Rad sa fajlovima

Fajl predstavlja kolekciju podataka trajno uskladištenih na disku pod određenim nazivom. Svaki fajl sačinjen je iz niza nula i jedinica, a njihova interpretacija daje im određeno značenje.

Prilikom rada sa fajlovima treba razlikovati rad sa onima koji su tekstualni i onima koji su binarni. Mada tekstualni fajlovi u svojoj osnovi sadrže podatke u binarnom formatu, oni sa sobom donose i određena pravila interpretacije zapisanih bajtova, koja nisu primenljiva na bilo koji binarni fajl. Stoga, neispravna manipulacija sadržajem fajla može dovesti do oštećenja samih podataka (data corruption).

Dalji tekst baviće se prikazom nekih operacija koje se mogu obavljati nad fajlovima, a koje pružaju paketi os, io, io/ioutil i bufio.

• Kreiranje

Novi fajl može biti kreiran pomoću funkcije Create() paketa os koja kao jedini parametar prima naziv fajla, tako da se navodi 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 koja takođe kao jedini parametar prima naziv fajla. Jedina povratna vrednost funkcije je greška koju treba obraditi.

```
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 dobaviti upotrebom funkcije Stat() paketa os navođenjem naziva fajla kao jedinog parametra koji funkcija prima. Sem greške, funkcija vraća i vrednost tipa FileInfo preko koje se mogu dobaviti navedene informacije.

```
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("Permissions:", fileInfo.Mode())
fmt.Println("Last modified:", fileInfo.ModTime())
fmt.Println("Is Directory: ", fileInfo.IsDir())
```

Na sledeći način može se proveriti da li fajl postoji:

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

- ## Preimenovanje

Fajl može biti preimenovan (premešten izmenom putanje) pomoću funkcije Rename() paketa os koja kao prvi parametar prima staru putanju fajla, a kao drugi novu putanju. Jedina povratna vrednost funkcije je greška

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

- ## Otvaranje i zatvaranje

Funkcija os.Open() sa unetim argumentom naziva fajla vrši otvaranje fajla (ukoliko on postoji) u read-only režimu i vraća pokazivač na otvoreni fajl i grešku. Zatvaranje fajla vrši se pozivom metode Close() nad fajlom.

```
file, err := os.Open(fileName)
if err != nil {
   panic(err)
}
file.Close()
```

Kako je sadržaj fajla često potrebno menjati, funkcija Open() nije pogodna za takve situacije. Funkcija os.OpenFile() je opštija funkcija i dozvoljava veću fleksibilnost. Prvi parametar funkcije je naziv fajla, nakon čega sledi flag koji naznačava u kom režimu se fajl otvara, posle čega se navode permission bits (pogledati file system permissions).

```
file, err = os.OpenFile("somefile", os.O_RDWR, 0666)
if err != nil {
   panic(err)
}
```

Dozvoljeno je navesti više flag-ova tako da se između njih nalazi operator | (OR). Moguće vrednosti flag-a (drugog parametra funkcije) su:

- O_RDONLY - otvara se read-only
- O_WRONLY - otvara se write-only
- O_RDWR - otvara se read-write
- O_APPEND - pisanje se vrši dodavanjem novog sadržaja na kraj
- O_CREATE - kreira se novi fajl ukoliko već ne postoji
- O_EXCL - u kombinaciji sa O_CREATE, zahteva se da fajl ne postoji
- O_SYNC - otvara se za sinhroni I/O
- O_TRUNC - skratiti fajl pri otvaranju ako je moguće

Neke od mogućih vrednosti koje se navode za permission bits su:

- 0000 - nema dozvola
- 0700 - samo vlasnik može da čita, piše i izvršava
- 0770 - vlasnik i grupa mogu da čitaju pišu i izvršavaju, ostali ne
- 0111 - svi mogu da izvršavaju
- 0222 - svi mogu da pišu
- 0333 - svi mogu da pišu i izvršavaju
- 0444 - svi mogu da čitaju
- 0555 - svi mogu da čitaju i izvršavaju
- 0666 - svi mogu da čitaju i pišu

- ## 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.

```
originalFile, err := os.Open(originalPath)
if err != nil {
   log.Fatal(err)
}
defer originalFile.Close()
newFile, err := os.Create(newPath)
if err != nil {
   log.Fatal(err)
}
defer newFile.Close()
bytesWritten, err := io.Copy(newFile, originalFile)
if err != nil {
   log.Fatal(err)
}
log.Printf("Copied %d bytes.", bytesWritten)
err = newFile.Sync()
if err != nil {
   log.Fatal(err)
}
```

- ## Pozicioniranje

Pozicioniranje na tačno određeno mesto u fajlu pred naredno čitanje ili pisanje moguće je vršiti 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 za flag su:

- 0 - U odnosu na početak fajla
- 1 - U odnosu na trenutnu poziciju
- 2 - U odnosu na kraj fajla

```
file, err := os.Open(fileName)
if err != nil {
   log.Fatal(err)
}
defer file.Close()
var offset int64 = 5
var whence = 0
newPosition, err := file.Seek(offset, whence)
if err != nil {
   log.Fatal(err)
}
fmt.Println("Just moved to 5:", newPosition)
```

- **Pisanje**

Pisanje u otvoreni fajl može se vršiti na više načina. Jedan od njih je pozivom metode Write() nad fajlom koja kao jedini parametar prima slice bajtova koje treba zapisati, a 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)
}
log.Printf("Wrote %d bytes.\n", bytesWritten)
```

Naredni način pisanja 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 koji funkcija prima je naziv fajla, nakon čega se navodi slice bajtova koje treba upisati, dok je poslednji parametar rezervisan za navođenje permisija koje će biti dodeljene fajlu u slučaju da on ranije nije postojao (funkcija vrši i kreiranje fajla ukoliko je to potrebno).

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

Paket bufio poseduje strukturu Writer koja nudi mogućnost baferisanog pisanja za io.Writer objekte (fajl implementira io.Writer interefejs). Writer struktura odlikuje se veličinom bafera (4096B by default) u koji se zapisuje sadržaj koji će biti upisan u fajl. Veličina bafera može se i zadati. Pozivom metode Write() vrši se upis u bafer, dok se metodom Flush() sadržaj iz bafera upisuje na disk čime se bafer prazni. Metode Size(), Buffered() i Available() pružaju informacije o ukupnoj veličini bafera, broju popunjenih bajtova i broju slobodnih bajtova. Ukoliko se bafer popuni, potrebno ga je isprazniti pre daljeg upisa ili pozivom Flush() metode ili resetovanjem sadržaja bafera što se može izvesti uz pomoć metode Reset().

```
file, err := os.OpenFile(fileName, os.O_WRONLY, 0666)
if err != nil {
   log.Fatal(err)
}
defer file.Close()

bufferedWriter := bufio.NewWriter(file)
bytesWritten, err := bufferedWriter.Write(
   []byte{65, 66, 67},
)
if err != nil {
   log.Fatal(err)
}
log.Printf("Bytes written: %d\n", bytesWritten)
bytesWritten, err = bufferedWriter.WriteString(
   "Buffered string\n",
)
if err != nil {
   log.Fatal(err)
}
log.Printf("Bytes written: %d\n", bytesWritten)

unflushedBufferSize := bufferedWriter.Buffered()
log.Printf("Bytes buffered: %d\n", unflushedBufferSize)
bytesAvailable := bufferedWriter.Available()
if err != nil {
   log.Fatal(err)
}
log.Printf("Available buffer: %d\n", bytesAvailable)
bufferedWriter.Flush()
bufferedWriter.Reset(bufferedWriter)
bytesAvailable = bufferedWriter.Available()
if err != nil {
   log.Fatal(err)
}
log.Printf("Available buffer: %d\n", bytesAvailable)

bufferedWriter = bufio.NewWriterSize(
   file,
   8000
)
bytesAvailable = bufferedWriter.Available()
if err != nil {
   log.Fatal(err)
}
log.Printf("Available buffer: %d\n", bytesAvailable)
```

- ## Čitanje

Kako fajl implementira interfejs io.Reader, moguće je nad njim pozvati metodu Read() koja kao parametar prima slice bajtova u koje će upisati sadržaj fajla. Ukoliko je sadržaj fajla veći od dužine slice-a, upisaće se prvih n bajtova (popuniti ceo slice), dok će u situaciji kada je slice duži od sadržaja fajla biti popunjeni samo prvih m elemenata kolika je i dužina sadržaja fajla. U slučaju kada Reader ne naiđe na sadržaj koji može da pročita, metoda će vratiti grešku.

```
file, err := os.Open("folder/somefile")
if err != nil {
   panic(err)
}
bytes := make([]byte, 2)
_, err = file.Read(bytes)
if err != nil {
   panic(err)
}
```

Funkcija io.ReadFull() vrši čitanje fajla koji se navodi kao prvi argument u slice koji se navodi kao drugi argument. Ukoliko je sadržaj koji se čita kraći od zadatog slice-a, funkcija će vratiti grešku.

```
file, err := os.Open("folder/somefile")
if err != nil {
   panic(err)
}
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. Podaci se učitavaju u zadati slice u popunjavaju se do dužine slice-a.

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

Svi prethodni načini učitavaju sadržaj u slice čija je dužina zadata pre samog čitanja. Funkcija ioutil.ReadAll() učitava sadržaj celog fajla koji je zadat i kao povratnu vrednost vraća slice u kom se nalaze podaci iz fajla.

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

Još jedan način za učitavanje sadržaja celog fajla je uz pomoć 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)
}
fmt.Println(bytes)
```

U situacijama zahtevnijim za obradu pogodno je koristiti bufio paket kao i pri pisanju u fajl. Reader struktura omogućava manipulaciju sadržajem io.Reader objekata (fajl implementira io.Reader interfejs). Metoda Peek() vraća narednih n bajtova u odnosu na trenutnu poziciju, tako da trenutna pozicija nakon poziva ostaje ista, dok metoda Read() takođe vraća narednih n bajtova, ali tako da se nakon poziva trenutna pozicija menja. Pozivom metode ReadByte() čita se naredni bajt, ukoliko postoji, ili se vraća greška ukoliko ne postoji. ReadBytes() i ReadString() metode kao parametar uzimaju vrednost delimitera i vraćaju sadržaj fajla do zadatog delimitera (zajedno sa njim) u obliku slica-a bajtova ili u obliku stringa.

```
file, err := os.Open(fileName)
if err != nil {
   log.Fatal(err)
}
defer file.Close()
bufferedReader := bufio.NewReader(file)
byteSlice := make([]byte, 5)
byteSlice, err = bufferedReader.Peek(5)
if err != nil {
   log.Fatal(err)
}
fmt.Printf("Peeked at 5 bytes: %s\n", byteSlice)

numBytesRead, err := bufferedReader.Read(byteSlice)
if err != nil {
   log.Fatal(err)
}
fmt.Printf("Read %d bytes: %s\n", numBytesRead, byteSlice)

myByte, err := bufferedReader.ReadByte()
if err != nil {
   log.Fatal(err)
}
fmt.Printf("Read 1 byte: %c\n", myByte)

dataBytes, err := bufferedReader.ReadBytes('\n')
if err != nil {
   log.Fatal(err)
}
fmt.Printf("Read bytes: %s\n", dataBytes)

dataString, err := bufferedReader.ReadString('\n')
if err != nil {
   log.Fatal(err)
}
fmt.Printf("Read string: %s\n", dataString)
```

Za obradu tekstualnog sadržaja pogodno je koristiti strukturu Scanner iz bufio paketa koja obrađuje fajl token po token. Podrazumevano ponašanje skenera je takvo da on učitava liniju po liniju teksta pozivom metode Scan(). Kraj tokena ne mora predstavljati novi red. Kao funkcija za split tokena može se registrovati bufio.ScanWords ili bufio.ScanRunes funkcija, ili bilo koja sopstvena funkcija koja je SplitFunc tipa.

```
file, err := os.Open(fileName)
if err != nil {
   log.Fatal(err)
}
defer file.Close()
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())
```