**Vrste Jezika:**

Programski jezici razlikuju se po skupu postupaka, koncepata i načela za koje je

utvrđeno da pomažu učinkovitom rješavanju neke vrste problema.

**Razlikovanje jezika s obzirom na strukturu** programa, strukturu stanja i metodologiju

procesiranja:• Blok strukturirani, procedurno orijentirani jezici (Pascal, C),• Objektno zasnovani, objektno orijentirani jezici (C++, Java) ,• Raspodijeljeni, konkurentni jezici (Erlang, Java),• Funkcijski jezici (Lisp, Clojure, F#),• Logički jezici (Prolog)

**Kategorizacija jezika**:Imperativni jezici (procedurni):• blok strukturirani i objektno zasnovani,• program je prikazan slijedom naredbi za promjenu stanja koje se odvija od početka prema kraju,• jezik (program) modelira arhitekturu računala,• specificira se KAKO treba raditi,• Deklarativni jezici (neprocedurni) :• funkcijski i logički ,• specificira se ŠTO treba raditi, neovisno o tome kako će se izvesti!

**Trodimenzionalni prostor modela procesiranja razvoja jezika:**

Promjena stanja(slijed promjena stanja izvodi se izvršavanjem naredbi: pohrani/dohvati, naredba određena ulaznim simbolom i tekućim stanjem (Turing), računala s pohranjenim programom spremaju instrukciju kao dio njenog stanja i dovode je u procesnu jedinicu prije izvođenja, za asemblerske i imperativne jezike promjena stanja je osnovni model)

Distribucija poruka(komunikacija između jedinki je osnovni model procesiranja: pošalji/primi su nedestruktivne operacije, • komunikacijski kanal, katkad sa spremnikom, ima ulogu memorije, • varijabla ima značenje kanala, • jedinka je definirana komunikacijskim sučeljem, neovisno o unutarnjem stanju, • kombinacija komunikacije i promjene stanja: programske jedinke s unutarnjim stanjem i komunikacijskim vratima sa spremnikom poruka za vanjsko ponašanje.)

Klasifikacija (procesiranje se promatra kao slijed koraka klasifikacije od kojih svaki služi da ograniči rezultat, a završava kada se izvede jedan element kao konačni rezultat, • objektno orijentirano programiranje podržava klasifikaciju objekata u klase.)

**Procesiranje izvornog programa:**• interpretiranje, kojim se ustanovljena naredba izravno izvodi te **,**• prevođenje, kojim se referenciranjem na skup strojnih instrukcija pridruženih izvornoj instrukciji generira prevedeni program i povezuje sa skupom strojnih instrukcija u biblioteci kako bi se dobio izvršni binarni program.

Imperativni se prevode(brži), a deklarativni inerpretiraju.

**Značajke programskih jezika glede primjene su:**

Lakoća pisanja (writability) • ograničena kod imperativnih jezika, jer modeliraju rad procesora (izvođenje programa), • od početka bolja kod deklarativnih jezika (zbog potreba područja umjetne inteligencije zbog kojeg su nastali).Lakoþa čitanja (readability) • važna kod velikih programskih sustava ,• deklarativni jezici su općenito lošiji za čitanje za što je razlog način primjene, a ne sam jezik.

**Konkurentnost** označava mogućnost istodobnog izvođenja dvije ili više aktivnosti. Dvije ili više aktivnosti se mogu izvesti istodobno ako ne utjeću jedna na drugu.

**Paralelizam** označava istodobno izvođenje dvije ili više aktivnosti. Konkurentne aktivnosti se mogu izvoditi paralelno ukoliko se raspolaže resursima za to. Konkurentne aktivnosti se mogu odvijati i slijedno.Pseudoparalelizam opisuje situaciju kad se raspolaže jednim procesorom, a d ijelovi programa se "bore" za dodjelu vremena za izvedbu, Stvarni paralelizam postiže se samo s više procesora koji istodobno izvode različite dijelove programa.

**Osnovnim modelom procesora** smatra se von Neumannov model. On se zasniva na memoriji s pohranjenim instrukcijama i podacima, te tokom operacija na temelju pojedinačne točke upravljanja opisane programskim brojilom. Njegove značajke određuju:• programsko brojilo koje označava instrukciju u izvođenju,• adresa koja opisuje podatak i pristup podatku.SKICA(memorija programa, točka upravljanja(programsko brojilo) strelica na točku kod instrukcije i, ima i j k,, zatim strelica na memoriju podataka koja sadri podatke a,b,c, strelica se zove adresa)

Ograničenje osnovnog modela : slijedno izvođenje instrukcija (jedna po jedna u slijedu) koje se odražava na brzinu rada i informacijski kapacitet. Neka rješenja za njegovo prevladavanje su:

• uvođenje konkurentnosti oko točke upravljanja (istovremeno izvođenje različitih faza "susjednih" instrukcija), • operacije nad skupom podataka (npr. vektor ili polje), • višeprocesorska organizacija (više po vezanih istovrsnih ili raznovrsnih procesora), • primjena optimizirajućih prevoditelja za konvencionalne (nekonkurentne) jezike, • uvođenje konkurentnih dijalekata konvencionalnih jezika, • uvođenje konkurentnih jezika.

**Sustavni prikaz arhitektura procesorskih sustava:** na dva načela: • broj instrukcija i podataka istodobno u obradi (Flyn), • funkcionalnost i tok informacija (Skillicorn).

**Flynu** to su modeli procesora u kojima se obrađuje istodobno: • jedna instrukcija - jedan podatak, SISD (engl. Single Instruction Single Data), • jedna instrukcija - više podataka, SIMD (engl. Single Instruction Multiple Data), • više instrukcija - jedan podatak, MISD (engl. Multiple Instruction Single Data), • više instrukcija - više podataka, MIMD (engl. Multiple Instruction Multiple Data). Za upravljačke primjene u stvarnom vremenu važne su arhitekture SISD i MIMD.

**Skillicornova** metodologija uvodi apstraktne modele slične onima koji se primjenjuju u procesorskom upravljanju telekomunikacijskim sustavima. Funkcijske jedinice apstraktnog procesora su: • instrukcijski procesor (IP), • instrukcijska memorija (IM), • podatkovni procesor (DP), • podatkovna memorija (DM), • komunikacijska jedinica, izvedena kao komutacijsko polje ili mreža (SW n x n) koja omogućuje povezivanje svake jedinice sa svakom.

Informacije koje izmjenjuju su: • adresa instrukcije, • instrukcija, • adresa podatka • podatak (operand, rezultat) i • stanje operacije.

**Skillicornovom redoslijed operacija:SKICA(**lijevo: DP,DM, desno: IP,IM**)** **1.** programsko brojilo u IP postavlja adresu instrukcije za IM ,**2.** instrukcija se očitava iz IM i priprema za izvršenje u IP ,**3.** IP dostavlja operacijski kod (OP kod) instrukcije i adrese operanada u DP , **4.** DP postavlja adresu operanda za DM ,**5.** vrijednost operanda očitava se iz DM, dovodi u DP i izvodi operacija ,**6.** DP postavlja adresu rezultata za DM ,**7.** DP pohranjuje vrijednost rezultata u DM ,**8.** DP vraća stanje operacije za IP čime označava da je završio izvođenje instrukcije.

**Moguće ubrzanje rada postiže se uz: 1.** Istodobno (paralelno) izvođenje više operacija za jednu instrukciju.(DP:pohrana rezultata u DM(7) i vraća stanje operacije u IP(8)),**2.** Istodobna obrada više instrukcija u različitim fazama.(jedna se dostavlja na izvođenje (3), druga dohvaća slijedeću instr u slijedu (1)),**3.** Povećanje broja procesora

**Povećanje broja procesora: a)**Procesor polja podataka (Array Processor) koji sadrži jedan IP i više DP-ova. Instrukcije u takvom procesoru određuju operacije sa skupovima podataka koje se izvode tako da se međusobno neovisna računanja provode istodobno, a za što treba više DP-ova.**b)** Višeprocesorski sustav ( Multiprocessor) s više međusobno povezanih osnovnih procesora.**c)** Procesor toka podataka (Data Flow Processor) koji nema IP i IM, a svu obradu provodi više DP-a.

**Slabe povezanosti :**(engl. loosely coupled).SKICA(lijevo:SWnxn povezan sa DP ovima, dva DP i DM , desno dva IP,IM) Slaba povezanost označava da svaki DP pristupa samo svojoj DM, a komunikacijom između DP-ova obavlja se izmjena podataka i usklađivanje rada. Slabo povezani sustav prikladan je za izvedbu mreže procesora koji međusobno komuniciraju izmjenom podataka kroz komunikacijsku mrežu.

U telekomunikacijama radi sa slabo povezanim procesorima (bez zajedničke memorije) i slabo povezanim procesima (bez zajedničkih podataka).

**Jako povezani (tightly coupled) procesori** su oni u kojima svaki DP može pristupiti svakoj DM, a ne samo vlastitoj (zajednička ili dijeljena memorija).SKICA: (DP-ovi su povezani sa DM preko SWnxn)

**Korutine** - dijelovi programa koji se izvode preklapajuþi se u vremenu (sl. 3.1). U svakoj točki tijekom izvedbe jedna korutina može obustaviti ili ponovno pokrenuti drugu. Ponovno pokrenuta korutina nastavlja se od točke u kojoj je bila obustavljena. Ovakav način izvođenja naziva se pseudoparalelnim, jer se u jednom trenutku izvodi samo jedna korutina, ali promatrač izvana dobiva dojam kako se dvije ili više korutina izvode istodobno. SKICA(strelica odozgora na dolje, lijevo je prog.1, u sredini ide strecila desno u prog.2, strelica dolje, pa desno natrag pa dolje))

**Paralelne naredbe** izvode se istodobno, što se postiže paralelizacijom uobi čajenih programskih konstrukata, npr. bloka (par begin – par end) ili petlje (parfor) (sl. 3.2). Sve naredbe paralelnog bloka izvode se istodobno, kao i sve iteracije paralelne petlje. Paralelizam razine naredbe naziva se fino-granuliranim paralelizmom (fine-grain).SKICA(između parbegin i parend su 4 naredbe koje se poslože horizontalno-istodobno)

**Proces** je najpotpuniji i najčešći konstrukt za predočavanje paralelizma. Proces je apstrakcija fizikalnog procesora. On izvodi naredbe u slijedu, ima svoje stanje i podatke. Paralelizam se postiže s više procesa koji se izvode na više procesora i međusobno komuniciraju ,prvo definira procesni tip procesa koji opisuje programski kod.,pa više procesa tog tipa stvara odvajanjem od osnovnog procesa (engl. fork) i oba procesa nastavljaju s paralelnim radom. Procese treba uskladiti (sinkronizirati). Najjednostavnije je rješenje s čekanjem osnovnog procesa da svi završe obradu i sastanu se (engl. join). Kako procesi izmjenjuju informacije, sinkronizacija se ne može promatrati odvojeno od komunikacije. SKICA: proces, u kvadratu: naredbe više njih, ispod podaci, ERLANG:Pid=spawn(Module,FunctionName, ArgList)

**Nit ili dretva** (thread) razlikuje se od procesa jer ne raspolaže vlastitim podacima, većih dijeli s procesima i drugim nitima. Nit izvodi naredbe u slijedu, ima svoje stanje obrade, ali ne i podatke (SKICA: proces iznad, u kvadratu: naredbe više njih, ispod podaci, dretva pored koja povezuje podatke )Java

**Kritični djelovi** - Dijelovi procesa u kojima se pristupa zajedničkim varijablama

**Razlika između monitora i varijable brave-** varijabla brava je jedna obična varijabla koja se provjeri prije nego se uđe u kritični odsječak (recimo semafor koji zna brojat samo do 1), a monitor je apstraktni tip podataka i baš u njemu su zatvoreni podaci i operacije nad njima.

**Sinkronizacijski primitivi za zajedničke varijable**

• **varijabla brava** (lock variable) je zajednička varijabla s dvije operacije: zaključaj - otključaj (lock - unclock); kad jedan proces zaključa drugi ne može pristupiti dok ponovno ne otključa

**• semafor** (semaphore) je zajednička varijabla s dvije operacije: povečaj za 1 - smanji za 1 ako vrijednost nije nula (V(s) - P(s)); operacijom V(s) proces onemogučuje pristup drugima, a proces koji uspije s operacijom P(s) nastavlja s obradom.

• **monitor** je apstraktni tip podatka u kojem su zatvoreni podaci i operacije nad njima, a samo jedan proces može izvoditi operaciju. S dvije operacije može se blokirati i reaktivirati blokirani proces (wait, signal). Kad je proces u monitoru siguran je da je jedini aktivan pa to mora trajati kratko kako bi i ostalima dopustio izvedbu.

**Izmjena poruka** SKICA(Proces Pid1 { Pid2!Message} strelica Message dp Proces Pid2 {receive Message->Actions;end.})

**Raspodijeljeno pridruživanje** ( distributed assignement) Naredbe za predaju i prijam poruke

U jeziku Erlang primitiv send je oblika: Pid!Message , gdje je Pid identifikator procesa kojem se šalje poruka, a Message poruka , a primitiv receive ovakav: receive Message1 [when Guard1] -> Actions1 ; Message2 [when Guard2] -> Actions2 ; end.

**Imenovanje** predajnog i prijamnog procesa može biti: • simetrično izravno; svaki proces imenuje onog drugog, SKICA(Proces S šalje procesu R) • asimetrično izravno; prijamnik prima poruku od svakog procesa, SKICA(Procesi S1 i S2 šalju procesu R) • posredno; uvodi se međuobjekt nazvan vrata (port) za izmjenu poruka od više procesa prema jednom ili međuobjekt poštanski sandučić (mailbox) za izmjenu poruka između više procesa. SKICA(Procesi S1 i S2 šalju procesu R preko porta P, ili preko mailboxa M),

Osnovni oblik send i receive primitiva opisuje asimetrično izravno imenovanje kojim predajni proces imenuje identifikatorom prijamni proces, dok prijamni proces ne zna za predajni.

**Suradnja procesa** određuje odnose jednog ili više predajnih procesa s jednim ili više prijemnih procesa.1:1 (• izmjena poruka između dva procesa u jednoj točki: kanal između procesa, izravno imenovanje procesa, • izmjena više poruka u više točaka: više kanala - izravno imenovanje kanala ili poštanski sandučić, n:1 • više procesa šalje poruke jednom procesu: asimetrično imenovanje procesa , 1:m • jedan od prijamnih procesa prima poruku (anycast), • svi prijamni procesi primaju poruku (broadcast), • neki od prijamnih procesa, odnosno procesi u definiranoj skupini primaju poruku (multicast)., n:m • svaki proces može poslati poruku svakom procesu, svaki proces može primiti poruku od svakog procesa

**Način izmjene poruka:**

**Sinkrona komunikacija** Komunikacija je sinkrona ako predajni proces nastavlja obradu tek nakon potvrde od prijamnog procesa. Sinkrono komuniciraju procesi u modelu CSP (imenovani kanal, jednosmjerno)

**Asinkrona komunikacija** Kod asinkrone komunikacije predajni proces nastavlja s radom odmah nakon predaje poruke. Da bi to bilo moguće potreban je spremnik za poruke, npr. poštanski sandučić, u kojem će poruka biti pohranjena dok je ne preuzme prijamni proces. U Erlang komuniciraju asinkrono.

**Način prijama poruke:**

**Eksplicitni** je prijam onaj kod kojeg prijamni proces kontrolira i određuje kad će primiti poruku. SKICA(proces R, čeka poruku, receive, kod receive je strelica ulazna, nastavlja obradu). U Erlangu.

**Implicitni prijam-**kad poruka stigne, na prijamnoj strani stvara novi podproces – nit koja obavi prijam, a odvija se pseudoparalelno s osnovnim prijamnim procesom i dijeli s njim podatke . Oni se sinkroniziraju kao u primjeru zajedničkih varijabli.SKICA(proces R ide do poruke, kod poruke <<t, nastavlja obradu do receive)

**Smjer komunikacije**

**Jednosmjerno** je pri normalnom odašiljanju i prijamu jedne poruke

**Dvosmjerno** kad se u povratnom smjeru daje odgovor na primljenu poruku. Često se rabi kod procesa u međusobnom odnosu klijent/poslužitelj, i u Erlangu

**Dvosmjerno u Erlangu:**

Klijent: klijent(Zahtjev)-> server!{self(), Zahtjev}, receive {server, Odgovor} -> R end. Poslužitelj: server(Data) -> receive {From, X} -> {R,Data1}=fn(X,Data), From!{server,Rjesenje} server(Data1) end.

**Dvosmjerna komunikacija rješava se na više načina:**

**• sastanak (rendezvous)** koji predočuje dvosmjernu komunikaciju porukom i potvrdom s eksplicitnim prijamom.(SKICA: Proces X šalje zahtjev procesu Y i čeka rezultat obrade. Y obrađuje i šalje rezultat. Nakon primitka rezultata,X nastavlja s radom.) Sinkrona dvosmjerna komunikacija uz eksplicitni prijam

• **poziv udaljene procedure (RPC - Remote Procedure Call)** prvenstveno je namijenjen komunikaciji između klijenta i poslužitelja na načelu implicitnog prijama Ideja je jednostavna i nalikuje običnom pozivu procedure, ali je izvedba teža, jer je riječ o proceduri koja se izvodi na udaljenom raĀunalu. Jezik treba omogućiti procesu klijentu da sve parametre smjesti u poruku (marshalling), a procesu poslužitelju da izvadi parametre iz poruke (unmarshalling). Poslužitelj izvozi (export) jednu ili više procedura za udaljeno pozivanje, a sučelje prema poslužitelju specificira procedure. SKICA(lijevo: procesi X1 i X1, dva puta X1, desno Y, X1 šalje upit prema Y i čeka odg, , Y primi obradi i šalje rezultat, drugi X1 šalje poruku za vrijeme obrade prvog upita od prvog X1, Y primi upit i obradi tek kad pošalje odgovor na prvi upit, i šalje odgovor) - Sinkrona dvosmjerna komunikacija uz implicitni prijam