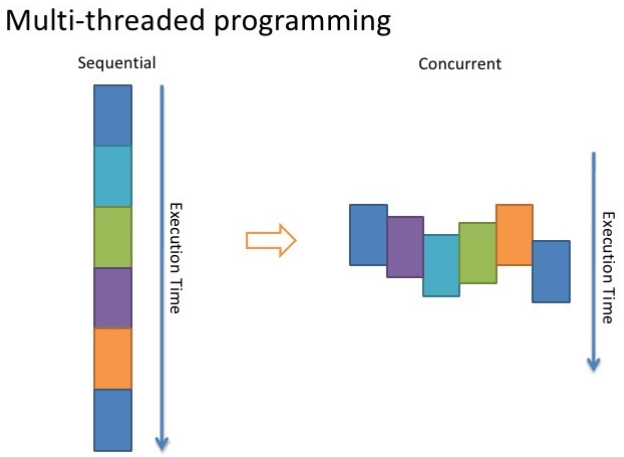
Konkurentna paradigma – Definicija, motivacija, veza sa programskim jezicima, skalabilnost i portabilnost

Jovan Milenković i Marija Katić
mr16006@matf.bg.ac.rs
mr16032@matf.bg.ac.rs

11. novembar 2018.

1 Kratak uvod, motivacija

Konkurentna paradigma predstavlja koncept izvršavanja više programa ili delova programa u istom vremenskom periodu (Slika 1). Sa aspekta kompleksnih programa, sa velikim brojem objekata koji međusobno i istovremeno interaguju, uvođenje konkurentnosti izvršavanja predstavlja prirodan način modelovanja. Koncepti konkurentnog programiranja, osim što olakšavaju modelovanje sistema i njegovu implementaciju, mogu doneti značajne uštede računarskih resursa u vidu bolje iskorišćenosti procesorskog vremena i raspoložive memorije. Kao dobre primere primene konkurentne paradigme možemo izdvojiti operativne sisteme i veb pregledače koji bi bili nezamislivi bez konkurentnosi kao centralnog koncepta njihovog dizajna [20].



Slika 1: Sekvencijalno i konkurentno izvršavanje programa

Međutim, konkurentnost uvodi nove komplikacije i direktno implicira mnoštvo važnih pitanja, nezavisno od nivoa programiranja i oblasti primene. Na koji način je realizovana komunikacija dva programa koji se konkurentno izvršavaju? Štaviše, ukoliko pomenuti programi imaju pristup zajedničkoj memoriji, sa pravom upisa, na koji način je upisivanje regulisano? Ukoliko dva programa istovremeno zahtevaju korišćenje deljenog resursa, na koji način se određuje prioritet [20]?

Razvoj operativnih sistema i višejezagrnih procesora za prirodnu posledicu imali su razvoj oblasti konkurentnog programiranja. Česta je praksa da se konkurentno programiranje prvi put detaljno izučava u okviru kursa iz oblasti operativnih sistema [20]. Imajući ovo u vidu, u nastavku je izložena kratka istorija konkurentne paradigme u okviru koje su detaljnije izloženi potreba za konkurentnim izvršavanjem i uticaj razvoja hardvera na razvoj paradigme. Izloženi pregled istorije predstavlja uvod u oblasti primene i podele u okviru paradigme, kao i glavne poteškoće pri dizajniranju pogramskih jezika koji podržavaju konkurentnost.

2 Pregled istorijskog razvoja konkurentnog programiranja

Konkurentno izvršavanje više procesa unutar jednog programa može se realizovati raspoređivanjem datih procesa na zasebne procesore ili deljenjem procesorskog vremena jednog procesora između više procesa. Deljenje procesorskog vremena podrazumeva particionisanje vremenskog intervala na jednake blokove i dodeljivanje svakog bloka jednom procesu. Politika distribucije opisanih vremenskih blokova jedan je od glavnih zadataka operativnog sistema [20].

Arhitektura ranih kompjutera, s kraja pedesetih godina prošlog veka, podrazumevala je jedan glavni procesor i više manjih, zasebnih, koji su bili zaduženi da paralelno sa glavnim tokom izvršavanja regulišu ulazno-izlazne operacije. Izvršavanje programa na ovim mašinama nije bilo konkurentno, i dalje je bilo neophodno sačekati kraj izvršenja trenutnog programa kako bi naredni program mogao ući u fazu izvršenja, jedina paralelizacija izvršavanja odvijala se u domenu ulazno-izlazne komunikacije. Šezdesetih godina 20. veka odvija se ubzani razvoj operativnih sistema, pojavljuje se koncept „planera“(eng. *scheduler*) kao centralnog dela operativnog sistema čiji je zadatak da koordiniše isprepletano izršavanje više zasebnih programa [20]. U ovom periodu nastaju mašine sa više „delimičnih procesora“ (eng *partial processor*). Na primer, jedna mašina je mogla imati dve ili više jedinice za rad sa brojevima u pokretnom zarezu. Time je bilo omogućeno paralelno izvršavanje više procesorskih instrukcija u okviru jedne programske naredbe [19]. Vremenom, konstanti pad cene računarskih resursa omogućio je praktičnu upotrebu koncepta „deljenja vremena“ (eng. *time-sharing*) koji podrazumeva istovremeno korišćenje jednog računara od strane više korisnika. Jedan od najpoznatijih operativnih sistema, Unix, razvijen je krajem šezdesetih godina sa ciljem realizacije koncepta „deljenja vremena“ na relativno jeftinim mašinama [20].

Period od pocečtka sedamdesetih, pa sve do ranih dvehiljaditih, obeležen je intenzivnim razvojem hardvera [19]. Značajno unapređenje performansi programa poboljšanjem hardvera i složenost impementacija konkurentnih konce-

pata stvorili su sentiment da je konkurentno programiranje isuviše komplikovano za programiranje korisničkih aplikacija [20]. Dva hardverska unapređenja omogućila su značajno bolje performasne već postojećeg softvera:

- eksponencijalni rast procesorske moći (Murov zakon, eng. *Mooore's law*) [17]
- implementacija „skrivenih" konkurentnih koncepata, nevidljivih programeru (eng. *Hidden concurrency*)

Skrivenim konkurentnim konceptima nazivaju se arhitekutralna poboljšanja procesora, poput:

- istovremene obrade više instrukcija u stilu „pokretne trake" (eng. *Instruction pipelining*), koja podrazumeva istovremeno izvršavanje prve, dekodiranje druge i dovlačenje treće instrukcije iz memorije
- upotrebe zasebnih kanala za protok informacija i podataka
- keširanja (eng. *caching*) instrukcija i podataka
- paralelizacija aritmetičkih operacija procesora

Sredinom dvehiljaditih na tržištu se pojavljuju prvi personalni kompjuteri sa višejezgarinim procesorima: čipom „Athlon 64 X2" [9] kompanije AMD i Intelovim Core 2 Duo [10]. U ovom periodu usporava trend eksponencijalnog rasta procesorske moći, raste broj jezgara po procesoru, a teret poboljšanja performansi programa premešta se na izdradu softvera i dizajn jezika koji će iskoristiti višejezgarne resurse [19]. Sve veća upotreba kompjutera kao multimedijalne mašine i „strimovanje" (eng. *streaming*) sadržaja sa interneta za čiju realizaciju je konkurentnost neophodna, značajno su uticali na razvoj konkurentnih mehanizama u okviru samih programskih jezika. Za kraj ovog istorijskog uvoda, primetimo da danas čak ni na tržištu mobilnih telefona više nema jednojezgarnih procesora.

3 Fizička i logička konkurentnost, hijerarhijska podela konkurentnosti i nivoi konkurentnosti

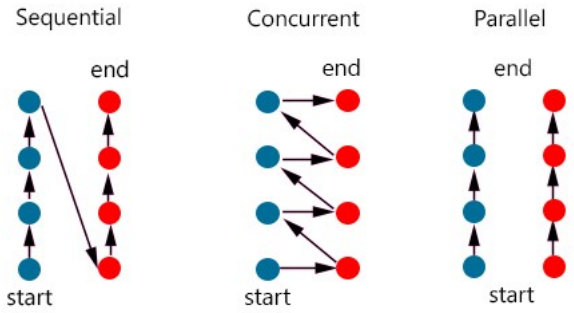
3.1 Fizička i logička konkurentnost

Konkurentne koncepte, zavisno od hardverske podrške, možemo podeliti u dve kategorije: fizičku i logičku konkurenost. Fizička konkurentnost podrazumeva izvršavanje više procesa na zasebnim procesorima, dok logička podrazumeva isprepletano izvršavanje više programa na jednom procesoru. Sa stanovišta programera i dizajna programskih jezika, logička konkurentnost ista je kao fizička. Zadatak je implementacije jezika da posredstvom operativnog sistema preslika logičku konkurentnost u odgovarajući hardver [20].

3.2 Hijerarhijska podela konkurentnosti

Tri su termina česta u upotrebi: konkurento, paralelno i distribuirano programiranje. Konkurentna paradigma podrazumeva svaki vid istovremenog izvršavanja

procesa ili programa na jednoj mašini. Paralelna paradigma je specijalizacija konkurentne paradigme, gde se svaki proces izvršava na zasebnom procesoru. (Slika 2) Distribuirana paradigma je specijalizacija paralelne paradigme u kojoj su procesori fizički razvojeni, na zasebnim mašinama, svaka sa zasebnom memorijom. Distribuirani sistemi danas su standard u mnogim oblastima informatike, poput distribuiranih baza podataka („Microsoft Azure Cosmos DB“ [8], „Google Cloud Spanner“ [4] i dr.) i internet igrica sa više igrača, gde svaki igrač pokreće igiricu na svom kompjuteru (npr. „League of Legends“ [7]).



Slika 2: Šematski prikaz sekvencijalno - konkurentno - paralelno izvršavanje

3.3 Nivoi konkurentnosti

Konkurentnost može biti realizovana na više načina i na različitim nivoima implementirana. Istorijski, četiri nivoa konkurentnosti prirodno su se izdvojila kao zasebne celine:

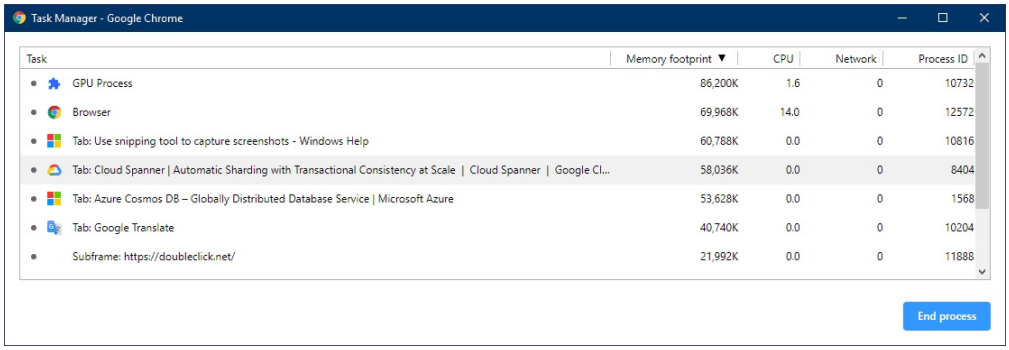
1. Nivo instrukcije – izvršenje dve ili više instrukcija. Dobar primer su već pomenuto paralelno izvršavanje više aritmetičkih operacija u okviru jednog izraza ili obrada pojedinačnih elemenata niza paralelno na više procesora [20].

2. Nivo naredbe – izvršavanje dve ili više naredbi jezika višeg nivoa istovremeno.

3. Nivo jedinica – izvršavanje dva ili više potprograma istovremeno. Dobar primer je pregledač „Google Chrome“ koji za svaku novootvorenu karticu pokreće novi proces u pozadini [3] (Slika 3)

4. Nivo programa – izvršavanje dva ili više programa istovremeno.

Prvi i četvrti nivo konkurentnosti ne utiču na dizajn programskog jezika. Implementacija konkurentnosti na nivou instrukcije zadatak je dizajnera kompajlera progamskog jezika, dok se implementacijom na nivou programa bave dizajneri operativnih sistema [19].



Slika 3: Prikaz procesa jedne instance Google Chrome pregledača

4 Veza sa programskim jezicima

Zadatak dizajnera programskih jezika je da obezbede konkurentnost na nivou naredbe i potprograma. Najznačajniji problemi u dizajnu jezika su rešavanje konflikata čitanja/pisanja nad resursima koje deli više procesa i sinhronizacija izvršavanja više procesa jednog programa. Pored pomenuta dva, nešto manje značajan, ali i dalje važan je problem uticaja procesa na odluke koje donosi „raspoređivač“ operativnog sistema (eng. *Process scheduler*). Danas, većina programskih jezika ima biblioteke i pakete za konkurentno programiranje, neki od njih su: C# Task Parallel Library [2], Java Thread [5] i Java Concurrent paketi [6], Python Concurrency paketi [12]. U slučaju interpretiranih programskih jezika poput Pajtona, nativna konkurentnost može biti samo logičkog tipa s obzirom da se program izvršava interpretiranjem instrukcija-po-instrukcija [19].

5 Skalabilnost i portabilnost

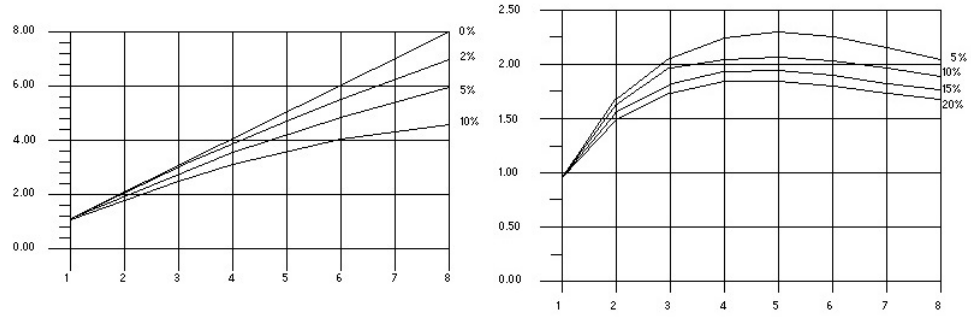
Jedan od ciljeva razvoja konkurentnog softvera je pisanje skalabilnih i portabilnih konkurentnih programa. Konkurentni algoritam je skalabilan ukoliko brzina njegovog izvšrenja raste sa porastom broja procesora. Poželjno je da pomenuto ubrzanje bude srazmerno povećanju broja procesora. Imajući u vidu porast broja procesora sa svakom novom generacijom mašina, skalabilnost je neophodna karakteristika svakog dobro-pisanog konkurentnog softvera [19].

Međutim, nije uvek moguće napisati linearno skalabilan program. Razlog tome jesu delovi programa koji moraju biti sekvencijalno izvršeni. Mera ubrzanja paralelizacijom data je čuvenim Amdalovim zakonom (eng. *Amdahl's law*). Amdalov zakon uspostavlja vezu izmedju ubrzanja, broja procesora i količine posla koja mora biti obavljena sekvencijalno:

$$U = \frac{1}{1 + \frac{1-s}{n}} \tag{1}$$

gde je U ubrzanje, s procenat broja instrukcija koje se ne mogu paralelizovati (npr. 10%), 1-s procenat instrukcija koje se mogu paralelizovati, a n broj raspoloživih procesora [16]. Prethodna formula grafički je prikazana na slici 4a. x-osa označava broj procesora **n**, na y-osi je ubrzanje **U**, dok su s desne strane

grafika naznačene količine sekvencijalnog posla **s** za svaku krivu na grafiku. U prethodnoj formuli nije uračunat posao sinhoronizacije rada više procesora. Kada se uračuna i posao sinhronizacije, dolazi se do zaključka da dodavanjem jezgara može doći i do degradacije performansi, što je i prikazano grafički na slici 4b [1].



Slika 4: a) Amdalov zakon bez uračunatog posla sinhronizacije procesora
b)Amdalov zakon sa uračunatim poslom sinhronizacije procesora

Životni vek hardvera je kratak i postoji mnoštvo različitih arhitektura, pa je iz tog razloga neophodno da napisani softver bude i portabilan. Međutim ovo je češće pitanje dizajna operativnog sistema, nego samog programa [19]. "IEEE standard POSIX 1003.1c Threadsđefiniše API koji bi operativni sistem koji podržava paralelno programiranje trebalo da implementira [15]. Među najpoznatijim operativnim sistemima koji implementiraju ovaj standard jesu macOS i Unix. Pored ovog standarda, česta je praksa da operativni sistem pruža i svoj nativni API za rad sa nitima i procesima [11] [13].

Sa pitanjem skalabilnosti programer se susreće već pri odluci kako posao raspodeliti na potprocese koji će se konkurentno izvršavati. Ovo je jedna od osnovnih odluka koju programer mora doneti prilikom pisanja programa. Jedna od uobičajenih strategija, pogodna na malim mašinama, je kreiranje posebne niti za svaki od glavnih zadataka ili funkicija programa. Na primer, u programu za obradu teksta zasebni procesi mogu biti: razdvajanje paragrafa na linije, obeležavanje strana, provera pravopisa i gramatike, prikazivanje slika, itd. Opisana strategija se često naziva paralelizam zadataka (eng. *Task parallelism*). Glavni nedostatak opisane strategije je loše skaliranje na veći broj procesora (biće iskorišćeno najviše onoliko procesora koliko ima zadataka). Rešenje ovog problema je primena tzv. paralelizma podataka (eng. *data parallelism*), gde se manje-više iste operacije primenjuju paralelno nad delovima nekog velikog skupa podataka koji se obrađuje. Na primer, program za obradu slika moze podeliti sliku na n manjih pravougaonika, a zatim na svakom od njih primeniti izabrani filter [18] [14].

Literatura

[1] Amdahl's law (Sun Studio 12: C User's Guide). on-line at: https://docs.oracle.com/cd/E19205-01/819-5265/bjaem/index.html.

[2] C# Task Parallel Library (TPL). on-line at: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl.

[3] Chrome processes. on-line at: https://developer.chrome.com/extensions/processes.

[4] Google Cloud Spanner. on-line at: https://cloud.google.com/spanner/.

[5] java.lang.thread Documentation. on-line at: https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html.

[6] java.util.concurrent Documentation. on-line at: https://docs.oracle.com/javase/8/docs/api/?java/util/concurrent/package-summary.html.

[7] League of Legends - Game Info. on-line at: https://na.leagueoflegends.com/en/game-info/.

[8] Microsoft Azure Cosmos DB – Globally Distributed Database Service. on-line at: https://azure.microsoft.com/en-us/services/cosmos-db/.

[9] PCMAG Encyclopedia - Athlon64 X2. on-line at: https://www.pcmag.com/encyclopedia/term/38109/athlon.

[10] PCMAG Encyclopedia - Intel Core 2 Duo. on-line at: https://www.pcmag.com/encyclopedia/term/56609/core-2.

[11] Processes and Threads | Microsoft Docs. on-line at: https://docs.microsoft.com/en-us/windows/desktop/procthread/processes-and-threads.

[12] Python Concurrency Library. on-line at: https://docs.python.org/3/library/concurrency.html.

[13] Threading Programming Guide | Apple Developer. on-line at: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/Multithreading/AboutThreads/AboutThreads.html.

[14] Turon Aaron. *Understanding and Expressing Scalable Concurrency*. PhD thesis, Boston, MA, USA, 2013. AAI3558728.

[15] Blaise Barney, Lawrence Livermore National Laboratory. POSIX Threads Programming. on-line at: https://computing.llnl.gov/tutorials/pthreads/.

[16] David A. Padua, editor. *Encyclopedia of Parallel Computing*. Springer, 2011.

[17] Schaller, Robert R. Moore's law: Past, present, and future. *IEEE Spectr.*, 34(6):52–59, June 1997.

[18] Michael L. Scott. *Programming Language Pragmatics, Third Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2009.

[19] Robert W. Sebesta. *Concepts of Programming Languages (3rd Ed.)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.

[20] Allen B. Tucker, Jr. and Robert E. Noonan. *Programming Languages: Principles and Paradigms*. McGraw-Hill Higher Education, 1st edition, 2001.