vrijeme za izmjenu konteksta kod Linuxa što je indikacija kako će se ponašati višedretvene aplikacije ovisno o svojoj veličini. Jedan od bitnijih problema je ovdje i potrebno vrijeme kod Solarisa kod povećanja veličine procesa (64K) i broja procesa. Naime, to znači da će kod što većih procesa pod Solarisom praktički eksponencijalno rasti vrijeme potrebno za potpunu obnovu konteksta, dok je ono kod Linuxa znatno manje, a krivulja glađa. Krivulja pokazuje Solarisovu tendenciju da se «uguši» kod velikog broja velikih višedretvenih procesa.



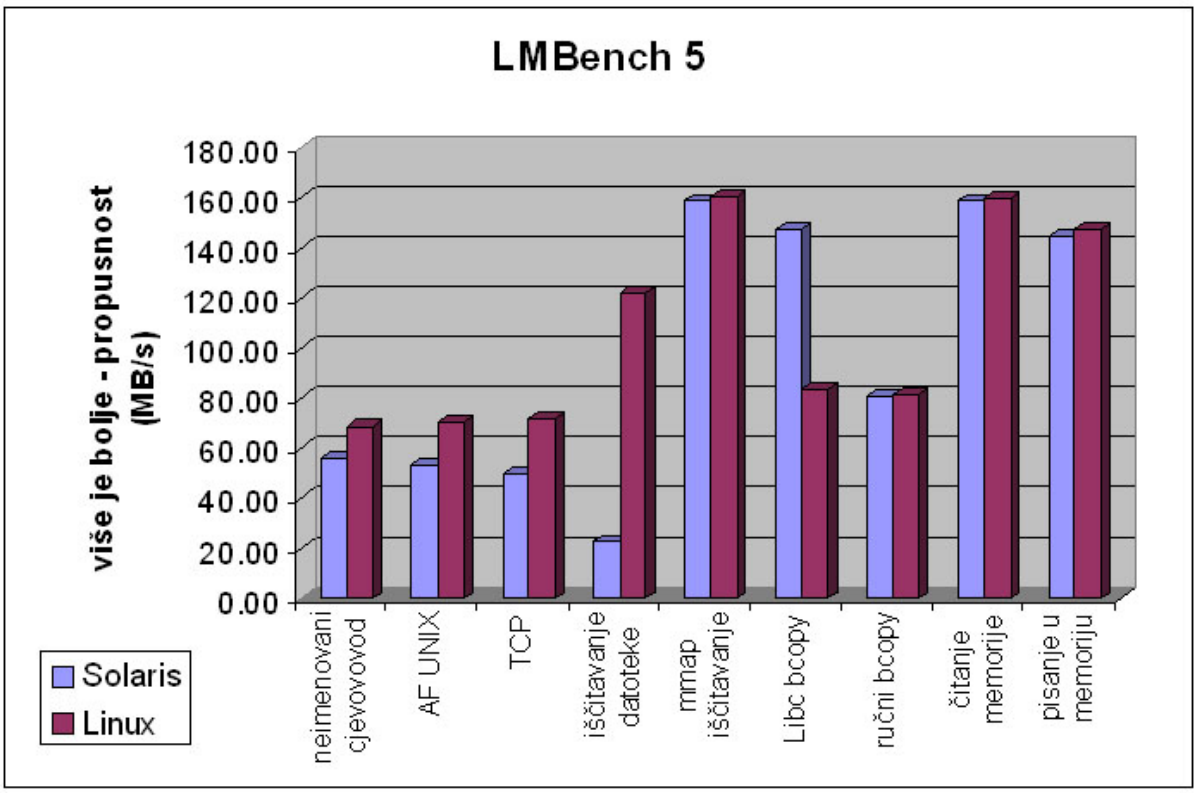
Gornji graf nam, pak pokazuje da su sporosti sistemskih odnosno jezgrinih poziva vrlo na strani Linuxa, a posebice moramo naglasiti razlike između stvaranja kloniranih procesa (primijetite logaritamsku skalu!). Kako se fork() naredba praktički konstantno koristi na poslužiteljima, ovakav rezultat jasno pokazuje da je Linux primjereniji ikakvom poslužitelju kod kojeg se pojavljuju uzastopni fork() pozivi (Apache, SSH, FTP poslužitelj, višekorisnički stroj, itd.). Nadalje, primjetan je problem i kod sinkronog multipleksiranja datotečnih opisnika (bilo kakav standardni poslužiteljski softver praktički danas koristi poll() ili select() pozive) kad se ne koristi Solarisov /dev/poll – što će reći da nemodificirane aplikacije koje barataju sa velikim brojem datotečnih opisnika rade bitno brže pod Linuxom. Jasno, to vrijedi i za ostale stupce/testove u gornjem grafu.



Slijedeći test je striktno baziran na mjerenjima sporosti međuprocesnih komunikacija. U ovom je testu se Solaris također pokazao značajno sporijim u svim kategorijama, a posebice u ostvarivanju TCP spoja (koji jer također iznimno bitan u radu jednog poslužitelja budući da se ostvaruje svaki put kad se TCP klijent spaja na poslužitelj – a ne moramo ni naglasiti kolika je brojnost TCP baziranih servisa i inih). Jasno, slično vrijedi i za RPC i UDP kao i AF Unix komunikaciju, ali u nešto manjoj mjeri.

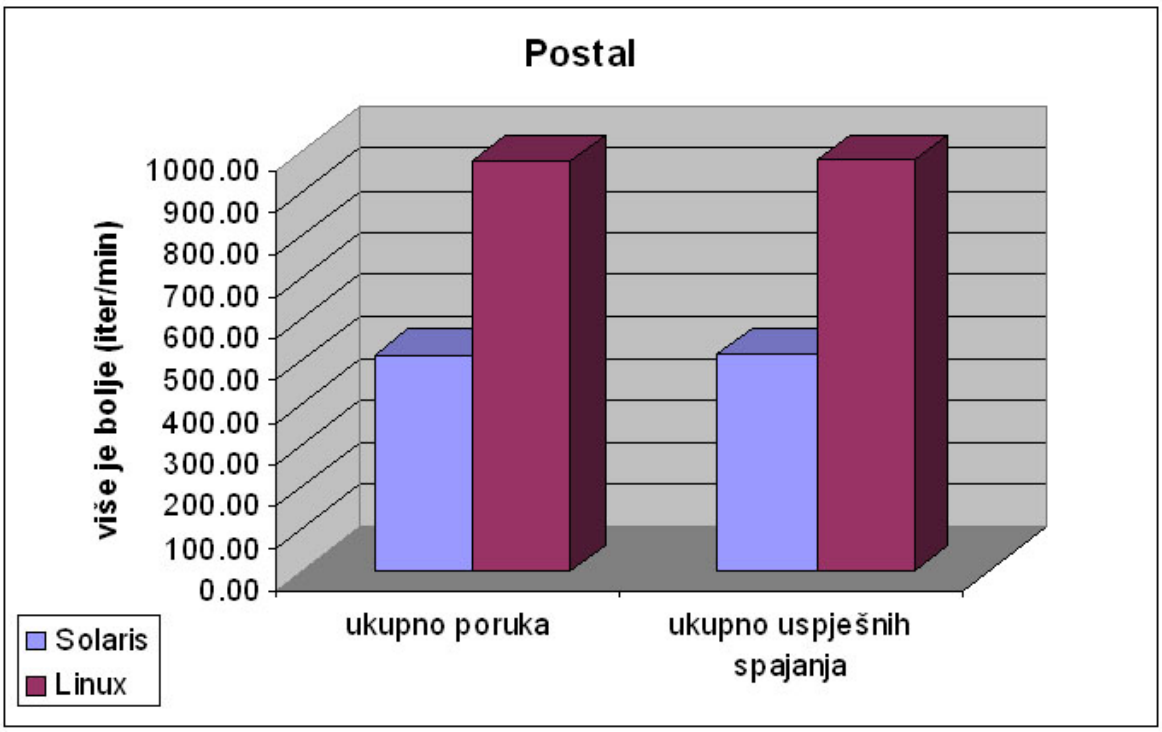


Opet, i ovaj test pokazuje koliko su jezgrini pozivi brži kad je riječ o Linuxu, a posebice možemo istaknuti vrijeme koje je potrebno Solarisu da se oporavi od «nepostojeće stranice». I ovdje se vidi nešto sporije sinkrono multipleksiranje, sporija detekcija sistemskih grešaka a i nešto sporije baratanje velikim brojem datoteka.



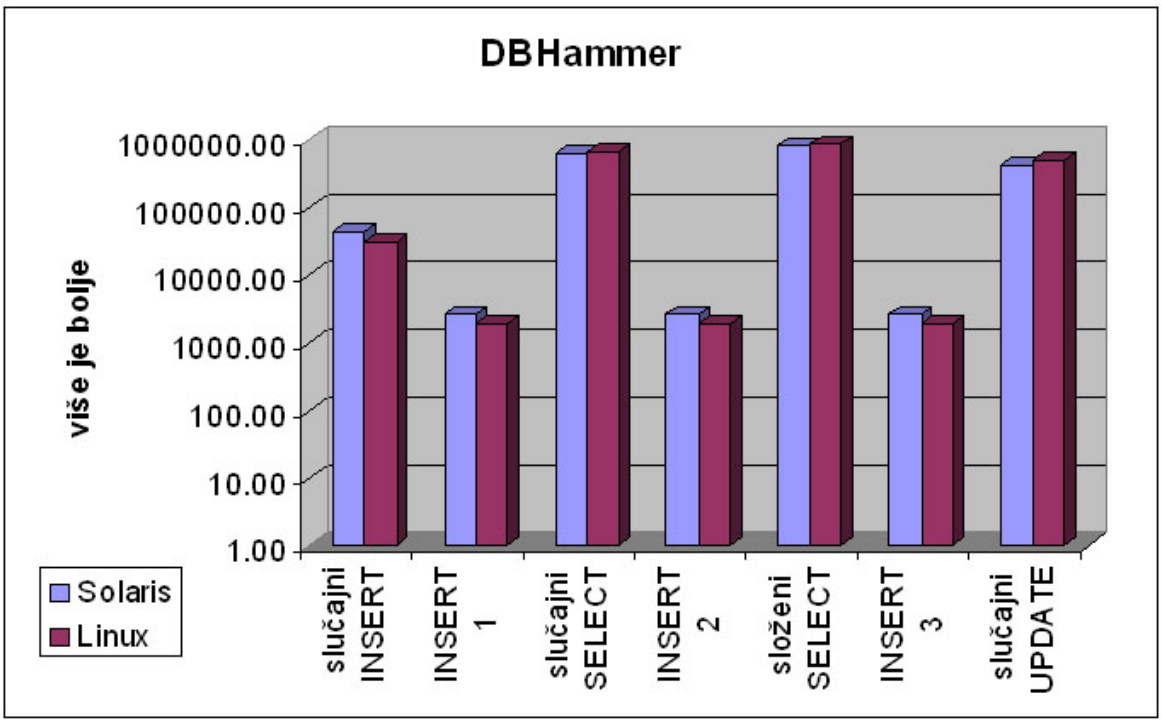
Posljednji test predstavlja testiranje propusnosti u kojima se Linux većinom pokazuje neznatno bolji (osim u iščitavanju datoteka u kojima je čak za red veličine brži). Solaris ~~začudo~~ ima u svojoj standardnoj sistemskoj C biblioteci nešto brže implementiran bcopy(), ~~međutim to ne predstavlja problem budući da se danas standardno najviše koristi mmap() poziv (GNU Libc za skoro sve memorijske pozive koristi mmap()) u kojem je Linux ipak brži.~~ Molim primijetiti vrlo dobre performanse Solarisove verzije bcopy() naspram one koja se izvršava pod Linuxom.

5) Postal je program za testiranje performansi (propusnosti) SMTP poslužitelja autora Russella Cokera, a mi smo ga koristili u verziji 0.62. Preciznije, dotični šalje na SMTP poslužitelj pseudoslučajne poruke što više i što brže može. U našoj konfiguraciji su se svi ti mailovi prosljeđivali u uređaj /dev/null, odnosno bili su automatski brisani, a koristio se Sendmail 8.10.2 u istim verzijama kako na Solarisu tako i na Linuxu.



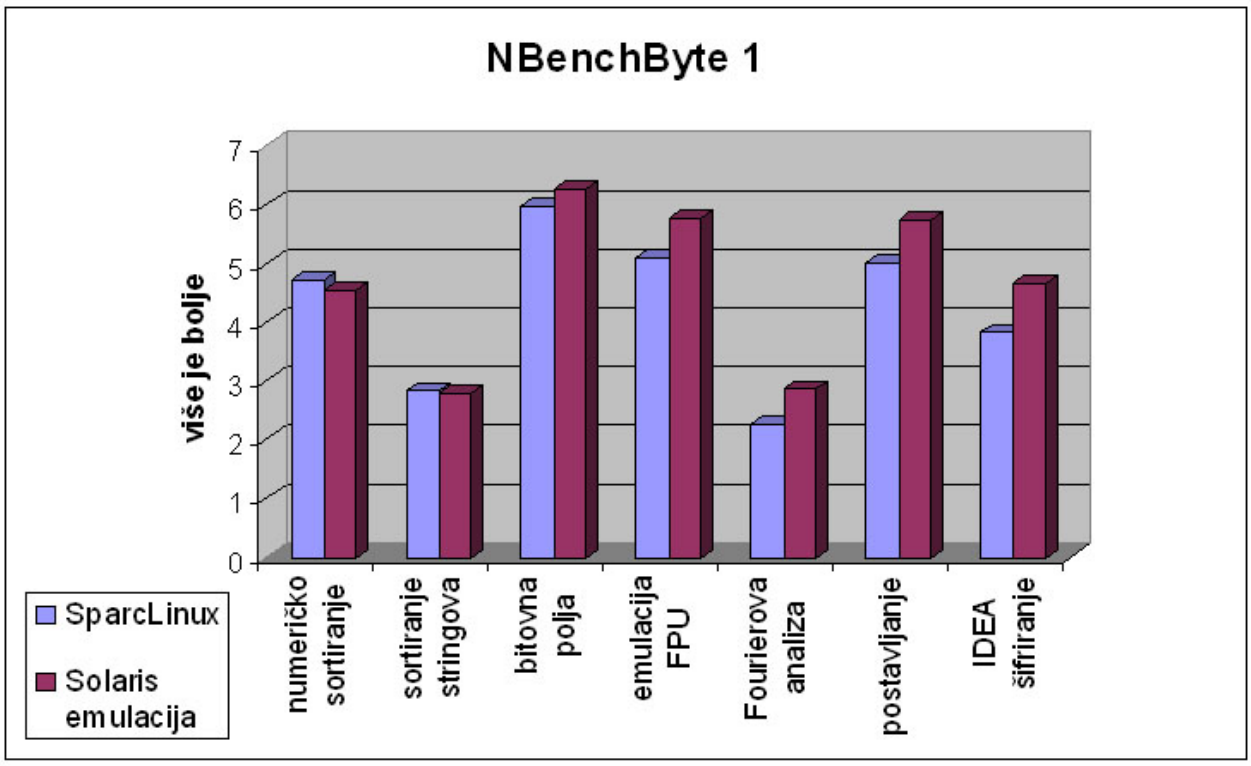
Kao finalni rezultat očita je velika prednost Linuxa nad Solarisom u radu sa identičnom konfiguracijom Sendmaila i identičnim programom za slanje. Ovo je ne samo rezultat brzine parsiranja nizova znakova, već i ponašanje sistema pod velikim zagušenjima i manjkom radne memorije (load-average postavke su u oba Sendmaila bile ugašene, te je sistemski load prelazio 15, a kod Solarisa je došao i do 20). Ovaj jednostavni aplikativni test pokazuje u praksi rezultate svih već navedenih testova, tj. da je Linux nedvojbeno bitno brži u baratanju podacima u memoriji (parsiranje, sortiranje, itd.), da brže barata malim datotekama i sl.

6) DBHammer predstavlja jedan od rijetkih aplikativnih testova, a mi smo ga odabrali isključivo radi činjenice da baratanje bazama podataka u ovom slučaju pokazuje TCP performanse, memorijske performanse, performanse datotečnog sustava kao i sposobnost parsiranja i analize nizova znakova; dakle svi detalji koji su do sada analizirani bi se trebali i ovdje vidjeti. Testna inačica bila je u verziji 0.3.1, a korištena baza MySQL 4.0.12 u istim verzijama kako na Linuxu tako i na Solarisu. Za ostvarenje komunikacije sa bazom koristilo se isto referentno računalo kao i za već navedene mrežne testove. Sami testovi se sastoje iz stvaranja tablica (CREATE), ubacivanja podataka u njih (INSERT), prihvaćanja podataka (SELECT), mijenjanja podataka (UPDATE) i naravno brisanja tablica (DROP). Dotični testovi se izmjenjuju, koriste se pseudoslučajni uzorci, itd.

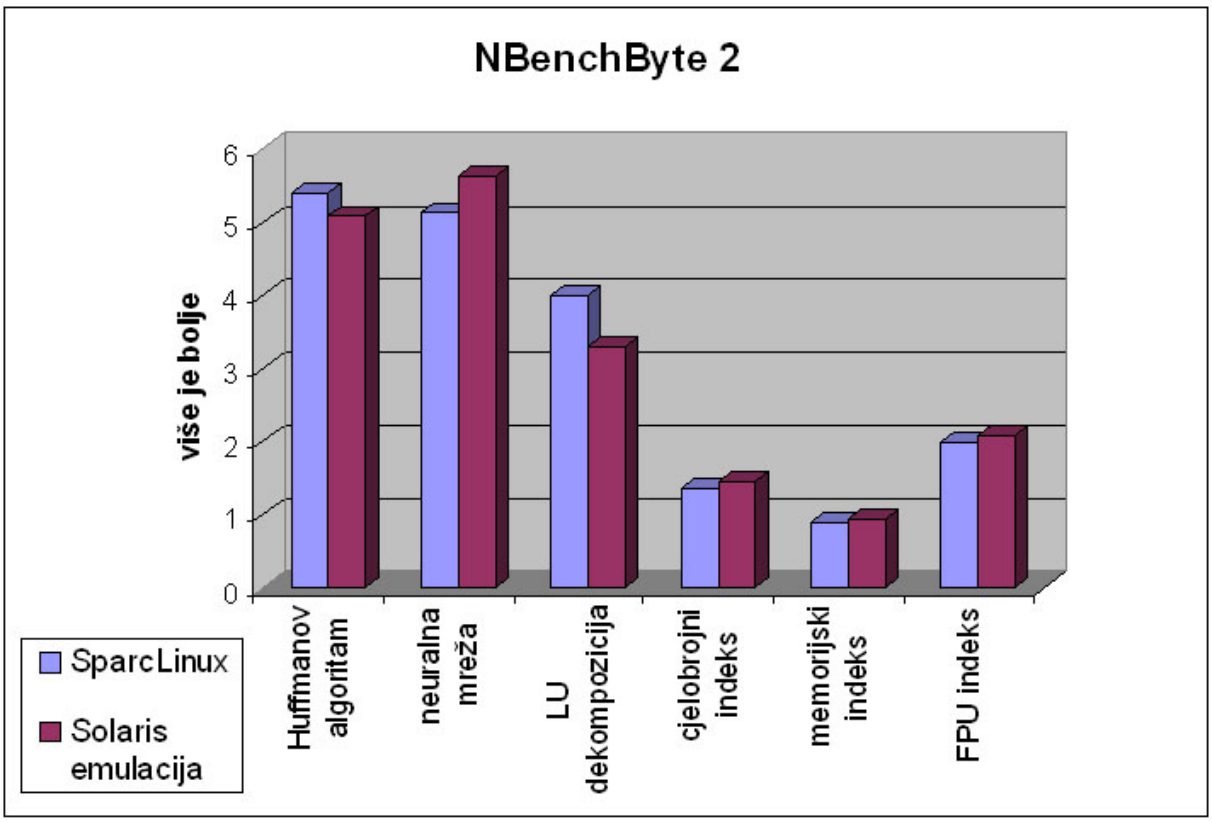


Zanimljivost je da su obje testirane platforme skoro identičnih performansi, odnosno da je razlika statistički praktički zanemariva. Dakle, odabir Linuxa ovdje ne donosi neke prednosti, ali ni ne unosi nikakav gubitak u performansama.

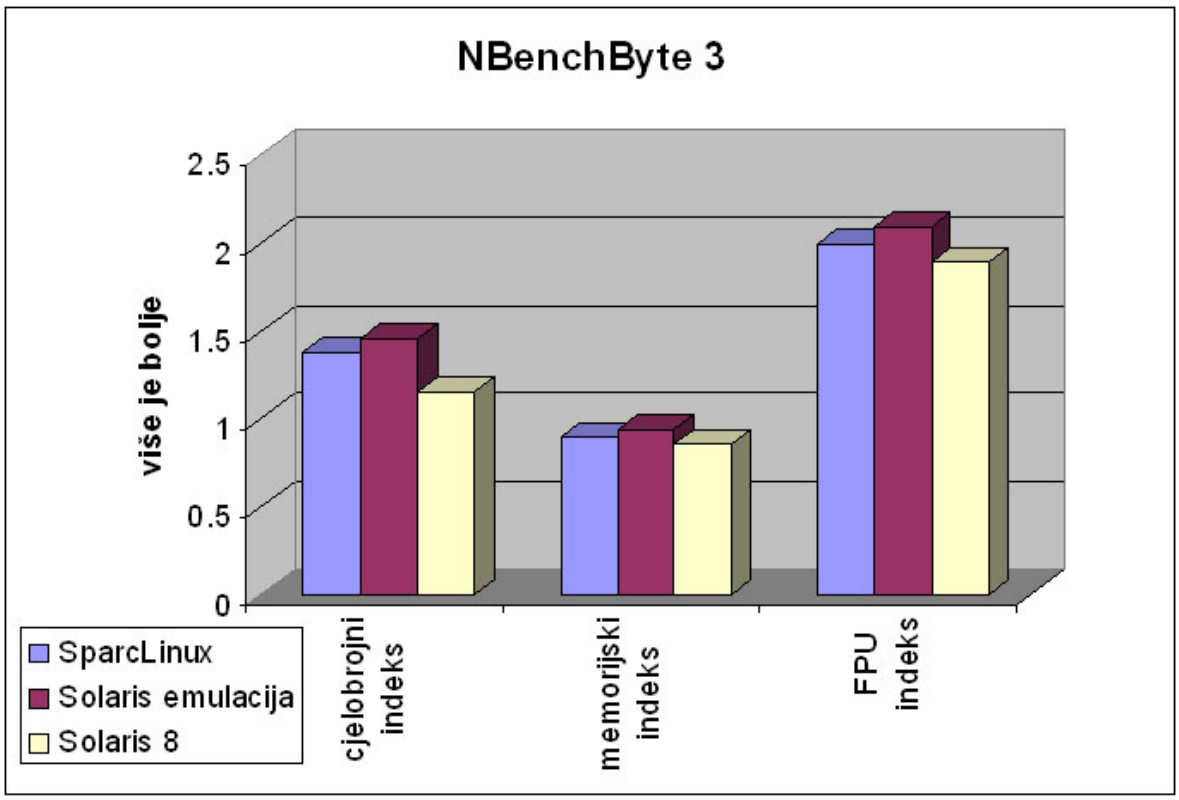
7) Emulacija je posljednji dio testova i predstavlja naš test koliko se dobro ponaša postojeća Solaris emulacija u Sparc Linux jezgri naspram originalnog Sparc Linux koda. Razlog ovog testa je utvrditi da li se mogu postojeće Solaris izvršne datoteke ponovno iskoristiti i na Sparc Linux instalacijama u slučajevima kad nije izvorni kod dostupan, npr. kod Legato Networker ili Sophos Sweep softvera. U tu svrhu smo iskoristili ponovo NBench u različitim verzijama – jedna je prekompilirana statički na testiranom Solaris 8 računalu, a druga na testiranom Sparc Linux računalu i obje su onda izvršene na potonjem računalu.



Iz grafa je očito ne samo da emulacija radi odlično na Linuxu, već i da je kominacija Solaris standardne C biblioteke i Linux jezgre donijela zanimljiv dobitak u performansama. Jasno, to znači i da možemo u idućim inačicama Linux Libc biblioteke očekivati sve bolje performanse. Nadalje, to znači i da je emulacija kvalitetno izvedena, te da se izvršne datoteke ponašaju statistički korektno (kao i što je očekivano).



I posljednji test pokazuje više ili manje identične rezultate uz zanemarive oscilacije, što će reći da Linux jezgra unosi značajna ubrzanja u radu, a da bi Linux programeri još mogli poraditi na vlastitoj Libc biblioteci. Usporedimo li testove iz originalnog NBench testiranja i ovih posljednjih rezultata, dobivamo činjenicu da se Solaris izvršne datoteke izvršavaju brže pod Linuxom nego pod Solarisom.



Preporuka:
Memorijski zahtjevi Solarisa su bitno veći od Linuxa – osnovna instalacija Solarisa 8 kao i Solarisa 9 zahtijeva prema Sunovom priručniku «Solaris 8 Advanced Installation Guide» i u praksi minimalnih 128MB radne memorije i 1GB diskovnog prostora, a sa svim servisima i bitno više. Dapače, pokazalo se i da su zahtjevi Solarisa 9 čak nešto veći nego deklarirani, te da bi njegova primjena izbacila iz upotrebe Ultra5 računala na ustanovama (a i ponegdje zaostala Ultra1 računala). Sa druge strane, Sparc Linux 2.4.20 sa Debian testing (kodno ime «Sid») verzijom i svim podignutim standardnim preporučenim servisima (NTPd, Apache, ProFTPd, SSHd, Cron, MySQL, itd.) zauzima svega 42MB radne memorije. Nadalje, prema Debian instalacijskom priručniku «Installing Debian GNU/Linux 3.0 For SPARC», Debian zahtijeva 12MB radne memorije i 110MB diskovnog prostora.
Nadalje, kad se koncentriramo na rezultate očito je da Linux doživljava konstantne promjene i poboljšanja koje su naposljetku rezultirale u bitno kvalitetnijim performansama u praktički svim relevantnim područjima za poslužiteljski rad: i diskovnim performansama, i brzinom baratanja memorijom i pogotovo mrežnim performansama. Spomenemo li još činjenicu da za Linux postoje posebni dodatni sigurnosni modeli koje je moguće uvesti i time dodatno ojačati računala, izbor je očit. No, to i nije jedini važni argument – pregledamo li forenzike koje je Sektor za operacijske sisteme kao podršku sistemcima izvršio u posljednjih godinu dana, dolazimo do poraznih rezultata: do sada je provaljen svega jedan Debian u CARNetovim ustanovama (i to zbog slabe brige sistemca i nekorištenja dostupnih sigurnosnih modela u jezgrama koje posebno priprema Grupa za izradu paketa). Za Solaris rezultati nisu tako dobri – budući da je izvršeno nebrojeno mnogo forenzika dotičnih, uglavnom uvijek zbog raznih exploita koji su poradi Sunovih kašnjenja i lošijeg dizajna uzrokovali sistemcima i forenzičarima glavobolje.

Za kraj spomenimo i činjenicu da je i zbog edukacije i mlađih sistemaca Linux nešto ugodniji i bolji, kao platforma koja je zbog svojeg otvorenog koda i dostupnosti vrlo brzo postala iznimno raširena. U svakom slučaju, Linux se predstavlja kao vrlo dobro i kvalitetno rješenje, dobrih performansi, brzog ritma promjena i poboljšanja i po autorovom mišljenju riječ je o platformi koja je dovoljno zrela za zamjenu Solarisa na low-end radnim stanicama i poslužiteljima.

Rezultati testova:
Nalaze se priloženi u izdvojenom html obliku: Testiranje-SolarisLinux-testovi.htm.