1 Teški procesi i komunikacija

1.1 Stvaranje novih procesa

Korišćenjem sistemske komande `fork` trenutni proces se duplira ("račva"). Novi proces je identičan sa originalnim, osim u svom identifikacionom broj. Komanda ne prima parametre i vraća jedan integer. Oba procesa se nakon račvanja nastavljaju odvijati u sledećem redu koda. Jedino po čemu se razlikuju je vrednost koju je vratio `fork`, ukoliko je u pitanju originalni proces ("roditelj") tada je ta vrednost jednaka proces identifikatoru (*pid*-u) "deteta", i može se koristiti za komunikaciju sa detetom, a u novom procesu je ova vrednost jednaka nuli. Budući da je najčešće potrebno da dete i roditelj rade različite stvari, to se obično rešava kodom sledećeg oblika:

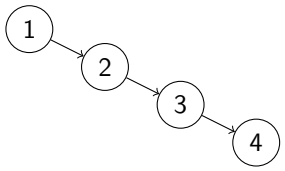
```
pid := fork();
IF pid = 0 THEN
   (* "detetove" operacije *)
ELSE
   (* "roditeljske" operacije *)
END;
```

Komanda `wait` se može pozvati u originalnom procesu sa efektom da se proces uspava dokle god neki od procesa dece ne završi sa radom. Veća kontrola nad ovim ponašanjem se može postići komandom `waitpid` u kojoj možemo precizirati tačno koji proces čekamo da se završi.

U sledećim sekcijama će biti ilustrovano nekoliko klasičnih primera organizacije procesa.

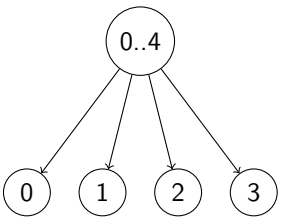
1.1.1 Lanac procesa

```
pid := 0;
i := 1;
WHILE (pid = 0) AND (i < n) DO
  pid := fork();
  IF pid # 0 THEN
    (* ... *)
  ELSE
    INC(i);
  END;
END;
```



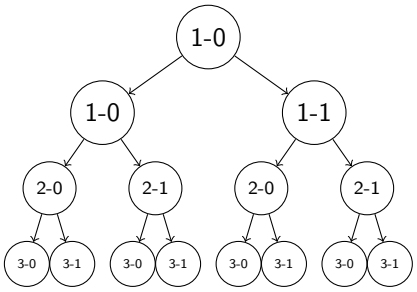
1.1.2 Roditelj sa puno dece

```
pid := -1;
j := 0;
WHILE (pid # 0) AND (j < m) DO
  pid := fork();
  IF pid # 0 THEN
    INC(j);
  ELSE
    (* ... *)
  END;
END;
```



1.1.3 Stablo procesa

```
dubina := 1;
brdece := 0;
WHILE (dubina < maxdubina) AND
      (brdece < potrebnodece) DO
  pid := fork();
  IF pid # 0 THEN
    INC(brdece);
  ELSE
    INC(dubina);
    brdece := 0;
  END;
END;
```



maxdubina=4, potrebnodece=2, čvorovi prikazuju vrednosti pri kreiranju

1.2 Pajpovi

Za komunikaciju između procesa se mogu koristiti pajpovi koje nam daje operativni sistem. Oni su jednosmerni i imaju kraj za čitanje i kraj za pisanje. U programu se po mnogim pitanjima koriste kao normalni otvoreni fajlovi, tj koristi se fajl deskriptor *(fd)* za pristup preko funkcija kao što su `read` i `write`. Prilikom čitanja, ako nema podataka u pajpu, proces će biti blokiran dok podaci ne stignu. Samim tim se ovo može koristiti za sinhronizaciju procesa. Isto tako ako se piše u pajp, ti podaci će ostati sačuvani dok ih drugi proces ne pročita.

Pajp se dobija pozivom funkcije `pipe` koja prima niz tipa `int` dužine 2. Na nulto mesto se upisuje fd za kraj za čitanje, a na mesto jedan se upise fd za pisanje. Ovo se može vizuelno zapamtiti da je kraj jedan višlje od kraja nula, te da podaci prirodno teku nadole. Funkcija vraća nulu ako je uspešno kreiran pajp.

Pajpovi postoje dokle god postoje procesi koji imaju otvoren fajl deskriptor za bar jedan kraj pajpa. Zbog toga se insistira na urednom zatvaranju krajeva. Kraj pajpa se zatvara funkcijom `close`. Na nivou operativnog sistema se ove stvari tipično rutinski oslobađaju kako se procesi završe, ali je uvek bolje da sami upravljamo resursima.

Tipičan scenario za korišćenje pajpa za komunikaciju je da se napravi pre poziva funkcije `fork`, te onda i dete i roditelj imaju pristup odgovarajućim fajl deskriptorima. Kao što je malopre objašnjeno, oba procesa će morati da zatvore oba kraja da bi se pajp zatvorio.

Primer u kome roditelj čita rezultate od deteta:

```
piper := pipe(pfd);
pid := fork();
IF pid # 0 THEN
   (* zatvorimo kraj koji nam ne treba *)
   close(pfd[1])
   roditelj := TRUE;
ELSE
   close(pfd[0]);
   roditelj := FALSE;
END;

(* u opstem slucaju se logika programa odvaja od
   formiranja procesa i komunikacija *)
IF roditelj THEN
   read(pfd[0], rez, sizeof(rez));
   close(pfd[0]);
   (* ... *)
ELSE
   (* ... *)
   write(pfd[1], rez, sizeof(rez));
   close(pfd[1]);
END;
```

U kompleksnijim scenarijima, kao što su malopre pomenuti lanci procesa, brzo može da se zakomplikuje praćenje otvorenih i zatvorenih procesa. Na primer kod lanca, ako se ne zatvaraju krajevi, svako sledeće dete će naslediti i otvorene krajeve originalnog procesa i svih procesa u međuvremenu. Preporuka za ovakve situacije je korišćenje sistemskog poziva dup, koji za prosleđeni fajl deskriptor pravi i vraća još jedan koji pokazuje na isti objekat, u ovom slučaju kraj pajpa. Onda je moguće zatvoriti originalno dobijeni fajl deskriptor i koristiti samo ovaj novi, koji je za sada na raspolaganju samo ovom procesu. No, i dalje je neophodno voditi računa da deca ovog procesa nasleđuju i ove krajeve, te da je potrebno zatvoriti i njih.

Primeri sa komentarima su dati u rešenjima zadataka sa vežbi.