

OS Vezbe

1. cas

Podsecanje iz P1 i P2, i ceste greske na ispitima.

- Neinicijalizovani pokazivac
- Stringovi bez terminirajuce nule kada se rucno radi sa stringovima
 - Rucno postavljanje nule na kraj stringa je obavezno da funkcije za ispis ne bi citale smece iz memorije
 - Svaka funkcija iz `string.h` automatski stavlja terminirajucu nulu na kraj stringa
- Definisanje makroa za proveru greska koji ce biti cesto koriscenj tokom kursa.

```
#define check_error(cond, msg)\
    do{\
        if(!(cond)){\
            perror(msg);\
            exit(1);\
        }\
    } while(0)
```

- Ovaj makro se koristi tako sto se kao cond stavi uslov koji ne sme da bude true, na primer kod rada sa File Descriptorima kada proveravamo `check_error(fd != -1)`, u slucaju da je fd -1 mi izbacujemo gresku. Takodje je jako bitno da koristimo `errno.h` zaglavlje koje nam omogucava detaljniji uvid u to sta je tacno greska.

TLPI poglavlja

3.4 Handling errors from Sys Calls

2. cas

Dobijanje inforamcija o korisniku, grupi i fajlu

Korisnici

- Na linux sistemima sve inforamcije o korisniku se cuvaju u `/etc/passwd` fajlu. Dok se informacije o grupi cuvaju u `/etc/group`

- Za informacije o korisniku moram da `#include <pwd.h>` zaglavlje koje omogućava citanje `/etc/passwd` fajla, zbog sigurnosnih razloga korisnicke lozinke odnosno hashevi isti su smesteni u `/etc/shadow` koji zahteva specijalna prava pristupa da bi se otvorio i citao.
- Da bi dobili informacije o korisniku moramo da koristimo staticku strukturu `struct passwd *userInfo`, `userInfo` je samo naziv strukture u ovom slucaju dok nju popunjavamo sistemskim pozivom `getpwnam()` koja kao argument prima niz karaktera koji je korisnicko ime u plain textu, postoji i druga funkcija koja vraca istu strukturu samo prima korisnicki id `uid_t` (User ID) i ona je `getpwuid()`.
- Za citanje celog `/etc/passwd` fajla koristimo funkcije `setpwent()` i `endpwent()` koje otvaraju i citaju i nakon toga zatvaraju `/etc/passwd` file.
- Trenutnog korisnika dobijamo tako sto pozovemo `getpwent()` i fajl citamo do kraj dok funkcija ne vrati `null`.

```
setpwent();
struct passwd *user = NULL;

while((user = getpwent()) != NULL){

    print_users(user);
}
endpwent();
```

- Primer kako bi kod koji cita ceo fajl trebao da izgleda.

```
void print_users(struct passwd* userInfo){

    fprintf(stdout, "\n");
    fprintf(stdout, "Username: %s\n", userInfo->pw_name);
    fprintf(stdout, "UID: %d\n", userInfo->pw_uid);
    fprintf(stdout, "GID: %d\n", userInfo->pw_gid);
    fprintf(stdout, "Home dir: %s\n", userInfo->pw_dir);
    fprintf(stdout, "Shell: %s\n", userInfo->pw_shell);
}
```

- Funkcija koja ispisuje korisnike u formatu koji je citljivi korisniku

Grupe

- Grupe se obradjuju na isti nacin kao i korisnici samo sto se za grupe koristi zaglavlje `grp.h`
- Informacije o grupi takodje mozemo dobijati na osnovu imena grupe i njenog ID-a koriscenjem funkcija `getgrnam()` i `getgrgid()` koje kao parametre primaju niz karaktera

(ime grupe), odnosno `gid_t` sistemski tip koji je Group ID

- Podaci o grupi se cuvaju takodje u staticki alociranoj strukturi `struct group* groupInfo` gde je `group` tip structure a `groupInfo` samo njen naziv.
- Ona se popunjava gore navedenim funkcijama.
- Kao kod `pwd.h` zaglavlja i `grp.h` omogućava citanje celog `/etc/group` fajla sa funkcijama `setgrent()`, `getgrent()` i `endgrent()`

```
setgrent();
struct group *currentGroup = NULL;

while((user = getgrent()) != NULL){

    print_group(currentGroup);
}
endgrent();
```

- Ovaj kod prolazi kroz ceo `/etc/group` fajl i ispisuje njegov sadrzaj onako kako mi odlucimo da ga formatiramo u funkciji `print_group()`

```
void print_groups(struct group *grinfo){

    fprintf(stdout, "\n");
    fprintf(stdout, "%s\n", grinfo->gr_name);
    fprintf(stdout, "%d\n", grinfo->gr_gid);

    for(int i = 0; grinfo->gr_mem[i] != NULL; i++){
        fprintf(stdout, "%s\n", grinfo->gr_mem[i]);
    }

}
```

- Ova funkcija za dati argument `*grinfo` ispisuje ime grupe, njen id i sve clanove koji su u toj grupi

Fajlovi

- Na linux sistemima postoji 7 vrsta fajlova;
 1. regularni fajlovi - `r`
 2. direktorijumi `d`
 3. blok fajlovi, oni su obicno hardverske komponente i nalaze su `/dev` direktorijumu `b`
 4. char fajlovi, ovo su ulazni i izlazni streamovi, terminal je jedan od char fajlova `c`

5. pipe-ovi, ovo su fajlovi koji sluze za redirekciju izlaza jednog programa u ulaz drugog `p`
6. simbolicki linkovi, oni sadrže stvarne putanje do nekog fajla `l`
7. Socketi, koriste se za komunikaciju izmedju aplikacija `s`

- Sve informacije o fajlu mozemo dobiti pomocu sistemskog poziva `stat()` , koji kao argumente prima putanju do fajla i pokazivac na strukturu `struct stat fileinfo` gde je `fileinfo` samo naziv.
- Iz `stat` funkcije mozemo da dobijemo informacije kao sto su prava pristupa u formatu `rw-rw-rw-` gde je `r` - read, `w` - write i `x` - executable. i redosled kojim su rasporedjeni oznacava prava korisnika, grupe i ostalih na sistemu
- Pre ovih 9 karaktera ide jedan karakter koji oznacava tip fajla (jedan od onih 7 sa pocetka).
- Struktura `stat` takodje sadrzi i informacije o velicini fajla, vremenu nastanka fajla, vremenu pristupa i vremenu modifikacije, `uid_t` korisnika koji je napravio fajl kao i `gid_t` kojoj taj korisnik pripada.

TLPI poglavlja i materijali

8.1 - 8.4, 15.1, 15.4-15.4.3
man 5 passwd i man 5 group

3. cas

Sistemi pozivi za rad sa fajlovima

Mkdir

- Poziv `mkdir` kreira direktorijum, kao argumente prima putanju na koji fajl treba biti kreiran i mod u kojem treba biti kreiran, odnosno `mode_t` promenljivu koja u sebi sadrzi prava pristupa u hexadecimalnom zapisu, na linuxu se prava pristupa oznacavaju trocifrenim hexadecimalnim brojem.

User	Group	Other
rwX	rwX	rwX
111	101	101

- Ovo je ekvivalent kao da smo napsali 0755 za prava pristupa, ovo takodje znaci da korisnik moze da cita, pise i izvrsava dok svi ostali mogu samo da citaju i izvrsavaju

Unlink i rmdir

- Brisanje fajlova se vrši pozivom `unlink` koji uklanja fajl sa date putanje. Dok se za brisanje direktorijuma koristi `rmdir`, da bi `rmdir` uspešno radio direktorijum koji brismo mora da bude prazan, kod neke implementacije mogli bi smo rekurzivno da prodjemo kroz zadati direktorijum pobrisemo sve fajlove sa `unlink` ili `rmdir` ako naidjemo na neki novi direktorijum i zatim obrisemo nas zadati direktorijum.

Otvaranje fajlova

- Svaki otvoren fajl na sistemu ima svoj fajl deskriptor, on nam govori koji je fajl otvoren, sa kojim pravima pristupa i sa kojim flagovima smo otvorili taj fajl odnosno sta mozemo sa njim da radimo
- Za otvaranje fajlova koristimo funkciju `open` kojoj saljemo putanju do naseg fajla, flagove koji odredjuju kako se kreira fajl i prava pristupa.

"Pasted image 20231030152556.png" is not created yet. Click to create.

- Ovo su flagovi sa kojima otvaramo fajlove.
- Nakon sto smo pozvali funkciju `open` ona vraća ceo broj `int` koji je nas file deskriptor koji nam omogućava da kasnije pisemo ili citamo otvoreni fajl.
- Fajl deskriptor ne sme da bude `-1`, u tom slucaju se dogodila greska pri otvaranju fajla.
- Na linux sistemima deskriptori pocinju od broja 3 posto su:
 1. `0` - `STDIN` - `0` je standardni ulaz
 2. `1` - `STDOUT` - `1` je standardni izlaz
 3. `2` - `STDERR` - `2` je standardni izlaz za greske

Read i write

`read` poziv pokušava da cita podatke iz datog file deskriptora `read(int fd, void buf[.count], size_t count);`, `fd` je file deskriptor, `buf` je memorija u koju upisujemo procitanih `count` bajtova fajla, ako `read` vrati `-1` znamo da citanje nije bilo uspesno.

```
{
    int fd = open(argv[1], O_RDONLY);
    check_error(fd != -1, "open failed");

    char buf[BUFFER_SIZE];
    int readBytes = 0;

    while(readBytes = read(fd, buf, BUFFER_SIZE)){
        check_error(write(1, buf, readBytes) != -1, "write failed");
    }

    check_error(readBytes != -1, "read failed");
}
```

```
    close(fd);  
}
```

- U ovom kodu otvaramo fajl sa `O_RDONLY` flagom koji nam omogućava citanje fajla zatim alociramo niz karaktera `char buf[]` u koji cemo da smesatamo procitane podatke.
- Funkcija `read` vraća broj procitanih bajtova fajla i sve dok broj procitanih bajtova nije 0 nastavljamo da citamo, u slučaju da je `readBytes = -1` to znači da je došlo do greske.
- Ova funkcija čita dati fajl i ispisuje ga na standardni izlaz, slično kao program `cat` u linuxu.