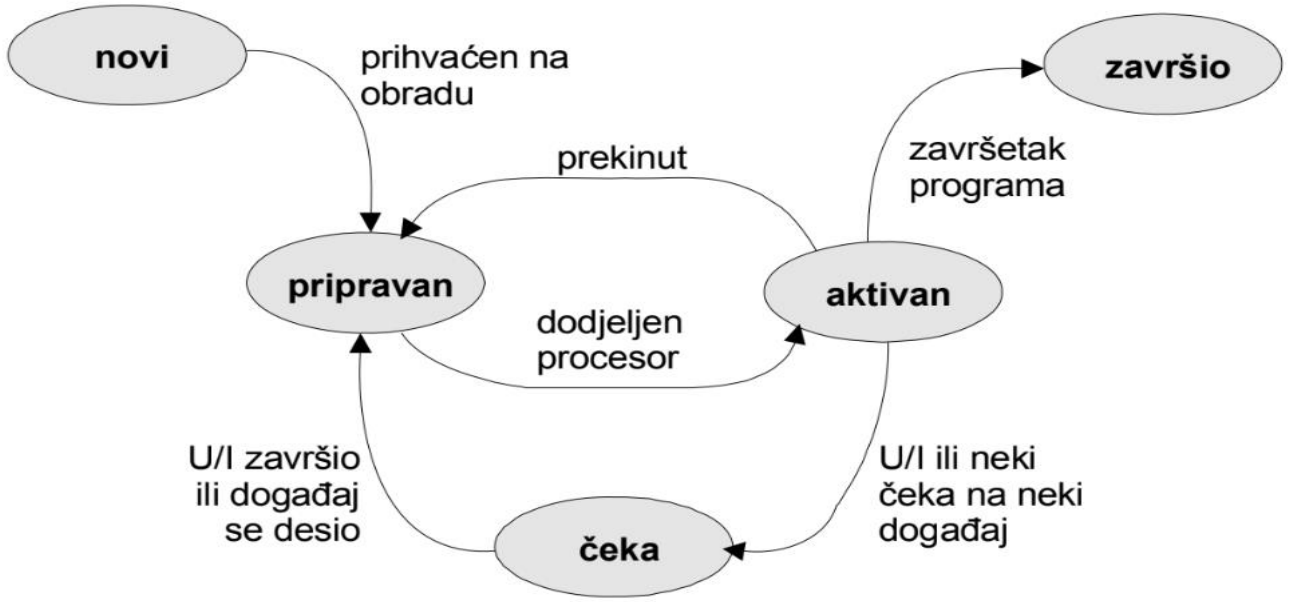
1. Objasnite kako OS vodi evidenciju o procesu ? – PCB
Operativni sustav vodi evidenciju o procesu u PCB-u (Process Control Block). To je struktura s informacijama o procesu. Svaki PCB sadrži id procesa, ime procesa, trenutno stanje procesa, image na disku, sadržaj registara, podatke o memoriji koju proces koristi, popis otvorenih datoteka, informacije o korisniku procesa, pokazivače(na parent, child procese , ako postoje)...

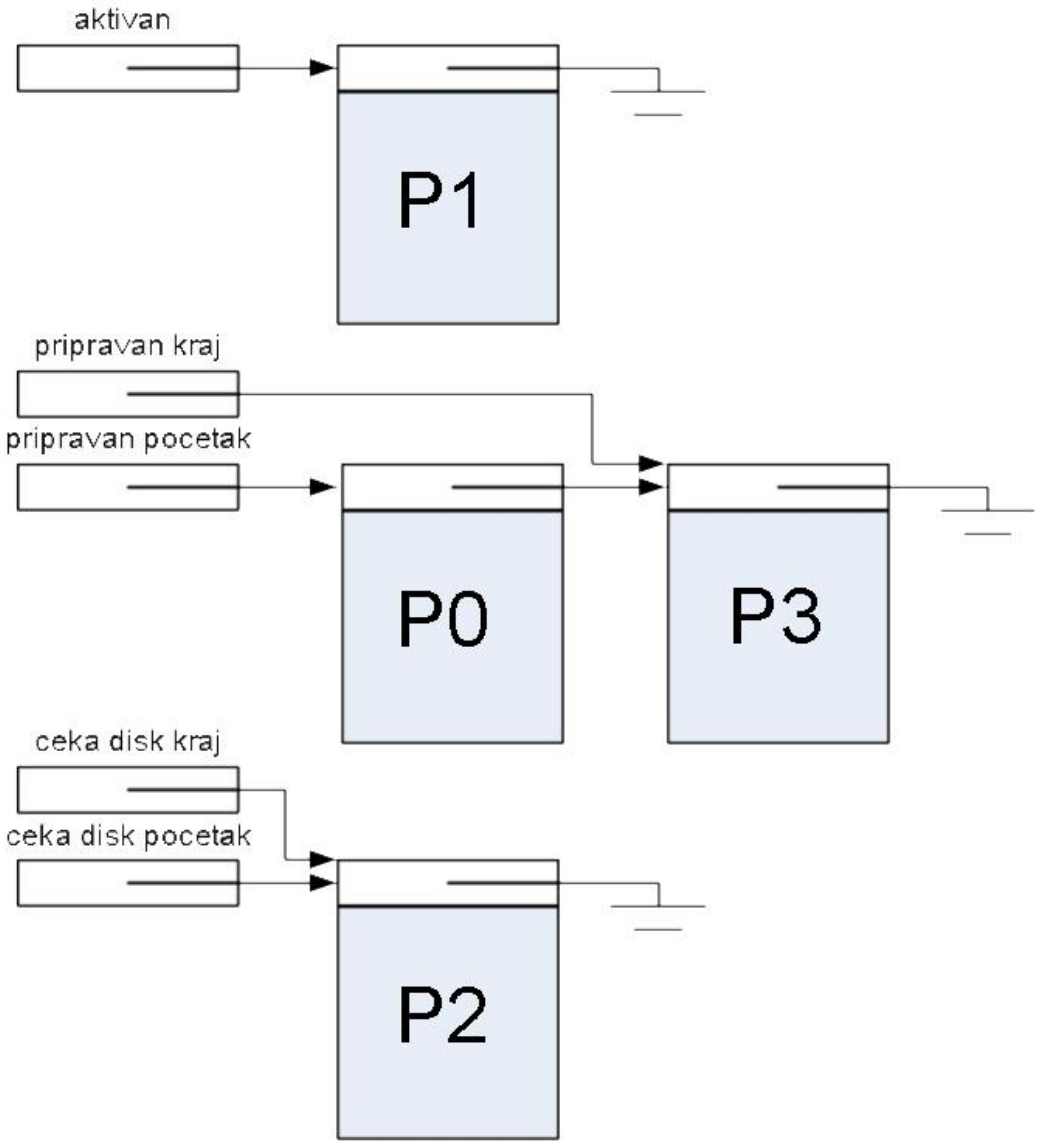
Proces je program u izvođenju jer mijenja sadržaj resursa. Sustav se sastoji od:
• korisničkih procesa - izvode korisnički kod
• procesa OS-a - izvode sistemski kod
Procesi se izvode paralelno međusobno dijeleći procesor. Obuhvaćaju trenutne aktivnosti u sustavu opisane sadržajima registara procesora i memorijskih lokacija koje koristi proces. Sadrži programski odsječak i stog procesa koji sadrži privremene i globalne podatke pohranjene u podatkovnoj memoriji. Proces je program u izvođenju sa programskim brojilom i pripadajućim skupom registara i memorijskih lokacija kao i resursa koje koristi.

2. Objasnite graf stanja procesa
Stanje procesa je određeno njegovom trenutnom aktivnošću.

Proces se može nalaziti u jednom od stanja:
• Novi (new) – stanje u kojem se proces nalazi nakon što je stvoren, kreira se PCB i proces ide u sljedeće stanje pripravan.
• Pripravan (ready) – označava proces koji se nalazi u redu za izvršavanje. Nakon nekog vremena, kada dobije pravo korištenja procesora, proces prelazi u stanje aktivan.
• Aktivan – označava proces koji se trenutno izvršava. Ukoliko OS prekine aktivan proces tada se taj proces prebacuje u stanje pripravan, a ukoliko proces mora obaviti neku I/O operaciju, tada ide u stanje čeka. (U oba slučaja se zapamti trenutno stanje popunjavanjem PCB-a).
• Čeka (waiting) – označava stanje procesa u kojem proces čeka na npr. Ulazno/Izlaznu operaciju.
• Završio (terminated) – označava stanje procesa u kojem je obrada završena, briše se PCB.
U jednom trenutku samo jedan proces može biti aktivan, dok ih više može biti u stanju pripravan ili čeka.



3. U sustavu je 4 procesa. P1 je aktivan, P3 i P0 su pripravni, a P2 čeka disk. Aktivni proces izvodi pisanje na disk. Nacrtajte stanje procesa, objasnite što će operacijski napraviti te se ovo programski rješava. (Objasnite kako OS izvodi prebacivanje iz stanja u stanje, Context Switch).



OS jednostavno prebacuje procese iz stanja u stanje tako da stavi proces u određen red. PCB među ostalim podacima sadrži i pokazivače koji će pokazivati na sljedeći i prethodni u redu, te polje u kojem je zapisano stanje procesa. Prebacivanje iz reda u red se obavlja jednostavnim preusmjeravanjem pokazivača.

Context switch je izmjena aktivnog procesa. Kod prebacivanja obrade s jednog procesa na drugi procesor mora sačuvati stanje prekinutog procesa tako da ga zapiše u PCB , te obnoviti stanje procesa (tako da učita sve potrebne podatke iz PCB-a) koji postaje aktivan. Sam context switch traje neko određeno vrijeme i nastojimo da to vrijeme bude što kraće. Problem je što u PCB upisujemo i informacije o dodijeli i korištenju memorije procesa. Taj proces dodatno usporava context switch. Zbog toga koristimo niti (threadove) koje smanjuju broj izmjena konteksta.

Koraci izmjene konteksta:
1. Sadržaj spremnika upiše se u PCB na koji pokazuje pokazivač aktivan
2. PCB iz reda aktivan stavi u red čeka
3. Prvi proces iz reda pripravan stavi u red aktivan
4. Obnovi stanje procesa koji postaje aktivan(prepiše stanja registara iz PCB-a).

4. Objasnite problem proizvođač – potrošač

Dva procesa su nezavisna ako nemaju zajedničke varijable i ne koriste iste dijelove memorije. Primjer zavisnih procesa je problem proizvođač-potrošač. Proizvođač proizvodi poruke, kojih može biti maksimalno n, a potrošač čita poruke i tako ih troši. Da se osigura da proizvođač ne unosi poruku u pun spremnik, odnosno da potrošač ne očitava poruku iz praznog spremnika uvodi se varijabla brojac koja se inicijalizira na vrijednost nula:

proizvođač

```
int in = 0;
while (1)
 {
        proizvedi podatak;
        while (brojac = = N) nop;
        stavi podatak u buffer[in];
        in++;
        if (in >= N) in = in%8;
        brojac++;
}
```

potrošač

```
int out = 0;
while (1)
 {
        while (brojac == 0) nop;
        pročitaj podatak iz buffer[out];
        out++;
        if(out >= N) out = out % 8;
        brojac --;
        potroši podatak;
}
```

Varijabla brojac je varijabla koju dijele oba procesa. U asemblerskom obliku naredbe za smanjivanje, i povećavanje brojača glase ovako:

proizvođač

```
mov     ax, brojac
inc     ax
mov     brojac, ax
```

potrošač

```
mov     ax, brojac
dec     ax
mov     brojac, ax
```

Ako npr. dođe do prekida nakon prvog reda izvođenja naredbe za inkrementaciju brojača i procesor se prebaci na izvođenje drugog procesa (potrošač), varijabla brojac ostaje nepromijenjena te dolazi do greške. Taj dio procesa, u kojem se prvo čita zajednička varijabla, zatim se obrađuje i na kraju ta izmijenjena vrijednost upisuje, se naziva kritičan odsječak. Kritičan odsječak općenito se definira kao dio procesa u kojem proces pristupa ili mijenja zajedničke varijable ili datoteke. Za operacijski sustav je bitno da osigura da kada je jedan proces u kritičnom odsječku tada niti jedan drugi zavisan proces ne smije izvoditi svoj kritičan odsječak.

5. Riješite problem proizvođač – potrošač pomoću semafora (2 semafora Spun, Sprazan, što je semafor i operacije nad semaforom)

Proizvođač:

```
while (1)
 {
        proizvodi poruku;
        while (brojac = = N) nop;

        stavi poruku u buffer;

        in++;
        if (in == N) in = 0;

        čekaj(S);
        brojac++;
        postavi(S);
}
```

Potrošač:

```
while (1)
 {
        while (brojac = = 0) nop;

        pročitaj poruku iz buffer;
        out++;
        if (out == N) out = 0;

        čekaj(S);
        brojac--;
        postavi(S);
}
```

Kritičan dio programa proizvođač, potrošač je dio u kojem se mijenja vrijednost zajedničke varijable brojac. Problem kritičnog dijela možemo riješiti koristeći semafor S. U ovom programu postoji i problem sinkronizacije u dijelu while (brojac == N) nop; i while (brojac == 0) nop. Taj problem rješavamo uvođenjem dvaju semafora Spun i Sprazan.

Proizvođač:

```
while (1)
{
        čekaj(Spun);

        proizvodi poruku;
        stavi poruku u buffer;

        postavi(Sprazan);
        in++;
        if (in == N) in = 0;
}
```

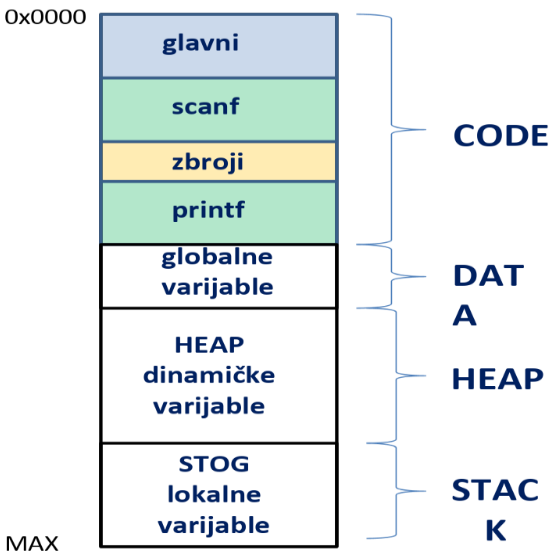
Potrošač:

```
while (1)
{
        čekaj(Sprazan);

        pročitaj poruku iz buffera;

        postavi(Spun);
        out++;
        if (out == N) out = 0;
}
```

Ako koristimo semafore, tada više nema potrebe za korištenjem naredbi while (brojac == N) nop; i while (brojac == 0) nop. Inicijaliziramo Spun.vrijednost = n (veličina buffera) i Sprazan.vrijednost=0. Kako se buffer puni (u programu proizvođač) tako se vrijednost varijable Spun.vrijednost smanjuje, a vrijednost Sprazan.vrijednost raste. Spun.vrijednost se može smanjivati maksimalno do -1 (to će ujedno značiti da je buffer pun) jer će tada funkcija čekaj(Spun) staviti proces proizvođač u red čekanja Spun i aktivirati proces potrošač. Proces potrošač: kako se buffer prazni tako funkcija čeka(Sprazan) smanjiva vrijednost Sprazan.vrijednost, a f-ja postavi(Spun) povećava vrijednost Spun.vrijednost. Vrijednost Sprazan.vrijednost se također može smanjivati samo do -1 kada f-ja čekaj(Sprazan) stavi proces potrošač u red čekanja na Sprazan i ponovo prebaci izvođenje na proces proizvođač. Glavna prednost ovakvog korištenja je u tome što se izbjegava prazan hod kod procesa proizvođač(kada je buffer pun) i kod procesa potrošač (kada je buffer prazan).

6. Kreiranje adresnog logičkog prostora

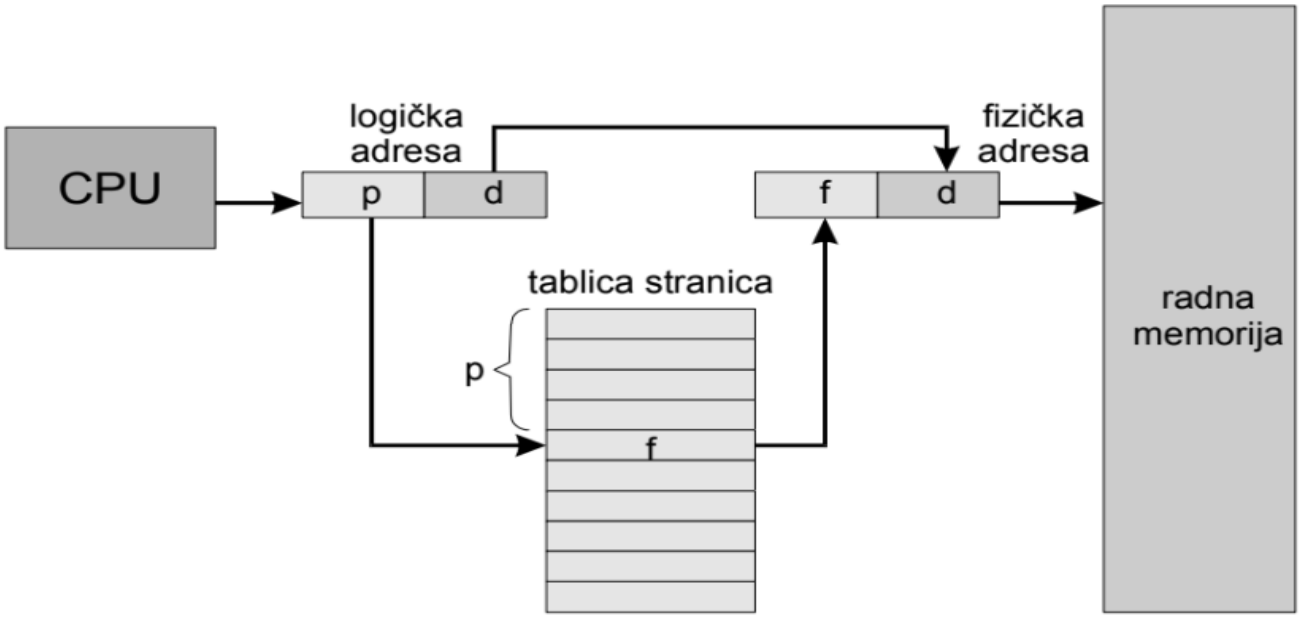
Otvori se editor (npr. Notepad) i napiše se izvorni (source, glavni.c zbroji.c, zaglavlje.c) kod. Nakon što je provjereno da li je kod ispravno napisan prevodi se (compile) izvorni u binarni kod i kreira se tablica simbola (symbol table). Sve zajedno je zapisano u datoteci glavnog koda koji se izvodi. U tablici simbola zapisane su sve funkcije i globalne varijable. Slijedi postupak povezivanje (Link) svih komponenata u jednu cjelinu. Nakon procesa kompajliranja i linkanja generira se kod i podaci koji započinju od adrese 0 do MAX.



7.Na jednostavan način objasniti dodjelu memorije po starnicama

Radna memorija dijeli se na blokove fiksne veličine koji se nazivaju okviri (frames). Kada se program upiše u memoriju, stranice se upisuju u slobodne memorijske okvire. Radi jednostavnosti prebacivanja programa s diska u radnu memoriju disk je podijeljen na okvire koji su jednake veličine kao okviri memorije.

Logička adresa koju generira procesor dijeli se na broj stranice p i pomak unutar stranice d. Broj stranice p predstavlja pokazivač na redak tablice stranica. U tablici stranica upisane su početne adrese okvira f. Kombinacija početne adrese okvira f i pomaka d određuje fizičku adresu memorijske alokacije.
Sklopovlje za dodjeljivanje memorije po stranicama:



Sustav za dodjelu memorije po stranicama djeluje na sljedeći način: Kad se program prihvati na izvođenje izračuna se potreban broj okvira i uspoređuje se s brojem slobodnih okvira u memoriji. Ukoliko je slobodan dovoljan broj okvira procese upisuje u memoriju stranicu po stranicu. Istovremeno se za svaku stranicu u tablici stranica upisuje i broj okvira u koji je ona upisana.