

OS

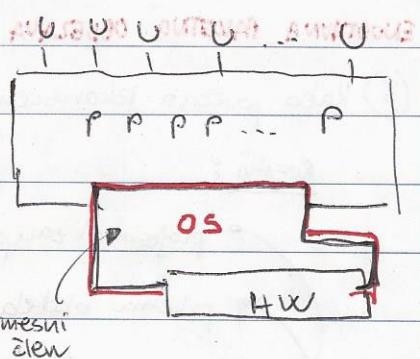
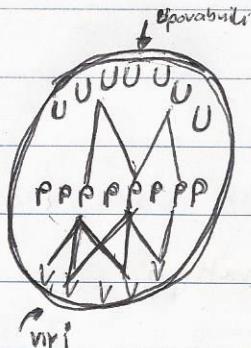
Kaj je OS?

Rac. sistem

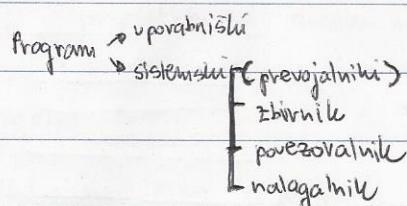
uporabniki (user)

uporabniki programi (applic. prog.)

strojna oprema (hardware)



OS nadzoruje vse to dogajanje



- OS:
- dodeljuje vire
 - rezervira konfliktor (\Rightarrow optimizacija)
 - nadzira vire in programe
 - olajšuje delo uporabnikov

program, ki je posrednik med uporabnikom in strojno opremo, ki sluša:

- (1) učinkovita raba strojne opreme
- (2) ustvarji prijazno okolje uporabniku

Zgodovina

Zgodnjii sistemi:

- veliki
- konzole
- programer = operator

postopek:

- napisal svoj program
- naložil program v pomnilnik
- napiše začetni naslov v FC
- in sproži izvajanje
- operira z dogajanjem
 - ↳ lahko prečine, odčita vsebine lokacij, popravi

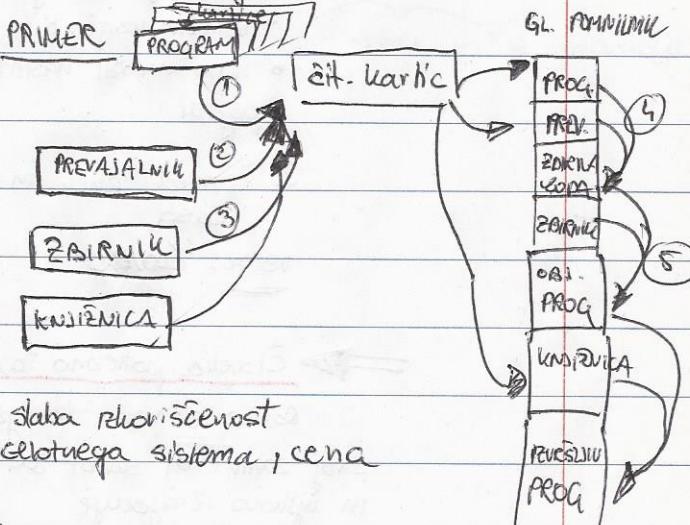
Kasnejši dodatna oprema:

- strojna oprema (čitalniki kartic, tiskalniki, ...)
- prog. oprema (zbirnik, nakagalnik, knjižnice, povezovalnik, gonilniki, prevajalniki)

\Rightarrow poenostavlja se programiranje

- zahtevnejša raba računalnika

PRIMER



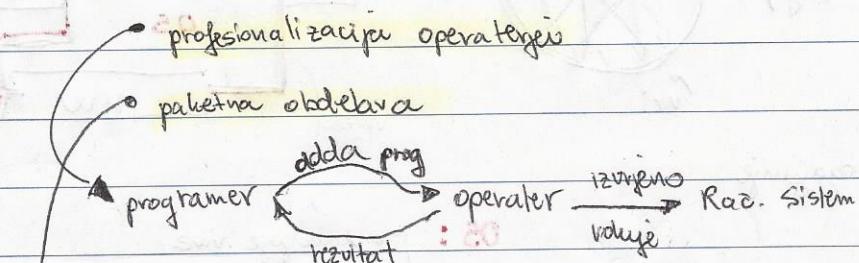
Nauč: slaba izkorisčenost
čelotnega sistema, cena

↳ ? Kako izboljšati izkorisčenost
rac. sistema?

ENOSTAVNA PAKETNA OBDELJAVA

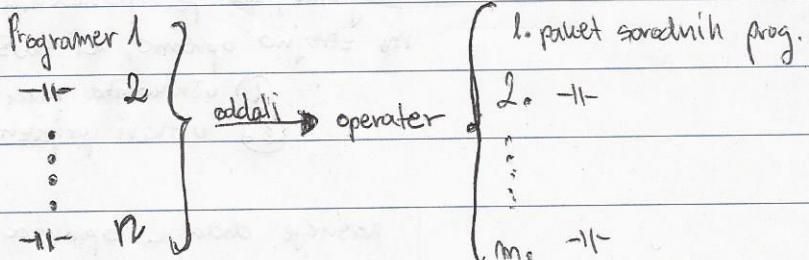
(?) Kako povečati izkoriscenost?

Razitvi:



(+) za krajša priprava prog. za zagovr

(-) ni bilo interakcije med programerjem in rač sistemom



! Izkoriscenost še vedno premajhna

• Razlogi:

- operator:
 - operira izvajanje pos. prog.
 - ugotavlja zakaj se je program ustavil
 - ustrezno reagirati
- nato naloži naslednji prog.
- pognal

Σ: med enim in drugim postopki preteče preteč
časa

vzrok: človek

➡ Človeka potrebitno zmanjšati

Raven z rač. ki ga upravlja

Stroj sam naj slabši za napredovanje in avtomatično nizanje programov/postrov in njihovo izvajanje

RESIDENTNI MONITOR (R.M.)

prvi preprosti OS

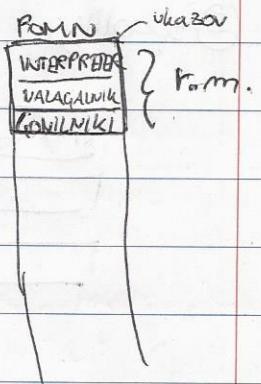
Nalaga: ob koncu posla avtomatično izvesti naslednji posel

Delovanje (načeloma):

↳ ob zagonu sistema naj se naloži R.M. v pomnilnik se začne in poskrbi za izvedbo prvega posla

Med izvajanjem posla R.M. ostane v pomn.

↳ bo se neli posel konča nadzor prevzame spet R.M.
in poskrbi za izvedbo naslednjega posla



Realizacija R.M.

(?) Kako naj R.M. izvede posel

Rешitev: Dodati je treba termilne kartice z ukazi namejenimi ravnemu R.M. -ju

latalniku

gornilniku

(+) izkorisčenost se je izboljšala

(-) ni interakcije še vedno

Presenečenje: Izkorisčenost še vedno ni zadovoljiva

Razlog: počasnost V/I naprav

Zakaj: Hitrost procesorja ... nekaj 10^6 ukazov/s

Hitrost V/I naprav ~ 17 kartic/s

Sčasoma hitrejše V/I naprave, toda pohitritev CPU še večja \Rightarrow razkorak se večji

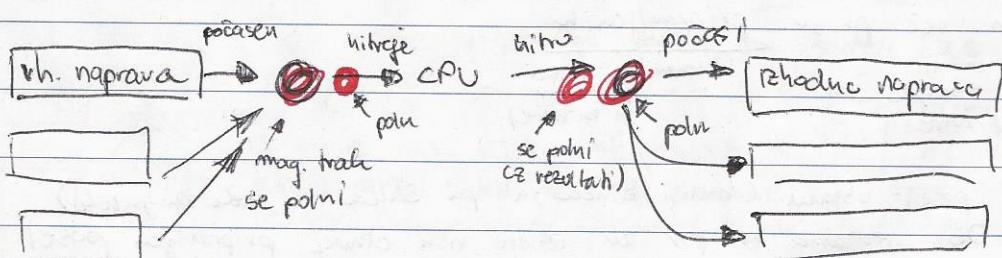
reda $10^5 \times$ počasnejši

(?) Kako to rešiti?

- organizacija delovanja

↳ 1. ločeni vhod/izhod

↳ 2. Spooling



medij z zaporednim
dostopom

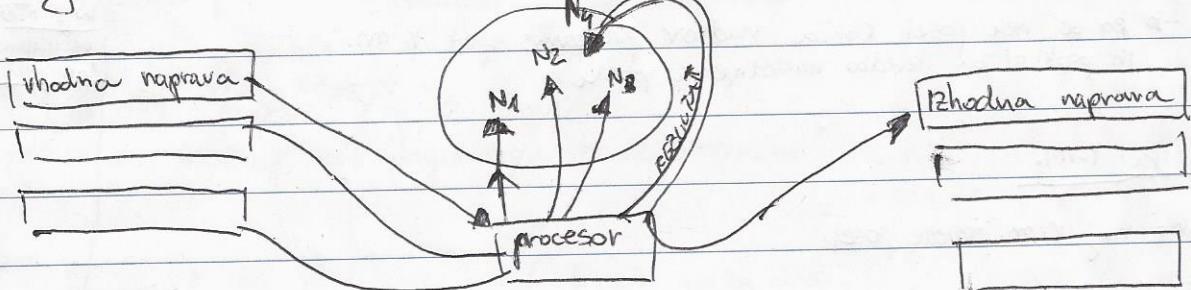
- + izkorisčenost je navesta
- ni interakcije

čas je vedno predolg (prevajanje magnetnega traku)

Pojav magnetnega diskusa !

- + - direktni dostop do podatkov

② spooling

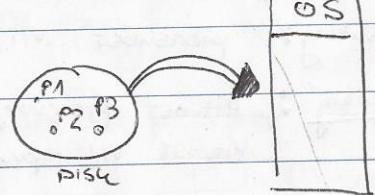


- + izkorisčenost dosti večja zaradi prekrivajočih V/I in pa računalnih op.

- ni interakcije

MULTIPROGRAMSKO IZVAJANJE

Spooling:

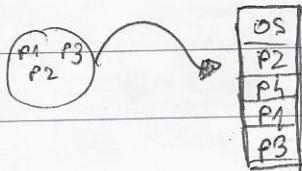


možno izbiranje naslednjega, ki se bo začel izvajati

nova naloga za OS \Rightarrow razvrščanje poslov
(job scheduling)

V gl. pomn. se je dalo naložiti
več poslov

Kako izkoristiti?
multiprogramirati



1) v pomn. več poslov pripravljenih

2) OS izbere enega in ga požene

\hookrightarrow menjata samo procesor
in bitne

3) če tekoči posel ustavi (bodisi konča, ali pa čaka na nečo drugače)
potem OS prevzame nadzor in izbere nek drugi pripravljen posel
in ga požene

\hookrightarrow preklapaja med posli (switch)

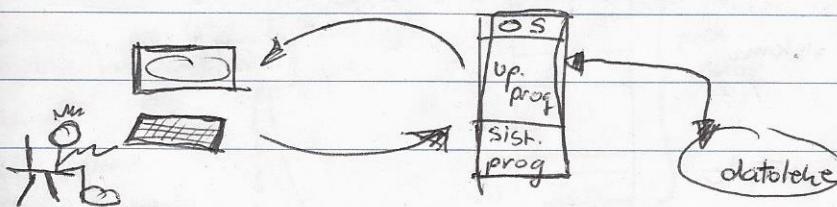
Nove naloge za OS

- RAZVRŠČANJE POSLOV
- UPRAVLJANJE S POMNILNIKOM
- Podpora multiprogramiranju (dodeljivanje procesorja)
- ZAŠČITA

INTERAKTIVNOST

Programer bi radil:

- tipkovnica → ukaz → V.S. → "takoj"
zaslon ← rezultati
- na enak način imeti na voljo tudi sistemski programi



Lastnik: bi moral nuditi:

- sprotni datotečni sistem
 - ↳ mnожice datotek (članik tipov)
 - ↳ ustrezno organiziranje
 - ↳ na značajem pomnilnika
posledica ⇒ visoka cena

Nevrnost; slaba izkoristljivost zaradi programerja ob višji ceni

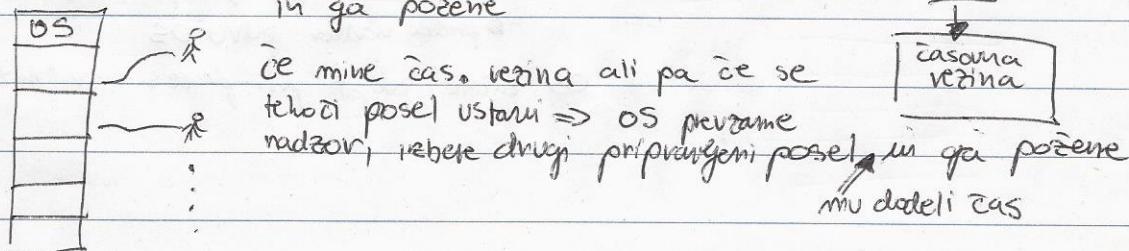
Časovno dodeljivanje (time sharing)

(vseopravljnost, multitasking)

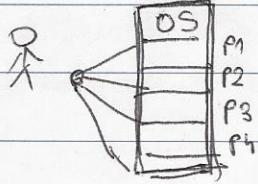
Kriterij: multiprogramiranje + dodeljivanje časa

- hkrati dela več uporabnikov, vsak s svojim poslom (vsaj 1)
- vsak posel se sme neprekiniti izvajati le amken čas

OS izbere npravjen posel, mu dodeli čas za izvajanje in ga počene



Proces (namesto posla)



↳ program v neki fazi svojega izvajanja

↳ proces izvajal ki ji recemo opravilo (task)

⇒ večopravilni (multitasking) OS

Nove naloge OS

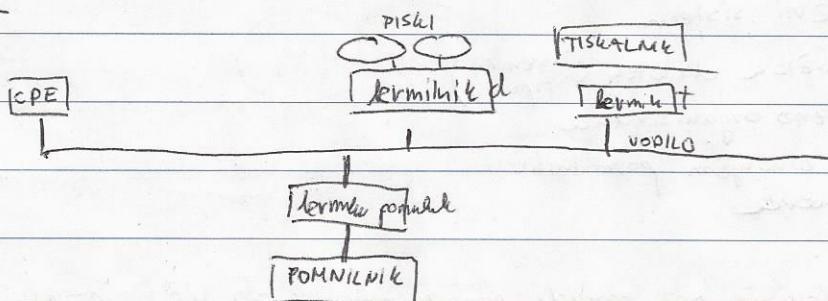
- datotečni sistem
- upravljanje z zunanjiimi pomnilniki
- upravljanje s internim pomnilnikom
- zaščita
- dodeljuvanje procesorja
- komunikacija med procesi, synchronizacija

~~ZDAVČNIK~~

ELEMENTI RAC. SISTEMA

- vsebina
 - delujejoči rac. sistema
 - V/I
 - pomnilnik
 - elementi strojne zaščite
 - sistemski klici

Zgradba



Krmilnik: nadzoruje ustrezno napravo ali več naprav.

ZAGON IN PELOVANJE

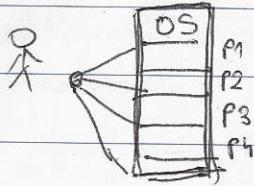
ZAGON

- ob vklpu se požene program, zagonski nalagatnik (bootstrap loader)

- ↳ majhen
- ↳ inicIALIZira konf. rac. sistema in jih pre... (pomn., kermihlik, ...)
- ↳ naloži OS (jedro) v pomnilnik
- ↳ predca nadzor podru OS

OS začne izvajati prvi proces (npr. "init")

Proces (namesto posla)



↳ program v neki fazi svojega izvajanja

naloge
Procesi izvajalci jih recemo opravilo (task)

⇒ večopravilni (multitasking) OS

Nove naloge OS

- datotečni sistem

- upravljanje z zunanjimi pomnilniki
- upravljanje s internim pomnilnikom
- zaščita
- dodeljivanje procesorju
- komunikacija med procesi, sinhronizacija

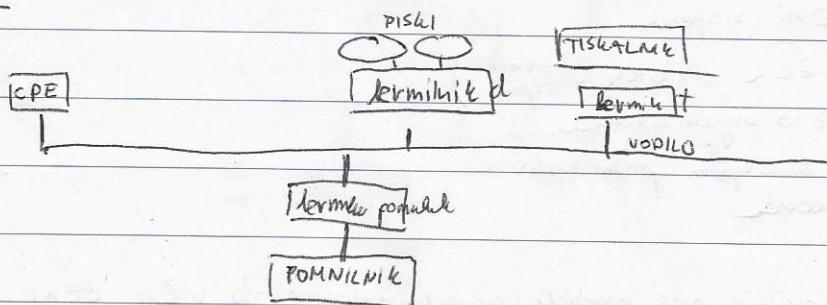
IZKUPNO

ELEMENTI RAC. SISTEMA

- vsebina

- delovanje rač. sistema
- V/I
- pomnilnik
- elementi strojne zaščite
- sistemski klici

Zgradba



Krmilnik: nadzoruje ustrezno napravo ali več naprav (monitor, tipkovnica, miš, zvočnik, ...)

ZAGON IN PELOVANJE

ZAGON

- ob vklopu se požene program, zagonski nalagalnik (bootstrap loader)

Prvotni

↳ inicIALIZIRa konf. rac. sistema in jih preveri (pomn., krmilnik, ...)

↳ naloži OS (jedro) v pomnilnik

↳ predaja nadzor jedru OS

OS začne izvajati prvi proces (npr. "init")

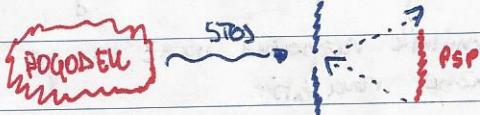
PEROVANJE: voden s prekinitvami

- ↳ OS čaka na dogodek in se nanje odziva
- ↳ o dogodku ga obvesti prekinutna zahteva

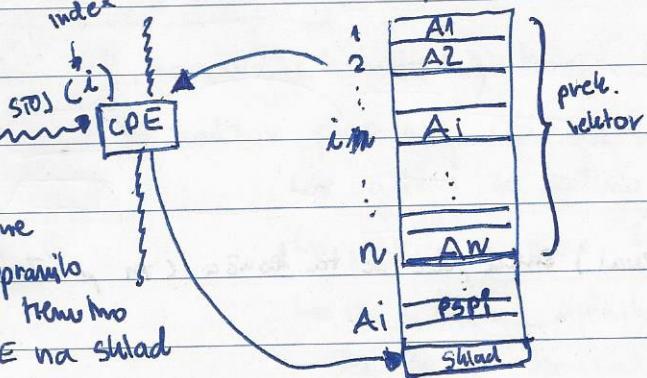
POTEC PREKINITEV

prekinitvena zahteva

- ↳ CPE prekine tekoče opravilo
- ↳ izvede PSP
- ↳ CPE nadaljuje prekinjeno delo



index velikosti



- CPE prekine tekoče opravilo in shrani trenutno stanje CPE na sklad

- uporabi indeksi da prebere naslov A_i (če naloži v PC)

- prične izvajati PSP (če konča prenese stanje CPE nazaj iz sklada)

- CPE nadaljuje prekinjeno delo

- v gl. pomm. je prek. velikost

vsebuje naslove $A_1 \dots A_n$

od $PSP_1 \dots PSP_n$

- hprava ve, da je PSP zanje /

i-ti komponenti prekinitvenega velikosti

- hprava zahteva pozornost, poslije

CPE nudi index 1

izveden izvedba

prog. izpraševanje (polling)

- vektorsko prekinitvuje

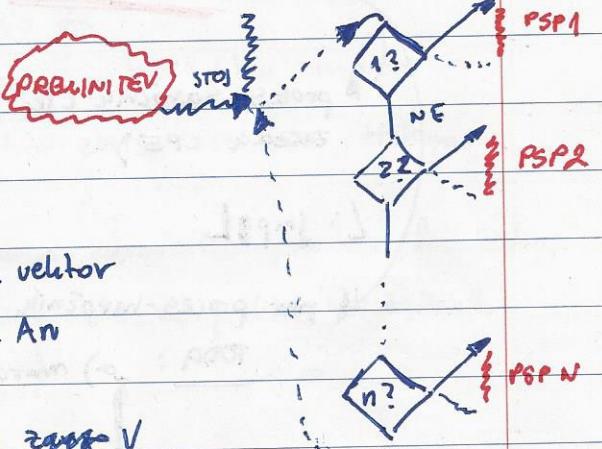
- CPE prekine tekoče opravilo

- prične izvajati skupni PSP

↳ vgoratori od teje je prišla prek. zahteva

- začne izvajati pristop po ~~CPE~~ PSP

- nadaljuje CPE



Greedene prekinitve

- (+) sistem boj odziven
- (-) zapletenost prioritete (ne bo vsake PSP prekiniten)

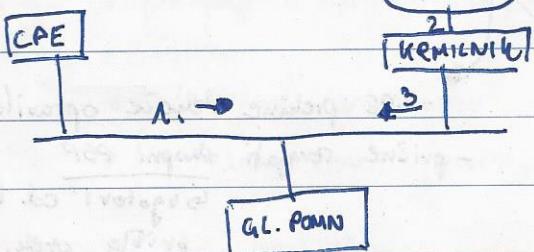
VHOD IN IZHOD

V/I sistem = V/I naprave + krmilniki + programi za V/I operacije

za prenos podatkov
stranski podatkov

Izbjava jih krmilniki
(podrobni opis, brane, ravesti, mrež
vklj.)

Potek V/I operacije



- 1) CPE vpiše podatke v reg. krmilnika
- 2) krmilnik razpozna ukaz in ga izvaja, izvajati
- 3) ko je operacija končana, krmilnik pošlje prek. zahavo v CPE

(?) Kaj v času med 1 in 3?

↳ sintroni V/I
↳ asinhroni V/I

proces-naročnik (če je zahteval) čaka, da se ta konča (in pri tem zaseda CPE)

L: jmp **L**

→ pa proces-naročnik nadaljuje svoje izvajanje

TODA: a) morda lahko proces-naročnik nadaljuje brez tečaj

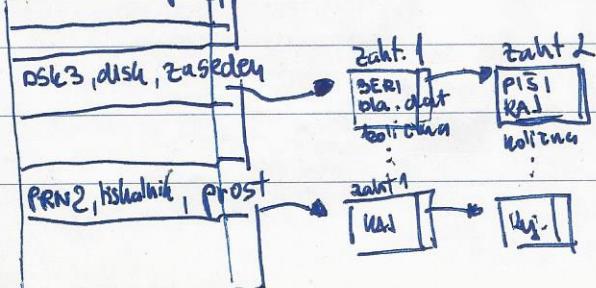
b) naročnik ne more nadaljevati (dolgo)

Npr. če naročnik potrebuje podatke takoj
ko jih je naročil
tedaj: takoj naročnik mora preiti v stanje zakejja
pri temer sprosti CPE

Asinhroni V/I omogoča več V/I op. hkrati

(?) kako z njimi upravlja OS

OS uporablja tabela stanj naprav, v tej tabeli, v komponenti so podatki o upravljanju



vseki V/I napravi, priпадa 1 komponenti

↳ ime
↳ tip
↳ stanje v katerem je naprava
↳ baze podatkov na seznam V/I zahit
naslovjem na to naprave

ZACETEK VII operacije

Proces naročnik želi VII op. Zato sprozi ustrezni sistemski klic.

Nadzor preverame OS, ker:

- ↳ ugotovi, za katere napravo gre
- ↳ pogleda v tabelo stanj naprav (v komponento te naprave)
- ↳ je naprava zasedena (njen seznam ni prazen)
- pokem
OS vključi novo zahtevo v ta seznam

sicer

OS vključi novo zahtevo v prazen seznam
in še poskrbi za zaitek te VII op.

(lahko te nalog preda gonilniku)

VONEC VII OPERACIJE

Ko VII naprava (gonilnik) konča svojo nalog, pošlje prek. zahtev CPE

↳ nadzor preverame OS

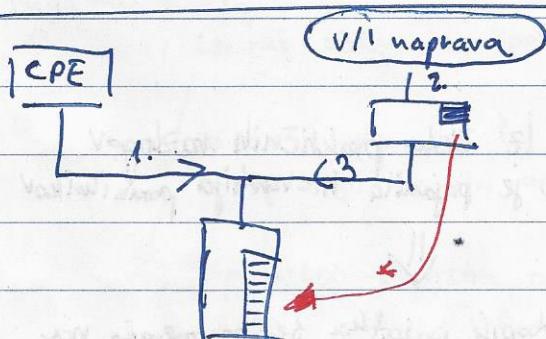
↳ ugotovi za kateno napravo gre

↳ pogleda v njeno komponento v tabeli stanj naprav

↳ iz seznama njenih zahtevov zbrise vozlišče, tistega
ki se je končal

↳ ob koncu se obresti njenega naročnika - proces ki tako

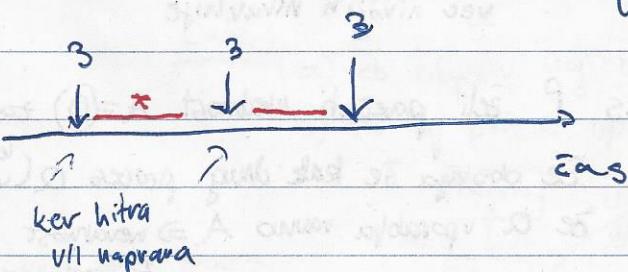
↳ izbere se naslednji VII zahtev in poskrbi za zaitek
njenega izvajanja (gonilnik)

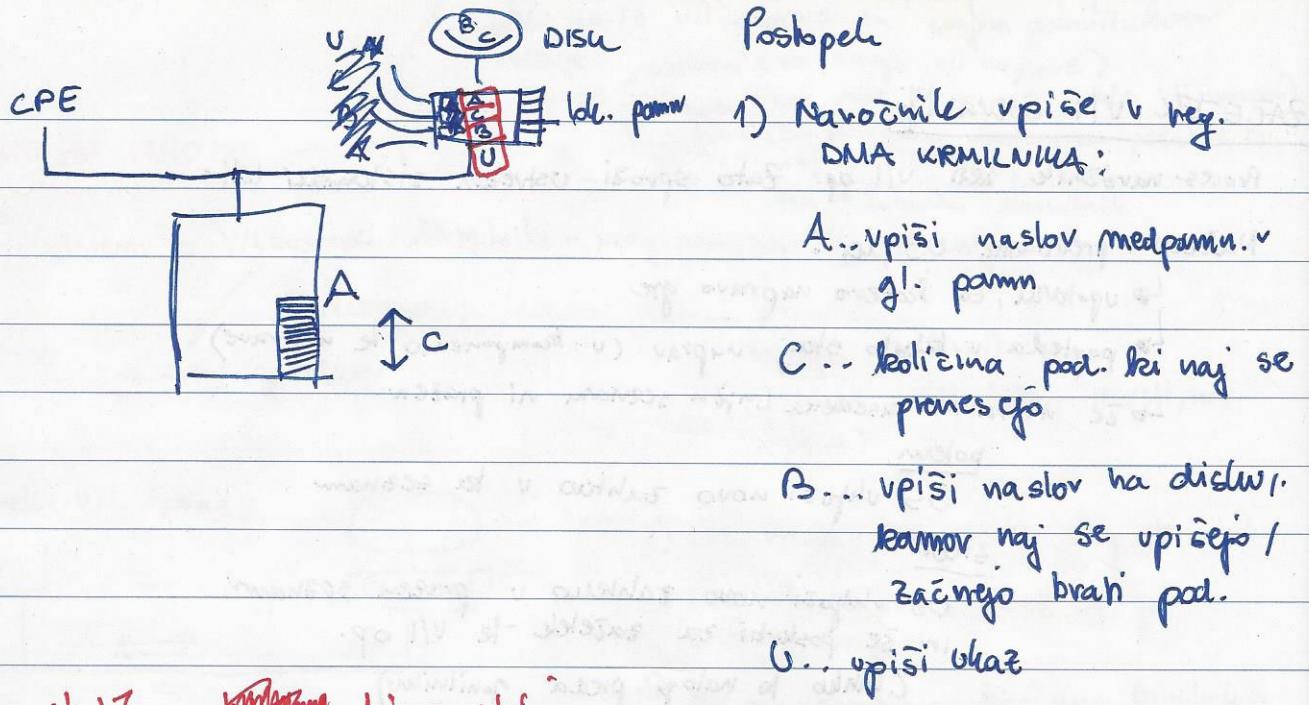


Ali je procesor s prenačajem
pod med gl. pomr. in loc. pomr. komunikacij
zelo obremenjen?

Če je VII naprava sposobna hitro
brati/pisati podatke asinhroni VII in
včinkovit

REŠITEV: DMA (direct memory access)

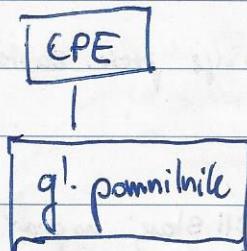




Pomnilniška hierarhija

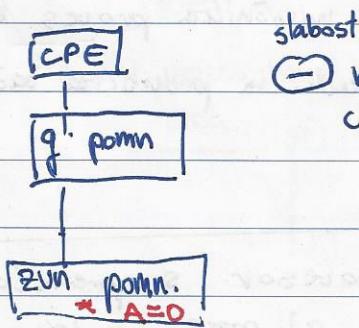
Tisti podatkov med CPE in gl. pomnilnikom

1.



→ gl. pomnilnik majhen
začasen

2. zunanjí pomnilniki



slabost
→ razkorak v hitrosti med
CPE in gl. pomnilnikom

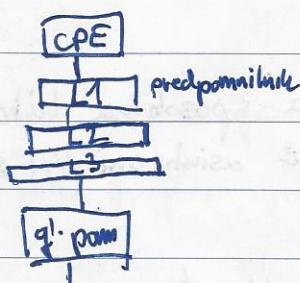
povezlek

Σ: Iz čisto praktičnih razlogov
se je pojavila hierarhija pomnilnikov

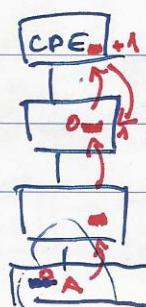


Kopije podatkov se pojavljajo na
več nivojih hierarhije

3.



Primer: Proces P želi poravnati vrednost $A=0$ za 1 *



če obstaja še kakš drug proces Q (ker imam
že Q uporablja ravno A ⇒ nevarnost os
za nekonsistencijo pod.)

prekinitev

Σ : OS postane črtor 2 zahtev

* zahteva po izkoristku in interaktivnosti je vodila do večopravilnosti

* prekinitev = zahtev so vodile do hierarhijske pomn.

večopravilnost + hierarhijska pomn. prineseta nevarnosti

\Rightarrow zaščita

Elementi strojne zaščite

Zaščita naj nudi OS

Toda pri tem potrebuje nekaj osnovnih mehanizmov, ki jih nudi

strojna oprema:

↳ vsaj dvojni način delovanja nač sistema

↳ privilegirani ukazi

↳ zaščita pomnilnika

↳ zaščita procesorja

Način delovanja

Nekateri ukazi so potencialno škodljivi za druge procese

Taki ukazi naj bodo privilegirani; tj. jih bo lahko izvedel samo OS

Implementacija:

↳ rac sistem naj pozna (usoj) 2 načina izvajanja

↳ sistemski (S) ~~privilegirani ukazi~~
v katerem lahko teče le OS in določeni sistemski programi

↳ HVR nudi bit N (mode)

ki opisuje trenutni način

↳ uporabniški (U), v katerem tečejo uporabniški programi

↳ poskrbeti je treba, da bo

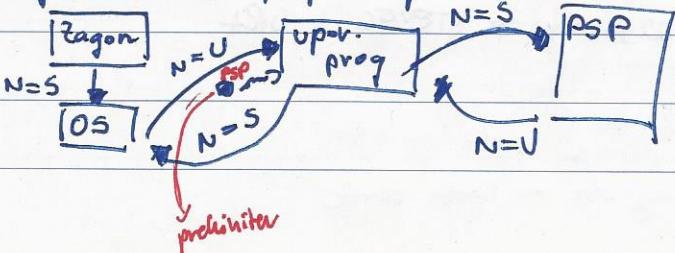
$N=U$ natanko takrat ko teče

uporabniški program

Zato:

a) ob zagonu naj bo $N=S$ sistemski, da se uvelodi OS, ...

b) OS naj požene upr. program pri $N=U$



↳ če upr. program proces posluša izvesti privilegiran ukaz, se sproži prest

ZAŠČITA VII

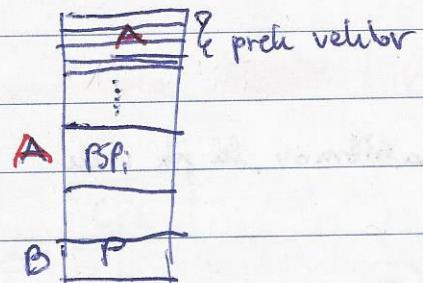
VII akciivnosti morajo biti zaščitene pred upo.

⇒ VII ukazi morajo biti privilegirani

ZAŠČITA GL. POMN.

Primer: proces p (uporabniški)

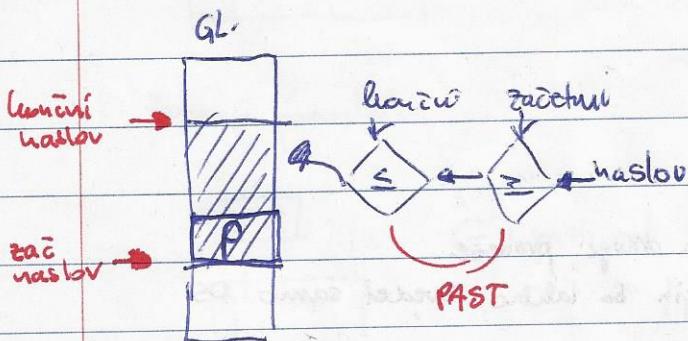
če bi p smel venadzorovano sprememiti prek vektor



Primer 2

če bi p smel venadzorovano spremeni spv PSP

glej polp [32]



ZAŠČITA PROCESORJA

CPE zaščiti pred upo. programom/procesami, da si ga ta ne more prisvojiti

IDEJA: v strojnih opremi je časovnik, ki periodično posilja zahteve po piknikih, ob vsaki prehodu se sporoči vel PSP, kar se oddosi dli ko temu procesu še dodelil CPE in te mu bo zmanjšal prioritet

$$\text{časovnik} = \text{STVEC} + \text{VFT}$$

sistem z dodavanjem osa

A. METODA ZA DODAVANJE

Ponavljaj

OS izbere v po-proces P, ki naj dobil CPU

• opravi vzdrezvalna dela:

↳ določi stevec

↳ nastavi ga

↳ n se leži

P

↳ koe časovne, dokler časovnik ne sproži prek zahoda

: dokler (P ustavlja) V (človec zas. režive)

OS opravi se kakšno vzdrezvalno delo

Dokler true

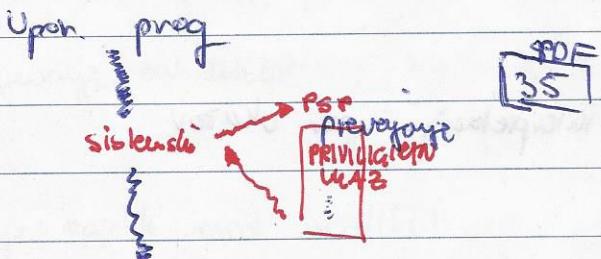
SISTEMSKI UKAZI

Obstaja veliko privilegiranih ukazov

(?) Kako naj vpo-prog. doseže, da se tak ukaz priigne izvajati?

ODG vpo-program zaprosi OS naj ta v njegovem imenu izvede tak ukaz zanj \Rightarrow sistemski ukaz

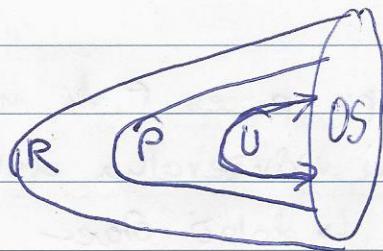
Vsek sistemski ukaz je ukaz, ki ga navedemo v prog.



ELEMENTI OP. SISTEMA

Pogled na OS:

- ↳ UPORABNIK (U)
- ↳ PROGRAMER (P)
- ↳ razvijalec (R)



uporabniški pogled:

- ↳ komponente OS
- ↳ storitve, ki jih nudi OS

prog. pogled:

- ↳ sistemski letci
- ↳ sistemski programi

razvijal. pogled

- ↳ zgradba OS
- ↳ tehnike in orodja za snovanje
- ↳ implementacija OS

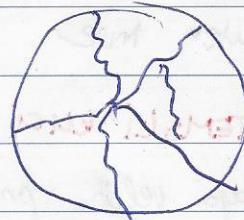
Komponente OS

Sistem OS ima več komponent

⇒ večja preglednost, obvladljivost, zanesljivost

- značilne komponente

- ↳ upravlja procesov
- ↳ upravlja gl. pomnilnik
- ↳ upravlja datoteke
- ↳ upravlja V/I sistema
- ↳ upravlja zem. pomnilnik
- ↳ delo v omrežju
- ↳ zaščita
- ↳ varnost
- ↳ sistem za interpretacijo upor. ukazov



Upravljanje procesov

Proces = vstavljen, teče, čaka, končan

rabi vire

komunicira ali pa ne

Naloge OS

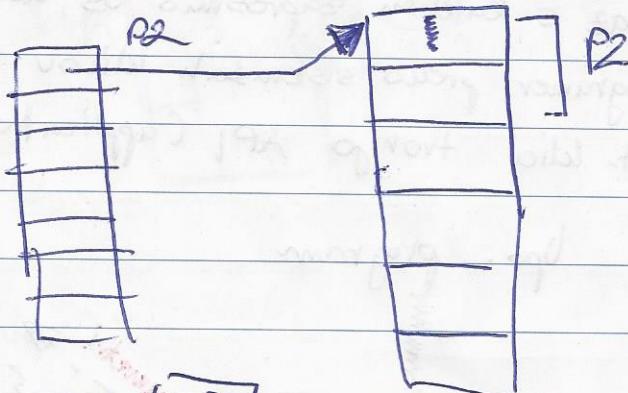
- ustvarjanje proces
- uničenje -||-
- nadzor -||-
- vstavljanje
- komunikacija med procesi
- rezervanje iz mrežnih zank

Upr. gl. pomnilnika

Naloge OS

- izbirajo kandidatov za sproščen del pomnilnika
- dodeljuje pomnilka in sproščanje
- nadzor: kdo ima kaj

FIZ. POM



Sistem za upravljanje z datotekami

pdj. [32]

datoteka = log. enota podatkov na tradicionalnem pomnilku

Naloge:

↳ ustvarjanje/brisanje datotek

↳ -/- dodeljanje

↳ prenesanje, kopiranje, -

↳ preslikovanje med datotekos in resničnim zvi. pomnilku

Upravljanje zvi. pom.

→ upravljanje s praznim prostorom

→ dodeljanje -/-

→ optimalno izpoljevanje zahtevkov

Dels v omrežju

zakrivlji/glojje razlik med vozilci

klicanje oddaljenih funkcij (RPC) RMI

- synchronizacija

Zaščita

Varnost

Interpretacija upor. ukazov

Ostali sist. prog

↳ hubski program, mu dodeljuje gl. pomn., VII naprave, CPE

→ smotri dodeljevanje virtov

SISTEMSKI KLICI

Vklj. s katerim zaprosimo OS, da pomaga (= izvede privilegirani vklj.)
 Programer preko sistemskih vklj. izkorisca OS
 Sist. vklj. tvorijo API (application programming interface)

Upo. program



Prenos parametrov pri sist. vklj. na zbirnem nivoju

↳ preko dejavnega reg. CPE

+ hitrost

= omogočnost prostorice

preko gl. pom. v reg. CPE se
vpise nastav, ker se začenjajo parametri

+ nedomejivi parametri

- hitrost

preko sklada

Vsih prog. jezikov nudijo funkcije sistemskih vklj. preko vklj. funkcij

Primer: C)

```
#include <fcntl.h>
```

```
#include <unistd.h>
```

Main ()

{

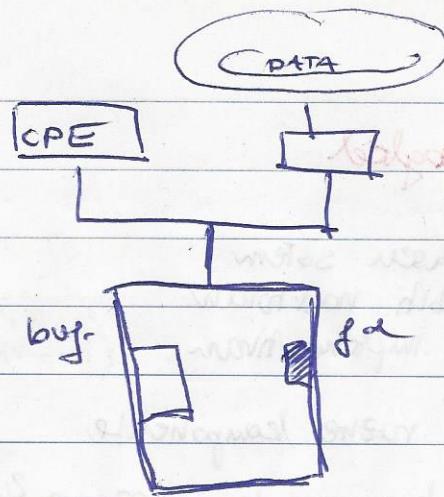
```
char buf[1024];  
int fd;
```

```
ssize_t nread;
```

```
fd = open("data", O_RDONLY);
```

```
nread = read(fd, buf, 1024);  
close(fd)
```

}



Sistemski klici so pogost

Posix standard

Shupine:

↳ upravlja procesor: kreirati/uničevati, matogati programa, končati program (exit), prednji konec (abort)

↳ -i- datoteke:

↳ -i- naprav

↳ uveljavljati in dostava info

↳ komunikacije

glej 41

↳ model

↳ shupni pomnilnik:

zaprošuje za shup. pomm., vracaže
shupnega pomm.

↳ prenosanje sporočil: glej 42

Primeri 42

```
pid = fork() -- ustvari nov proces, vrnji njegovo id
```

```
waitpid(a, &b, c) --
```

```
exec(...)
```

```
exit(status)
```

```
p = getpid()
```

```
r = brk(n)
```

napaka v
skripti

enzivijalni pogled

OS - kompleksen sistem
mora biti načrtovan
pažljivo implementiran

- vsebuje razne komponente

Rake so te komponente povezane?

Vrake za težave začetnih OS je bila vzhova enovita zgradba

Zgradba OS

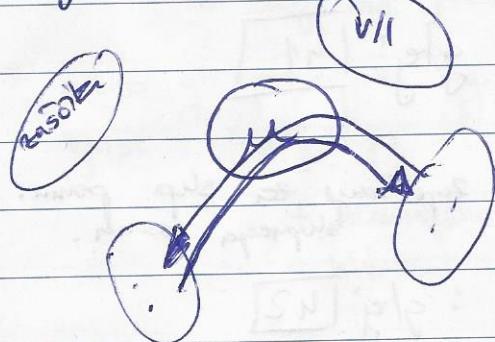
↳ enovita: zaradi neenostavnega dopoljevanja je bil OS ogromen program, ki je bil težko razširjiti
⇒ nestabilen, nizanesljiv

razenjena zgradba

← upo vnesek

← strojna oprema

μ jedro



μ = osnovno upr. program
in gl. posn.

- osnovna komunikacija

PROCESI

Motivacija:

v št. včasih veliko dogajanja

Skupne značilnosti teh dogajanj (aktivnosti)

↳ v usmerju nek program

↳ v potrebuje rač. vire (CPE, gl. pomo, ...)

↳ v ima svoj izvajevalski cikel (se rodi, teče, čaka, se konča)

↳ program lahko usmerja več razl. dejavnosti

Te aktivnosti so torej programi v raznih fazah (stanjih) izvajanje
rečemo jim procesi

Program: je pasivna entiteta: vsebina neke datoteke

Proces: je več kot program: poleg programa zajema tudi trenutno stanje (situacijo)
nekaterih drugih entitet, ki se spremenijo med izvajanjem programa

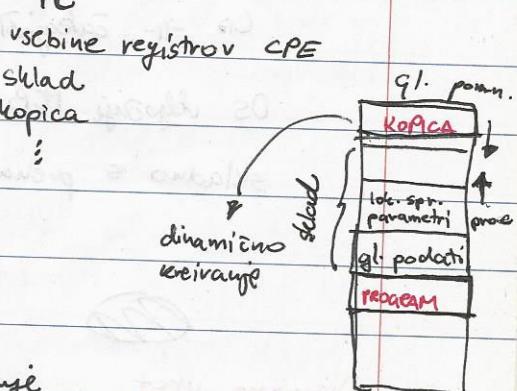
Proces := statični del + dinamični del

Torej: isti program lahko povodi različne procese.

Vsi tečejo po tem programu, razlikujejo pa se
v dinamičnih delih

Dinamični del (oz. podatki) so morski:

gl. pomo.
nadzorni blok (PCB)



Stanja procesa:

V času svojega obstoja vsake proces prehaja iz stanja v stanje

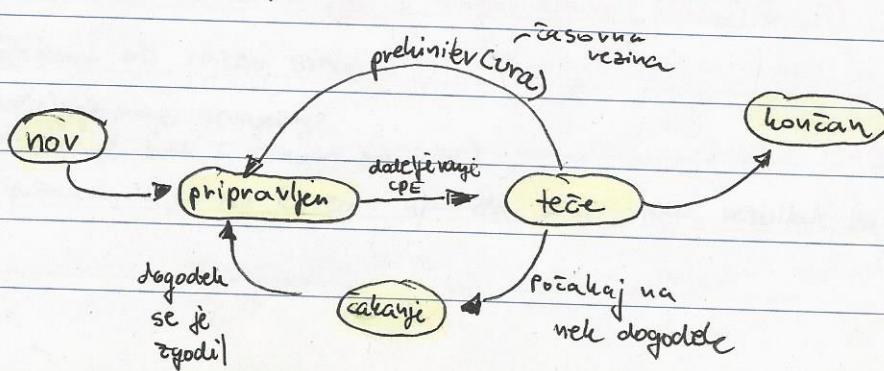
↳ nov ... os ga vzpostavlja

↳ pripravljen ... ce čaka na procesor

↳ teče ... ce CPE izvaja njegove ukaze

↳ čaka ... ce čaka na nek dogodek

↳ končan ...



Predpostavka:

procesor

ce krejemo 1 proces

Nadzorni blok

PCB process control block

V OS je proces predstavljen s svojim nadzornim blokom (PCB).
Ta vsebuje vrsto dinamičnih (in statičnih) podatkov o procesu.

PCB str. 49

- stanje procesa	- računalniški podatki
- podatki za razvrščanje	↳ id procesa
- podatki o procesorju	↳ št. nadzor
- podatki o skladu	↳ čas/datum kreiranja
- podatki o podatkovnem delu	- ostali podatki
- II- za upravljanje spomelatom	↳ id lastnika
- II- podatki o V/I	↳ id. sinov tega procesa

Vrste:

PCB vsebuje karalce za povezovanje v razne vrste.

↳ vrsta vseh procesov (ready queue) (job que) - s PCB-ji vsebuje trenutno obstoječih procesov

↳ -II- pripravljenih (ready queue) -II- čakajo na procesor

↳ -II- čakajočih na V/I napravo -II- čakajo na dano napravo

OS vtežuje PCB premo procesa v razne vrste ga iz njih izloča

skladno s prehajanjem procesa iz stanja v stanje

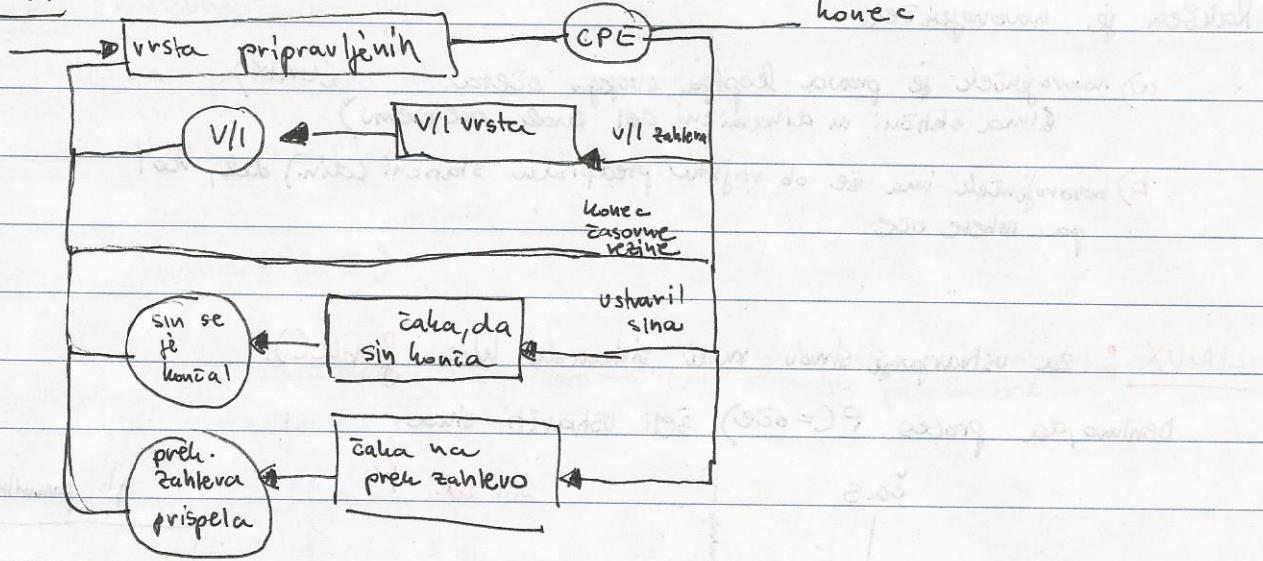
glej slika 49
(slika 40) =

Diagram vrst

prehajanje procesov (oz. upihovih PCB-jev) iz

vrste v vrsto opisuje diagram vrst

slika 4)

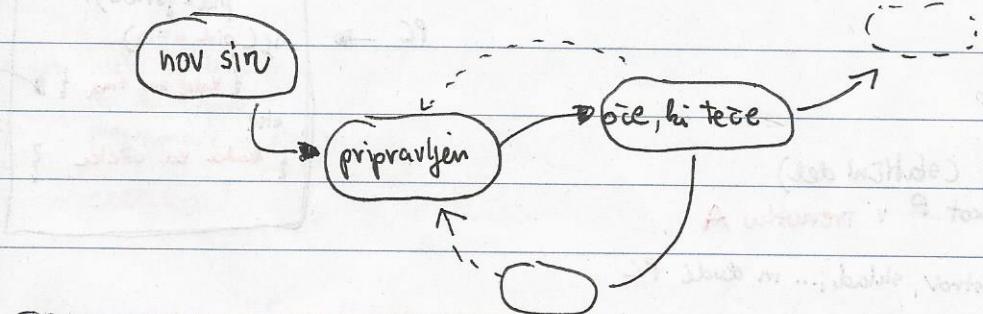


OPERACIJE NAP PROCESI

- ustvarjanje / končanje procesov
- razvrščanje procesov

USTVARJANJE PROCESA

Proces, ki teče lahko ustvari nov proces; sin



(?) Kako oče ustvari sina?

Odg: s sistemskim delcem (unix: fork())

Kdo dodeli sinu potrebne računske vire (prostor, čas, ...)?
možnosti:

- dodeli mu jih kar OS
- priskrbi mu jih kar oče
- oče odstopi del svojih virov (resource partitioning)
- oče si vire deli s svojim sinom (tekmje zanje)

Kaj se zgodi z očetom ob rojstvu sina?

- oče nadaljuje svoje izvajanje
(in tekmje tudi s sinom za vire)
- oče počaka, da se sin konča. npr. da vrne velice rezultate, ki jih oče nabi

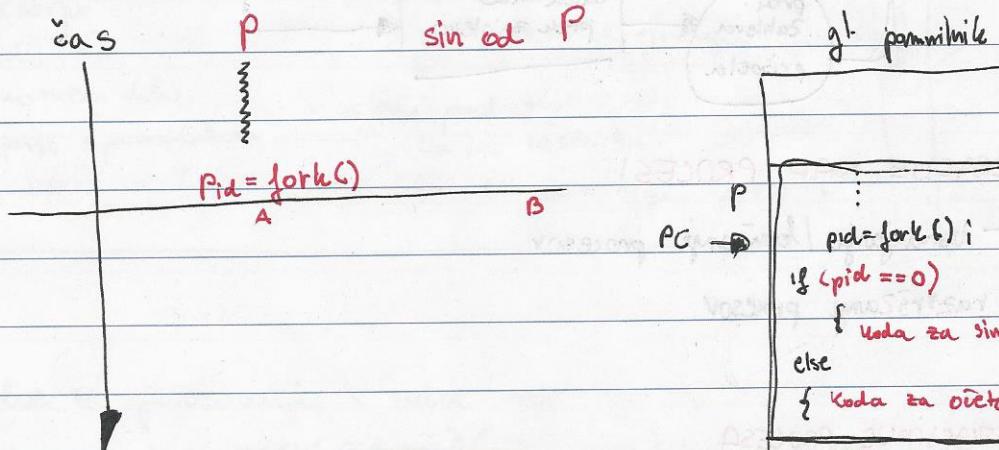
Kakošen je novorjeniček

a) novorjeniček je prava kopija svojega očeta (UNIX)
(ima stični in dinamični del enake očetovemu)

b) novorjeniček ima že ob rojstvu predpisani stični (din.) del, kot ga izbere oče

LINUX: za ustvarjanje sinov nudi sistemski funkciji `fork()`.

Definimo, da proces P (=oče) želi ustvariti sina



Ko se sin rodil, postane pripravljen (B)

Sin od P je identičen očetu B

Zato ima sin od P:

- ↳ enak program kot P (stični del)
- ↳ enak dinamični del kot P v trenutku A

↳ vsebine registrrov, sklad,... in sledi PC

Med izvajanjem forkja ji OS dodeli novorjeničku nov identifikator (pid), ki je > 0 .

Ta pid se upiše v spremenljivko pid očeta P

Zdaj je sin od P pripravljen

Ko sin od P dobi procesor, steče od lokacije dalje, ker mora končati (sinov) PC

Ker nebo redel pid=fork(); zato ne bo še svojega sina in tudi ne bo

v svoj pid vpisal nicesar. Ob rojstvu sina je os nastavil pid na 0

* Torej oče ima vrednost pid = "identifikator sina", sin pa ima pid = 0

Na podlagi te razlike oče in pa sin ustvari, ali je oče, ali sin

EDINA RAZLICA

Primer : Enostavna lupina

lupina prebere vpo. ukaz (s parametri),
ustvari sina (lupine), ki izvede ta ukaz

while (TRUE)

```
{
    beri_ukaz (ukaz, parametri); // lupina s terminala prebere ukaz
    p=fork(); // in parametre
}
```

if (p == 0)

```
{
    execve (ukaz, parametri, 0);
```

{

else

{

waitpid (p, &status, 0)

}

testirati če
& nreč
parametri
potom ni treba

→ ukaz parametri ↗
↗

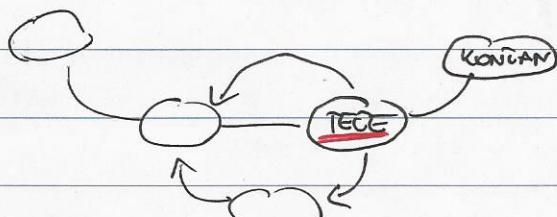
npr. sistem . klic za
vzhod na čakajida
se sin konča
waitpid (...)

→ npr. sistemski klic
za zamenjavo
programa (od sina)
exec (...)

KONČANJE PROCESA

Proces zahteva svoj konec s sistemskim klicem

UNIX: EXIT



OS: - procesu odstranimo vse virje

- sprosti sistemski pod. strukturi, kjer je evidenca o tem procesu (npr. PCB)

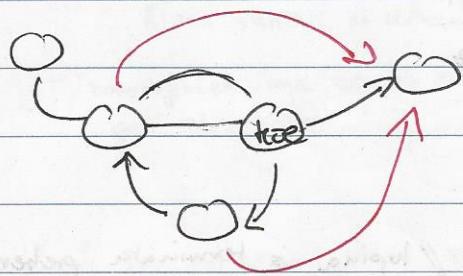
UNIX: - OS pošle očetu signal, da se je sin končal
- če oče na konec sina že zakaj, lahko os

upiše status končanca v &status
opis Sinovega končca

- če oče se ne zakaj, OS shrani opis
končca v svojo sistemsko sp.

Potek je sin ZOMB

Pričilno končanje drugega procesa (ubijava)



Proces ki teče lahko zahteva, da se konča v drug proces

Razlogi:

- ↳ sin ni več potreben
- ↳ sin prekoračil dovoljenega količina nekega virja
- ↳ če oče želi končati in ima sinov, pa OS ne dovoli obstoja sinov

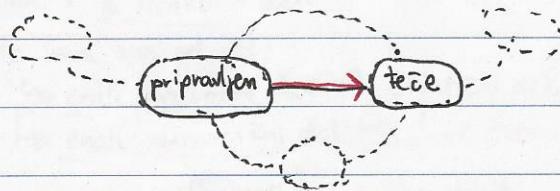
UNIX: **Init** (pid=1) posvoji tales sirotov

RAZVRŠČANJE

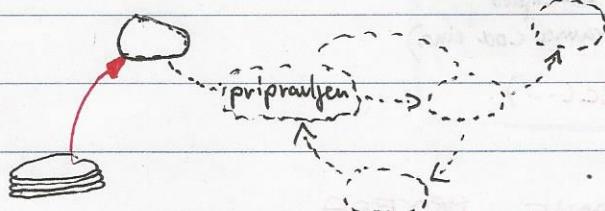
3 vrste

- razvrščanje na procesorju (short-term)
- razvrščanje "postav" (long-term)
- menjovanje (swapping)

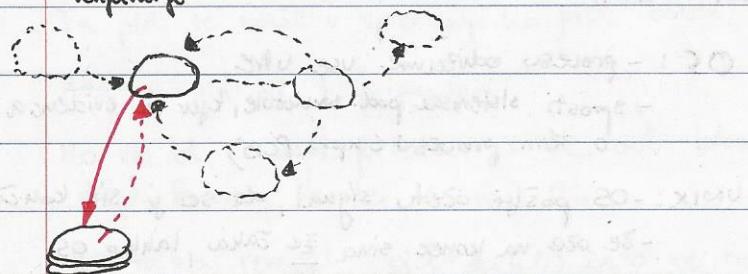
Razvrščanje na procesorju



Razvrščanje postav



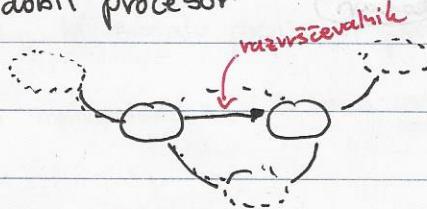
Menjovanje



Razvrščanje na procesorju

CPU - scheduling

Razvrščevalnik (scheduler) je del OS, ki izbere enega od pripravljenih procesov, ki bo naslednji dobil procesor.

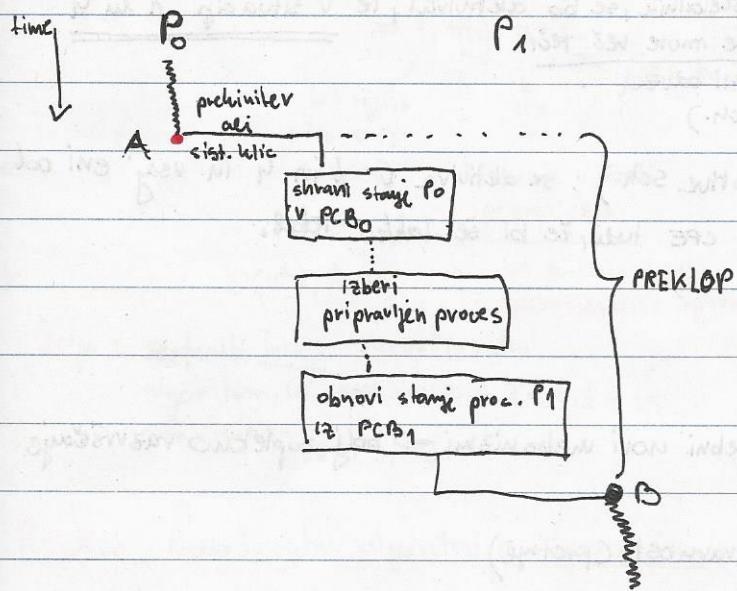


Značilnosti:

- razvrščevalnik se kljice relativno pogosto
- ⇒ zato mora biti čim hitrejš!

Preklop: (menjava obroča) → context switching

Razvrščevalnik dodeli CPE izbranemu pripravljenemu procesu, pravimo, da je CPE zamenjal obroč, v katerem izvaja ukaze.



Pri preklopu sodelujejo dva dela procesa

- razvrščevalnik (scheduler) izbere pripravljen proces
- dodeljevalnik (dispatcher) izbrano dejansko dodeli CPE

Preklopni čas

(= čas med A in B)

context switch time

je za uporabnika zista rezija, v tem času OS ne počne nič neposredno koristnega za proces

Ker je preklopljajenje pogosto je važno, da velikega pomena, da je t_{AB} čimkratiji

$$\frac{t_{\text{slice}}}{t_{\text{slice}} + t_{AB}} = \frac{20}{20+10} = 66\%$$

Kaj vpliva na t_{AB}

↳ hitrost pomnilnika (ki je so PCB)

↳ št. reg. v procesorju

↳ arhitektura (nabor ukazov)

↳ zgradba CPE

↳ podatki o pomnilniku (način upravljanja pamn.)

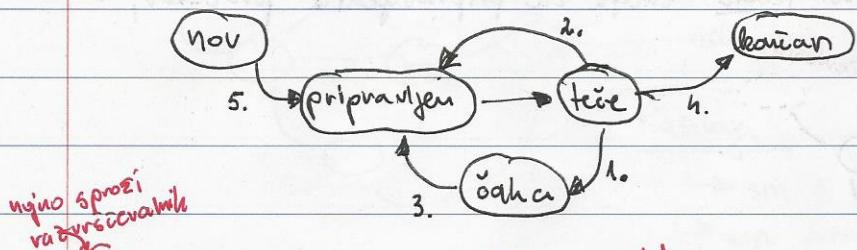
↳ algoritmi za izbiranje naslednika (razvrščevalnik)

↳ ostala občasna opravilca

eden od razlogov za pojavi niti → Preklop je ozko grlo v OS

Razvrščanje z brez odvzemanjem

(?) Kdaj je smiselno in kdaj celo nujno poticati razvrščevalnike



1.) Proses je onemogočen CPE je prost!

2. Proses je porabil svojo rezino. Bo nadaljeval on ali kdo drug

3. Proses gre v pripravljene, ali že takoj zaslvi CPE?

4. Proses ki je končal. CPE je prost!

5. Prispel je nov proces. Ali že takoj zaslvi CPE?

Dva pristopa k razvrščanju na procesorju

↳ razvrščanje brez odvzemanjem: razvrščevalnik se bo aktiviral, le v situaciji 1 in 4 proces zapusti CPE, le, če me more več teči.
Torej se temu procesu CPE ni odvezel
(non preemptive sch.)

razvrščanje z odvzemjem (preemptive sch.): se aktivira v 1 in 4 in vsaj eni od ostalih (2,3,5). Proses ^{lahko} zapusti CPE tudi, če bi se lahko tekel.

Razvrščanje z odvzemjem

⊕ fleksibilnost, odzivost

⊖ odpira nekatere nevarnosti \Rightarrow potrebi novi mehanizmi \Rightarrow bolj zapleteno razvrščanje

Razvrščanje z odvzemanjem odpina nekatere nevarnosti (pretežje)

Primer: Danci je tabelo A [1 ... n] števil

Program P₁: V komponento polci t implementira.

Program P₂: Seštej vse komponente polci A, in repsi rezultat

Scenarij:

1) zaženemo P₁ in P₂ in ustvarita se 2 procesa P₁, P₂

2) razvrščevalnik dodeli CPE procesu P₁

3) se preden P₁ poveča vse komponente polci A ga časovniki prehine in CPE se mu odvzame (razvrščanje z odvzemjem)

4) razvrščevalnik dodeli CPE procesu P₂

5) P₂ sešteje komponente polci A, ki pa še ni bilo do konca obdelano s programom P₁ \Rightarrow P₂ vrne napaden rezultat

Vir težave: P_2 je smel uporabiti polje A, čeprav P_1 ni končal spremiščanja polja A

Torej: Polje A bi moralo ostati nekako zaklenjeno po tem ko je bil P_1 prekuhan
tako, da ga P_2 ne bi mogel uporabiti

Ali pa: nekaj bi moralo prepričiti procesu P_2 , da sploh steče, preden se je končal P_1

Naučni potrebeni bodo mehanizmi za časovno usklajen dostop do A.

! - synchronizacija procesov

ALGORITMI ZA RAZVRŠČANJE NA PROCESORJU

Vpliva na: [prehodni čas

"zadovoljstvo" uporabnikov

Kako vrednotiti njihovo kakovost?

Možna meritva (proljet časa):

a) izkorisčenost procesorja

b) propustnost sistema $= \frac{\# \text{končanih-procesov}}{\text{časno obdobje}}$

c) čas obdelave = čas od rojstva procesorja, do smrti

$$= t_1 + t_2 + t_3 + t_4$$

čas rojstva
čas starta
(potreben)

čakanje v vrsti
pripravljenih

$\Theta(n^k)$, $k < ..$
na to lahko daber
razvrščevalnik upriva

d) čakalni čas t_2

e) odzivni čas = čas od rojstva procesorja, do njenega prvega
izhoda
(vražno, za interaktivnost)

f) pravilčnost = V proces naj dobri
ustrezen čas CPE

Zanimajo nas povprečne vrednosti
npr. \bar{t}_2

Želja: seščaviti hitri razvrščevalni
algoritem, ki maksimizira $\{a_i, b_i, f_i\}$ in
minimizira $\{c_i, d_i, e_i\}$

Realnost: hitri algoritmi vracajo te

suboptimalne rešitve

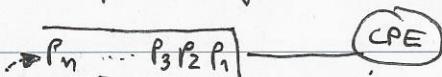
Kar je še redno težko, zato se
osredotočimo na 1 samo mero
 (t_2, \bar{t}_2)

Posledica: Razvrščevalni algoritmi so heuristični

Alg: Prej pride prvi pmeje (FCFS)

Ideja: Kdor prvi zahteva CPE, ga prvi doli (brez odzemanja)

Implementacija: pripravljeni procesi so v neli vrsti (FIFO)



Proces bo izgubil CPE samo, če je končal, ali je
blokiran

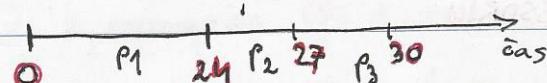
"Analiza": \bar{T}_2

Primer 1: Procesi P_1, P_2, P_3 zahtevajo $24, 3, 3$

Denimo, da so vstopili v vrsto:



Gantov diagramm:

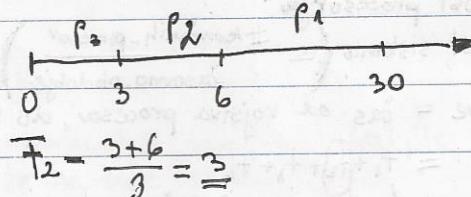
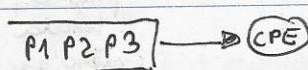


Sklep: \bar{T}_2 lahko zelo varira
 \bar{T}_2 v splošnem ni minimalen

$$\bar{T}_2 = \frac{0+24+27}{3} = 17$$

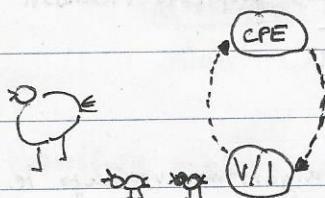
Primer 2: Enaki procesi

Denimo da vstopijo:



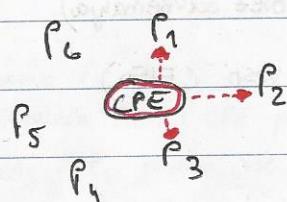
$$\bar{T}_2 = \frac{3+6+30}{3} = 13$$

Pojav konvoja



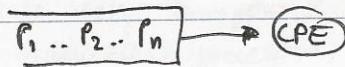
Alg. Krožno razvrščanje (RR)^{round robin}

Ideja: vrsta pripravljenih si predstavljamo kot krožni seznam. Proses dobri CPE za n časovno rezino. Ob izteku čas. rečine radi pa ob onemogočevju ali končajuji CPE dodeli naslednjemu na seznamu



Razvrščanje z odvzemajem

Implementacija:



"Analiza"

n procesov
 r dolžina čas. rezine
 p prečaknji čas

Proces bo ponovno dobit CPE čez $\leq (n-1)(r+p)$ časa

če bi proces idealno rabil + časa, bo
v resnici končal žete po

$$++ \left(\frac{r}{r}-1\right) \cdot (n-1)(r+p)$$

-če je r premajhen se relativno
preči časa trabi za prečakanje
če je r prevelike to občuti upr. uporabnih
(interaktivnost)

Alg. Loto

Ideja: če proces dobi "dovolilnice"
dovolilnica ima vpisano ime procesov

Veliko razvrščevalnih alg. uporablja paper priorte

Ideja: vsak proces ima neko prioriteto. CPE se dodeli procesu, ki ima danes najvišjo prioriteto
Prioriteta so lahko stalne (stacionarne) ali spremenljive (dinamične)

Razvrščanje, ki uporablja prioritete je lahko z odzemanjem / brez.

Npr. če med pripravljenimi vstopi proces Q, ki ima višjo prioriteto od trenutnega P

- pri razvrščanju brez odzemanja, to ne bo prekinilo procesa P
- pri razvrščanju z odzemanjem pa to lahko prekine P (se P-pi odzame CPE)
in se ga dodeli Q

Pri razvrščanju po prioritetah je nevarnost: da proces nikoli ne pride do CPE, ker ga v vrsti
stalno prehitavajo novi, bolj pomembni procesi t.i. stradanje (starvation)

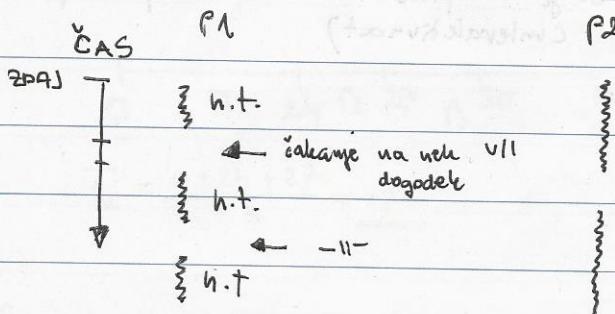
Zdravilo: OS periodično poveča prioriteto vsem pripravljenim procesom. t.i. staranje (aging)
Housekeeping opravila

Prioriteta: definirana bodisi na podlagi notranjih parametrov ali pa zunanjih parametrov

Alg. najkrajši nemoteni telo (shortest job algorithm)

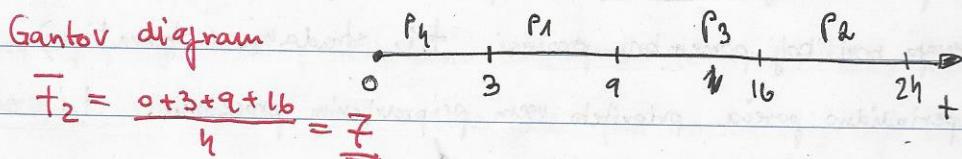
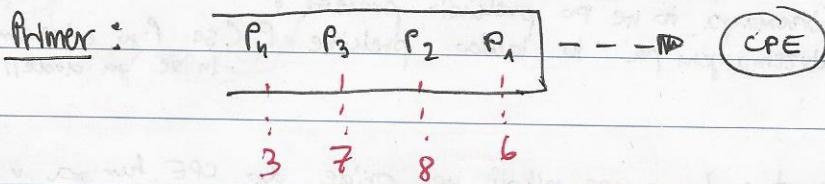
Nemoteni telo (n.t.) → CPU burst

↳ je tel procesa od trenutka, ko je dobil procesor do trenutka
ko se blokira



Heuristika: Proses, ki oblikuje
najkrajši naslednji nemoteni telo naj dobi procesor

Alg: CPE se dodeli tiskrnu pripravljenemu procesu, ki bo ~~dalje~~ nemoteno telo
najkrajši čas. To je algoritem brez odzemanja (če je najkrajši že izbran, zakaj
bi ga prehinali,...)



Trditev: SJF vrši minimalni \bar{T}_2 (med vsemi alg. brez odzemanja)

Dokaz: P_1, \dots, P_N
 t_1, \dots, t_N

če $t_1 < t_2 < \dots < t_N$ potem je poprečni čakalni čas \bar{T}_2 min

? Kako procesu določiti dolžino naslednjega nemotenega teca?

ODG: V lesnici (splošno) ne moremo natančno

Ali je zato SJF neuporaben?

Ideja: poslušimo naslednji me. tel. kar seda dobro oceniti (napovedati)

Kako? Po zgledu: "če za jutri napovem enako vreme, kot je danes, se bom v pop. malovrat zmotil"

Oziroma: "predpostavimo, da bo naslednji nem. tel. trajal podobno dolgo kot zadnji nem. tel."

To bo res. delovala če se vreme v resnici malovrat spreminja oz. če se n.t. ^{dolžina} redko spremeni (pri danem procesu)

Kaj pomeni "podobno dolgo"?

Naj bo: $t_n \dots$ dejanska dolžina n.t. zadnjega ~~procesa~~ n.t. za ta proces, ki ga opazujemo

$T_{n+1} \dots$ napovedana dolžina naslednjega n.t.

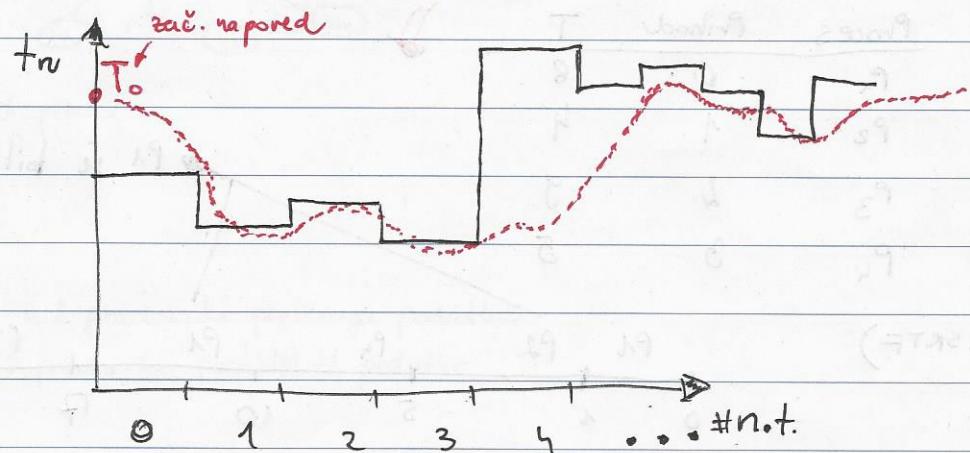
Naj bo: $T_{n+1} = d t_n + (1-d) T_n$, $0 < d < 1$

$\frac{d}{1-d}$ dejanska dolžina zadnjega predstavlja prejšnja napovedana vrednost

Primer

$$T_{n+1} = d t_n + (1-d) \alpha t_{n-1} + \dots + (1-d)^j \alpha^j t_{n-j} + \dots + (1-d)^{\infty} \alpha^{\infty} T_0$$

Koeficient α izberi tako, da se bo funkcija $T(n)$ čim hitje prilagajala $t(n)$ da bomo T_{n+1} izračnali hitro



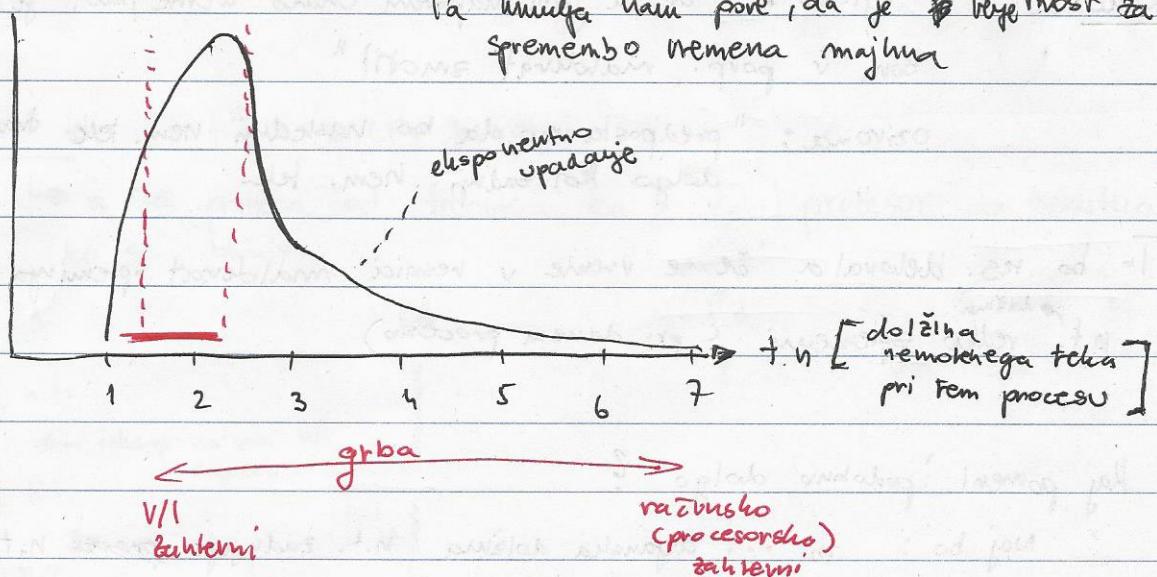
št. nemotenjega telca

bilo kot bi bilo "vreme"
nestabilno bolj slaba bi bila
napovedovalka redča funkcija
napovedi

Eksperimenti: Nuj bo P poljubem proces

pogostost
n.t. = to
dolžina

Ta kurva nam pove, da je verjetnost za spremembu vremena majhna



Alg. SRTF ($= SJF + \text{odzema inje}$) shortest remaining time first)

Ko nov proces Q vstopi med pripravljeni, se lahko zgodi, da je njegov $T(Q)$ manjši kot vira do konca svojega napovedanega n.t. telovir proces P

$T(P)$ $T(Q)$

SJF to je predine procesa P , se mu ne odzame CPE
SRTF pa v tem primeru CPE odzame P_1 in ga da Q -ju

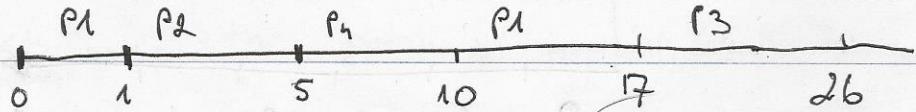
PRIMER :

[SRTF daje boljši \bar{T}_2 od SJF]

Proces	Prihod	T
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

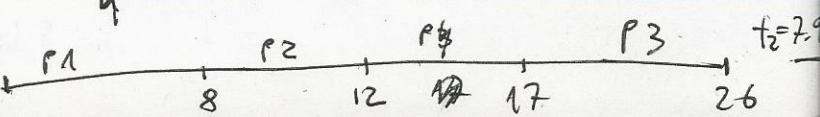
P_1 je bil vmes odset CPE

Gant (SRTF)

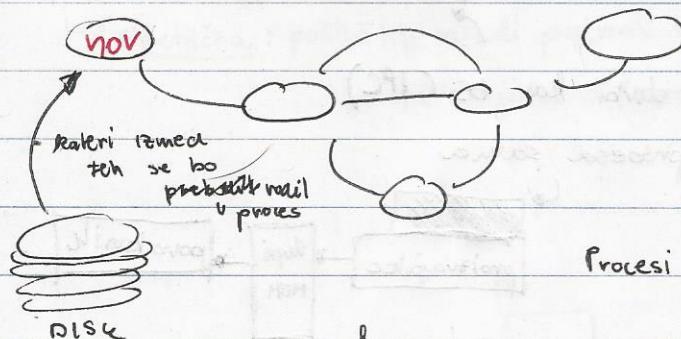


ČASOVNE $\bar{T}_2 = (0-0) + (1-1) + (5-3) + (10-1) + (17-2) = 65$

GANTH (SJF)



Razvrščanje poslov



Odločanje relativno redka (v primerjavi z razvrščanjem na procesorju)

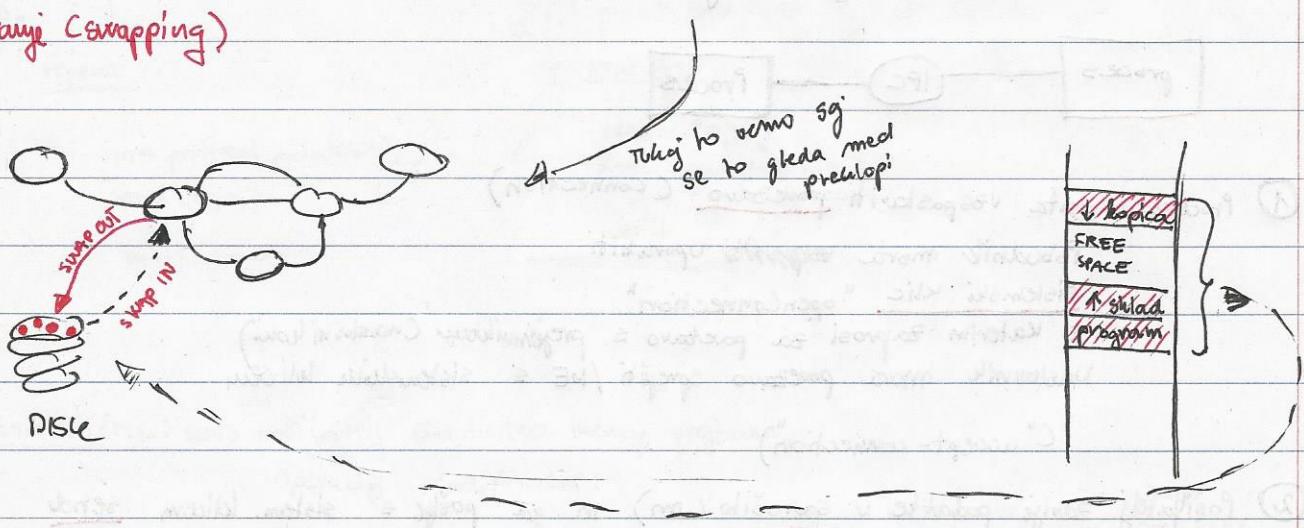
✓ lahko bi si privoščili tehničke odločitve
čemu dati pozornost?

Procesi v grobem dveh vrst:

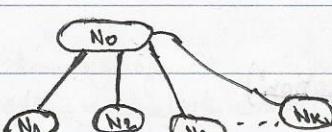
- ↳ procesorsko zahtevni
- ↳ v/I zahtevni

Razvrščevalnik bi lahko storil za vrednotenje obremenitev vseh enot računalnika (load balancing)

Menjanje (Swapping)



SODELOVANJE MED PROCESI



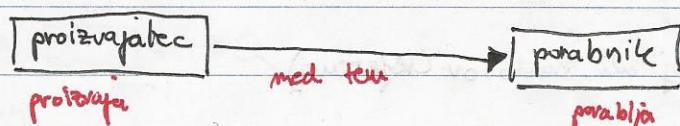
1. sinhronizacija med procesi

2. komunikacija

Proizvajalec - porabnik

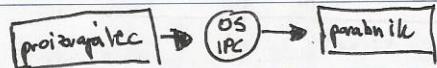
proizvajalec : proces, ki proizvaja podatke

porabnik : proces, ki porablja podatke



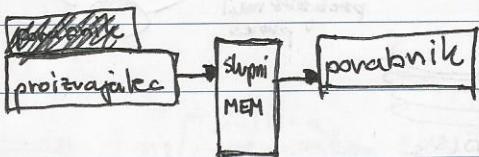
Kako poteka komunikacija med sosednjima procesoma

Vrste komunikacije (modeli komunikacije)



Dva modela:

- 1) za prenos podatkov med procesoma skrbijo kar OS (IPC)
- 2) za prenos med procesoma skrbita proces sama



IPC - inter-process communication

Del OS; tipično v jedru

Običajno realiziran s prenašanjem sporočil
(message passing)



- ① Procesa morata vzbudit povezano (connection)

Pobudnik mora ~~zuspostaviti~~ uporabiti

sistemski klic "openConnection"

s katerim zaprosi za povezavo s prejemnikom (nastavnikom)

Nastavnik mora povezavo sprejeti / NE s sistemskim klicem

("accept-connection")

- ② Posiljalci zavije podatke v sporočilo (m) in ga pošuje s sistem. klicem send (komu, m)

- ③ Prejemnik ima sistemski klic receive (od koga, m)
- s katerim prejme sporočilo

- ④ Povezava se prekine s sistemskim klicem "close-connection"

Povezava (glejamo logično)

- neposredne
- posredne
- simetrične
- asimetrične
- s takšnim ali drugačnim kopiranjem podatkov
- s posifanjem kopij ali naslovov (referenc)

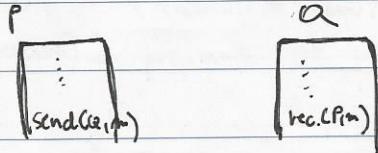
Posredna komunikacija

↳ procesi se pri komuniciraju eksplicitno i imenujejo (navedejo)

↳ simetrična: posiljalatelj in tudi prejemnik drug drugega eksplicitno imenuje operacije send/receive:

P: send(Q, m); ... pošti procesu Q sporocilo m

Q: receive(P, m); ... prejmi od P sporocilo m



Primer (proizvajalec, ponabnik)



P:

repeat

m = proizvedi_podatek();
send(Q, m)
until

Q:

repeat

receive(P, m)
parabi - podatek(m)
until

↳ asimetrična: samo posiljalatelj eksplicitno imenuje prejemnika
Operacije read/receive:

P: send(Q, m) ... pošti m Q-ju

Q: receive(x, m) ... prejmi sporocilo od kogarkoli

primer
STREŽNIK

Posredna komunikacija

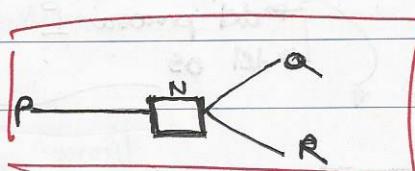
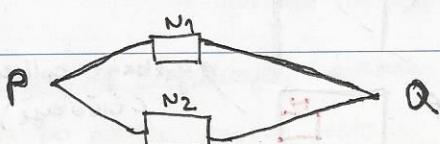
↳ procesi, ki komunicirajo se ne imenujejo (navedejo), pač pa se imenujejo neh usmesni medij - nabiralnik (mailbox)

nabiralnik ima enolični nastav; nabiralnikov je lahko več. Ideja je enostavna: posiljalatelj vrsti sporocilo v nabiralnik, od tam pa ga prejemnik vzame / preberi

Operacije send/receive

P: send(N, m) ... vrzi sporocilo v nabiralnik N

Q: receive(N, m) ...



Upodobitev, nevarnosti

(glej primer)

→ Idma za Sporočilo

Primer (proizvajalec - porabnik)

P:

repeat

```
m = proizvedi-podatki();
send(N,m);
:
until
```

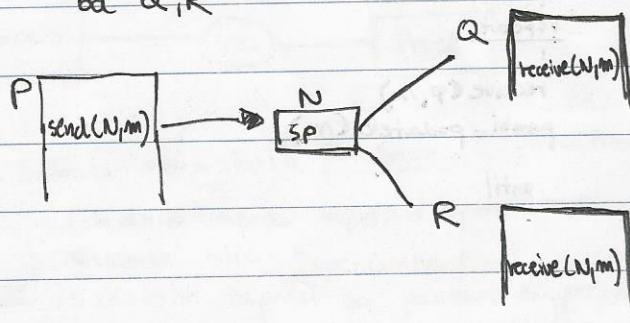
Q:

repeat

```
receive(N,m);
porabi-podatki();
until
```

Tekma za sporocilo:

- Denimo: P pošlje sporocilo v N = namenom, da ga bo prebral le eden od Q, R



potresni scenarij:

① Q prejme sporocilo sproži svoj receive
Receive uspe prenesti sporocilo Q-ju

② Q izgubi procesor, preden je receive uspel zbrisati
Sporocilo iz N

③ procesor dobri R, Recimor da tudi ta zazne svoj
receive

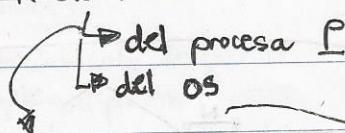
⇒ Q in R dobila sporocilo

⇒ receive bi se moral dokončati, preden se zadre drugi receive (\Rightarrow kritični opseci)

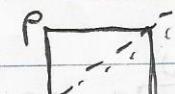
Kje je nabitavilnik?

Denimo, da P ustvari nabitavilnik N

Moznosti 2:



1.



sist. p

garbage collection
(čiščenje)

Sinhronizacija send - receive

send() je :

- ↳ sinhrona (blocking send) : pošiljatelj čaka dokler prejemnik sporočilo ne prispe
- ↳ asinhrona (non-blocking send) : pošiljatelj ne čaka (tj. pošiljatelj lahko nadaljuje kar po

ke naslednjemu

receive() :

- ↳ sinhron (blocking receive) : prejemnik čaka, dokler sporočilo ne prispe
(čodisv = njeni / naravniki)
- ↳ asinhron (non-blocking receive) : prejemnik ne čaka

stopen programu

Možne so razne kombinacije teh send - receive

Rendezvous: (zmeni) : Kadar pošiljatelj sihrono pošlje in prejemnik sihrono prejme

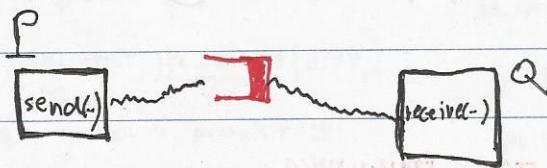
pošiljatelj

prejemnik

sihron receive

Kopiranje

Povezava med dvema procesoroma :



Gleda na kapaciteto vrste ločimo povezave

→ brez kopiranja (zero-capacity oz. no-buffering)
(povezava ne more shraniti nobenega sporočila)

↳ če a še ni sprejel prejšnjega sporočila,
potem P ne more poslati naslednjega

→ z omejenim kopiranjem (bounded capacity)

(povezava je sposobna shraniti N sporočil
oz sporočila soš skupno dolžino n)

→ z neomejenim kopiranjem (povezava ima potencialno neomejeno kapaciteto)

OS po potrebi posreča vrsto

POŠILJANJE KOPIJ ALI NASLOVOV

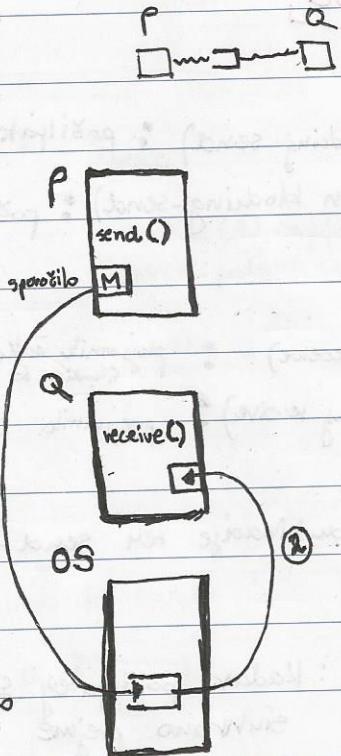
Kaj se prenosa pri komunikaciji

① prenos kopije sporočila

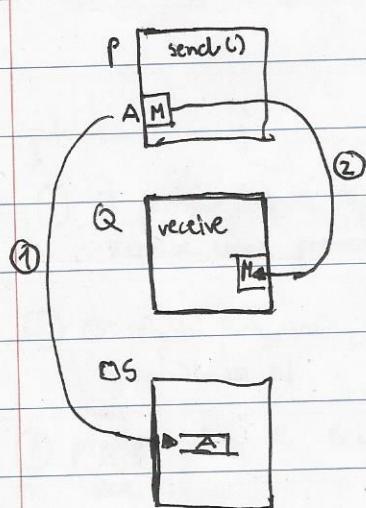
- ① → Ko se začne send(), OS shupira M v svojo (sistemska) vrsto

- ② → Ko Q sporoči receive
OS shupira M iz svoje vrste, v naslovni prostor Q-ja

- ⊖ Sporočilo se kopira 2x (prostori)
če so sporočila dolga in se pošiljajo pogosto, se za komunikacijo rabi, relativno veliko časa



② Prenos naslova sporočila



- ① → Ko P zahteva send(), OS shupira A v svojo vrsto

- ② Ko Q zahteva receive(), takrat OS shupira sporočilo M iz naslovnega prostora P-ja v naslovni prostor Q-ja

- ⊕ Sporočilo se kopira 1X

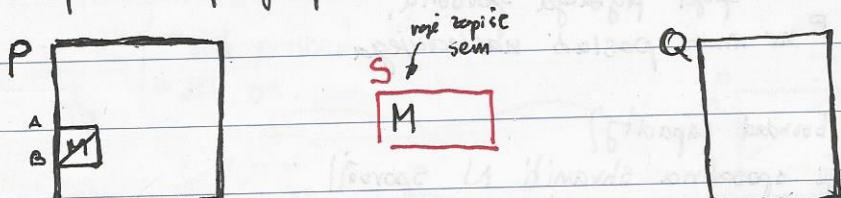
- ⊖ Če se konča treba shraniti M (OS) podobno kot ①

KOMUNIKACIJA PREKO SLEPNEGA POMNILNIKA

Motivacija: Pri IPC je potrebno vsaj 1x kopirati sporočilo (čas!!)

(?) Kako bi proces P posredoval podatke mreži procesu Q, brez kopiranja

Idee: preko slavnega pomnilnika P in Q



Denimo, da je M pripravljen na lokacijah A-B. P bi mora obvestiti OS, da "odpira" del M med A-B procesu Q (dovoljuje Q-ju dostop do M-ja)

OS bi moral omogočiti Q-ju, da ne posega po tem naslovnem prostoru, ki je del P, ne, da bi se ujel v past (zaščita pomnilnika)

- V praksi :
- ① P zaprosi OS naj poišče pomnilnik S (izven nasl. prostora P-ja) in Q-ja, dane velikosti, ki bo skupen. OS vrne njegovo ime (lokacija)
 - ② P si najdeni S priključi v svoj naslovni prostor
 - ③ Tudi Q si lahko priključi S v svoj nasl. prostor
 - ④ Ko ga ne rabita več lahko S vrneta (detach)

Primer (unix)

P (pošiljatelj) Q (prejemnik) // te podatke prebere in jih uporabi
// P naj zaprosi za skupni pomnilnik in upiše vaj podatke

```
#include <sys/shm.h>
#define KLJUC 555
#define VELIKOST 100 (size_t)
```

```
main()
{
    int id_shp; // kazalec na S
    char *kaz_shp; // niz znakov
    char *podatki;
```

// prosi za rezerviranje slv. pomnilnika
// dane velikosti in dostopnih pravic
// od OS pridobi identifikator tega slv. pomn.

id_shp = shmget(KLJUC, VELIKOST, IPC_CREAT | 0777);

// prosi OS, da razširi naslovni prostor s
// tem skupnim pomnilnikom, OS bo ob tem
// vrnil naslov, kjer se začenja ta skupni pomn.

kaz_shp = shmat(id_shp, NULL, 0);

podatki = "1234";

strcpy(kaz_shp, podatki);

shmctl(id_shp, IPC_RMID, NULL) // pomnilnik
// vrnemo OS

```
#include <sys/shm.h>
#define KLJUC 555
#define VELIKOST ((size_t) 100).
```

```
main()
{
    int id_shp;
    char *kaz_shp;
```

// Q zaprosi za identifikator obstoječega
// skupnega pomnilnika, OS mu ga vrne

id_shp = shmget(KLJUC, VELIKOST, 0777);

kaz_shp = shmat(id_shp, NULL, 0)

printf("Izpis iz slv. pomn.: %s", kaz_shp);

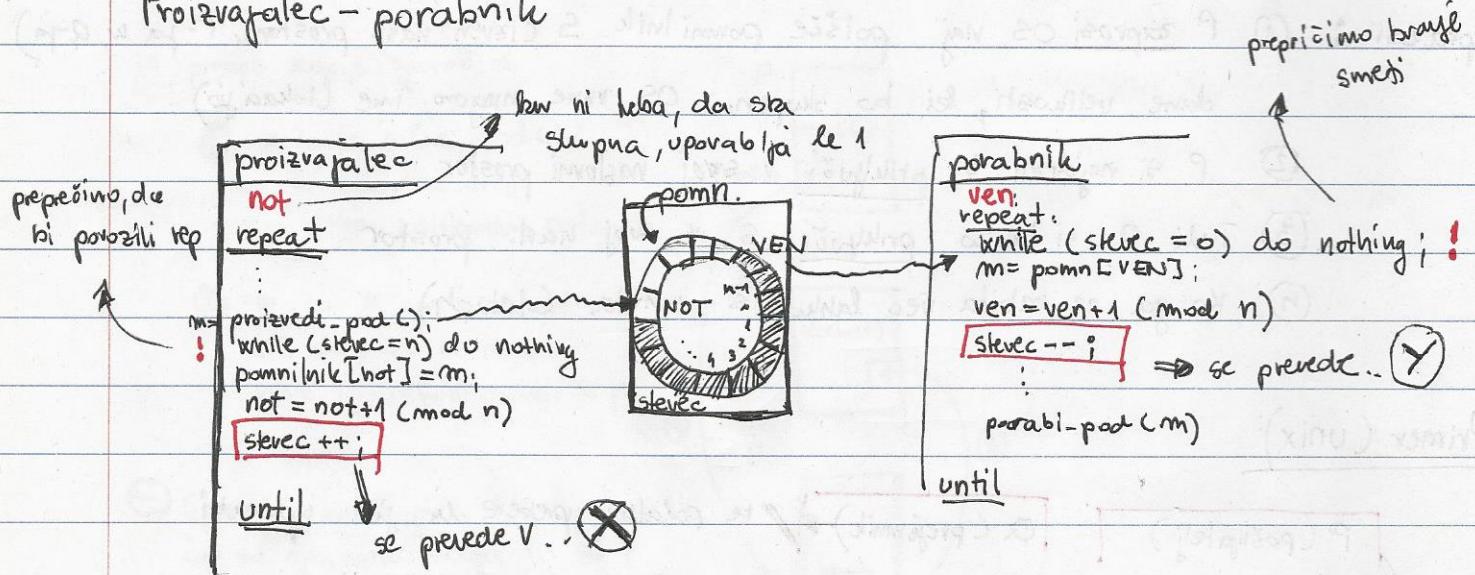
~~shmat~~

shmat(id_shp)

... loči se od
skupnega pomn.

SINHRONIZACIJA

Proizvajalec - porabnik



(X) $\xrightarrow{PC_1}$:
A1: $R1 \leftarrow stevec$
A2: $R1 \leftarrow R1 + 1$
A3: $stevec \leftarrow R1$
:

(X) $\xrightarrow{PC_2}$:
B1: $R2 \leftarrow stevec$
B2: $R2 \leftarrow R2 - 1$
B3: $stevec \leftarrow R2$
:

Predpostavka: PC1 in PC2 sta že pred A1 in B1
 $stevec = 5$ (tedaj)

Izvajanje st. 1

$stevec = 5$

A1
A2
A3
:

... $stevec = 6$

B1
B2
B3
:

... $stevec = 5$

$\xrightarrow{\text{prelup}}$

Izvajanje st. 2

$stevec = 5$

A1

A2

$\xrightarrow{\text{prelup}}$

(shrani $R1 = 6$ na sledil, ...)

iz $stevca = 5$ se prenese v $R2$

B1

B2

B3

:

$\xrightarrow{\text{prelup}}$ (nacaljuje \times)

A3

... $stevec = 6$!!!

Izvajanje 3

$stevec = 5$

A1

A2

$\xrightarrow{\text{prelup}}$

B1

B2

$\xrightarrow{\text{prelup}}$

A3

$\xrightarrow{\text{prelup}}$

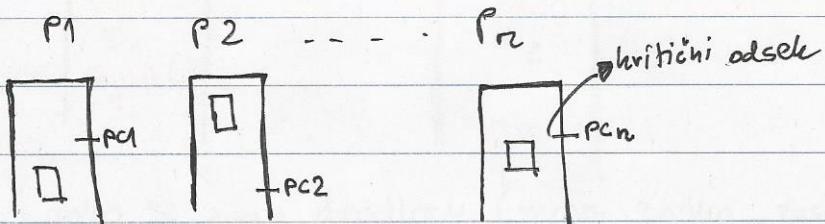
B3

$stevec = 4$

Problem kritičnih odsekov

Danih je n sočasnih procesorov P_1, \dots, P_n

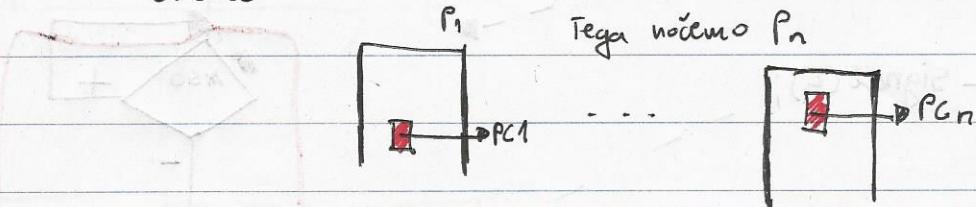
Vsak od njih ima v svojem programu vsaj en kritični odsek



Naloga: Zagotoviti sočasno izvajanje (preklapljanje) procesov $P_1 \dots P_n$, da bodo izpolnile naslednje zahteve:

1. Vzajemno izključevanje procesov (mutual exclusion)

sočasno je lahko levočemo en proces s svojim PC znotraj svojega kritičnega odseka



2. Omejeno čakanje na vstop v kritični odsek

→ Nihče nekontrolno dolgo (neomejen) na vstop v svoj k.o.

3. Omejen vpliv na izbor naslednjega, ki vstopi v svoj k.o.

→ na izbor vplivajo le trenutni tekmeci (kandidati) za vstop

4. Splošnost rešitve

→ Rešitev naj velji za vsake N , $N \geq 2$ (N = št. procesorov)

→ Neodvisnost od tehničnih značilnosti računalnika

Naši programi $P_1 \dots P_n$ imajo zgradbo

P_i : repeat:

:

k.o.

:

until:

Rešitev bomo iskali v obliki dveh dodatkov, s katerima bomo zavarovali k.o.

P_i : repeat:

:

vstopni del

k.o.

:

izstopni del

until

Tu se P_i dovoli/ne da vstopi v svoj k.o.

Tu se obvesti ostale, da je P_i zapustil svoj k.o.

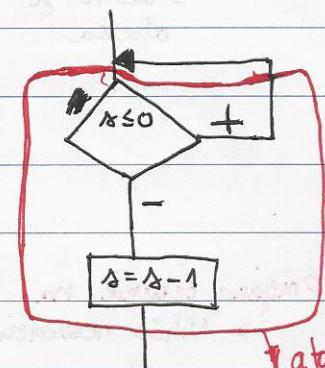
Načini reševanja problemov teritičnih odsekov

- čisto algoritmično / programerski (banker) → nepraktično
- Test & set (\times), swap (A, B) → atomarna izvedba (cel vek ali nič)
- semajorji
- monitorji

SEMAFOR (krožni → spinlock sem.)

Dijkstra

- celoštevilška spr. $s \in \mathbb{Z}$
- nad s dovoljeni 2 operaciji: (počet inicializacije)
 - wait(s);
 - signal(s);



$$s = s + 1$$

atomarna izvedba

izvede se v celoti, se ne prehine → prehlop!

Uporaba semajorjev

Problem: Proizvajalec - Potrobnik

Proizvajalec $s=1$

repeat:

wait(s)

stevc = stevc + 1;

signal(s)

until

Potrobnik

repeat:

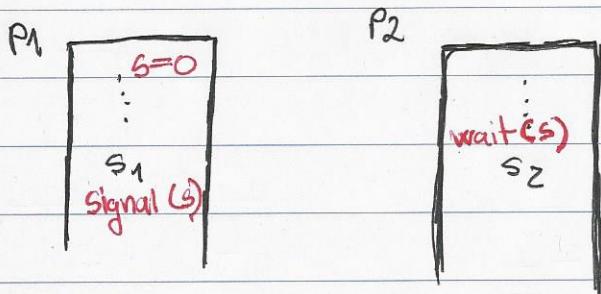
wait(s)

stevc = stevc - 1;

signal(s)

until

Sinhronizacija procesov:



Naloga: s_2 se sme izvesti šele potem, ko se ji končal s_1

$(s_1 \prec s_2)$

$\nearrow s_1$ se mora prej končati

Lahko se zgodi deadlock (mrtna zauha, zastoj).

