Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, maj 2023.
Datum: 7. 5. 2023.

Drugi 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 nalik sistemu Unix postoji sistemski poziv `signal` čiji je potpis dat dole.[1] Ovim pozivom proces može da preusmeri obradu signala datog prvim parametrom na rutinu (*signal handler*) datu drugim parametrom. Ukoliko uspe, ovaj sistemski poziv vraća pokazivač na rutinu koja je bila postavljena za dati signal pre ove promene, dok u slučaju neuspeha vraća `SIG_ERR`. Proces dete nasleđuje rutine za obradu signala od procesa roditelja, ali može nezavisno da ih redefiniše.

Koristeći sistemske pozive *fork, exit, signal* i *kill*, kao i funkcije *printf* i *getchar*, napisati C program koji, kada se nad njim pokrene (roditeljski) proces, pokrene jedan proces dete, a onda taj proces dete gasi slanjem signala `SIGTERM`. Proces dete izvršava praznu petlju sve dok ne stigne ovaj signal. Kada primi signal za gašenje `SIGTERM`, proces dete treba da postavi pitanje korisniku ispisom na standardni izlaz rečenice „Da li ste sigurni da želite da prekinete izvršavanje? (D/N)“. Ukoliko se na standardni ulaz procesa deteta unose znak 'D' ili 'd', proces dete treba da se ugasi, a u suprotnom da nastavi da izvršava praznu petlju i tako dalje. Da bi se izbeglo utrkivanje u slučaju da proces roditelj pošalje signal pre nego što je proces dete preusmerio rutinu za njegovu obradu, proces roditelj treba najpre da preusmeri svoju rutinu za ovaj signal, potom pokrene proces dete koje će to naslediti, a onda sebi preusmeru obradu ovog signala na staru rutinu. Sve greške koje onemogućavaju dalje funkcionisanje kako je opisano obraditi ispisom poruke o grešci i završetkom procesa.

```
typedef void (*SIGHANDLER) (int);
SIGHANDLER signal (int signal, SIGHANDLER newHandler);
```

Rešenje:

[1]Ovo je stari i prevaziđen Unix sistemski poziv. Noviji oblik je nešto složeniji i robusniji, ali principijelno sličan.

2. (10 poena)

Neki procesor pri obradi prekida, sistemskog poziva i izuzetka prelazi na sistemski stek. Taj stek alociran je u delu memorije koju koristi kernel, a na vrh tog steka ukazuje poseban registar SSP procesora koji je dostupan samo u privilegovanom režimu.

Prilikom obrade ovih situacija procesor ništa ne stavlja na stek, već zatečenu, neizmenjenu vrednost registra PC kog je koristio prekinuti proces sačuva u poseban, za to namenjen registar SPC (procesor nema PSW), a vrednosti registara SP i SSP međusobno zameni (*swap*). Za pristup steku procesor tako uvek koristi SP. Prilikom povratka iz prekidne rutine instrukcijom `iret` procesor radi inverznu operaciju. Procesor je RISC, sa *load/store* arhitekturom i ima 32 registra opšte namene (R0..R31).

Kernel je višenitni, a svakom toku kontrole (procesu ili niti), uključujući i niti kernela, pridružen je poseban stek koji se koristi u sistemskom režimu. Promenu konteksta u kernelu radi data operacija `yield`. Kontekst procesora kernel čuva na steku, dok se informacija o vrhu steka čuva u polju `PCB::sp` čiji je pomeraj u odnosu na početak strukture PCB u asembleru označen istoimenom simboličkom konstantom. U kodu kernela postoji statički pokazivač `oldRunning` koji ukazuje na PCB tekućeg procesa, kao i pokazivač `newRunning` koji ukazuje na PCB procesa koji je izabran za izvršavanje.

Napisati kod funkcije `initContext` koju koristi kernel kada inicijalizuje procesorski kontekst za proces sa datim PCB-om tako da ovaj proces može dobiti procesor funkcijom `yield`. Na vrh već alociranog steka procesa (prvu praznu lokaciju) koji se koristi u neprivilegovanom režimu ukazuje `usrStk`, dok na vrh već alociranog steka procesa koji se koristi u kernelu ukazuje `krnlStk`, a početna adresa programa je `startAddr`. Stek raste ka nižim adresama. Svi pokazivači su 32-bitni, a tip `uint32` predstavlja 32-bitni ceo neoznačen broj.

```
void yield () {
  asm {
  ; Save the current context
  push      spc ; save regs on the process stack
  push      ssp
  push      r0
  push      r1
  ...
  push      r31
  load      r0, oldRunning ; r0 now points to the running PCB
  store     sp, [r0+PCB::sp] ; save SP

  ; Restore the new context
  load      r0, newRunning
  load      sp, [r0+PCB::sp] ; restore SP
  pop       r31 ; restore regs
  ...
  pop       r0
  pop       ssp
  pop       spc
  }
}
```

```
void initContext (PCB* pcb, void* usrStk, void* krnlStk, void* startAddr);
```

Rešenje:

3. (10 poena)

U školskom jezgru promenu konteksta obavlja operacija `yield` data dole. Korišćenjem ove funkcije za promenu konteksta implementirati binarni semafor za međusobno isključenje (*mutex*) klasom `Mutex` sa sledećim ograničenjima:

- Operaciju `wait` ne sme da pozove nit koja je zaključala *mutex*.
- Operaciju `signal` sme da pozove samo nit koja je zaključala *mutex*.

Ukoliko neko od navedenih ograničenja nije zadovoljeno treba baciti celobrojni izuzetak `ERR_MUTEX`.

```
void yield () {
  Thread* oldt = Thread::running;
  Thread* newt = Thread::running = Scheduler::get();
  if (setjmp(oldt->context)==0) longjmp(newt->context,1);
}
```

R
e
š
e
n
j