1. Što je OS, koji su osnovni zadaci?

Operativni sustav je programski paket koji djeluje kao sučelje između korisnika i hardvera, to je ujedno i skup programa koji vode kontrolu korištenja resursa(procesora, glavne memorije..) ; kontrolira izvođenje programa, dodjeljivanje memorije, upravljanje podacima.
Glavni cilj OS-a je da omogućava

- Jednostavnije korištenje računala → tako da korisnik pri radu ne mora voditi računa o specifičnosti hardvera
- Efikasnije korištenje resursa

2. Koji su zadaci Os-a?

Zadaci Os-a su:

- **Upravljanje procesima**: Proces je program koji se izvodi. Os mora obavljati stvaranje i poništavanje korisničkih i sistemskih procesa, odgađanje odnosno prekidanje i ponovno aktiviranje procesa, sinkronizaciju, komunikaciju među procesima i rješavanje deadlocka.
- **Upravljanje radnom memorijom:** Os mora: pratiti koji su memorijski dijelovi trenutno zauzeti i od koga; odlučivati kojem procesu će dodijeliti oslobođeni memorijski prostor; dodjeljivati i oslobađati memorijski prostor po potrebi.
- **Upravljanje sekundarnom memorijom**: Os mora: upravljati slobodnim memorijskim prostorom; dodjeljivati memoriju; upravljati zahtjevima za pristup sekundarnoj memoriji.
- **Upravljanje ulazom/izlazom**
- **Upravljanje datotekama:** Os mora omogućiti: stvaranje i uništavanje datoteka, direktorija; osnovne operacije s datotekama; organizirati sustav pristupa.
- **Zaštita korištenja sustava**
- **Otkrivanje pogreški u radu**
- **Rad s mrežom**
- **Tumačenje korisničkih naredbi**

3. Što je proces, čime je određen i kako OS vodi evidenciju o procesu?

Proces je program u izvođenju jer mijenja sadržaj resursa. Sustav se sastoji od:

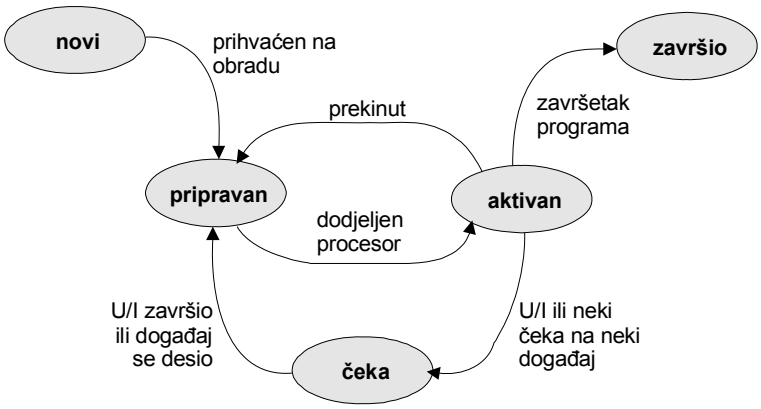
- korisničkih procesa - izvode korisnički kod
- procesa OS-a - izvode sistemski kod

Procesi se izvode paralelno međusobno dijeleći procesor. Obuhvaćaju trenutne aktivnosti u sustavu opisane sadržajima registara procesora i sadržajima 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.

Operativni sustav vodi evidenciju o procesu u **PCB**-u (Process Control Block). To je struktura u kojoj su smještene informacije 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)...

4. Navedi i objasni stanja u kojima se proces nalazi tijekom izvođenja!

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.

5. Kako OS organizira procese kako bi ih prebacivao iz stanja u stanje

Stanja pripravan, čeka na disk, I/O.. su u memoriji prikazana kao redovi realizirani pomoću vezanih lista. U Data segmentu OS-a nalaze se pokazivači:

- Pripravan početak (pokazuje na prvi proces u redu pripravan)
- Pripravan kraj (pokazivač na kraj reda pripravan)
- Aktivan (pokazivač na trenutno aktivan proces)
- Za svaki red čeka pokazivač na početak i kraj (slično kao i pripravni pokazivači, samo za red čeka)

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.

6. Objasni context switch

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).

7. Strategije dodjele procesora.

Postoje različiti algoritmi za dodjelu procesora. Kriteriji po kojima uspoređujemo koliko je dobro neko rješenje su:

- Iskoristivost hardvera
- Vrijeme čekanja što kraće – tako da korisnik dobije što bržu uslugu

Algoritmi za dodjelu procesora:

1. **Prvi u redu prvi poslužen (FIFO**) – ima najgore $t_{čeka}$
2. **Kraći poslovi prvi** (shortest jobs first, **SJF**) – problem trošenja vremena na računanje vremena izvršavanja pojedinih procesa
3. **Round – Robin** - time sharing kojemu je temelj FIFO uz uvođenje ograničenja vremena koje proces može provesti u izvođenju
4. **Ograničavanjem vremena u aktivnom stanju na vremenski kvant ΔT** - $t_{čeka}$ je relativno dobro. Treba odabrati optimalno vrijeme kvanta ΔT(ne smije biti premalo zbog izmjene konteksta). Ako se izvršavanje procesa prekine zbog isteka ΔT onda ga ponovno vraćamo u red pripravan ali mu povećamo ΔT.

8. Na primjeru proizvođač-potrošač objasni kritični odsječak.

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;
         procitaj podatak iz buffer[OUT];
         out+ +;
        if (out >= N) out = out % 8;
 brojac - -;
 potroši podatak;
}
```

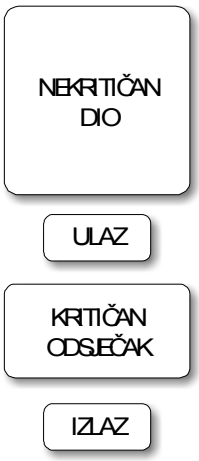
Varijabla *brojac* je varijabla koju dijele oba procesa. U asemblerskom obliku naredbe za smanjivanje, odnosno za povećavanje brojača glase ovako:

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.

9. Kako se rješava kritični odsječak pomoću *test&set*?



Naredbom test&set proces signalizira procesoru da ulazi u kritični odsječak i time osigurava da ga se, u tom dijelu, ne prekine u izvršavanju. Prilikom ulaska u kritičan odsječak proces provjerava da li je neki drugi proces u kritičnom odsječku i ako je, čeka da taj proces izađe i tek onda ulazi u njega. Kada proces izađe iz kritičnog odsječka jednostavno signalizira da je izašao.
Na ulazu u kritičan dio koristimo: while (test&set(flag)) nop; - s tim kodom provjeravamo da li je neki proces već u kritičnom odsječku i ako je čekamo da izađe iz njega.
Na izlazu iz kritičnog odsječka koristimo flag=0;

10. Objasni semafor

Korištenje semafora efikasno rješava:

- problem ulaska procesa u kritičan odsječak
- probleme vezene uz sinkronizaciju među procesima

Sam semafor je realiziran kao struktura koja se sastoji od cjelobrojne varijable (koja je u početku inicijalizirana na npr. 1) i pokazivača na PCB (na početku null). Problem ostalih algoritama je u tome što proces koji nije dobio dozvolu za ulazak u kritični odsječak aktivno čeka trošeći vrijeme procesora. Kod semafora taj se problem rješava primjenom operacija **čekaj** i **postavi.**

```
Čekaj (semafor S)
{
   s.vrijednost--;
   if(s.vrijednost<0)
   {
      upiši stanje procesa u PCB
      stavi PCB procesa u red čekanja na S;
      aktiviraj prvi proces iz reda pripravan;
   }
}
```

```
Postavi (semafor S)
{
   s.vrijednost++;
   if(s.vrijednost<=1)
   {
      prvi proces iz reda čeka semafora S
      stavi u red pripravan
   }
}
```

Proces koji izvodi operaciju **čekaj** i zaključi da ne smije nastaviti izvođenje mora sam prekinuti svoje izvođenje i stati u red čekanja na semaforu. Potom se izvođenje predaje operacijskom sustavu koji aktivira neki proces iz reda aktivnih procesa. Uz svaki oblik kritičnih odsječaka vezan je jedan semafor koji ima svoj red čekanja. Kad proces izvodi operaciju **postavi** na semaforu; jedan proces, koji čeka u redu na taj semafor, se aktivira i prebaci u red pripravan. Same naredbe promjene vrijednosti semafora moraju biti nedjeljive i njima direktno upravlja OS.

11. Kako pomoću semafora riješiti problem proizvođač potrošač?

```
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 buffera;
 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. Sada program izgleda:

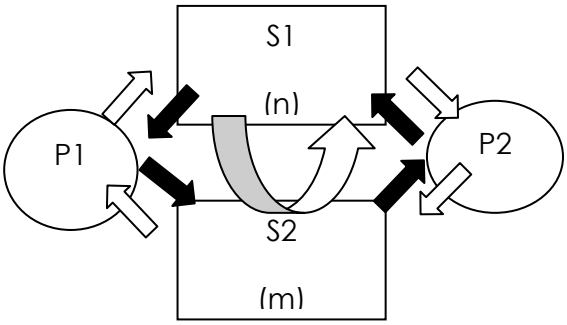
```
Proizvođač:
while (1)
{
 čekaj(Spun);

 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).

12. Problem potpunog zastoja (deadlock)

Više procesa može istovremeno zahtijevati uporabu istih resursa računarskog sustava. Primjer: Imamo dva procesa P1 i P2. P1 rezervira resurs S1 i izvršavanje se prebaci na P2 koji rezervira resurs S2. U idućem prebacivanju izvođenje se vrati na P1 koji sada želi rezervirati S2 ali taj resurs je rezerviran od strane P2 pa P1 prebacimo u stanje čekanja. Procesor prebaci izvršavanje na P2 koji sada želi rezervirati S1, ali ne može pa se i on prebaci u stanje čekanja. Ne izvršava se niti jedan proces. Ovakva situacija se naziva potpuni zastoj (deadlock).
n, **m** označavaju koliko istih resursa imamo.



Kvadrati S1 i S2 označavaju resurse(hardverske i softverske), dok krugovi **P1** i **P2** označavaju procese. Potpuna petlja označava da postoji zastoj.