

# Pomoc za kolokvijum

## Prag i jos po nesto

Prag za kolokvijum je 40% odnosno dva u potpunosti tacna zadatka (obrada gresaka, alokacija, dealokacija, otvaranje/zatvaranje fajlova, format ispisa, exit code itd.). Da bi se prag uspesno ostvario potrebno je detaljno spremati sistemske pozive koji su radjeni na vezbama i koji se provlace po slajdovima.

Specijalna napomena za sistemski poziv *open()*, bitni flag-ovi su:

- *O\_CREAT* - pravice se novi fajl ako ne postoji.
- *O\_EXCL* - kombinuje se sa *O\_CREAT* i ukoliko fajl postoji doci ce do greske.
- *O\_TRUNC*, *O\_APPEND* - *O\_TRUNC* ce obrisati sadrzaj fajla, a *O\_APPEND* ce usloviti da se sav sadrzaj koji se dopisuje dodaje na kraj fajla (bez obzira na poziciju *file pointer-a*).
- *O\_RDONLY*, *O\_WRONLY*, *O\_RDWR* - opisuju za koju operaciju se fajl otvara.

Specijalna napomena za kreiranje fajla sa odredjenim privilegijama:

- Da biste uspesno kreirali fajl sa nekim odredjenim privilegijama, npr. 0733 potrebno je da obezbedite da vrednost trenutne maske (*umask*) to omogucava ili da rucno postavite privilegije nakon *open()* sistemskog poziva:

1. Prvi nacin:

```
int oldMask = umask(0); // stavljamo da umask bude 0
int fd = open("1.txt", O_CREAT, 0733); // pravimo fajl
if (-1 == fd) {
    perror("open failed");
    exit(EXIT_FAILURE);
}
umask(oldMask); // vracamo staru vrednost maske
```

2. Drugi nacin:

```
int fd = open("1.txt", O_CREAT, 0733); // pravimo fajl
if (-1 == fd) {
    perror("open failed");
    exit(EXIT_FAILURE);
}
if (-1 == chmod("1.txt", 0733)) { // osiguravamo se
    perror("chmod failed");        // da ima dobra prava
    exit(EXIT_FAILURE);            // pristupa
}
```

## Na kolokvijumu

1. Pazljivo procitajte sekciju pre samih zadataka.
2. Pazljivo citajte tekst zadatka, ukoliko nesto u zadatku nije eksplicitno zabranjeno – dozvoljeno je. Posebno obratite paznju na rad sa memorijom. Ukoliko se ne zna dimenzija ulaza **morate koristiti dinamicku alokaciju**.
3. Kada ste ubedjeni da vam programi rade korektno proverite da li imate curenje memorije i fajlove koji nisu zatvoreni. Te stvari su najgluplje izgubljeni poeni.
4. Format zadataka je takav da su test primeri dati kroz ulaz u program, izlaz iz programa, kod kojim program treba da završi izvršavanje (exit code) i poneke dodatne informacije. Budite sigurni da ste ispostovali sve navdene uslove (**exit code ulazi u proveru ispravnosti zadatka i jednako je vazan kao i proizvedeni rezultat**).

## Obrada gresaka

Greske se moraju obradjivati nakon svakog poziva funkcije ili sistemskog poziva. Minimalna obrada gresaka za funkcije koje vraćaju -1 u slučaju greske je:

```
if (-1 == sistemski_poziv(arg0, arg1, arg2)) {  
    perror("neka smisljena poruka o gresci");  
    exit(EXIT_FAILURE);  
}
```

Za funkcije koje vraćaju pokazivac, odnosno NULL u slučaju greske:

```
if (NULL == poziv_funkcije(arg0, arg1, arg2)) {  
    perror("neka smisljena poruka o gresci");  
    exit(EXIT_FAILURE);  
}
```

Greske se ne moraju proveravati za sistemske pozive *close()* i *umask()*.

## Exit code-ovi i ispis greske

*Exit code* je broj sa kojim program završava izvršavanje. Najcesca 2 nacina za postavljanje *exit code*-a su return iz main funkcije i poziv exit() funkcije. Dakle, ako imamo program (u daljem tekstu **test.c**):

```
#include <stdio.h>  
  
int main() {  
    printf("Zavrsicemo exit code-om 13.\n");  
    return 13;  
}
```

njegov *exit code* ce biti 13. Slicno da je umesto return 13 pozvana funkcija exit(13) efekat bi bio isti po pitanju *exit code*-a.

Kojim se *exit code*-om vas program završio mozete proveriti iz terminala. Recimo da imamo ovakvu sekvencu komandi:

```
$ gcc -o test test.c
```

```
$ ./test
```

```
$ Završicemo exit code-om 13.
```

```
$ echo $?
```

```
$ 13
```

```
$
```

Naredba napisana masnim slovima služi da se ispise *exit code* poslednje pokretane komande (programa).

Sto se ispisa gresaka tice, **ispis gresaka uvek ide na standardni izlaz za greske**. Dakle, pitanja vezana za to da li je OK ispisati gresku sa `printf("%s\n", strerror(errno))` – **nije OK!** Medjutim, `fprintf(stderr, "%s\n", strerror(errno))` je vec u redu, **mada bih preporucio *perror()* jer podrazumevano ispisuje sistemsku poruku o gresci + vasu poruku i sve to na standardni izlaz za greske**.

## Koriscenje *man* strana i literature

Na kolokvijumu cete imati dostupnu knjigu APUE i man strane. U knjizi ima sve sto smo radili ukljucujuci i dosta nekih drugacijih primera. **Nemojte se oslanjati na to da kolokvijum moze da se uradi tako sto ce se prepisati odgovarajuci delovi iz knjige. Nemate vremena da trazite nesto sto ne znate gde je.**

Man strane su jako korisne, pa sledi par napomena:

- kada ne mozete da se setite tacnog imena funkcije otvorite man stranu neke srodne funkcije i scroll-ujte do sekcije SEE ALSO (na primer treba vam *strstr()*, ali ste zaboravili kako se tacno zove, a znate *strchr()*, otvorite man za *strchr()* i u SEE ALSO sekciji cete videti *strstr()*).
- kada ne mozete da se setite nekog flag-a, znate sta radi, ali ne i kako se tacno zove mozete pretraziti man stranu po nekoj kljucnoj reci. Na primer za *open()* i *O\_EXCL* flag, pretrazite man stranu za *open()* po stringu *exist*. Pretraga ide tako sto stisnete kosu crtu (*slash*), ukucate string koji trazite i zatim pritisnete enter. Za naredno pojavljivanje istog stringa pritisnete slovo *n*. Naravno, pretragu vrsite u okviru otvorene man strane.
- kada niste sigurni kako se koristi neka funkcija, desava se da postoji primer upotrebe u okviru man strane za tu funkciju (jedan primer takve funkcije je *nftw()*)

## Greske u kompilaciji i izvorsavanju programa

- Ako dobijete gresku ili upozorenje da neka funkcija, konstanta ili makro nije definisana:
  1. ukljucite zaglavlja iz man strane te funkcije odnosno tog sistemskog poziva
  2. ako to ne pomogne, definisite makro: `#define _XOPEN_SOURCE 700` na vrhu

*source* fajla iznad svih *#include* direktiva.

- Ako dobijete SEGFAULT:
  1. prevedite kod sa opcijom *-g*
  2. pokrenite debager sa *gdb ./zad*
  3. pokrenite izvršavanje programa sa *run arg1 arg2* (*arg1* i *arg2* su neki argumenti komandne linije koje vas program očekuje, ako ne očekuje nikakve argumente dovoljno je samo *run*)
  4. *gdb* će vam izbaciti poruku da je doslo do SEGFAULT-a, vi treba da ukucate komandu *bt* i da stisnete enter, a *gdb* će vam ispisati liniju koda gde je doslo do greske
- Dodatno kada ste pokrenuli debager, *breakpoint* se postavlja pre *run* komande sa *b ime\_fajla.c:broj\_linije*, na naredni korak u izvršavanju se prelazi sa *n*, izvršavanje programa se nastavlja sa *c*, vrednost promenljive se stampa sa *p ime\_promenljive*.

Ovaj dokument je radna verzija i bice azuriran u skladu sa vasim nedoumicama, kako bi svi imali sve tehnicke informacije na jednom mestu.