Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Računarska tehnika i informatika
Kolokvijum: Prvi, april 2023.
Datum: 7. 5. 2023.

Prvi kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 90 minuta. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____% = _____/15

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Ocenjivanje unutar potpitanja je po sistemu "sve ili ništa", odnosno nema parcijalnih poena. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**.

1. (10 poena)

U nekom sistemu sa straničnom organizacijom memorije virtuelni i fizički adresni prostor su iste veličine, po 16 MB, adresibilna jedinica je bajt, a veličina stranice je 4 KB. U donjoj tabeli data je mapa fizičkog adresnog prostora, s tim da prva data oblast memorije (BIOS) počinje na fizičkoj adresi 0, a svaka sledeća data oblast počinje odmah iza prethodne, na prvoj sledećoj novoj stranici. Iznad datih oblasti nalazi se prostor na raspolaganju za procese.

Operativni sistem preslikava navedene oblasti fizičke memorije u virtuelni adresni prostor svakog procesa, i to na isto mesto, na najviše adrese virtuelnog adresnog prostora procesa, ali u obrnutom redosledu od datog (BIOS oblast je na najvišim virtuelnim adresama, MMIO odmah ispod njega itd.).

a)(5) U prazna polja date tabele za svaku oblast upisati heksadecimalne vrednosti njene veličine, početne fizičke adrese i početne virtuelne adrese u prostoru svakog procesa.

Simbolički naziv	*Vrsta i sadržaj*	*Veličina*	*Veličina (hex)*	*Početna fiz. adr. (hex)*	*Početna v. adr. (hex)*
`BIOS`	ROM, BIOS	8 KB		0	
`MMIO`	Memorijski preslikan ulaz-izlaz	512 KB			
`SRAM`	Statički perzistentni RAM, konfiguacija	4 KB			
`KTXT`	DRAM, kod kernela	64 KB			
`KDATA`	DRAM, podaci kernela	436 KB			

b)(5) Implementirati funkciju `mapKernelVMem` koju jezgro poziva prilikom inicijalizacije memorijskog konteksta novog procesa i koja treba da inicijalizuje deo PMT procesa koji se odnosi na opisane oblasti. Na raspolaganju je funkcija `initPMTEntry` koja inicijalizuje jedan ulaz u datom PMT za datu stranicu, tako da se ta stranica preslikava u dati okvir. Prava pristupa definišu se poslednjim parametrom na način kao i u POSIX-u, pri čemu su definisane simboličke konstante `PROT_READ`, `PROT_WRITE`, `PROT_EXEC` sa istim značenjema kao i tamo, ali i `PROT_USR` koja dozvoljava definisan pristup stranici i u korisničkom režimu. Tip `size_t` je celobrojni tip koji može da prihvati vrednosti virtuelne ili fizičke adrese.

```
void mapKernelVMem (PMT* pmt);
initPMTEntry(PMT*, size_t page, size_t frame, int prot);
```

Rešenje:

2. (10 poena)

Implementirati funkciju `handleProtectionFault` koju jezgro operativnog sistema poziva pri obradi hardverskog izuzetka generisanog zbog bilo kakvog prestupa u pravima na zahtevanu operaciju sa memorijom. Prvi parametar ove funkcije ukazuje na deskriptor tekućeg procesa, drugi sadrži broj stranice koja je adresirana, a treći parametar koduje zahtevanu operaciju u bitima koji su maskirani odgovarajućim simboličkim konstantama kao u POSIX-u. Ova funkcija treba da vrati kod greške dat sledećim simboličkim konstantama u sledećim slučajevima:

- `ERR_ILLEGAL_ADDRESS` u slučaju da adresirana stranica ne pripada nijednom logičkom segmentu virtuelnog adresnog prostora datog procesa;
- `ERR_ILLEGAL_OP` u slučaju da je proces izvršio operaciju nedozvoljenu za tu stranicu;
- `ERR_ILLEGAL_USR_OP` u slučaju da je proces izvršio (inače dozvoljenu) operaciju za stranicu koja nije dostupna u korisničkom režimu rada procora;
- u slučaju da se radi o okidaču za kopiranje pri upisu (*copy on write*) treba da pozove funkciju `copyOnWrite` koja će uraditi ovo kopiranje i da vrati ono što i ova f-ja vrati.

Na raspolaganju su sledeće funkcije:

- `SegDsc* getSegDesc(Process*, size_t page)`: za datu stranicu datog procesa vraća deskriptor logičkog segmenta, odnosno *null* ako stranica ne pripada nijednom;
- `int getSegProt(SegDsc*)`: vraća bite prava pristupa u datom deskriptoru logičkog segmenta, kodovane u bitima kojima se pristupa kao u POSIX-u, simboličkim konstantama `PROT_READ`, `PROT_WRITE`, `PROT_EXEC` sa istim značenjema kao i tamo, ali i `PROT_USR` koja označava dozvoljen definisan pristup stranici i u korisničkom režimu;
- `PMT* getPMT(Process*)`: vraća PMT za dati proces; uvek uspeva;
- `PageDsc* getPageDesc(PMT*, size_t page)`: vraća deskriptor date stranice u datom PMT; uvek uspeva za validne argumente;
- `int getPageProt(PageDsc*)`: vraća bite prava pristupa u datom deskriptoru stranice u PMT, kodovane u bitima na isti način kao i za funkciju `getSegProt`;
- `int copyOnWrite(PageDsc*)`: obavlja kopiranje date stranice; vraća kod greške u slučaju neuspeha.

```
int handleProtectionFault (Process* proc, size_t page, int op);
```

Rešenje:

3. (10 poena)

U nekom sistemu postoje sledeći sistemski pozivi vezani za deljene biblioteke sa dinamičkim vezivanjem (DLL):

- `int mapDLL (const char* dllName)`: po potrebi učitava DLL sa zadatim imenom i mapira ga u virtuelni adresni prostor pozivajućeg procesa; u slučaju uspeha, vraća pozitivnu celobrojnu vrednost koja predstavlja identifikator datog DLL-a u kontekstu procesa; u slučaju greške, vraća negativnu vrednost;
- `int mapDLLSymbol (int dll, const char* symbolName)`: u DLL-u koji je prethodno mapiran pronalazi simbol sa zadatim imenom i vraća njegovu (virtuelnu) adresu; u slučaju neuspeha, vraća *null*.

U DLL-u „mydll.dll“ definisane su dve funkcije, `f1` i `f2`, čiji su potpisi dati dole. Simboli za njih kodovani su za povezivanje kao `f1@int@int*@int` i `f2@double@X*`, respektivno (ime f-je – povrtani tip – tipovi parametara). Korišćenjem datih sistemskih poziva realizovati „patrljke“ (*stub*) za ove dve funkcije koje program treba da poziva da bi pristupio njihovim implementacijama u DLL-u. U slučaju greške pozvati funkciju `handleError` koja ima isti potpis i ponašanje kao bibliotečna funkcija `printf`, samo što ispis šalje na standardni izlaz za greške i potom završava pozivajući proces.

```
int f1 (int*, int);
double f2 (X*);
```

R
e
š
e
n
j