

Napredni algoritmi i strukture podataka

Rad sa fajlovima, serijalizacija



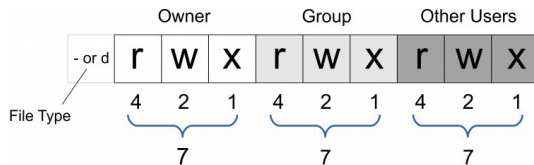
Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Vrste fajlova

- ▶ Fajl predstavlja kolekciju podataka trajno uskladištenih na disku pod određenim nazivom
- ▶ Sadržaj svakog fajla je niz nula i jedinica, a način njihove interpretacije je ono što razlikuje različite formate datoteka
- ▶ Osnovna podela datoteka jeste na binarne i tekstualne
- ▶ Tekstualni fajlovi sačinjeni su iz niza karaktera transformisanih u nizove bitova prema određenom enkodingu (UTF-8, UTF-16, ASCII ...)
- ▶ Za binarne fajlove ovo pravilo ne važi i za svaki format se definiše koji tip sadržaja čuva i na koji način se taj sadržaj zapisuje i interpretira

Permisije nad fajlovima

- ▶ Većina fajl sistema svakom fajlu dodeljuje attribute koji definišu koji korisnik sme da vrši koje akcije nad tim fajlom
- ▶ Unix sistemi razlikuju tri akcije odnosno dozvole - čitanje, pisanje i izvršavanje
- ▶ Dozvole se dodeljuju odvojeno za tri klase - vlasnika, grupu i ostale korisnike



Kreiranje

- ▶ Novi fajl može biti kreiran pomoću funkcije `Create()` paketa `os`
- ▶ Jedini parametar funkcije je naziv fajla, navodi se relativna ili apsolutna putanja do odredišnog direktorijuma, a zatim i sam naziv fajla
- ▶ Funkcija vraća pokazivač na novokreirani fajl i grešku

```
file, err := os.Create("somefile")
if err != nil {
    panic(err)
}
```

Brisanje

- ▶ Brisanje postojećeg fajla obavlja se funkcijom `Remove()` paketa `os`
- ▶ Jedini parametar funkcije je naziv fajla, navodi se relativna ili apsolutna putanja do odredišnog direktorijuma, a zatim i sam naziv fajla
- ▶ Funkcija vraća samo grešku do koje je potencijalno došlo

```
err := os.Remove("somefile")
if err != nil {
    panic(err)
}
```

Dobavljanje informacija

- Informacije o fajlu kao što su njegov naziv, veličina u bajtima, kada je poslednji put izmenjen, da li je direktorijum itd. mogu se dobiti upotrebom funkcije Stat() paketa os

```
fileInfo, err := os.Stat("somefile")
if err != nil {
    panic(err)
}
fmt.Println("File name:", fileInfo.Name())
fmt.Println("Size in bytes:", fileInfo.Size())
fmt.Println("Is Directory: ", fileInfo.IsDir())
```

- Na sledeći način proveravamo da li fajl postoji

```
if _, err := os.Stat(fileName); err != nil {
    if os.IsNotExist(err) {
        return false
    }
}
```

Izmena naziva i premeštanje

- ▶ Fajl može biti preimenovan (premešten izmenom putanje) pomoću funkcije `Rename()` paketa `os`
- ▶ Prvi parametar je stara, a drugi nova putanja do fajla
- ▶ Funkcija vraća samo grešku do koje je potencijalno došlo

```
err := os.Rename("somefile", "somefile2")
if err != nil {
    panic(err)
}
```

Otvaranje i zatvaranje

- ▶ Funkcija `os.Open()` otvara fajl (ukoliko on postoji) u read-only režimu i vraća pokazivač na otvoreni fajl i grešku
- ▶ Zatvaranje fajla vrši se pozivom metode `Close()` nad pokazivačem na fajl

```
file, err := os.Open(fileName)
. . .
file.Close()
```
- ▶ Kada želimo da menjamo sadržaj fajla, moramo koristiti funkciju `os.OpenFile()`
- ▶ Prvi parametar funkcije je naziv fajla, nakon čega sledi flag koji naznačava u kom režimu se fajl otvara, posle čega se navode dozvole
- ▶ Moguće vrednosti flag-a su `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_APPEND`, `O_CREATE`, `O_EXCL`, `O_SYNC`, `O_TRUNC`
- ▶ Može se navesti više od jednog flag-a, između kojh se navodi OR (`|`) operator

```
file, err = os.OpenFile("somefile", os.O_RDWR | os.O_CREATE, 0666)
```


Kopiranje

- ▶ Kopiranje sadržaja fajla vrši se tako što se otvori postojeći i kreira novi fajl, a zatim pozove funkcija `os.Copy()` koja kao parametre prima pokazivač na novi fajl i pokazivač na postojeći fajl
- ▶ Povratnu vrednost čine broj kopiranih bajtova i greška
- ▶ Nakon toga potrebno je pozvati metodu `Sync()` nad novim fajlom kako bi izmene bile zapisane na disk

```
bytesWritten, err := io.Copy(newFile, originalFile)
. . .
err = newFile.Sync()
```

Pozicioniranje

- ▶ Pozicioniranje na tačno određeno mesto (bajt) u fajlu pred naredno čitanje ili pisanje vrši se metodom `Seek()` koja se poziva nad fajlom
- ▶ Prvi parametar funkcije je offset (za koliko bajtova se vrši pomeranje), dok je drugi parametar flag koji određuje u odnosu na šta se vrši pomeranje
- ▶ Moguće vrednosti flag-a:
 - ▶ 0 - U odnosu na početak fajla
 - ▶ 1 - U odnosu na trenutnu poziciju
 - ▶ 2 - U odnosu na kraj fajla

```
// pomeramo se na peti bajt u odnosu na početak fajla  
newPosition, err := file.Seek(5, 0)
```

Pisanje I

- ▶ Jedan od načina za zapisivanje sadržaja u fajl je pomoću Write() metode pokazivača na fajl
- ▶ Jedini parametar funkcije je niz bajtova koje treba zapisati
- ▶ Povratne vrednosti su broj zapisanih bajtova i greška

```
file, err := os.OpenFile(fileName, os.O_WRONLY, 0666)
if err != nil {
    log.Fatal(err)
}
defer file.Close()
byteSlice := []byte("Bytes!\n")
bytesWritten, err := file.Write(byteSlice)
if err != nil {
    log.Fatal(err)
}
```

Pisanje II

- ▶ Drugi način oslanja se na paket `io/ioutil` i njegovu metodu `WriteFile()` koja je pogodna za brzo upisivanje sadržaja u fajl kada on nije preveliki
- ▶ Prvi parametar funkcije je naziv fajla, nakon čega se navodi niz bajtova koje treba upisati, dok je poslednji parametar rezervisan za navođenje permisija koje će biti dodeljene fajlu u slučaju da on ranije nije postojao

```
err := ioutil.WriteFile(fileName, []byte("Hi\n"), 0666)
if err != nil {
    log.Fatal(err)
}
```

Pisanje III

- ▶ Treći način zapisa koristi bafer paket koji nudi mogućnost baferisanog zapisivanja u promenljive čiji tip implementira `io.Writer` interfejs
- ▶ Bafer ima maksimalnu veličinu, predefinisana vrednost je 4096 bajtova, a može se i menjati
- ▶ Sadržaj se dodaje u bafer metodom `Write()`
- ▶ Metodom `FLush()` sadržaj se zapisuje na disk u slučaju fajla, a bafer se nakon toga prazni
- ▶ Sadržaj bafera se može obrisati pozivom metode `Reset()`
- ▶ Metode `Size()`, `Buffered()` i `Available()` pružaju informacije o ukupnoj veličini bafera, broju popunjenih bajtova i broju slobodnih bajtova

Pisanje IV

```
bufferedWriter := bufio.NewWriter(file)
bytesWritten, err := bufferedWriter.Write(
    []byte{65, 66, 67},
)

bytesWritten, err = bufferedWriter.WriteString(
    "Buffered string\n",
)

bufferedWriter.Flush()
bufferedWriter.Reset(bufferedWriter)

bufferedWriter = bufio.NewWriterSize(
    file, 8000)
```

Čitanje I

- ▶ Pokazivač na fajl (i svaki drugi tip koji implementira interfejs `io.Reader`) poseduje metodu `Read()`
- ▶ Sadržaj se upisuje u niz bajtova koji se prosleđuje kao argument
- ▶ Poziv funkcije vraća grešku kada se na početku čitanja nalazimo na kraju fajla

```
bytes := make([]byte, 2)
_, err = file.Read(bytes)
if err != nil {
    panic(err)
}
```

Čitanje II

- ▶ Funkcija `io.ReadFull()` učitava sadržaj fajla koji se navodi kao prvi argument u niz koji se navodi kao drugi argument
- ▶ Ukoliko je sadržaj koji se čita kraći od zadatog niza, poziv funkcije će vratiti grešku

```
bytes := make([]byte, 2)
_, err = io.ReadFull(file, bytes)
if err != nil {
    panic(err)
}
```

- ▶ Funkcija `io.ReadAtLeast()` učitava najmanje `n` bajtova gde je `n` zadato i vraća grešku ukoliko ne može da pronađe barem toliko bajtova

```
bytes := make([]byte, 2)
_, err = io.ReadAtLeast(file, bytes, 1)
if err != nil {
    panic(err)
}
```


Čitanje III

- ▶ Svi prethodni načini učitavaju sadržaj u niz čija je dužina zadata pre samog čitanja
- ▶ Funkcija `ioutil.ReadAll()` učitava ceo fajl i kao povratnu vrednost vraća niz popunjen sadržajem

```
bytes, err := ioutil.ReadAll(file)
if err != nil {
    panic(err)
}
```

- ▶ Još jedan način za učitavanje sadržaja celog fajla je uz pomoću `ioutil.ReadFile()` funkcije
- ▶ Razlika je ta da se u ovom slučaju navodi putanja fajla, a ne pokazivač na fajl

```
bytes, err := ioutil.ReadFile("folder/somefile")
if err != nil {
    panic(err)
}
```

Čitanje IV

- ▶ U situacijama zahtevnijim za obradu pogodno je koristiti bufio paket kao i pri pisanju u fajl
- ▶ Metoda Peek() vraća narednih n bajtova u odnosu na trenutnu poziciju, tako da trenutna pozicija nakon poziva ostaje ista

```
bufferedReader := bufio.NewReader(file)
byteSlice := make([]byte, 5)
byteSlice, err = bufferedReader.Peek(5)
```

- ▶ Metoda Read() takođe vraća narednih n bajtova, ali tako da se nakon poziva trenutna pozicija menja

```
numBytesRead, err := bufferedReader.Read(byteSlice)
```

Čitanje V

- ▶ Pozivom metode `ReadByte()` čita se naredni bajt, ukoliko postoji, ili se vraća greška ukoliko ne postoji
`myByte, err := bufferedReader.ReadByte()`
- ▶ `ReadBytes()` i `ReadString()` metode kao parametar uzimaju vrednost delimitera i vraćaju sadržaj fajla do zadatog delimitera (zajedno sa njim)
`dataBytes, err := bufferedReader.ReadBytes('\n')`
`dataString, err := bufferedReader.ReadString('\n')`

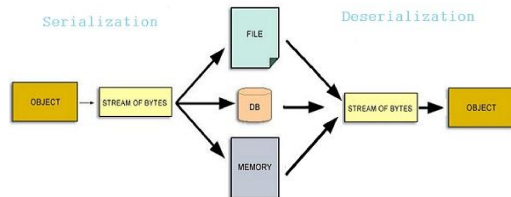
Čitanje VI

- ▶ Za obradu tekstualnog sadržaja pogodno je koristiti strukturu Scanner iz bufio paketa koja obrađuje fajl token po token
- ▶ Podrazumevano ponašanje skenera je da učitava liniju po liniju teksta pozivom metode Scan()
- ▶ Kao funkcija za podelu tokena može se registrovati bilo koja funkcija SplitFunc tipa (postojeće su ScanWords i ScanRunes)

```
scanner := bufio.NewScanner(file)
scanner.Split(bufio.ScanWords)
success := scanner.Scan()
if success == false {
    err = scanner.Err()
    if err == nil {
        log.Println("Scan completed and reached EOF")
    } else {
        log.Fatal(err)
    }
}
fmt.Println("First word found:", scanner.Text())
```

Pojam serijalizacije

- ▶ Serijalizacija predstavlja postupak prevođenja stanja objekta u niz bajtova sa ciljem njihovog trajnog skladištenja ili prenosa preko mreže
- ▶ To stanje kasnije može rekonstruisati procesom deserijalizacije, čestoi u drugačijem okruženju
- ▶ Formati za serijalizaciju su brojni i mogu se podeliti na tekstualne (JSON, XML, CSV, YAML) i binarne (BSON, MessagePack, protobuf)

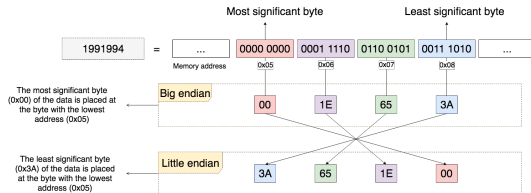


encoding/binary paket

- ▶ Paket encoding/binary implementira jednostavnu translaciju između brojeva i niza bajtova
- ▶ Translacija se vrši čitanjem i pisanjem vrednosti fiksne dužine
- ▶ Vrednost fiksne dužine ili imaju tip fiksne dužine (bool, int8, int16, uint8, float32, complex64 ...) ili predstavljaju niz ili strukturu koja sadrži samo vrednosti fiksne dužine
- ▶ Predstavlja pogodan način za kodiranje struktura podataka sa numeričkim vrednostima u njihovu binarnu reprezentaciju
- ▶ Može predstavljati i osnovu za kreiranje sopstvenih binarnih formata i protokola

ByteOrder I

- ▶ Redosled bajtova diktira na koji će način biti kodirane vrednosti dužine veće od jednog bajta
- ▶ Implementacije ByteOrder interfejsa su LittleEndian i BigEndian
- ▶ Treba voditi računa da se prilikom serijalizacije i deserijalizacije koristi ista implementacija kako bi se očuvala semantika podatka



ByteOrder II

```
b := []byte{1, 2, 3, 4}
val := binary.LittleEndian.Uint32(b)
fmt.Printf("Little endian: %b\n", val)
//output: Little endian: 0000010000000001100000001000000001
val2 := binary.BigEndian.Uint32(b)
fmt.Printf("Big endian: %b\n", val2)
//output: Big endian: 00000001000000010000000011000000100
binary.LittleEndian.PutUint32(b, val2)
fmt.Println(b)
//output: [4 3 2 1]
```


Serijalizacija struktura

- Paket poseduje funkcije Read i Write kojima se iz io.Reader-a čita ili u io.Writer piše određeni podatak, uz navođenje odgovarajućeg ByteOrder-a

```
vertex := Vertex{3, 2}
err = binary.Write(file, binary.LittleEndian, vertex)
if err != nil {
    panic(err.Error())
}
file.Seek(0, 0)
newv := &Vertex{}
err = binary.Read(file, binary.LittleEndian, newv)
if err != nil {
    panic(err.Error())
}
fmt.Println(*newv)
//output: {3 2}
```

Zadaci I

- ▶ Napisati funkciju koja za fajl sa zadatim nazivom ispisuje njegovu veličinu kada je poslednji put izmenjen. Ukoliko fajl ne postoji, treba ga kreirati u okviru funkcije.
- ▶ Napisati funkciju koja:
 - ▶ Kreira fajl na zadatoj putanji
 - ▶ U njega upisuje tekst "Hello from the file!"
 - ▶ Kopira njegov sadržaj u novi fajl sa nazivom "copy.txt"
 - ▶ Briše originalni fajl
- ▶ Napisati funkciju koja čita i na konzoli ispisuje prve tri reči iz "copy.txt" fajla.
- ▶ Napisati funkciju koja iz "copy.txt" fajla čita vrednost m-tog i n-tog bajta (parametri funkcije) i njihov zbir ispisuje na konzoli.
- ▶ Napisati funkciju koja se pozicionira na sredinu "copy.txt" fajla i naredna četiri bajta menja vrednostima [10, 21, 103, 15].
- ▶ Napisati funkciju koja svaki treći bajt u "copy.txt" fajlu menja vrednošću 0.

Zadaci II

- ▶ Kreirati strukturu Triangle sa poljima Edge1, Edge2 i Edge3 tako da je svako polje tipa float32. Upotrebom encoding/binary paketa:
 - ▶ Serijalizovati nekoliko vrednost tipa Triangle u fajl pod nazivom "triangles.bin"
 - ▶ Deserijalizovati drugi upisani trougao i ispisati vrednosti njegovih stranica na konzoli
 - ▶ Izmeniti taj trougao tako da mu sve stranice postanu duplo veće, a zatim te vrednosti izmeniti i u fajlu
 - ▶ Pomoću Read funkcije u slice triangles učitati poslednja dva trougla iz fajla
- ▶ Kreirati strukturu Sport sa poljima Name (string), IsTeam(boolean) i MatchDuration (int16). Upotrebom encoding/binary paketa:
 - ▶ Serijalizovati nekoliko vrednosti tipa Sport u fajl pod nazivom "sports.bin"
 - ▶ Pravilno deserijalizovati sve vrednosti iz fajla "sports.bin" i ispisati ih na konzoli