Sustavi za rad u stvarnom vremenu

8. Posebnosti izgradnja programske potpore za SRSV-e pitanja za ponavljanje i odgovori

1. Ada vs C u kontekstu SRSV?

ada ima mehanizme za SRSV-e ugrađene u sam jezik:

- podrška za rad u stvarnom vremenu
- upravljanje dretvama
- podrška radu UI naprava

s druge strane C ima proširenja (biblioteke) za podršku SRSV-ima

ada ima vrlo strogu provjeru tipova podataka: možeš definirat da prima vrijednosti samo između 5 i 10, prevoditelj baca grešku ak skuži da trpaš neš drugo

2. koja svojstva prevoditelja je moguće ili potrebno koristiti pri izgradnji programske potpore za SRSV-e

- javlja greške (samo sintaksne tho) koje treba popraviti
- javlja upozorenja - pametno ih je popravit, iako su neka glupa ne želiš da ti se jedno pametno ugura u more glupih

  zastavice:

  - Wall: aktiviraj sve razine upozorenja
  - Werror: tretiraj upozorenja kao errore, nedaj dalje dok ih programer ne riješi

- optimira: npr preskače neku petlju u kojoj se niš ne desi, ali i pametnije stvari. ima više razina

  zastavice:

  - O1: osnovna razina optimizacije
  - O3: max razina optimizacije
  - Os: ???
  - O0: isključi optimizaciju

3. Kad ugrađivati funkcije na mjesto poziva (definirati ih s inline), a kada ne?

btw to nekad i prevoditelj sam radi kad optimira (ugradi funkciju na mjesto poziva)

da → kad želimo ubrzati rad programa

ne → kad želimo da nam program bude kratak, malen

4. Čemu služi oznaka volatile pri definiranju varijabli? koje probleme rješava?

ako npr imamo praznu petlju prevoditelj će je u postupku optimizacije vjerojatno izbaciti

ako je na silu želimo ostaviti uvedemo spremničku barijeru s tom oznakom volatile i onda ju prevoditelj ne dira

5. za kakve funkcije kažemo da nisu prikladne za višedretveno okruženje?

za one funkcije koje koriste globalne ili statičke varijable

6. granice zapisa cijelih tipova podataka / koja je preciznost realnih brojeva?

nećemo to učit, treba samo znati koja je veća od koje pa kad u zadatku dođe jedna i pita te se koji su mogući problemi i rješenja → ta se može over/underflowat i rješenje je uzet veću

7. Na što je sve potrebno paziti pri korištenju višeprocesorskih i višejezgrinih sustava (kod raspoređivanja dretvi) u SRSV okruženju?

- može se dogodit da se dvije dretve (različitih prioriteta) paralelno izvode na dvije različite procesorske jedinke
- ako dretva manjeg prioriteta radi intenzivne promašaje (traži podatke koji nisu u priručnom spremniku procesora)
    - onda dretva većeg prioriteta može bit usporena

8. Kako se ponekad može smanjiti vremenska složenost postupka?

korištenjem tablica - vrijednosti se izračunaju prije pokretanja kad vrijeme nije kritično za sve moguće vrijednosti ulaza, al time se poveća prostorna složenost, a nekad nije ni moguće tako

9. što je to prostorna složenost?

za n ulaznih podataka gornja ograda potrebnog prostora će biti proporcionalna s onim što je u zagradi (npr. n^2)

10. kako ispitni uzorci mogu utjecati na razvoj programske potpore?

- često predstavljaju scenarije uporabe i mogu biti pokretač izgradnje, nešto oko čega će se oblikovati arhitektura sustava i komponente
- mogu poslužiti kao provjera specifikacije, je li definirano zaista ono što sustav treba raditi

11. Za zadani odsječak koda navesti nedostatke (u kontekstu korištenja u SRSV-ima).

```
printf("Unesi parametar A (5-10):");
scanf("%d", &A);
a = obrada (A);
if ( a > 5 )
exit(-1);
else
posalji(A, a);
```

?????

12. Za dodjelu jedinstvenih identifikacijskih brojeva koristi se funkcija: int id (){ static int broj = 0; return broj++; }. Koje nedostatke ima navedena funkcija?

koristi se statički deklarirana varijabla, kada se paralelno pozovu 2+ dretve mogli bi dvaput dodijelit isti id

rješenja:

- zaključati od istovremenog korištenja

- svakoj dretvi davati svoje podatke na način da ih šaljemo u funkciju

13. Navesti nedostatke navedena koda (u raznim "okruženjima") i mogućnosti za njihovo rješavanje.

```
int op ( ... ) {
static int brojilo = 0;
... /* neka operacija koja ne koristi varijablu brojilo */
brojilo = brojilo + 1;
return brojilo;
}
```

ista stvar ko gore, statički deklarirana varijabla.

možemo prije i poslije kritičnog odsječka (brojilo = brojilo + 1) dodat pthread_mutex_lock/unlock(&KO)

možemo koristit nedjeljive operacije nad operandima tipa "fetch and add" koji ne razumem

14. Odrediti složenost idućeg algoritma u O (veliko O) notaciji (funkcija1 je linearne složenosti – O(N)).

```
for ( i = 0; i < N; i++ )
      for ( j = i + 1; j < N; j++ )
            A[i][j] = funkcija ( i, j, N );
```

→ O(n^3)

15. Popraviti sljedeću funkciju tako da općenito bude maksimalno moguće precizna uz zadane tipove podataka (bez mijenjanja tipova).

```
double funkcija2 ( double a, double b ){
      double c;
      c = 1 - a / 1e50;
      if ( b > 0 )
            c = b * (c - 1);
      return c
```

Mislim da tu može bit fora da iskoristiš dve varijable umjesto jedne za spremanje vrijednosti (kao kad smo koristili 2 registra u arh1)

Druga opcija kod takvih zadataka je pretumbat redosljed izvođenja akcija da se zadrži preciznost, a ne promijeni logika.

16. Koji problemi preciznosti mogu nastati izvodenjem programa, ako su sve varijable:

a) cjelobrojne

b) realne?

```
x = a - b*c;

y = d - x;
```

a) underflow -> za velike b,c se može dogodit da x,y bude toliko mali da je opet velik

b) gubitak preciznosti, nemrem napamet smislit primjer sad ?????

17. Između očitanja neke ulazne naprave treba proći izmedu 20 i 50 sabirničkih ciklusa.

Kako to ostvariti ako se može programirati u asembleru, te kako ako se mora koristiti

C?

Nisam 100% za ovaj odgovor, ali mislim da: u asembleru samo korištenjem prazne petlje, u C-u prazna petlja s memorijskom barijerom.