LABORATORIJSKE VEŽBE IZ PREDMETA PARALELNA OBRADA

1. KONKURENTNO PROGRAMIRANJE U PROGRAMSKOM JEZIKU BACI

Zadatak 1. Napisati program koji konkurentno izvršava procedure Da i Ne. Obe ove procedure ispisuju 50 puta reč "D", odnosno "N" bez prelaska u novi red.

1. Izvršiti program nekoliko puta. Prokomentarisati dobijene rezultate.
2. Reci "D" i "N" zameniti rečima "Da" i "Ne". Program rekompajlirati i ponovo izvršiti nekoliko puta. Prokomentarisati dobijene rezultate
3. Izvršavati program korak po korak, tako da se na izlazu dobije najpre 50 puta reč "Da" a zatim 50 puta reč "Ne".

Zadatak 2. Procedura `P` povećava vrednost promenljive `x` za 1 ukupno 100 puta. Napisati program koji izvršava konkurentno dva poziva procedure `P`. Na kraju, program ispisuje konačnu vrednost promenljive `x`.

1. Izvršiti program nekoliko puta i uočiti vrednosti promenljive x koje program ispisuje. Objasniti zašto su ove vrednosti različite, kad pri proizvoljnom scenariu izvršenja naredbi dva poziva procedure `P`, vrednost promenljive `x` je na kraju uvek 200.
2. Izvršavati sada program naredbu po naredbu (malo jedan, malo drugi poziv procedure P) i uočiti vrednost promenljive `x` posle N izvršenja naredbe `x:=x+1` (u oba poziva). Ponoviti postupak za drugu vrednost broja N. Zašto se u ovom slučaju dobijaju iste vrednosti za vrednost promenljive `x`?
3. Restartovati program, i uočiti PCODE za posmatrani program. Šta se može primetiti vezano za broj naredba u PCODE-u i u programu? Da li su naredbe programa atomične? A šta je sa instrukcijama u PCODE-u?
4. Ponovo izvršavati program naredbu po naredbu (ponovo malo jedan, malo drugi poziv procedure P), ali sada koristeći taster "Step PCODE". Koja se vrednost promenljive x sada dobija posle N poziva naredbe `x:=x+1`? Zašto?
5. Konstruisati scenario tako da je vrednost promenljive `x` jednaka 5 posle ukupno 10 poziva naredbe `x:=x+1`. Proveriti konstruisan scenario izvršavanjem programa instrukciju po instrukciju PCODE-a.

Zadatak 3. (prvo rešenje problema kritične sekcije) Implementirati rešenje problema kritične sekcije izmedju dva procesa u kome procesi dele zajedničnu promenljivu red i na osnovu nje pristupaju kritičnoj sekciji. Prvi proces najpre čeka sve dok promenljiva red ima vrednost 2, zatim ulazi u kritičnu sekciju i na kraju postavlja vrednost promenljive red na 2. Analogno radi i drugi proces. Neka oba procesa u kritičnoj sekciji 10 puta ispišu svoj redni broj.

1. Izvršiti program više puta.
2. Da li je prilikom nekog od izvršenja došlo do deadlock-a? Zašto?
3. Da li su dva procesa u nekom izvršenju bila zajedno u kritičnoj sekciji? Objasniti način na koji ste došli do zaključka?
4. Da li ova dva procesa pristupaju kritičnoj sekciji konkurentno? Šta će se desiti ukoliko jedan od procesa nastrada neposredno po izlasku iz kritične sekcije? Simulirati ovu situaciju izvršavanjem programa naredbu po naredbu na sledeći način: najpre izvršavati naredbe prvog procesa sve do izlaska iz kritične sekcije, a zatim naredbe drugog procesa.

5. Dodati prvom procesu deo koda na izlazu iz kritične sekcije koji će da uspori njegovo izvršavanje (npr. petlju dužine 200 koja promenljivi tmp dodeljuje vrednost 1). Izvršiti program.

Zadatak 4. (drugo rešenje problema kritične sekcije) Da bi popravili nedostatak rešenja izloženog u prethodnom zadatku, svakoj proceduri ćemo dodeliti sopstveni ključ za ulazak u kritičnu sekciju. To su globalne promenljive `c1` i `c2`. Njihova vrednost 0 znači "zaključano", tj. da je proces u kritičnoj sekciji dok vrednost 1 znači "otključano", tj. da proces nije u kritičnoj sekciji. Prvi proces najpre čeka sve dok je drugi u kritičnoj sekciji a zatim ulazi. Pre nego što udje, postavlja vrednost svog ključa na 1. Kritična sekcija je identična kao u predhodnom zadatku.

1. Izvršiti program više puta.
2. Da li su dva procesa u nekom izvršenju bila zajedno u kritičnoj sekciji?
3. Da li postoji scenario u kome sinhronizacija radi korektno?
4. Da li je veća verovatnoća (pod uslovom da oba procesa imaju iste prioritete i da se izvršava samo jedna iteracija ciklusa) da procesi ne budu zajedno u kritičnoj sekciji ili da budu?

Zadatak 5. (treće rešenje problema kritične sekcije) Uvedimo indikatore `c1` i `c2`, pri čemu oni označavaju da odgovarajući proces zeli da udje u kritičnu sekciju. Neka sada prvi proces postavljanjem `c1` na 0 pokazuje nameru da udje u kritičnu sekciju, a onda ispituje da li je drugi u svojoj kritičnoj sekciji, i ako jeste, čeka.

1. Izvršiti program više puta.
2. Može se uočiti da ovo rešenje pati od deadlock-a. Konstruisati jedan scenario koji vodi u deadlock i to proveriti izvršavanjem programa naredbu po naredbu.
3. Da li postoji scenario pri kom program korektno radi? Ukoliko postoji, konstruisati jedan takav i proveriti ispravnost pomoću BACI interpretera.
4. Da li su procesi bili zajedno u kritičnoj sekciji u nekom od izvršenja?

Zadatak 6. (četvrto rešenje problema kritične sekcije) U predhodnom rešenju deadlock nastaje kada oba procesa čekaju da ovaj drugi završi kritičnu sekciju i da postavi svoj indikator na 1. Jedno rešenje ovog problema je da proces prilikom čekanja, za trenutak odstupi od svoje namere da udje u kritičnu sekciju (prvi proces izvršavajući naredbe `c1:=1; c1:=0`) kako bi drugi mogao da u tom trenutku napusti ciklus čekanja i udje u kritičnu sekciju.

1. Izvršiti program više puta.
2. Da li sada progam pati od deadlock-a i da li procesi mogu zajedno da se nadju u kritičnoj sekciji?
3. Konstruisati beskonačno dug scenario pri kome ipak dolazi do deadlock-a. Izvršavati program naredbu po naredbu prema konstruisanom scenariju nekoliko iteracija.
4. Kolika je verovatnoća deadlock-a u ovom slučaju?

Zadatak 7. Implementirati Dekerov algoritam kojim se uspešno rešava problem kritične sekcije dva procesa.

1. Izvršenjem programa nekoliko puta uočiti da ovo rešenje korektno radi.
2. Dodati prvom procesu deo koda na izlazu iz kritične sekcije koji će da uspori njegovo izvršavanje (npr. petlju dužine 200 koja promenljivi tmp dodeljuje vrednost 1). Izvršiti program.
3. Prokomentarisati brzinu izvršenja programa i broj prolaza kroz kritičnu sekciju oba procesa.

Zadatak 8. Implementirati Dijkstrin algoritam kojim se uspešno rešava problem kritične sekcije više od dva procesa. Ponoviti korake kao u predhodnom zadatku.

2. SEMAFORI I MONITORI

Zadatak 1. (problem kritične sekcije) Procesi `A` i `B` se izvršavaju konkurentno, i svaki treba da izvrši proceduru `crit` koja predstavlja kritičnu sekciju (procesi `A` i `B` ne smeju u isto vreme izvršavati proceduru `crit`). Napisati program kojim se, koristeći semafore, ovaj uslov realizuje. Redosled kojim `A` i `B` izvršavaju kritičnu sekciju nije bitan i ne sme biti odredjen programom.

1. Neka procedura `crit` ispisuje 20 puta broj `x` koji je njen jedini argument. Procedure (procesi) `A` i `B` pozivaju ovu proceduru, pri čemu su vrednosti argumenta 1 i 2 respektivno. Izvršiti program nekoliko puta.
2. Na osnovu izlaza programa ustanoviti da li program radi korektno i objasniti način na koji se to može ustanoviti.
3. Kakve posledice nastaju ukoliko se vrednost semafora ne inicijalizuje korektno? Kako će se program ponašati ukoliko je početna vrednost semafora jednaka 0, odnosno 2?
4. Na sličan način formirati treći i četvrti proces (`C` i `D`) i izvršiti potrebnu sinhronizaciju.
5. Uporediti ovo rešenje pomoću semafora sa rešenjima zadataka iz predhodne oblasti.
6. Dodati da se unutar funkcije `crit` inkrementira vrednost globalne promenljive g za 1. Na početku programa se ova vrednost inicijalizuje na 0 a na kraju ispisuje njena vrednost. Da li se ova vrednost razlikuje pri različitim izvršenjima programa?

Zadatak 2. (multipleks) Posmatrajmo modifikovani problem kritične sekcije za više od dva konkurentna procesa. Pri tome pretpostavimo da funkciju `crit` mogu da izvršavaju više procesa istovremeno, pri čemu je gornja granica za taj broj procesa ograničena brojem n. Drugim rečima, ne više od n procesa mogu da izvršavaju kritičnu sekciju konkurentno. Napisati program kojim se to obezbedjuje, ako je ukupan broj procesa $N = 10$ i $n = 2$.

1. Implementirati proceduru `crit`, tako da ona prvo ispisuje vrednost argumenta, zatim izvršava neko računanje a zatim ponovo ispisuje vrednost argumenta. Izvršiti program nekoliko puta.
2. Na osnovu izlaza programa ustanoviti da li program radi korektno i objasniti način na koji se to može ustanoviti.
3. Odrediti (na osnovu izlaza programa) koliko je procesa bilo u odredjenom trenutku u kritičnoj sekciji i koji je maksimalni broj procesa koji su u nekom trenutku bili u kritičnoj sekciji.

Zadatak 3. (barijera) Predpostavimo da N procesa mora da izvrši dve procedure: `rendezvous` i `critical_point`, pri čemu proceduru `critical_point` proces može da se izvrši tek kad svi ostali procesi izvrše proceduru `rendezvous`. Napisati program koji rešava ovaj problem, pri čemu osim pomenutog uslova nikakav drugi redosled izvršenja procedura ovih procesa nije bitan i ne sme biti definisan programom. Neka je broj procesa jednak 10.

1. Implementirati procedure `rendezvous` i `critical_point` tako da ispisuju redom "Rendezvous" i "Critical Point", kao i vrednost argumenta. Procesi redom pozivaju ove procedure sa argumentima *1, 2, … , N*. Izvršiti program nekoliko puta.
2. Modifikovati program tako da procesi izvršavaju navedene procedure više puta. Pronaći scenario pri kome program ne radi korektno.
3. Ispraviti ovo rešenje uvodjenjem još jedne "barijere".

Zadatak 4. (problem proizvodjača i potrošača). Imamo dve vrste procesa, pri čemu se svi procesi izvršavaju konkurentno. Prva vrsta procesa (proizvodjači) proizvode (nabavljaju) neke podatke dok druga vrsta procesa (potrošači) te podatke troši (obradjuje). Kada neki proizvodjač proizvede podatak, on ga ubaci u bafer iz kog potrošač uzima podatke. Za vreme dok se neki podatak dodaje u bafer ili uzima iz bafera, bafer je u nekonzistentnom stanju, pa proces koji dodaje ili uzima podatak mora imati ekskluzivno pravo pristupa. Takodje, ukoliko potrošač

pokuša da uzme podatak iz praznog reda, on se blokira sve dok neki proizvodjač ne doda podatak u red. Napisati program kojim se uspostavlja opisana sinhronizacija izmedju potrošača i proizvodjača.

1. Implementirati bafer kao FIFO red (pomoću niza i dva pokazivača). Inicijalizovati elemente baferskog niza na vrednost -1, i prilikom uzimanja elementa iz bafera, umesto preuzetog elementa staviti -1. Podaci koji se proizvode su slučajni brojevi od 0 do 9, a potrošači ispisuju podatak koji su dobili. Izvršiti program nekoliko puta.
2. Utvrditi korektnost implementacije i prokomentarisati način na koji je to ustanovljeno.
3. Objasniti koji deo koda u konkretnom slučaju predstavlja kritičnu sekciju.

Zadatak 5. (problem filozofa koji večeraju, Dijkstra 1965). U jednom manastiru postoji 5 filozofa čiji se život sastoji u razmišljanju i jelu. U trpezariji postoji okrugli sto sa 5 stolica, 5 tanjira, 5 viljuški i jednom velikom činijom sa špagetama u sredini. Svakom filozofu su potrebne dve viljuške da bi jeo. Označimo filozofe i viljuške ciklično brojevima 0, 1, 2, 3 i 4 kao na slici. Da bi i-ti filozof jeo, on mora da uzme i-tu i i+1-vu viljušku (pretpostavimo da je sabiranje po modulu 5). Svaki filozof naizmenično jede i misli. Napisati program koji simulira ovakav život filozofa.

1. Implementirati rešenje ovog problema korišćenjem niza semafora koji označavaju zauzetost pojedine viljuške, pri čemu i-ti filozof **prvo** uzima i-tu, **pa onda** i+1-vu viljušku.
2. Uočiti scenario koji implementirano rešenje vodi u deadlock. Izvršavati program po ovom scenariju (korak po korak) i pokazati da zaista nastaje deadlock.
3. Modifikovati predhodni metod uvodjenjem dodatnog semafora `table`. Filozofi najpre sednu za sto, a zatim uzimaju viljuške. Pri tome, najviše 4 filozofa može u istom trenutku sedeti za stolom.
4. Izvršiti modifikovano rešenje nekoliko puta. Da li sada dolazi do deadlock-a?
5. Modifikovati sada polazno rešenje tako da filozofi ne uzimaju viljuške u naznačenom redosledu (npr. filozofi sa parnim rednim brojevima poštuju redosled, a filozofi sa neparnim rednim brojevima uzimaju viljuške u suprotnom smeru). Da li sada dolazi do deadlock-a?

Zadatak 6. Posmatrajmo drumski most koji je toliko uzan da na njemu postoji samo jedna traka za kretanje vozila. Preko mosta može preći i više vozila u isto vreme, pod uslovom da se svi kreću u istom smeru. Ako bi istovremeno krenula vozila koja se kreću u suprotnim smerovima, dogodio bi se zastoj (ako ne i sudar). Dakle, ako neko vozilo želi da predje preko mosta, vozač mora biti siguran da niko trenutno ne prelazi most u suprotnom smeru. Napisati program koji pomoću monitora simulira prelazak vozila preko mosta.

1. Implementirati dve procedure koje predstavljaju vozila koja idu u direkntom odnosno suprotnom smeru. Prilikom prelaska mosta procedure ispisuju 3 puta slova ‘f’ odnosno ‘b’. Izvršiti program nekoliko puta.
2. Da li program radi ispravno? Zašto?
3. Izvršavanjem programa korak po korak pokazati da je pristup monitoru kritična sekcija i da u istom trenutku samo jedan proces može izvršavati poziv procedure unutar monitora. Koja je razlika izmedju `condition` promenljivih i semafora?

Zadatak 7. (simulacija običnog semafora preko monitora) Izvršiti simulaciju jedne promenljive tipa `semaphore` pomoću monitora. Implementirati procedure `PP` i `VV` koje će imati istu funkciju kao procedure `wait` i `signal`. Pomoću ove implementacije rešiti zadatak 2.

1. Izvršiti program nekoliko puta i prokomentarisati dobijene rezultate.
2. Da li su ovako implementirane procedure `PP` i `VV` atomične? Zašto?

3. KONKURENTNO PROGRAMIRANJE U PROGRAMSKOM JEZIKU ADA

Zadatak 1. Napisati program u programskom jeziku ADA koji se sastoji od dve grupe taskova (dva task tipa). Svaki task iz prve grupe ispisuje karakter x, a iz druge, karakter * ukupno 10 puta.

1. Formirati dva taska iz prve i jedan iz druge grupe. Izvršiti program nekoliko puta i posmatrati rezultat na izlazu.
2. Pošto imamo samo jedan task iz druge grupe, da li je neophodno deklarisati task tip?

Zadatak 2. Rešiti problem kritične sekcije pomoću Randez-Vous komunikacije u programskom jeziku ADA. Kritična sekcija se sastoji od inkrementiranja vrednosti zajedničke promenljive s. Na kraju programa ispisati vrednost promenljive s.

1. Semafor realizovati kao poseban task koji ima dva entry-ja, `wait` i `signal`. U telu taska semafor, naizmenično se prihvataju komunikacije preko ova dva entry-ja.
2. Izvršiti program nekoliko puta i ustanoviti da li ispravno radi.
3. Da li je na ovaj način ispravno simuliran binarni semafor? Ukoliko jeste, pokazati da su ispunjeni svi uslovi koje zadovoljavaju binarni semafori, ukoliko nije naći scenario koji u kome se ovako definisan semafor ne ponaša na uobičajen način.

Zadatak 3. Izvršiti simulaciju semafora pomoću sistema Randez-Vous u programskom jeziku ADA. Semafor predstaviti kao poseban task a operacije `signal`, `wait` i `initialsem` neka predstavljaju svaka po jedan entry. Pomoću tako definisanog semafora rešiti problem multiplex (zadatak 2 iz predhodnog odeljka). Neka ima ukupno 20 taska i neka najviše 3 mogu da budu u isto vreme u kritičnoj sekciji.

1. Implementirati telo taska semafor i pomoću naredbe select rešiti problem iz predhodnog zadatka.
2. U kom se redosledu bude uspavani procesi na ovom semaforu? Da li je ovako definisan semafor slab ili jak?
3. Implementirati kritičnu sekciju na isti način kao u zadatku 2 u predhodnom odeljku. Izvršiti program nekoliko puta i prokomentarisati dobijeni rezultat.

Zadatak 4. Napisati program na programskom jeziku ADA koji simulira studentske konsultacije. Student uči. Kad mu nešto nije jasno, odlazi na fakultet na konsultacije kod bilo kog slobodnog profesora. Ako su svi profesori zauzeti, student čeka dok neki od profesora ne postane slobodan, a zatim ulazi. Ima 30 studenata i 3 profesora i svaki student ide po 10 puta na konsultacije.

1. Implementirati telo taska `student` tako da student redom ispituje da li je svaki od profesora slobodan, i opredeljuje se za odredjenog profesora samo u slučaju da je on u tom trenutku slobodan. Ukoliko nema slobodnih profesora u odredjenom trenutku, student nastavlja da proverava sve dok se neki profesor ne oslobodi.
2. Izvršiti program nekoliko puta i utvrditi da li ispravno radi. Ispisivanje poruke na ekran implementirati kao kritičnu sekciju uvodjenjem novog taska `ispis`.
3. Šta se može reći o brzini i efikasnosti ove implementacije?
4. Modifikovati predhodno rešenje uvodjenjem taska `pusher`. Uloga ovog taska je posredovanje izmedju studenata i profesora. Student pomoću poziva entry procedure ovog taska dobija informaciju koji je profesor slobodan, odnosno čeka dok se bilo koji profesor ne oslobodi. Nakon završetka konsultacija, profesor obaveštava task `pusher` o tome pozivom entry procedure.
5. Izvršiti rešenje nekoliko puta i prokomentarisati efikasnost i korektnost modifikovanog rešenja.

Zadatak 5. Rešiti problem proizvodjača i potrošača (zadatak 4 u predhodnom odeljku). Napraviti task tipove `producer` i `consumer`, kao i task `buffer` koji predstavlja ograničeni bafer. Čitanje i upis podataka u bafer odvija se pomoću entry-ja `add` i `take`. Bafer realizovati kao kružni `FIFO` red dužine `size=10`. Ukupno ima 3 proizvodjača i 2 potrošača. Proizvodjači proizvode podatke pomoću funkcije `produce` a potrošači ih obradjuju pomoću procedure `doit`. Ova procedura ne radi ništa.

1. Prilikom ubacivanja podatka u bafer, kao i uzimanja podatka iz bafera ispisati trenutnu zauzetost bafera. Izvršiti program nekoliko puta i prokomentarisati dobijene rezultate.
2. Dodati unutar procedure `doit` čekanje od 0.01s. Izvršiti program nekoliko puta i ponovo prokomentarisati dobijene vrednosti za zauzetost bafera.

Zadatak 6. Napisati program na programskom jeziku ADA kojim se simulira susret dva prijatelja u parku. Prijatelji stižu u slučajnim trenucima, pri čemu jedan stigne, čeka drugog 10 sekundi. Ukoliko drugi ne dodje u tom vremenskom intervalu, prvi odlazi. Oba prijatelja dolaze u periodu od 0 do 30 sekundi. Za generisanje slučajnih brojeva koristiti funkciju `random`, koja ima jedan argument `x` i generiše jedan slučajni broj u intervalu 0..`x`.

1. Izvršiti program nekoliko puta i prokomentarisati dobijene rezultate?
2. Koji oblik select naredbe se koristi prilikom implementacije?

4. PROGRAMSKI JEZIK PARALLAXIS

Zadatak 1. Napisati program na programskom jeziku PARALLAXIS kojim se u procesorskom polju sa konfiguracijom nepovezane liste, primenom redukcije podataka, računa vrednost odredjenog integrala

$$\int_0^{+\infty} \frac{4}{1+x^2}\,dx$$

Raspolažemo sa ukupno 1000 procesna elementa.

Zadatak 2. Napisati program na programskom jeziku PARALLAXIS kojim se računa vrednost kosinusa korišćenjem Tejlorovog reda

$$\cos x = \sum_{n=0}^{+\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

Uporediti dobijenu vrednost sa vrednošću standardne funkcije COS za isti argument. Predvideti učitavanje argumenta x i tačnosti ε. Predvideti štampanje vrednosti argumenta x, zadate tačnosti, vrednosti izračunate sume, vrednosti standardne funkcije COS, razlike ovih vrednosti i broja korišćenih procesorskih elemenata. Za rešavanje problema koristiti jednodimenzionalno procesorsko polje sa sprežnom strukturom linearne liste.

Zadatak 3. Napisati program kojim se na programskom jeziku PARALLAXIS nalazi sve proste brojeve manje ili jednake 200 pomoću Eratostenovog sita.

Zadatak 4. Napisati program za izračunavanje integrala:

$$I = \int_0^{+\infty} \frac{4}{1+\sin^2(x)}\,dx$$

Monte-Carlo metodom na 1000 procesnih elemenata. Kod Monte-Carlo metode, vrednost integrala se izračunava aproksimativno izrazom

$$\int_a^b f(x)dx \approx \frac{f(x_1)+\ldots+f(x_n)}{n}$$