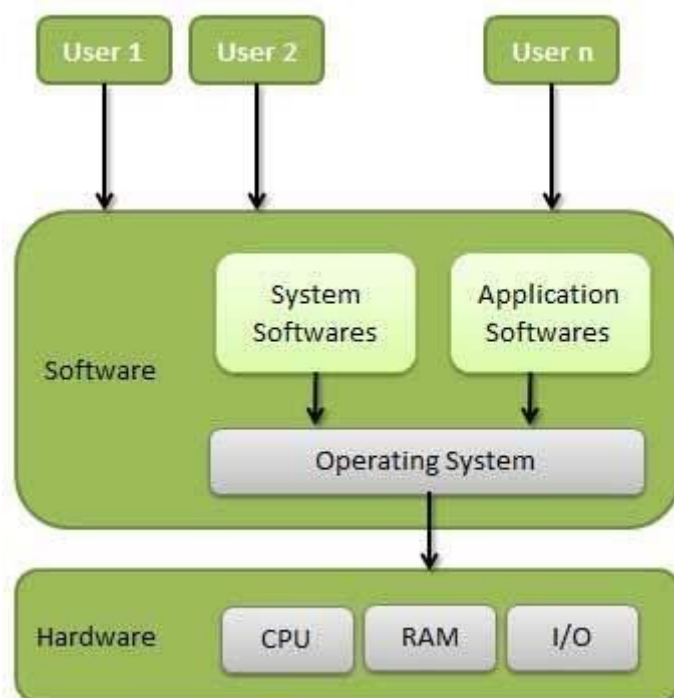


OPERACIJSKI SISTEMI – fun!

Kaj je OS?

Računalniški sistem sestavljajo:

- Uporabniki (users) - skušajo rešiti svoje računske probleme (ljudje, računalniški sistemi, numerično krmiljeni stroji,...)
- Programi (application software)
 - Uporabniški programi – opisujejo kako rešiti nek problem s pomočjo računskih virov (resources - lahko je pomnilniški prostor, procesorji, računski čas,...)
 - Sistemski programi – (urejevalnik), prevajalnik, zbirnik, povezovalnik, nalagalnik,...
- Strojna oprema (hardware) – zagotavljajo vse vire (CPE, pomnilnik, V/I naprave)



Dinamično, pojavljajo se konflikti (več zahtev za isto stvar) - Kdo usklajuje vse to dogajanje?

OS!

Kaj počne OS?

- Dodeljuje vire programom v izvajanju
- Rešuje konfliktne situacije
- Optimizira delovanje
- Nadzira dogajanje

Delovna definicija: OS je program, ki posreduje med uporabnikom in strojno opremo. Pri tem poskuša zagotoviti učinkovito rabo strojne opreme in prijazno okolje za uporabnika.

ZGODOVINA OS

Zgodnji računalniški sistemi:

- Zelo veliki, imajo konzolo
- operater = programer

- napiše program, ga naloži v glavni pomnilnik preko konzole ukaz za ukazom (dvojiški jezik), nastavi programski števec na začetni naslov programa. Opazuje izvajanje programa, ga prekine če je potrebno, na koncu odčita rezultat. Zelo zamudno, **MUČNO!**

Kmalu kasneje (?? 😊):

- dodatna oprema:
 - strojna:
 - čitalci luknjanega traku oz. kartic
 - programska:
 - zbirnik
 - nalagalnik (loader)
 - knjižnice programov
 - povezovalnik (linker – povezuje iz knjižnic)
 - gonilniki naprav
 - začetni debuggerji



enostavnejše programiranje



bolj »zapletena« (zamudna) raba računalniškega sistema

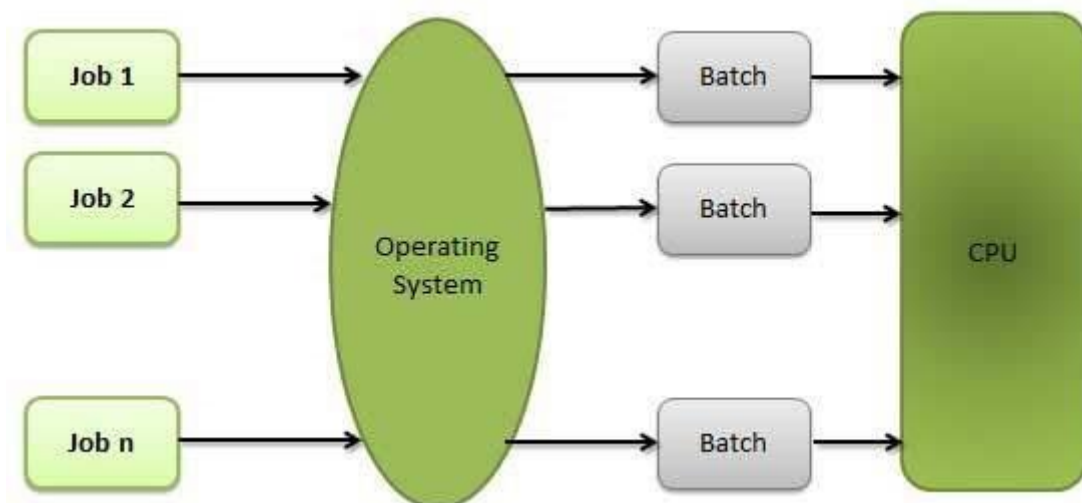
ZAKAJ?

Zaradi priprave poslov (POSEL (job) = vsi programi, ki so potrebni za izvajanje programa v računalniku)

ČAKO POVEČATI IZKORIŠČENOST RAČUNALNIŠKEGA SISTEMA?

Enostavna paketna obdelava:

- profesionalizacija operaterja
 - + krajši pripravljalni čas (čas od vnosa programa do začetka izvajanja)
 - ni interakcije program - programer
 - **paketna obdelava** – operater programe razvrsti v pakete (batch) po nekaterih kriterijih (jezik,...), paketi gredo en za drugim v računalnik – ni potrebno za vsak program posebej nalagati knjižnic,... Skrajša se pripravljalni čas.
- OPTIMIZACIJA** števila korakov pri izvedbi posla.



- + krajši pripravljalni čas
- ni interakcije program - programer



išče vedno premajhna izkoriščenost celotnega računalniškega sistema!

ZAKAJ?

Operater opazuje izvajanje poslov, če se zaradi napake ustavi, operater posreduje, poskrbi za ustrezen izpis. Ročno sproži naslednji posel v paketu.

Σ: Med enim in drugim poslom ter med paketi preteče preveč časa, ko računalniški sistem praktično miruje. Zavora je zopet človek.

SKLEP: človeka je treba zamenjati z nekim strojem (ne z Gregorjem).

S čim? **IDEJA:** z računalnikom samim. Ta računalnik naj skrbi za neprekinjeno avtomatično izvajanje zaporednih poslov.

[27.02.2018]

KAKO?

REZIDENTNI MONITOR (RM)

Naloga: rezidentni monitor naj do konca nekega posla poskrbi za avtomatično izvedbo naslednjega posla v paketu

Delovanje (načeloma):

- ob zagonu računalniškega sistema se RM naloži v glavni pomnilnik, se ga zažene, poskrbi za izvedbo 1. posla
- med izvajanjem posla RM ostaja v glavnem pomnilniku
- po izvedbi posla nadzor prevzame RM in poskrbi za izvedbo naslednjega posla v paketu

Kako naj RM poskrbi za izvedbo posla?

- uvedba kontrolnih kartic
 - kontrolna kartica je vsebovala navodila za RM
 - nekdo mora razpoznati -> interpreter kontrolnih kartic v RM
 - nalaganje se vedno dogaja -> nalagalnik v RM
 - gonilniki tudi stalno prisotni -> tudi v RM

RM je prvi OS (zgodovinsko)



boljša izkoriščenost vsega sistema



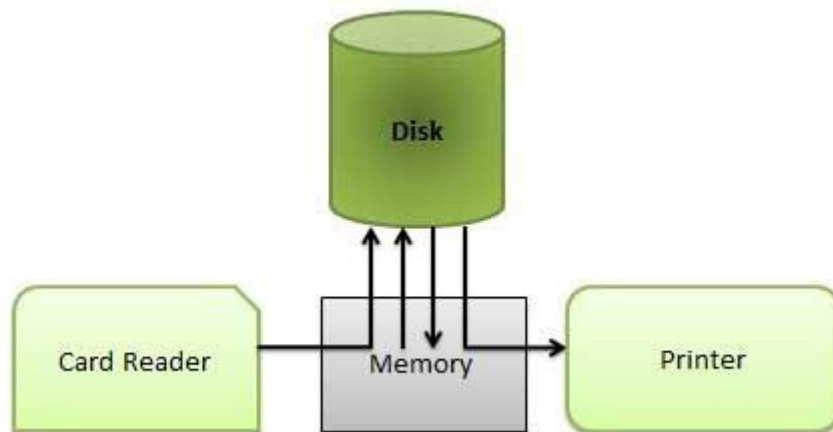
ni interakcije med programerjem in programom

Procesor je preslabo izkoriščen.

RAZLOG: počasnost V/I naprav (mehanskih) – procesor je velik del časa med V/I operacijami neizkoriščen.

REŠITVI (začasno izboljšanje):

- **Ločen V/I** (offline processing)
- **SPOOLING** (**S**imultaneous **P**eripheral **O**peration **O**n-**L**ine)
 - Se pojavi ob pojavu magnetnih diskov (disk je medij z neposrednim dostopom)
 - Čitalno/brisalna glava se lahko zelo hitro (v »zanemarljivem« času) premakne nad želene podatke
 - Magnetni disk se je lahko uporabil kot vmesnik za začasno hranjenje podatkov



Boljša izkoriščenost procesorja (in HW)



Še vedno ni interakcije

Opazili so, da izkoriščenost procesorja **še vedno ni zadostna!**

ZAKAJ? Nekateri programi imajo velike izhodno/vhodne zadeve, zato je veliko časa porabljenega za prenašanje podatkov med diskom in glavnim pomnilnikom.

MULTIPROGRAMSKA DODELAVA

Spooling omogoči, da je na disku več poslov pripravljenih

- Disk nudi neposredni doseg do vsakega od poslov
 - Lahko izberemo kateri naj se izvede naslednji!

Kateri P_i naj se izvede naslednji?

- **RAZVRŠČANJE POSLOV** (Job Scheduling) – to počne OS!

Kako še izkoristiti, da je na disku pripravljenih več poslov?

MULTIPROGRAMSKO IZVAJANJE poslov

- V glavnem pomnilniku je OS in nekaj pripravljenih poslov
- OS izbere enega izmed njih (po nekem kriteriju) in ga zažene
- Če se tekoči posel ustavi (ker čaka ali je končal – procesor »ne naprej«) OS prevzame nadzor in izbere nek drug pripravljen v glavnem pomnilniku ter mu preda nadzor



Boljša izkoriščenost procesorja

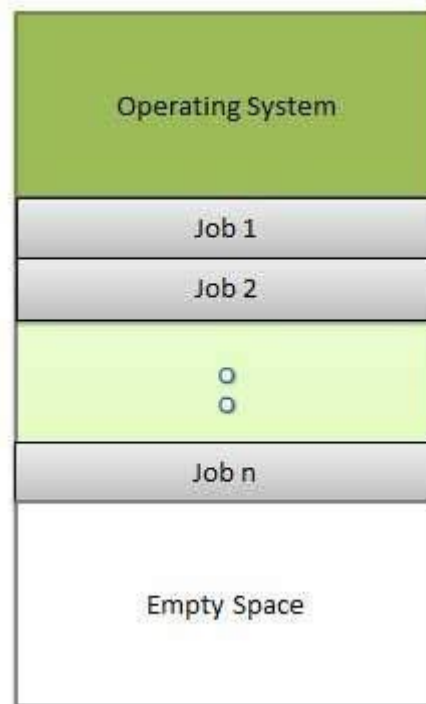


Še vedno ni interakcije

Σ: Ko se posel ustavi (= čaka ali konča) OS »takoj« dodeli procesor drugemu pripravljenemu poslu v glavnem pomnilniku. ← **RAZVRŠČANJE NA PROCESORJU** (Processor Scheduling)

DODATNE NALOGE OS:

- Razvrščanje poslov
- Razvrščanje na procesorju
- Upravljanje z glavnim pomnilnikom
- Zaščita



more
posel

ČASOVNO DODELJEVANJE (time sharing)

Interaktivnost, želje uporabnikov:

- Uporabnik ukazuje preko tipkovnice, OS izvaja ukaze ter poroča na zaslon
- Pri tem še uporaba interaktivne sistemske programske opreme

Interaktivnost razvijalca računalniškega sistema/lastnika:

- Potreben bo sproten datotečni sistem kjer bodo datoteke s programi /podatki ustrezno organizirane (zaradi dostopa...)
 - Cena narašča
- Človek se odziva počasi → procesor ga bo čakal
 - Izkoriščenost sistema pada

¿Ali je interaktivnost sistema sploh uresničljiva?

DA -> z uporabo časovnega dodeljevanja in multiprogramiranja

1. Hkrati dela več uporabnikov, vsak ima vsaj en svoj posel, toda vsak posel se sme neprekinjeno izvajati le omejen čas
2. OS izbere posel, mu dodeli časovno rezino in mu preda nadzor
3. Ko mine časovna rezina ali se posel ustavi (konča ali čaka) prevzame nadzor spet OS in nadaljuje s točko 2.

Tak OS se imenuje **večopravilni (multi-tasking)**



Boljša izkoriščenost celega sistema
Interaktivnost!!! **¡JEEJ, KONČNO!**

TERMINOLOGIJA:

Namesto posel (job) = proces (process)

INTUITIVNO:

Proces = program v neki fazi svojega izvajanja

NOVE NALOGE (VEČOPRAVILNEGA) OS:

- Dotočni sistem (file system)
- Upravljanje zunanjih pomnilnikov
- Upravljanje navideznega pomnilnika
- Zaščita
- Dodeljevanje procesorja
- Sinhronizacija & komunikacija med procesi

ELEMENTI RAČUNALNIŠKEGA SISTEMA

• DELOVANJE

- Sodoben računalniški sistem:
 - Vodilo
 - CPE
 - Glavni pomnilnik
 - Disk (druge zunanje enote – tiskalnik,...) --- krmilnik
 - KRMILNIK (controller)
 - Nadzoruje ustrezno napravo (ali več naprav)
 - Pri dostopanju do vodila »tekmujejo« s CPE (cycle stealing)
- Zagon
 - Ob vklopu se zažene **zagonski nalagalnik** (bootstrap loader)
 - Majhen program
 - Inicializira računalniški sistem

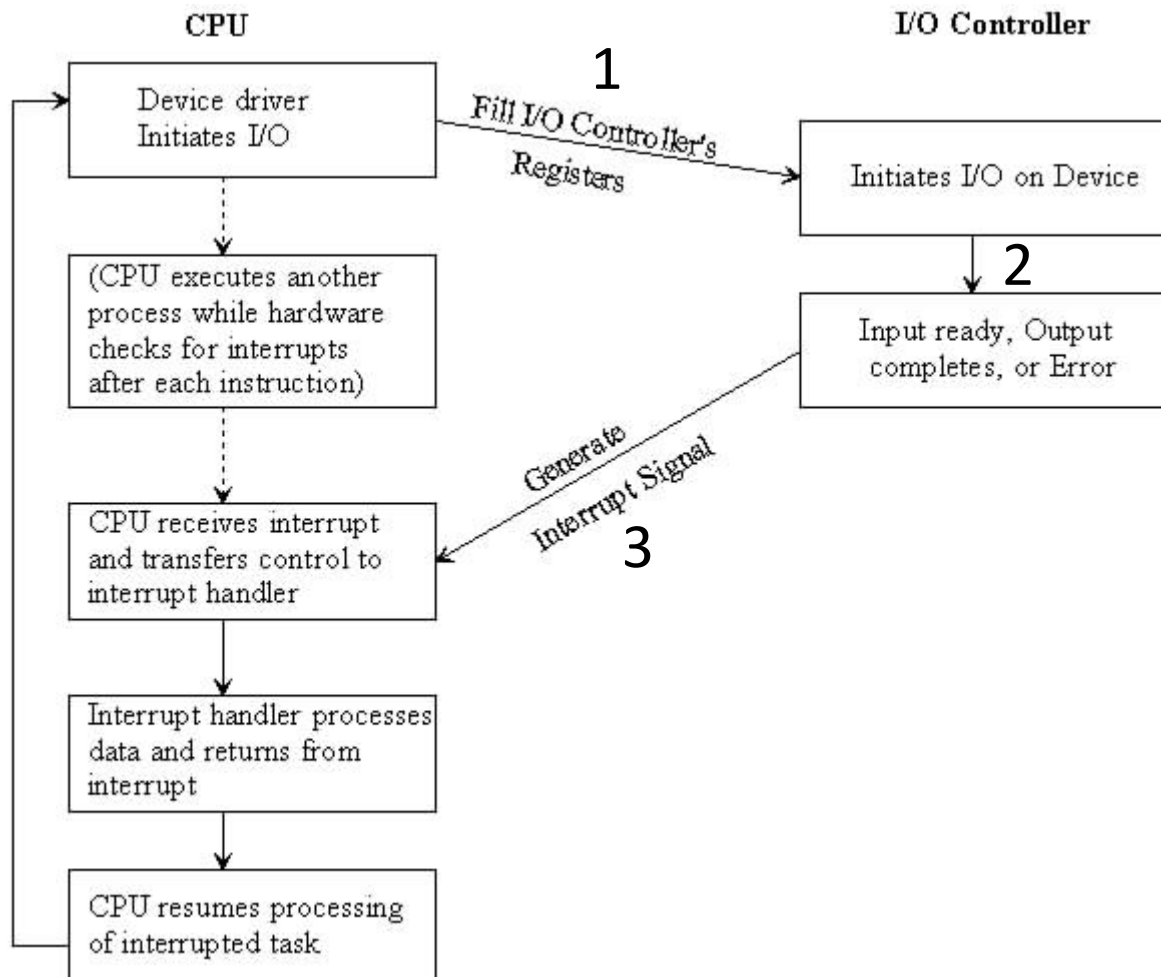
- Registre krmilnikov in ostale. Pač registre ki jih je treba nastavit 😊
- Naloži jedro operacijskega sistema
- Preda nadzor OS
- Delovanje (OS so vodeni z dogodki)
 - OS čaka dogodke in se nanje odziva
 - O dogodku ga obvesti prekinitvena zahteva (interrupt request)
 - Prekinitev je lahko:
 - Strojna – sproži jo HW (lahko je neuspešen doseg v glavni pomnilnik, požar??)
 - Programska – sproži jo tekoči proces
 - Past :((trap, exception)
 - Sistemski klic
 - V obeh primerih se OS odzove na prekinitveno zahtevo – prekinite tekoče opravilo (proces) in se odzove na prekinitveno zahtevo
 - **¿Kako OS določi kako naj se odzove?**
 - Potek prekinitve:
 - Tekoče opravilo, nek dogodek reče STOJ! (prekinitvena zahteva), zažene se **PSP** (prekinitveni servisni program), ko se konča se nadaljuje tekoče opravilo
 - **Izvedbe prekinitve:**
 - Programsko izpraševanje (polling)
 - Ko se pojavi prekinitveni zahtevek CPE prekinite tekoče opravilo (se shrani trenutno stanje (vsaj) registrov procesorja)
 - začne se izvajati **skupni PSP** (interrupt handler) in ugotovi kdo je dal prekinitveno zahtevo
 - začne se izvajati ustrezeni PSP
 - vrne se na izvajanje prekinjenega opravila
 - Vektorsko prekinjanje
 - V gl. pomnilniku je prekinitveni vektor, ki vsebuje naslove A1, A2,... ,An prekinitvenih servisnih programov PSP1, PSP2,... ,PSPn
 - Naprava ve, da je naslov njenega PSPi v i-ti komponenti vektorja
 - Ko naprava zahteva prekinitve, pošlje CPE-ju tudi svoj i
 - CPE prekinite tekoče opravilo (si shrani registre, sklad,...)
 - Uporabi indeks i, da prebere naslov Ai PSP-ja, ki ga mora izvesti
 - Prične izvajati PSPi
 - Ko se PSPi konča, se obnovi stanje procesorja ob prekinitvi in se nadaljuje prekinjeno opravilo

[13.3.2018]

• VHOD/IZHOD

- V/I sistem = V/I naprave + krmilniki + V/I programi
 - V/I naprave
 - Za prenos podatkov *ali*
 - Za hranjevanje podatkov – zunanji pomnilniki (disk, magnetni trak, optični pomnilniki, USB,...)
 - Krmilniki (device controller)
 - **Naloga:** skrbijo za podrobnosti pri prenosu podatkov med V/I napravo in svojim pomnilnikom
 - **Vsebuje:**

- Registre (zagotovo)
 - Statusni register – pove v kakšnem stanju je V/I naprava (lahko je zasedena (busy), prosta (done), napaka (error))
 - Ukazni register – tu je ukaz, ki naj ga izvede krmilnik (beri, piši,...)
 - Podatkovni register – podatki
- Pomnilnike (ni nujno)
- V/I programi
 - Izvajajo jih krmilniki, da razbremenijo procesor
- V/I operacija



- Potek:
 1. CPE vpiše ukaz in parametre v krmilnik
 2. Krmilnik razpozna ukaz in ga začne izvajati
 3. Ko krmilnik konča, pošlje CPE-ju zahtevo po prekinitvi
- Kaj med (1) in (3) počne CPE?
 - Sinhroni V/I
 - Proces – naročnik (CPE) **čaka**, da se (2) konča in se pojavi (3)
 - »wait«
 - »loop«
 - »busy wait« - naročnik stalno preverja statusni register pomnilnika



Hitrost



Slaba izkoriščenost CPE

- Asinhroni V/I
 - Proces – naročnik sprva nadaljuje svoj program
 - Nadaljuje brez težav do bridkega konca *ali*
 - »kmalu« (lahko tudi takoj) ugotovi, da ne more nadaljevati

- Tedaj se naročnik »blokira« (s sistemskim klicem) (→ sprostí CPE) in čaka do (3), OS dodeli CPE drugemu pripravljenemu procesu

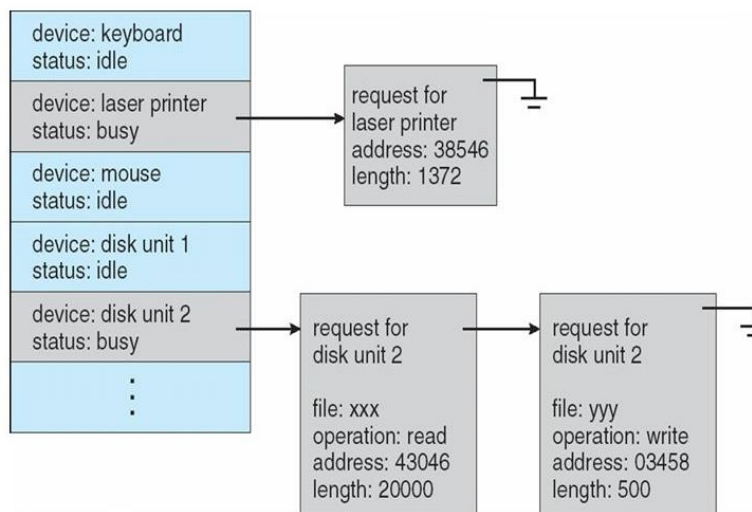


Boljša izkoriščenost celega sistema
Več možnih zahtev za isto napravo



»večja zapletenost«

- Omogoča več V/I operacij hkrati
 - **Kako s tem upravlja OS?**
 - Tabela stanj naprav (device-status table)
 - Vsaki V/I napravi OS dodeli ustrezno komponento tabele, ki vsebuje podatke o napravi
 - Ime/naslov naprave
 - Tip naprave
 - Stanje naprave
 - Kazalec, ki kaže na seznam vseh zahtevkov, ki so trenutno v igri za to napravo



- **Začetek V/I operacije**
 - Proces – naročnik želi V/I operacijo, zato sproži ustrezen sistemski klic (Kaj želi? Od koga?)
 - Nadzor prevzame OS, ki:
 - Ugotovi za katero napravo gre
 - Pogleda v tabelo stanj naprav (v komponento te naprave)
 - Če je naprava zasedena (vrsta ni prazna), potem OS vključi novo V/I zahtevo v vrsto zahtev
 - Če naprava ni zasedena, potem OS vključi novo V/I zahtevo v prazno vrsto zahtev in poskrbi za začetek V/I operacije
 - Če je naprava v okvari OS ustrezno odreagira
 - Nadzor vrne naročniku
- **Konec I/V operacije**
 - Ko V/I naprava konča, pošlje prekinitveno zahtevo (3)
 - Nadzor prevzame OS, ki:
 - Ugotovi za katero napravo gre
 - Pogleda v tabelo stanj naprav
 - Iz seznama zahtevkov za to napravo izloči končano V/I zahtevo in o koncu operacije obvesti (in obudi) njenega naročnika

- Če seznam za to napravo ni prazen (še čakajo), potem OS izbere eno od čakajočih zahtev in poskrbi za njeno aktiviranje
- DMA
 - Ali je CPE s prenašanjem podatkov med glavnim pomnilnikom in lokalnim pomnilnikom krmilnika obremenjen?
 - Če je V/I naprava hitra, pogosto pošilja (3) v CPE
 - DA!
 - Rešitev: **DMA** (Direct Memory Access)
 - Prenos med glavnim pomnilnikom in lokalnim pomnilnikom krmilnika izvede krmilnik mimo CPE
- **POMNILNIK**
 - Osnovna von Neumannova arhitektura:
 - Procesor
 - Pomnilnik
 - Začasen
 - Premajhen
 - Zato se doda zunanji pomnilnik
 - Stalno hranjenje podatkov
 - Velik
 - Glavni pomnilnik je prepočasn glede na procesor, to omilijo predpomnilniki (hiter, manjši)
 - **SKLEP:** zaradi praktičnih razlogov se pojavi pomnilniška hierarhija
 - Posledica: podatek (in njegove kopije) se lahko pojavijo na več mestih hierarhije
 - Če bi bil prisoten le en proces, P, se v procesor prenese najvišje ležeča vrednost (= najbolj ažurna)
 - Pri **večopravilnem OS** pa je potrebna previdnost!
 - Recimo, da je bil CPE procesu P odvzet (P čaka – ob prekinitvi Pja se stanje CPE shrani na sklad, torej A iz registra). Zažene se proces Q. Če Q tudi uporablja spremenljivko A, jo prebere iz zunanjega pomnilnika, kjer pa je proces P ni uspel dokončno ažurirati (**ojoj... Drama!**) → **NEKONSISTENTNOST PODATKOV!!** (zdravilo: HW in OS (Kako?))

[20.03.2018]

- **ELEMENTI STROJNE ZAŠČITE**
 - OS je postal »žrtev« raznih zahtev:
 - Zahteva po izkoriščenosti in interaktivnosti vodi do **VEČOPRAVILNOSTI**
 - Prekinitvene zahteve vodijo do **POMNILNIŠKE HIERARHIJE**
 - **ČAKO ZAGOTOVITI DA PROCES NE BO ŠKODIL DELOVANJU?**
 - **ODG:** za to naj poskrbi OS – toda potreboval bo nekaj (mehanizmov) strojne opreme:
 - **DVOJNI NAČIN DELOVANJA in PRIVILEGIRANI UKAZI**
 - **IDEJA:** nekateri ukazi so potencialno škodljivi, zato bodo privilegirani – smel jih bo uporabljati samo OS
 - **POSLEDICA:** računalniški sistem bo vedno delal v enem od dveh možnih načinov:
 - **SISTEMSKI (S)** – teče le OS in nekateri temu pridruženi programi, ki smejo izvajati privilegirane ukaze
 - **UPORABNIŠKI (U)** – tečejo uporabniški programi, ki ne smejo izvajati privilegiranih ukazov
 - Strojno opremo nudi bit način N, ki pove kateri je trenutni način delovanja. Poskrbeti moramo, da bo sistem prehajal med načini. Če bo proces poskušal izvesti privilegirani ukaz in je hkrati N=U, potem se proces ujame v PAST (→HW)
 - **ZAŠČITA V/I**
 - V/I ukazi se smejo izvajati le v sistemskem načinu (OS), večinoma so privilegirani
 - **ZAŠČITA POMNILNIKA**
 - Najpreprostejša zaščita – ukaz za spreminjanje obeh registrov (začetnega in končnega, znotraj katerih pomnilnik lahko »spreminjamo«, drugje pa ne) mora biti privilegirani

▪ ZAŠČITA PROCESORJA (ČASOVNIK)

- Če se program zacikla si bo bodisi prilastil procesor ali pa ga bo izkoriščal po nepotrebem. **KAKO TO PREPREČITI?**
- IDEJA: uvedemo časovnik (timer) ki bo periodično pošiljal prekinitvene zahteve. Ob vsaki zahtevi se bo pognal nek PSP, ki bo imel možnost posredovati
 - ne more posredovati tako, da bi analiziral ali se je program res zaciklal (ker tak algoritem ne obstaja (problem ustavitve)) in bi ga potem ustavil)
 - uvedemo raje **prioriteto** (če nek program dela dolgo časa mu zmanjša prioriteto in tako ta dobiva manj/manjše rezine procesorja)
 - PONAVLJAJ:
 - OS:
 - Izberi naslednji uporabniški program P
 - opravi vzdrževalna dela
 - vpiši v števec začetno vrednost
 - ...
 - Predaj procesor programu P
 - Čakaj
 - PROGRAM:
 - Teče dokler se ne izvede do konca (konča) || se izteče časovna rezina
 - Preda nazaj OS

Časovnik odšteva, ko se izteče generira zahtevo po prekinitvi

• SISTEMSKI KLICI

- Uporabniški program ne more izvesti nobenega od privilegiranih ukazov, ker se ob takem poskusu ujame v past
- **???Kaj če bi uporabniški program želel utemeljeno (z dobrim razlogom) izvesti nek privilegiran ukaz???**
- **REŠITEV:** uporabniški program mora zaprositi OS (oz. OS pooblastiti) da ta namesto njega strokovno izvede privilegiran ukaz (ker otroček ne sme uporabljati noža, prosi mami, da mu odreže kos kruha 😊). To pooblastilo/prošnjo izda v obliki ukaza, ki se imenuje SISTEMSKI KLIC (System call)

ELEMENTI OS

UPORABNIKOV POGLED:

- Komponente OS
- Storitve ki jih nudi OS

PROGRAMERSKI POGLED:

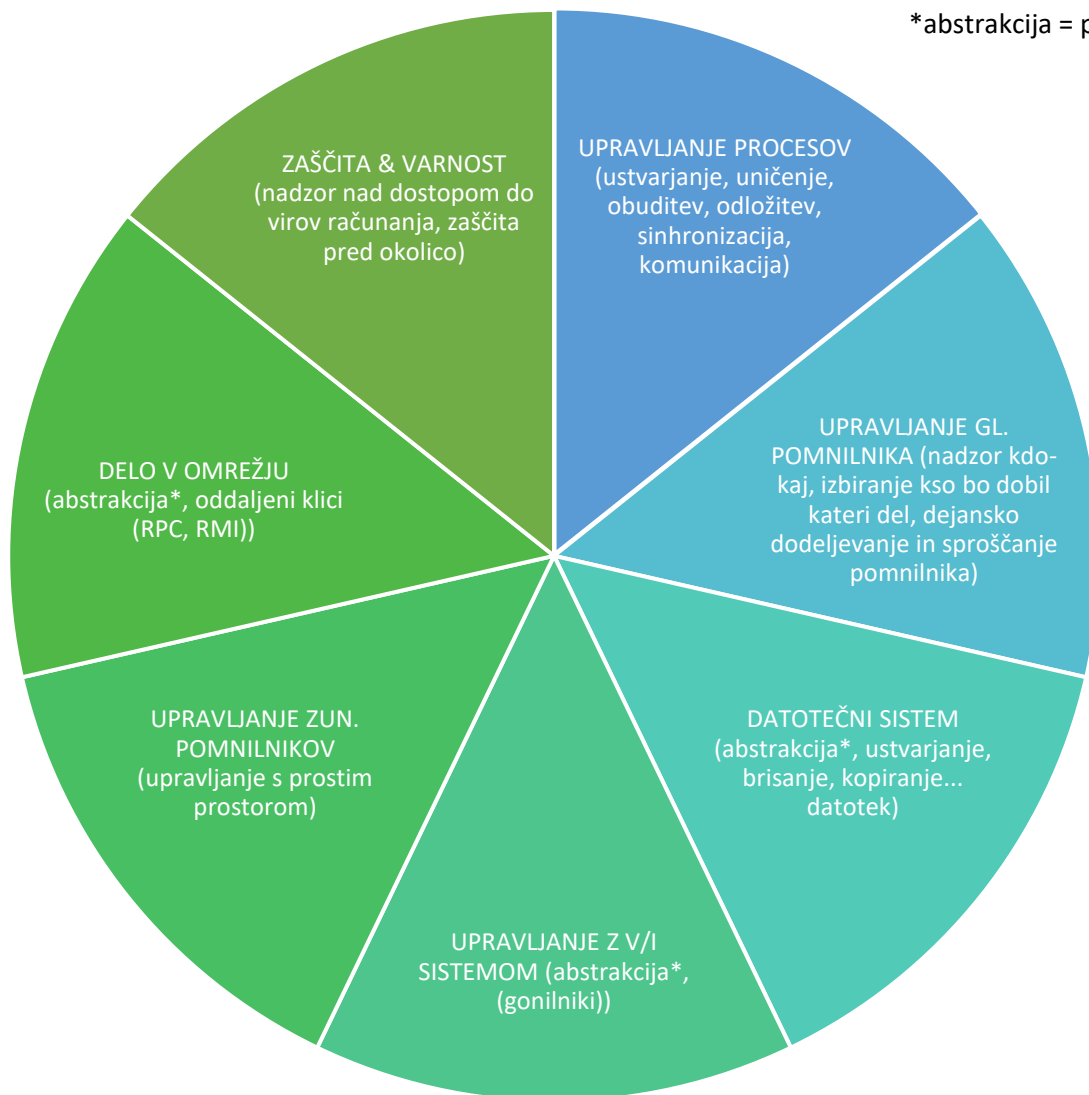
- Sistemski klici
- Sistemski programi (nalagalniki, povezovalniki, prevajalniki,...)

RAZVIJALČEV POGLED:

- Zgradba OS
- Virtualni stroj
- Snovanje OS
- Implementacija OS

KOMPONENTE OS

*abstrakcija = poenoten pogled na ____



[27.3.2018]

PROGRAMERJEV POGLED NA OS



SISTEMSKI KLICI:

- Programer s sistemskim klicem zaprosi OS, da ta v njegovem imenu izvede neko operacijo
- So način, da izkorišča OS
- Vsi skupaj tvorijo API (application program interface)
- Sistemski klic običajno potrebuje vhodne parametre
 - KAKO POSREDOVATI PARAMETRE?
 - Parametre vpišemo v vnaprej določene registre procesorja
 - Parametre vpišemo v pomnilnik, naslov pomnilnika vpišemo v register procesorja
- Višji programerski jeziki nudijo napredne sistemske klice kot običajne klice funkcij
- Sistemski klici so pogosti
- Delijo se v skupine:
 - Za upravljanje procesov
 - Ustvarjanje, končevanje, ustavljanje,...
 - Za upravljanje datotek
 - Za upravljanje naprav
 - Request, release
 - Za vzdrževanje in dostavo informacij
 - Ura, datum, uporabniki, kateri procesi tečejo,...
 - Za komunikacijo

- Standard POSIX (Portable Operating System Interface)

RAZVIJALČEV POGLED NA OS

Arhitektura (zgradba) OS:

- Nastane z nenehnim dodajanjem nalog → velik, zapleten, nepregleden, nesistematično organiziran (npr. DOS) → napake, nestabilnost, zelo težavna razširljivost
- Sistematično grajeni:
 - Razslojena zgradba  izboljšana  čas razširljivost
 - Nizko-jedrni OS
 - Zamisel: jedro izvaja **osnovne** naloge OS (upravljanje procesov, upravljanje glavnega pomnilnika, upravljanje komunikacije)
 - Vse ostale naloge so izven jedra



jedro je majhno → obvladljivo → stabilno



??

[3.4.2018]

PROCESI

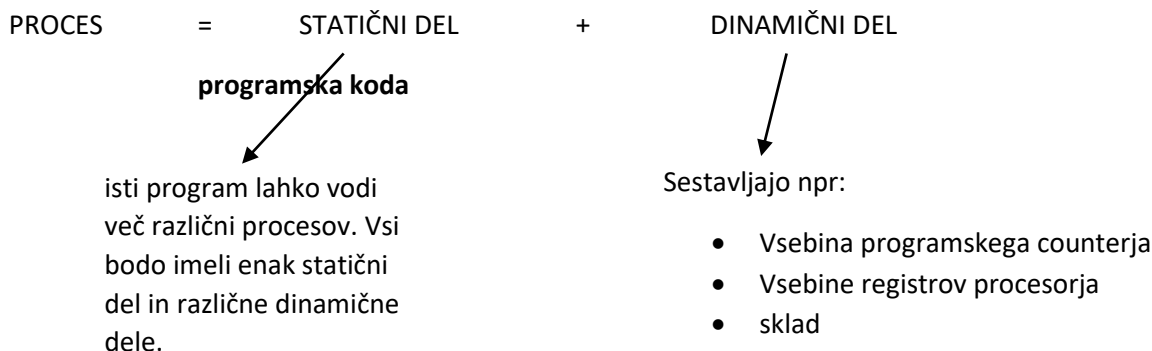
UVOD

V vsakem trenutku je v računalniškem sistemu množica aktivnosti, vse aktivnosti imajo nekaj skupnih lastnosti:

- Vsako usmerja nek program
- Vsaka je vedno v nekem stanju
- Vsaka rabi neke računske vire
- En program lahko usmerja več različnih aktivnosti

Aktivnost je torej program, ki je v nekem stanju → **PROCES**

Program je statična entiteta (vsebina neke datoteke). Proces je več od programa – poleg programa zajema še trenutno stanje neke druge, dinamične, entitete.



¿Kje se nahajajo vsi podatki iz statičnega in dinamičnega dela procesa?

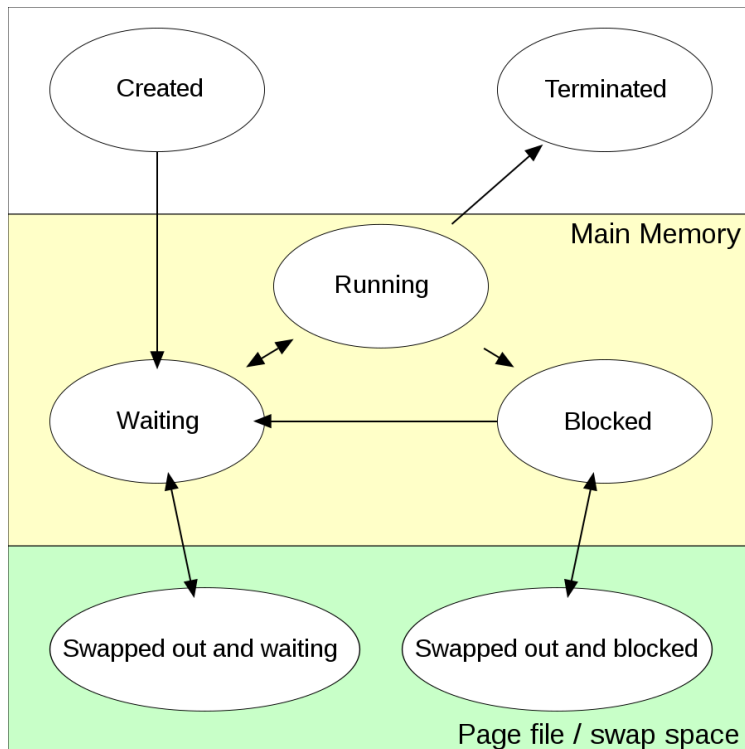
Marsikje → razpršeno po različnih HW enotah (glavni pomnilnik, procesor,...), v PCB-ju (nadzorni blok procesa),...

Stanje procesa

- Med svojim obstojem proces prehaja v različna stanja:
 - **nov**
 - če ga OS ravno ustvarja (zanj OS pripravlja vse, kar bo proces rabil med obstojem (tko kot je za novorojenčka treba kupiti dudo 😊))
 - **pripravljen**

- ko mu manjka le procesor
- **teče**
 - ko izvaja ukaze svojega programa
- **čaka**
 - ko čaka na nek dogodek
- **končan**

DIAGRAM PREHAJANJA STANJ *PROCESS STATES*



Nadzorni blok procesa (PCB)

Znotraj OS je vsak proces reprezentiran s svojim nadzornim blokom (PCB), ta vsebuje vse podatke o tem procesu (ko ta ne teče).

- Stanje procesa
- Podatki za razvrščanje
 - Prioriteta
 - Porabljen procesorski čas ...
- Podatki o procesorju
 - Vsebine PCB, registrov
- Podatki za upravljanje z glavnim pomnilnikom
 - Tabela strani, segmentov
- Stanje V/I naprav
 - Seznam dodeljenih naprav
- Računovodski podatki
 - ID procesa ...
- ...
- Ostalo
 - Lastnik procesa
 - Seznam sinov ...

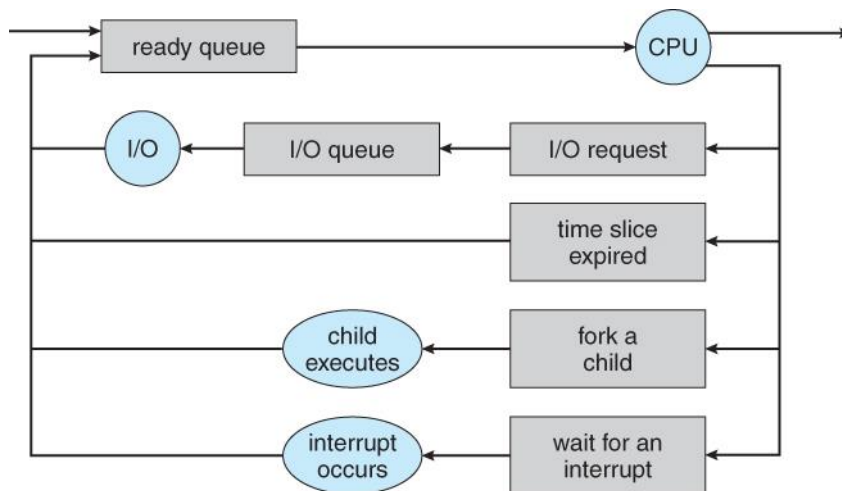
Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc....

VRSTE PCB-jev

PCB-ji imajo razne kazalce za vključevanje v razne vrste:

- Vrsta vseh (obstojećih) procesov (job queue)
- Vrsta vseh (trenutno) pripravljenih procesov (ready queue)
- Vrsta čakajočih procesov na V/I napravo
- ...

Očitno je lahko nek PCB v več vrstah hkrati → OS vključuje PCB procese v razne vrste oziroma ga iz njih izloča, skladno s prehajanjem procesa iz enega stanja v drugo. To je včasih predstavljeno s t.i. diagramom vrst.



[10.4.2018]

OPERACIJE NAD PROCESI

Ustvarjanje procesa

- Proces (oče) lahko ustvari novega (sina)
 - ĆKako oće ustvari sina? Sistemski klic (fork)
 - ĆKdo sinu priskrbi računske vire?
 - Da mu jih oće (podari ali tekmuje)
 - Da mu jih OS
 - ĆKakšen je sin?
 - Samosvoj
 - Kopija očeta
- **Ustvarjanje sina v UNIX**
 - P želi ustvariti sina (sistemski klic fork())
 - OS začne izvajati sistemski klic fork
 - Rezervira prostor v gl. pomnilniku
 - Ustvari PCB sin
 - Dodeli IDsin sinu
 - Medtem P čaka, njegov PC kaže takoj za i=fork()
 - Kakšen je sin? → kopija očeta v trenutku klica fork()
 - Ima enak program kot P
 - Ima enak dinamični del kot P
 - PC mu kaže takoj za fork()
 - Ko OS konča ustvarjanje Psin, izpiše IDsin v i v programu P, v sinovem programu pa ostane i=0
 - OS bo spremenil P v stanje pripravljenosti in Psin v stnje pripravljenosti
 - P in Psin nadaljujeta, ko dobita procesor
 - Preden nadaljujeta, se razlikujeta le v vrednosti i v programu P in Psin

Končanje procesa

- Proces z ustreznim sistemskim klicem (exit()) doseže, da ga OS konča, pospravi za njim
 - Odvzamemo vse vire
 - Sprosti podatkovne strukture (npr. PCB)

- **UNIX:** OS pošlje signal očetu, če oče že čaka na konec tega sina, OS vpiše v status opis, kako se je sin končal

Uničenje procesa

- Pripravljen, čaka → konec

Razvrščanje procesov


- Na procesorju (processor scheduling)
 - Komponente za upravljanje s prenosom OS vsebuje dva pomembna programa:
 - Razvrščevalnik (scheduler): izbere enega od pripravljenih procesov
 - Dodeljevalnik (dispatcher): izvede dodelitev CPE (...) izbranemu procesu
 - **¿Od česa je odvisen preklonni čas?**
 - Razvrščevalni algoritem
 - Kako se stanje procesorja shrani
 - Vsi registri hkrati
 - Vsak posebej
 - Hitrost pomnilnika (zaradi branja in zapisovanja)
 - Zgradba, arhitektura procesorja
 - Ostale komponente OS


[17.4.2018]

- *Razvrščanje z/brez odvzemanja procesorja*

¿Kdaj se lahko aktivira razvrščevalnik?

- Razvrščanje se sproži samo kadar procesor res ne more nadaljevati (drugje pa ni nujno).
- Glede na to ločimo dve vrsti razvrščanj:
 - Razvrščanje se sproži takrat ko CPE ne more nadaljevati procesa (ko čaka na nekaj ali se je proces čisto končal (1 in 4)) – **razvrščanje brez odvzemanja (non-preemptive)**
 - Razvrščanje se sproži (poleg 1 in 4) še v vsaj eni od ostalih situacij (2,3,5), tudi ko bi proces sicer še lahko nadaljeval – **razvrščanje z odvzemanjem (preemptive)**

 manjša prilagodljivost

 večja prilagodljivost

 Povzroča nevarnosti

ALGORITMI ZA RAZVRŠČANJE PROCESOV NA PROCESORJU

¿Na podlagi česa vrednotiti algoritme za razvrščanje?

Razna merila:

- Izkoriščenost CPE
- Propustnost sistema
- Čas obdelave procesa (od trenutka ko se je kreiral, do trenutka ko se je končal)
 - $\overline{t} = (t_1 + t_2 + t_3 + t_4)$
 - t_1 -> čas ustvarjanja, t_2 -> čas čakanja v vrsti pripravljenih, t_3 -> čas izvajanja, t_4 -> čas čakanja, ko je bil blokiran
- $\overline{t_2}$ čakalni čas (waiting time)
- Odzivni čas (čas od rojstva do prvega izhoda)
- Pravičnost (fairness)

Nekatera merila so med seboj odvisna (vplivajo eno na drugega) – če izboljšamo enega se lahko drugi poslabšajo

ŽELJA: Imeli bi hiter (polinomska časovna zahtevnost) razvrščevalni algoritem, ki bo maksimiziral mere a, b, f, in hkrati minimiziral mere c, d, e.

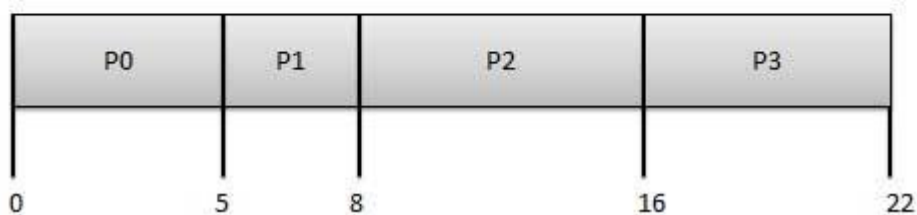
OPTIMIZACIJSKI PROBLEM, ki je NP-težek → razvrščevalni algoritmi so hevristični

- *PRVI PRIDE PRVI MELJE (FCFS)*

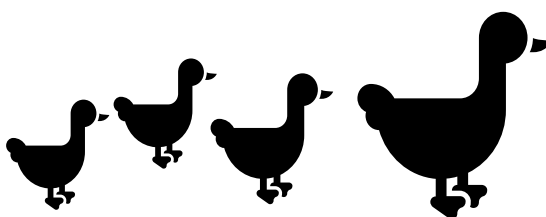
- Proces, ki je prvi zahteval CPE, prvi dobi CPE

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Gantov diagram:



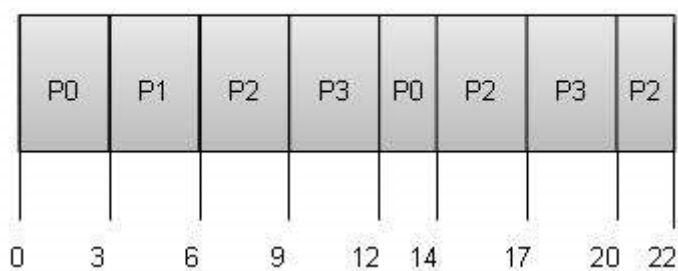
- Razvrščanje brez odzemanja
- **Pojav konvoja (convoy effect)** ← to bo na izpitu



- *KROŽNO RAZVRŠČANJE (Round robin scheduling?)*

- Krožni seznam procesov – proces dobi CPE za neko časovno rezino

Quantum = 3

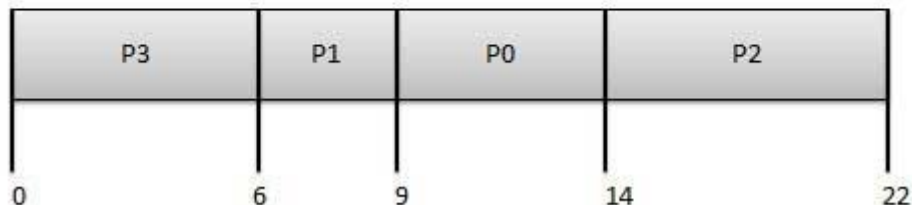


- Razvrščanje z odvzemanjem
- Po koliko časa dobi proces naslednjo rezino
- Če proces potrebuje neto t časa takoj, bo končal v družbi ostalih

- *RAZVRŠČANJE PO PRIORITETI (Priority scheduling)*

- Proces dobi neko prioriteto, CPE se dodeli pripravljenemu procesu z najvišjo prioriteto

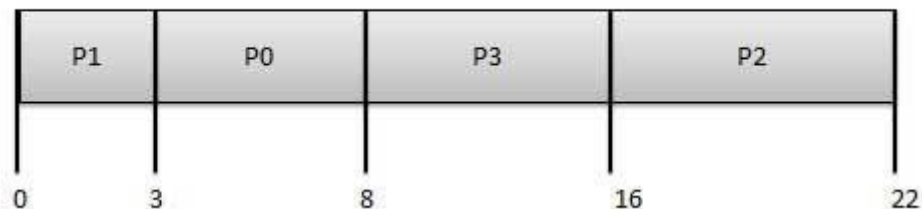
Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



- Razvrščanje z ali brez odvzemanja
- Če med pripravljene procese vstopi proces Q in ima svojo prioriteto večjo od tekočega procesa P tedaj:
 - Pri razvrščanju brez odvzemanja to ne bo prekinilo P
 - Pri razvrščanju z odvzemanjem, to lahko prekine proces P (ki se mu CPE odvzame)
- **NEVARNOST:** pripravljeni proces nikoli ne pride do CPE, ker ga stalno prehitvajo novi pripravljeni z višjimi prioritetami → proces **strada** (starvation)
- **ZDRAVILO:** proces nakrmimo (se postara, zato mormo zanj bolj skrbet ♥) → pripravljenim procesom se periodično povečajo prioritete (aging) ← to je eno izmed vzdrževalnih del (housekeeping) ob preklopu
- **SHORTEST JOB FIRST (SJF ali SJN)**
 - Nemoteni tek procesa je proces od trenutka, ko je proces dobil CPE, do trenutka ko se blokira ali konča
 - CPE naj se dodeli pripravljenemu procesu, katerega nemoteni tek bo najkrajši (najkrajši CPU burst)

LAHKO BI TEKU,
AMPAK NI, KER
SO DRUGI TEKL.
»Life motto, R.
Robič«

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



- Razvrščanje z ali brez odvzemanja
- Algoritem SJF daje minimalni povprečni čakalni čas t_2 (najboljši med vsemi algoritmi za razvrščanje brez odvzemanja)
- **¿Kako določiti dolžino nemotenega teka?** (to se šele bo zgodilo)
 - Ne moremo... 😞
 - **¿Ali ga lahko ocenimo?**
 - Predpostavimo, da bo dolžina naslednjega nemotenega teka podobna dolžini zadnjega nemotenega teka

- **¿Kaj pomeni podobno?**
 - t_n ... dolžina zadnjega nemotenega teka
 - T_n ... naša ocena dolžine zadnjega nemotenega teka
 - T ... ocena dolžine naslednjega nemotenega teka
 - $T = \alpha * t_n + (1-\alpha) * T_n$; $0 \leq \alpha \leq 1$
- V napovedi se zgodovina upošteva, toda čedalje manj, ko gremo bolj v preteklost

[24.4.2018]

- **SHORTEST REMAINING TIME FIRST (SRTF)**

- **¿Kaj če bi želeli še odvzeti procesor?**
- Ko nov proces R, ki vstopi med pripravljene, se lahko zgodi, da je dolžina njegovega neposrednega nemotenega teka krajša, kot ima do izstopa svojega nemotenega teka tekoči proces P
- To je kombinacija SJF in odvzemanja = SRTF

Process	Arrival Time (T_0)	Time required for completion (ΔT) (CPU Burst Time)
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Gantt Chart :

P0	P1	P2	P1	P3	P0	
0	1	3	5	9	13	22

- **ALGORITEM LOTERIJA**

- ZAMISEL: vsak proces dobi neko število dovolilnic (svojih), vse dovolilnice so v skupnem košu. Razvrščevalnik vsakič iz koša naključno potegne listek in dodeli procesor lastniku izžrebanega listka.

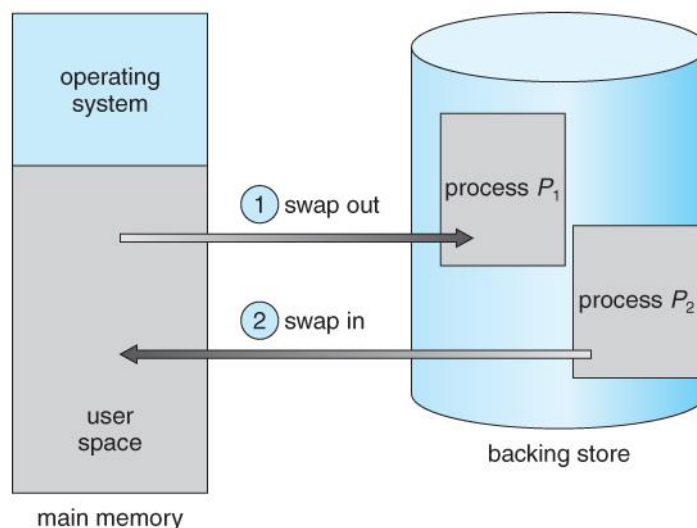


- * potreben je generator (psevdo) naključnih števil
- * posojanje listkov

RAZVRŠČANJE POSLOV (Job Scheduling)

- **RAZVRŠČEVALNIK POSLOV (job scheduler)**

- sproži se relativno redko → lahko bolj razmišlja in vrne bolj kakovostne odločitve
- vzdržuje dobro razmerje CPE in V/I zahtevnih procesov



- **SWAPPER** (del OS komponente za upravljanje s procesi)
- OS mora spremljati obremenjenost enot računalniškega sistema (razmerje med CPE- in V/I-zahtevnimi procesi)
→ to je del vzdrževalnih nalog med preklopi

PROCESI II

- Sodelovanje med procesi
 - S sporočili (IPC)
 - Preko skupnega pomnilnika

SODELOVANJE PROCESOV

- Včasih morajo procesi eksplicitno sodelovati (npr. deli in vladaj z več procesi)
- Pri tem sodelovanju pride do:
 - Komunikacije med procesi
 - Sinhronizacije procesov

[9.5.2019]

- **Problem proizvajalec-porabnik** (producer-consumer)
 - Dana sta 2 procesa:
 - Proizvajalec (P) – proizvaja podatke
 - Porabnik (Q) – uporablja te podatke
 - Proizvajalec proizvaja podatke, medtem jih porabnik že porablja (vse poteka sočasno)
 - *čKako med njima poteka komunikacija?*
 - Dva modela komunikacije:
 - Za prenos podatkov skrbi OS (IPC skrbi za vse)
 - Za prenos podatkov skrbita P in Q sama in sicer preko skupnega pomnilnika (shared memory)
 - **IPC – inter process communication**
 - Potek:
 - P in Q zaprosita za pridobitev povezave med njima (connection)
 - P: open_conn(Q) sistemski klic
 - Q: accept_conn(P)
 - P podatek zavije v sporočilo (m - podatek + ostale informacije (o podatku, cilju,...)) in ga odpošlje
 - Send(Q,m) sistemski klic
 - Q čaka na sporočilo s, ko ta pride, ga shrani v x
 - Receive(P,x) sistemski klic

- P in Q prekinita povezavo
 - P: done.conn(Q)
- **Povezava** je lahko:
 - **Neposredna** ali **posredna** ((in)direct)
 - **Simetrična** ali **asimetrična** ((a)symmetrical)
 - **S kopičenjem** ali **brez kopičenja** ((no) buffering)
 - Prenos _____ ali _____
- **NEPOSREDNA POVEZAVA** (direct connection)
 - Procesi se pri komunikaciji eksplicitno imenujejo
 - Lahko je:
 - Simetrična
 - Pošiljatelj in prejemnik (dva procesa) drug drugega eksplicitno imenujeta:

P: send(Q,m) pošlji Q-ju sporočilo m

Q: receive(P,x) sprejmi sporočilo od P
 - Povezava služi le dvema procesoma, med procesoma je kvečjemu ena taka povezava
 - Asimetrična
 - Samo pošiljatelj eksplicitno imenuje prejemnika (prejemnik pa pošiljatelja ne):

P_{1,2,3...}: send(Q,m) pošlji Q-ju sporočilo m

Q: receive(p,x) sprejmi sporočilo x od kogarkoli in v p shrani ime pošiljatelja

[16.5.2019]

...

[23.5.2019]

- **Problem kritičnih odsekov** (critical section)
 - **Scenarij:** recimo, da imata oba od P in Q svoj PC tik pred izvedbo A₁ oziroma B₁
 - **Ker sta P in Q sočasna** (oba tekmujeta za procesor) se lahko zgodi, da se ukazi na A₁, A₂, A₃ in B₁, B₂, B₃ prepletejo (zaradi preklapljanja).

Scenarij 1

(stevec=5)

A1

(R1=5)

A2

(R1=6)

A3

(stevec=6)

: preklap na Q

(stevec=6)

B1

(R2=6)

B2

(R2=5)

B3

(stevec=5)

Scenarij 2

(stevec=5)

A1

(R1=5)

A2

(R1=6)

: preklap na Q (R1=6 se shrani)

(stevec=5)

B1

(R2=5)

B2

(R2=4)

B3

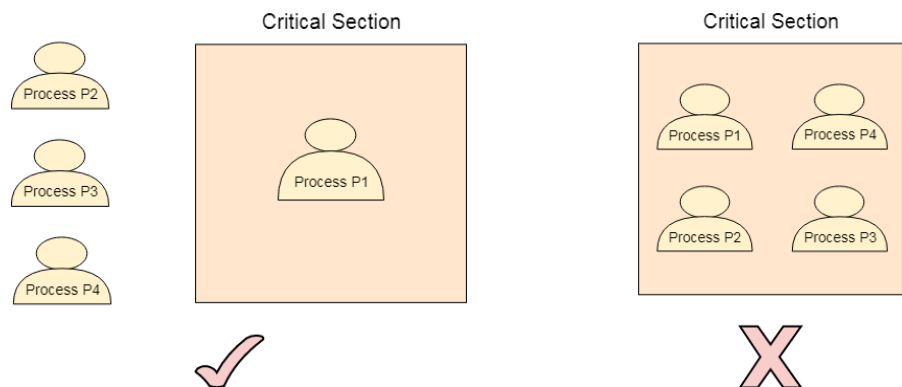
(stevec=4)

: preklap na P (ohrani se R1=6)

A3

(stevec=6)!!!

- **Vzrok:** preklapljanje med P in Q in nesrečno prepletanje ukazov A_1, A_2, A_3 in B_1, B_2, B_3 .
- **Sklep:** zagotoviti moramo, da se eden od njiju ne bo začel izvajati, če se drugi še ni končal.
- Pravimo, da ukazi A_1, A_2, A_3 tvorijo kritični odsek
- **Kako zagotoviti pravilno delovanje kritičnih odsekov?**



- Sočasno se lahko izvaja kvečjemu en proces (nič prepletanja med kritičnimi odseki teh procesov) – vzajemno izključevanje (mutual exclusion)
- Procesu ne smemo preprečevati vstopa v njegov kritični odsek neskončno dolgo
- Na izbor kdo naj naslednji vstopi v svoj kritični odsek vplivajo le kandidati za vstop
- Protokol naj bo splošen (za poljuben n in neodvisen od tehnologije)
- **METODE REŠEVANJA PROBLEMA KRITIČNIH ODSEKOV:**
 - Zamisel: procesi (njihovi programi) naj imajo naslednjo zgradbo:

```

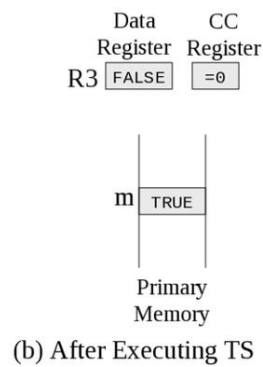
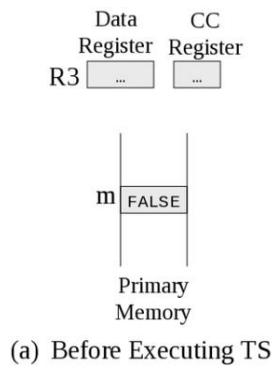
Pi:   repeat
      ...
      [vstopni del] ← štiti ta proces in ostale, da se kritični
                     odseki ne prepletajo (Pi-ju se prepreči oziroma dovoli
                     vstop v k.o.)
      [kritični odsek]
      [izstopni del] ← (ostalim Pj-jem pove, da se je kritični
                       odsek končal in lahko vstopajo ostali)
      ...
      until
  
```

- Nekatere možnosti za napolnitev **[]**:
 - Čista algoritmična/programska rešitev (v jeziku, ki ga uporabljamo)
 - »Bakery algorithm«,...
 - Velika časovna zahtevnost (ni preveč praktično)

- Uporaba posebnih, enostavnih ukazov, ki so bili razviti za ta namen (Test_and_Set, Swap)

opisano je v visokem jeziku, toda to ni prava implementacija!

- Test_and_Set



```
boolean Test_and_Set( boolean memory[m] )
{ [
    if( memory[m] )           // lock denied
        return True;
    else {                     // lock granted
        memory[m] = True;
        return False;
    }
]
```

- Swap

```
repeat
    var = true
    while (var == true) swap (lock,var);

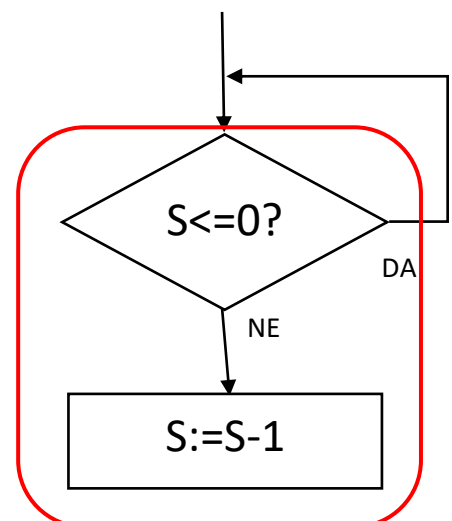
    CRITICAL SECTION

    Lock = false;

    REMAINDER SECTION
until false
```

- Oba delujeta atomarno
- Porabimo relativno veliko časa pred vstopom v kritični proces - če je k.o. kratek sta oba precej nepraktična
- Je rešitev, ampak ne glih za raketo al pa jedrsko elektrarno 😊
- Uporaba posebnih, višjih orodij sinhronizacije (semafor, monitor)
 - **Semafor:**
 - Najenostavnejši semafor (spinlock):
 - celoštevilaska spremenljivka (S)
 - Tri operacije:
 - Prireditve (npr. S:=1)
 - **Wait(S);**

Ta del je atomaren (pri zanki DA pa lahko pride do prekinitve)



- **Signal(S);** [S:=S+1]

- Uporaba pri problemu kritičnih odsekov
 - Več procesov ima isti semafor S (inic na 1)

```
repeat
    ...
    Wait(S);
    k.o.
    Signal(S);
    ...
until
```

- Uporaba pri drugih sinhronizacijskih problemih
 - Za usklajevanje procesov
 - U2 (k.o. procesa P2) se sme izvesti šele ko se je izvedel U1 (k.o. procesa P1) ($U1 < U2$)
 - **Za izpit:** imaš P1, P2 in P3 z U1, U2 in U3. Kako bi izvedli, da se najprej izvede U1, potem U2 in potem U3? Namig: potrebuješ 2 semaforja!!