# Lygiagretaus konspektas

# Atviri klausimai egzamine – lygiagretumas

## CPU lygiagretumo galimybės;

## Funkcinis lygiagretumas ir kaip jis sprendžiamas įvairiomis priemonėmis;

Funkcinis lygiagretumas Lygiagretumo modelis, kai visos gijos atlieka skirtingus veiksmus su tais pačiais duomenimis. Pagrindinė sprendžiama problema — su dideliu kiekiu duomenų reikia atlikti keletą skirtingų veiksmų. Atliekant keletą veiksmų lygiagrečiai gaunamas pagreitėjimas, nes vienu metu vykdomas ne vienas veiksmas, o keletas

Funkcinio lygiagretumo problemos Kaip išlygiagretinti skaičiavimus, kai vieno veiksmo rezultatai priklauso nuo kito veiksmo rezultatų? Jei veiksmai priklauso vienas nuo kito, gali kilti sunkumų. Kaip išskirti tokius veiksmus, kurie gali būti atliekami lygiagrečiai? Veiksmus lengviau lygiagretinti tada, kai jie nepriklausomi.

## Duomenų lygiagretumas ir kaip jis sprendžiamas įvairiomis priemonėmis;

Duomenų lygiagretumo problemos Kiek gijų sukurti? Kai gijų per mažai, skaičiavimai vyksta lėčiau, kai gijų per daug — vyksta konteksto perjungimas, gijų valdymas sunaudoja didelę dalį skaičiavimų laiko. Kaip išdalinti duomenis gijoms? Jei duomenys išdalinti nelygiai — vienos gijos skaičiuos ilgiau, kitos — trumpiau; skirtingų elementų apdorojimo laikas gali būti skirtingas. Kaip valdyti pašalinius gijų vykdomų funkcijų poveikius, jei gijose vykdomos ne grynos funkcijos ir naudoja tuos pačius resursus?

Duomenų paskirstymą gijoms galima spręsti panaudojant gijų telkinį (thread pool). Sukuriant gijų telkinį paleidžiamas fiksuotas kiekis gijų. Gijų telkiniui galima priskirti darbą, kurį gijos pasidalija pačios. Gijų telkinio privalumai — nereikia kiekvieną kartą kurti gijų, gijos pačios pasiskirsto darbus.

fork-join šablonas Duomenų lygiagretumo šablonas, kuris paskirsto gijoms užduotims ir po to susirenka visų gijų rezultatus. Susideda iš tokių pagrindinių žingsnių: 1 Padalinti užduotį į keletą sub-užduočių, kurios vykdomos lygiagrečiai; 2 Palaukti, kol paleistos gijos baigs darbą; 3 Apjungti gijų suskaičiuotus rezultatus.

## Gamintojo – vartotojo uždaviniai ir jų sprendimas;

## Tikras lygiagretumas ir užduočių perjungimas;

## Lygiagrečiojo programavimo sukeliamos problemos;

Lenktynių sąlygos dėl netinkamo interfeiso: Apsaugoti visus metodus nuo lygiagrečios prieigos gali nepakakti, norint apsaugoti duomenis nuo lygiagrečios prieigos. Svarbu, kaip metodai yra kviečiami.

Pavojus kyla, kai iš funkcijos ar metodo grąžinama nuoroda į monitoriumi saugomus duomenis. Jei yra metodas, kuris tiesiogiai grąžina tokius duomenis, kitoje programos vietoje, kur grąžinta reikšmė yra naudojama, galima su ta reikšme atlikti veiksmus už kritinės sekcijos ribų.

Metodas get\_storage() grąžina rodyklę į objektą, kuris saugomas monitoriaus, ir per grąžintą rodyklę objektą kitos gijos gali tvarkyti kaip nori — nėra garantijų, kad grąžintas objektas bus modifikuojamas su kritinės sekcijos apsauga. Geriausias variantas — sukurti monitoriaus viduje esančio sąrašo kopiją ir grąžinti ją, tuomet nebeliks bendrų duomenų. Galima operacijas, kurias su vidine struktūra norima atlikti už monitoriaus ribų, perkelti į monitoriaus klasę, kad ji būtų vienintelė, kuri tvarko duomenis.

Metodas, tvarkantis monitoriaus saugančius duomenis, gali priimti parametrą — funkciją, kuri bus kviečiama su vidine monitoriaus struktūra. Parametru perduota funkcija gali „pavogti“ nuorodą į vidinę struktūrą.

Kada nenaudoti lygiagrečiojo programavimo? Kai gaunamas programos pagreitėjimas nevertas: sudėtingesnio programos palaikymo; didesnių kūrimo kaštų; Kiti faktoriai: OS gijų valdymas užima laiko; Per trumpas vienos gijos užduoties vykdymo laikas; Per daug gijų gali išnaudoti visus sistemos resursus (gijos dėklas užima ~1MB atminties); Per dažnas konteksto perjungimas užima laiko;

# Atviri klausimai egzamine – monitoriai

## Galimi kritinės sekcijos variantai visose nagrinėtose kalbose;

Kritinė sekcija yra kodo fragmentas, kuris yra vykdomas ne daugiau kaip vienos gijos vienu metu. Monitorius yra duomenų struktūra, skirta nuo lygiagrečios prieigos saugoti duomenis. Jei reikia kurti monitorių pačiam, tam dažnai naudojamos kritinės sekcijos. Realiose programose dažniausiai kritinės sekcijos reikalingos būtent duomenų apsaugai, dėl to programuotojo tikslas ir turimos priemonės kartais prasilenkia.

Daugumoje programavimo kalbų sukurtos priemonės kritinei sekcijai kurti. Ja pasinaudojant galima susikurti savo monitorių. Rust programavimo kalboje yra tik monitoriaus duomenų struktūra (savo kurti nereikia), kuri kartu teikia ir kritinės sekcijos apsaugą. C++ klasės mutex objektai – teikia kritinės sekcijos sąsają C# kiekvienas objektas gali veikti kaip monitorius – teikia kritinės sekcijos sąsają Go tipas sync.Mutex – teikia kritinės sekcijos sąsają Rust tipas sync::Mutex – teikia monitoriaus sąsają

## Užraktų panaudojimas, aklavietės;

Jei naudojamas kritinės sekcijos variantas, tokia priemonė vadinama užraktu. Užraktas gali leisti pažymėti kritinės sekcijos ribas — tokiu atveju reikalingas rakinimo kintamasis. Jei skirtingose programos vietose užraktams naudojamas tas pats rakinimo kintamasis, tai vienu metu galima vykdyti tik vieną iš kodo blokų. Jei kalbos priemonės leidžia naudoti kritinės sekcijos žymėjimą, rekomenduojama jį ir naudoti, nes, pvz., įvykus išimtinei situacijai (exception) kritinės sekcijos viduje užraktas nelieka užrakintas, nors kritinės sekcijos kodas ir nebaigtas vykdyti.

Nepriklausomai nuo to, ar užraktas užrakinamas priėjus sinchronizuotą bloką, ar naudojantis monitoriaus metodais, jei viena gija yra užrakinusi užraktą ir kita gija bando tą patį užrakinti, ji yra blokuojama, kol pirmoji gija neatrakins užrakto.

C++ turi 3 užraktų klases, reikia susikurti norimos klasės objektą: mutex — rakinimas vykdomas kviečiant metodus; lock\_guard — rakinimas vykdomas sukuriant / sunaikinant objektą; unique\_lock — pirmų dviejų kombinacija, lanksčiausias. C# turi klasę Monitor, rakinimas vykdomas kviečiant statinius metodus (objektas nekuriamas, rakinimui naudojamas kitas objektas). Go turi tipą Mutex, analogiškas C++ mutex.

## Lenktynių sąlygų sprendimo būdai;

Lenktynių sąlygos dėl netinkamo interfeiso: Apsaugoti visus metodus nuo lygiagrečios prieigos gali nepakakti, norint apsaugoti duomenis nuo lygiagrečios prieigos. Svarbu, kaip metodai yra kviečiami.

Pavojus kyla, kai iš funkcijos ar metodo grąžinama nuoroda į monitoriumi saugomus duomenis. Jei yra metodas, kuris tiesiogiai grąžina tokius duomenis, kitoje programos vietoje, kur grąžinta reikšmė yra naudojama, galima su ta reikšme atlikti veiksmus už kritinės sekcijos ribų.

Metodas get\_storage() grąžina rodyklę į objektą, kuris saugomas monitoriaus, ir per grąžintą rodyklę objektą kitos gijos gali tvarkyti kaip nori — nėra garantijų, kad grąžintas objektas bus modifikuojamas su kritinės sekcijos apsauga. Geriausias variantas — sukurti monitoriaus viduje esančio sąrašo kopiją ir grąžinti ją, tuomet nebeliks bendrų duomenų. Galima operacijas, kurias su vidine struktūra norima atlikti už monitoriaus ribų, perkelti į monitoriaus klasę, kad ji būtų vienintelė, kuri tvarko duomenis.

Metodas, tvarkantis monitoriaus saugančius duomenis, gali priimti parametrą — funkciją, kuri bus kviečiama su vidine monitoriaus struktūra. Parametru perduota funkcija gali „pavogti“ nuorodą į vidinę struktūrą.

# Atviri klausimai egzamine – OpenMP

## Gijų paleidimas;

OpenMP – tai papildomos priemonės, suteikiančios galimybes kurti lygiagrečias programas Fortran, C, C++ kalbomis; OpenMP skirta kurti lygiagrečias programas, kuriose procesai (gijos) naudojasi bendra atmintimi;

Lygiagrečioje srityje vykdomų gijų skaičių galima nurodyti programos vykdymo metu. Visos lygiagrečios srities gijos vykdo tą patį struktūrinį bloką. Visos lygiagrečios srities gijos pradedamos vykdyti tuo pačiu metu ir vykdomos lygiagrečiai. Lygiagreti sritis baigiama vykdyti, kai baigiamos vykdyti visos tos srities gijos. Kiekvienai lygiagrečios srities gijai suteikiamas unikalus numeris; master gijos numeris – 0. Programos vykdymo metu nėra galimybių keisti gijos numerį.

## Sinchronizacijos priemonės

OpenMP kritinės sekcijos apsaugai naudojama kompiliatoriaus direktyva omp critical. Direktyva pažymimas kodo blokas, kuris turi vienu metu būti vykdomas vienos gijos. critical direktyvai galima nurodyti pavadinimą. Blokai su vienodais pavadinimais laikomi tos pačios kritinės sekcijos dalimi. Visos critical sritys be vardo taip pat laikomos tos pačios kritinės sekcijos dalimi.

omp\_lock\_t tipas turi su savimi susietą užraktą, kuriuo galima manipuliuoti programos vykdymo metu. omp\_lock\_t tipas nėra klasė ir savo metodų neturi. Užrakto valdymui naudojamos OpenMP funkcijos, kurioms perduodama rodyklė į omp\_lock\_t kintamąjį.

# Atviri klausimai egzamine – Go

## Žinučių siuntimas, aklavietės;

Paskirstytos atminties modelis — toks atminties modelis, kai kiekviena gija naudojasi tik savo atskira atmintimi. Gijos nesinaudoja jokiais bendrais resursais (bendrais kintamaisiais, failais ir kt.). Komunikacijai tarp gijų naudojamas koks nors žinučių apsikeitimo mechanizmas (tinklas, kanalai ar kt.)

Apsikeičiant pranešimais vienas procesas siunčia žinutę, kitas — laukia žinutės. Jei procesas laukia žinutės, kurios niekas nesiųs arba siunčia žinutę, kurios niekas nelaukia, gali susidaryti aklavietė (deadlock).

## Gijų paleidimas

Paskirstytos atminties modelis — toks atminties modelis, kai kiekviena gija naudojasi tik savo atskira atmintimi. Gijos nesinaudoja jokiais bendrais resursais (bendrais kintamaisiais, failais ir kt.). Komunikacijai tarp gijų naudojamas koks nors žinučių apsikeitimo mechanizmas (tinklas, kanalai ar kt.

Naudojant paskirstytos atminties modelį nebereikalinga duomenų apsauga nuo lygiagrečios prieigos. Kadangi monitorius saugo duomenis nuo lygiagrečios prieigos, o jos paskirstytos atminties modelyje nebūna, tai monitoriai paskirstytoje atmintyje nereikalingi.

Kadangi bendros atminties nėra, o procesai siuntinėja vienas kitam žinutes, tai galima paleisti keletą procesų (ne gijų) skirtinguose kompiuteriuose, sujungti juos tinklu, ir skaičiavimų pajėgumai nebus apriboti vienu kompiuteriu.

Dažnai paleidžiant procesą jam parametru perduodami reikalingi kanalai.

Viena gija yra vienas nuoseklus procesas, o tarpusavyje jie veikia lygiagrečiai. Gijos viena kitai siunčia žinutes pasinaudojant kanalais (channels). Kanalas yra žinučių siuntimo mechanizmas. Gijos gali į kanalą rašyti arba iš kanalo skaityti duomenis.

Atviri klausimai egzamine – MPI

## Superkompiuteriai;

**Superkompiuteris** supercomputer. Kompiuteris, savo paleidimo metu esantis vienas pirmaujančių pasaulyje pagal skaičiavimo galią.

Superkompiuterių OS:

Top 500 superkompiuterių operacinių sistemų pasiskirstymas: Linux — 500 Linux yra efektyvi sistema aukštos spartos kompiuterių tinklui. Linux yra nemokama — Windows licencijos kainuotų daug pinigų. Jei kažkas neveikia Linux sistemoje, galima pataisyti pačiam. Perkrauti reikia labai retai.

## Procesų sinchronizacija;

MPI standartas neapibrėžia sinchronizavimo pilnumo. Sinchronizavimas gali būti:

Pilnai sinchronizuotas — siuntėjas laukia, kol gavėjas gaus, gavėjas laukia, kol siuntėjas išsiųs. Buferizuotas — siuntėjas laukia, kol žinutė bus įrašyta į buferį, gavėjas laukia, kol žinutė atsiras buferyje.

Kolektyvinė komunikacija sukuria sinchronizacijos tašką tarp procesų, t.y., visi procesai turi pasiekti tam tikrą tašką kode prieš tęsdami darbą

## Procesų paleidimas;

## Kolektyvinė komunikcija (Broadcast, Scatter, Gather);

**Kolektyvinė komunikacija** (angl. collective communication) — tokia komunikacija tarp procesų, kurioje dalyvauja visi komunikatoriaus procesai. Ji sukuria sinchronizacijos tašką tarp procesų, t.y., visi procesai turi pasiekti tam tikrą tašką kode prieš tęsdami darbą.

MPI transliacija (**broadcast**) - transliacija yra kolektyvinės komunikacijos rūšis. Vienas procesas visiems komunikatoriaus procesams siunčia tą pačią žinutę. MPI transliacija dažnai naudojama perduoti naudotojo įvestus duomenis ar konfigūracijos parametrus visiems procesams

**Scatter** - persiunčia duomenis, saugomus viename procese, dalimis visiems komunikatoriaus procesams. priima duomenų masyvą ir jo elementus paskirsto visiems procesams jų numerio didėjimo tvarka. Procesas, iš kurio siunčiama, taip pat gaus dalį duomenų, nepaisant to, kad jame yra visi duomenys

**Gather** - surenka duomenis, saugomus skirtinguose procesuose, viename procese į bendrą duomenų rinkinį. Gather yra funkcija, atvirkščia Scatter. Gather priima kiekviename procese esantį masyvą ir jų elementus surašo į nurodytą proceso masyvą. Duomenys bus paimami ir iš surenkančio proceso.

# Atviri klausimai egzamine – asinchroninis programavimas

## C# Task ir JavaScript Promise paskirtis ir darbo su jais principai;

**C#:**

.NET karkasas turi klasę Task. Klasė yra naudojama asinchroninėms užduotims kurti ir vykdyti. Taip pat palaikomi async ir await raktiniai žodžiai, skirti palengvinti darbą su asinchroninėmis užduotimis. Kuriant asinchronines programas nėra būtina pačiam kurti Task objektų ir naudoti tęsinio metodų ar kitų priemonių. Užduotis kurti ir rezultatus iš jų paimti galima naudojantis raktiniais žodžiais async ir await. Metodą galima pažymėti raktiniu žodžiu async. Tokio metodo grąžinamas tipas turi būti įdėtas į Task tipą, kitaip C# kompiliatorius nekompiliuos programos. async metodo return sakinys turi grąžinti ne patį Task objektą, o reikšmę, kuri jame saugoma — C# kompiliatorius ją įdės į Task objektą ir grąžins jį.

**JavaScript Promise:**

ES6 standartas palengvino darbą su asinchroninėmis funkcijomis. Buvo pridėtas API, panašus į C# Task modelį. Asinchroninės funkcijos rezultatą, kuris dar nebūtinai suskaičiuotas, savyje turi Promise objektas. Darbas su Promise rezultatu vykdomas tam objektui kviečiant metodą then. then metodas grąžina naują Promise objektą, todėl then kreipinius galima jungti į grandinę, panašiai kaip C# ContinueWith

Naudojant Promise nedidėja funkcijų sudėjimo gylis. Lengviau pačiam susikurti asinchroninę funkciją. Vis tiek reikia naudoti callback funkcijas then parametrams. JavaScript varikliai palaiko tik vieną giją, tad asinchroninės funkcijos vykdomos pakaitomis įvykių cikle, o ne lygiagrečiai vienu metu. Dėl šios priežasties asinchronines funkcijas JavaScript verta naudoti tada, kai jų vykdymas blokuoja giją, nes reikia laukti, pvz., atsakymo iš serverio

## async ir await paskirtis ir darbo su jais principai;

Naudojant async / await nebelieka callback funkcijų. Klaidų apdorojimas tampa analogiškas nuosekliam kodui: įvykus išimtinei situacijai galima ją suvaldyti su try-catch bloku. JavaScript, asinchroninės funkcijos vykdymo metu įvykus išimtinei situacijai, grąžina tą pačią klaidą, o ne įdeda į kitą klaidą kaip C#. Kodą lengva skaityti, nes jis panašus į nuoseklų kodą

## Įvykių ciklas;

JavaScript varikliai palaiko tik vieną giją, tad asinchroninės funkcijos vykdomos pakaitomis įvykių cikle, o ne lygiagrečiai vienu metu. Dėl šios priežasties asinchronines funkcijas JavaScript verta naudoti tada, kai jų vykdymas blokuoja giją, nes reikia laukti, pvz., atsakymo iš serverio

## Gijos naršyklėje – kokia sąsaja naudojama, kokie trūkumai;

Asinchroninis programavimas naršyklėje leidžia išnaudoti vieną CPU giją. Naršyklė suteikia programuotojui sąsają kurti gijas. Gijos kuriamos pasinaudojant WebWorker sąsaja. WebWorker nėra ECMAScript standarto dalis, jį aprašo W3C ir WHATWG. WebWorker tipo objektas turi sąsają kurti gijoms ir apsikeisti žinutėmis tarp pagrindinės gijos ir sukurtos su WebWorker. Gijoje paleistas kodas turi apribojimų, pvz., jis negali dirbti su naršyklėje rodomu turiniu, kodas turi būti atskirame JavaScript faile.

# Atviri klausimai egzamine – programavimas GPU

## GPU lygiagretumo galimybės;

CPU ir GPU naudoja skirtingą atmintį, todėl duomenis reikia kopijuoti iš vienos atminties į kitą. Darbas su GPU atmintimi CUDA vykdomas naudojant CUDA funkcijas atminties valdymui.

## Gijų paleidimas;

GPU funkcija iš CPU kviečiama nurodant gijų bloko, kuriame bus vykdoma programa, dydį, bei gijų kiekį bloke. Jei funkcija kviečiama nurodant blokų ir gijų kiekius 2, 5, bus paleidžiama 10 gijų (2 blokai po 5 gijas). GPU gijas vykdo srautiniai multiprocesoriai, kurie priima gijų blokus ir juos vykdo. Bloko gijos vykdomos multiprocesoriuje lygiagrečiai, o kiekvienas multiprocesorius gali vykdyti vieną bloką vienu metu.

## Atominės operacijos;

CUDA remiasi bendros atminties modeliu. Rašymas iš kelių CUDA gijų į tą pačią atmintį yra neapibrėžtas (undefined behaviour). CUDA turi atominių operacijų rinkinį, skirtą operacijų atomiškumui garantuoti. Visos CUDA atominės operacijos vykdomos viena tranzakcija ir iš kitų gijų tarpinė operacijos būsena nematoma. Visos CUDA atominės operacijos dirba tiek su globalia, tiek su bendra atmintimi. Dauguma atominių operacijų priima rodyklę į atmintį, ten saugomą reikšmę modifikuoja ir grąžina seną reikšmę

## Duomenų paskirstymas CUDA gijoms;

CPU ir GPU naudoja skirtingą atmintį, todėl duomenis reikia kopijuoti iš vienos atminties į kitą. Darbas su GPU atmintimi CUDA vykdomas naudojant CUDA funkcijas atminties valdymui. cudaMalloc - GPU atmintyje išskiria nurodytą kiekį atminties. cudaFree - Atlaisvina GPU išskirtą atmintį. cudaMemcpy - Kopijuoja duomenis tarp CPU ir GPU. cudaMemcpyHostToHost iš CPU į CPU cudaMemcpyHostToDevice iš CPU į GPU cudaMemcpyDeviceToHost iš GPU į CPU cudaMemcpyDeviceToDevice iš GPU į GPU

## Duomenų perkėlimas tarp CPU ir GPU atminties Thrust;

Thrust palaiko vektorius CPU ir GPU atmintyje. CPU vektorių atitinka klasė host\_vector. GPU vektorių atitinka klasė device\_vector. Priskiriant device\_vector objektą host\_vector objektui arba atvirkščiai, kopijos konstruktorius pasirūpina duomenų perkėlimu tarp CPU ir GPU.

# Atviri klausimai egzamine – funkcinis programavimas

## map, filter operacijų lygiagretinimas;

Naudotis lygiagrečiaisiais map, filter ir reduce galima daugeliu atvejų, tačiau nevisais.

**map ir filter lygiagretinti** galima tada, kai parametru paduodamos funkcijos gali būti vykdomos lygiagrečiai ir nesukelti lenktynių sąlygų. Daugeliu atvejų tai būna grynos funkcijos, bet nebūtinai turi būti tokios. reduce funkcijoms kyla papildomas reikalavimas: jos turėtų būti monoidai

## Grynų funkcijų lygiagretinimas;

Visos funkcijos, jei nepažymėta kitaip, yra grynos, dėl to jų kvietimą galima pakeisti grąžinama reikšme, jei funkcija kviečiama su tais pačiais parametrais. Jei dvi funkcijos yra grynos ir viena nenaudoja kitos rezultatų, jas visada galima vykdyti lygiagrečiai be jokių apsaugos priemonių.

# Atviri klausimai egzamine – Python

## Gijų ir procesų telkiniai;

Gijos Python atvaizduojamos Thread klase. Python gijos tinka tada, kai lygiagrečiai atliekami veiksmai yra daugiausiai įvestis/išvestis (IO bound). Python turi modulį multiprocessing, kuris leidžia kurti procesus. Kiekvienas procesas yra atskiras Python interpretatoriaus procesas, todėl jais naudojantis galima išnaudoti sistemos lygiagrečias galimybes.

Klasė Process turi tuos pačius metodus, kaip Thread klasė, bet paleidžia naują procesą, ne giją.

Apsikeitimas duomenimis tarp procesų vykdomas tokiomis priemonėmis: Eilėmis Queue; Kanalais Pipe

## Globalus interpretatoriaus užraktas;

CPython interpretatorius turi globalų interpretatoriaus užraktą (global interpreter lock (GIL)), kuris apsaugo python objektus nuo lygiagrečios prieigos.

GIL reikalingas, nes python atminties valdymas nėra saugus lygiagrečiai prieigai. Dėl GIL python programos yra vykdomos vienoje gijoje, o programos viduje sukurtos gijos yra vykdomos keičiant kontekstą vienoje CPU gijoje.

## Daugiaprocesiškumo modelis;

Python turi modulį multiprocessing, kuris leidžia kurti procesus. Kiekvienas procesas yra atskiras Python interpretatoriaus procesas, todėl jais naudojantis galima išnaudoti sistemos lygiagrečias galimybes. Jis veikia paskirstytos atminties principu

## Daugiaprocesiškumui naudojamos eilės ir kanalai;

Apsikeitimas duomenimis tarp procesų vykdomas tokiomis priemonėmis: Eilėmis Queue; Kanalais Pipe

**Duomenų perdavimas naudojant eiles**. Sukuriamas multiprocessing. Queue objektas, kurio sąsaja yra beveik identiška standartinės Python eilės sąsajai. Į eilę objektai įrašomi metodu put. Iš eilės objektai išimami metodu get. Vienas procesas į eilę rašo duomenis, kitas iš eilės duomenis ima

**Duomenų perdavimas naudojant kanalus**. Kviečiama funkcija multiprocessing. Pipe, kuri sukuria du kanalo galus (Connection tipo) ir juos grąžina: parent\_conn, child\_conn = Pipe() Kanalo galais galima naudotis kviečiant jų metodus send ir recv. Komunikacija kanalu galima naudotis siunčiant ir gaunant žinutes iš bet kurio kanalo. Kanalai yra One2One tipo — bandymas į tą patį kanalą rašyti ar iš to paties kanalo skaityti iš kelių procesų vienu metu gali sugadinti siunčiamus duomenis.

## Python gijos ir jų galimybės bei trūkumai;

Gijos Python atvaizduojamos Thread klase. Klase galima naudotis dviem būdais — paveldint klasę ir užklojant run metodą, arba kurti Thread objektą jam perduodant norimą vykdyti funkciją target parametru. Klasė turi klasikinius metodus: start() join() is\_alive()

Python gijos tinka tada, kai lygiagrečiai atliekami veiksmai yra daugiausiai įvestis/išvestis (IO bound).

# Atviri klausimai egzamine – Aktorių modelis

## Priemonės duomenų modifikavimui iš kelių procesų;

Elixir Funkcinio programavimo kalba, veikianti su Erlang virtualia mašina, skirta lengvasvoriams procesams vykdyti. Tinka realaus laiko sistemoms, paskirstytoms sistemoms, klaidas toleruojančioms sistemoms. Turi pritaikytą sintaksę žinučių siuntimui ir priėmimui.

## Apsikeitimas duomenimis tarp aktorių;

Kiekvienas aktorius turi savo „pašto dėžutę“, į kurią gali priimti žinutes. Kiekvienas aktorius gali kitam aktoriui siųsti žinutes, iš kitų aktorių priimti žinutes, jas apdoroti. Žinučių apdorojimas vykdomas nuosekliai. Žinutės yra siunčiamos tiesiogiai procesui, t. y., egzistuoja kintamasis, kuris identifikuoja procesą (proceso ID, procesą atitinkantis objektas ar pan.), ir tas identifikatorius naudojamas žinutei siųsti. Žinutės gavėjas „mato“ tik pačias žinutes; jei reikia žinoti, kuris procesas siuntė, tai reikia tą realizuoti pačiam.

## Aktorių pašto dėžučių veikimo principai;

Procesų pašto dėžutėse laikomos visos dar neapdorotos žinutės. Procesas vykdomas kaip žinučių apdorojimo ciklas. Pašto dėžutė elgiasi kaip buferis, t. y., siunčiantis procesas žinutę įrašo ir toliau tęsia darbą, o gavėjas žinutę apdoros, kai ateis tos žinutės eilė. Jei pašto dėžutė tuščia, o procesas bando apdoroti sekančią žinutę, jis yra blokuojamas, kol ta žinutė ateis. Dėl šios priežasties gali susidaryti aklavietės situacija (deadlock).

Pašto dėžučių programuotojas tiesiogiai nepasiekia. Pašto dėžučių buferiai elgiasi panašiai, kaip buferizuoti kanalai, t. y., galima siųsti žinutes, kol jos nepriimamos.

Aktorių modelyje pašto dėžutė paslėpta nuo programuotojo, negalima nustatyti jos dydžio, kaip sinchronizuotiems kanalams. Programuotojui reikia pačiam užtikrinti, kad nebus perpildoma pašto dėžutė ir jai užteks atminties.

Praktinė užduotis:

Man buvo su c#, c++, Go.. pasirinktinai padaryti stacka, 10 threadu paleidi, kiekvienas po 5 kartus į console rašo savo id ir background threadas 11 tas kol anie rašo savo id, jis printina maxa iš stacko, stacko dydis lygus 10threadu \* 5. Antra dalis removint, riboti stacka iki 10 ir dėti taip pat, tik, kad dabar backr threadas removina 40 kartų.

Teoriniai klausimai:

Teoriniai tai realiai kaip elgtumeisi įvair. situacijose. Pvz vienas buvo - Turi dvi situacijas A - tavo threadai atlieka sunkius taskus, ilgai trunkančius B - greitai dirba, bet daug laiko praleidžia laukdami atsakymo iš serv Tai klausima kaip optimaliausiai išnaudotum savo cpu

Tokiose situacijose, kur vieni thread'ai atlieka sunkias užduotis, o kiti yra efektyvūs, bet laukia atsakymo iš serverio, gali būti naudinga panaudoti lygiagrečio programavimo metodikas. Štai keletas galimybių, kaip optimaliausiai išnaudoti savo CPU šioje situacijoje:

1. **Lygiagrečių thread'ų naudojimas:**
   * Panaudokite kelis thread'us (gijas) atlikdami sunkias užduotis (A).
   * Kitus thread'us panaudokite greitoms, bet laukiantiems užduotims (B).
2. **Lygiagrečių procesų naudojimas:**
   * Jei galite, galite panaudoti lygiagrečių procesų modelį, kur kiekvienas procesas vykdo savo užduotis.
   * Tai gali būti naudinga, jei užduotys yra nepriklausomos ir nesidalina resursais.
3. **Asinchroninio programavimo modelis:**
   * Panaudokite asinchroninio programavimo modelį, kuris leidžia vienam thread'ui tęsti darbą kitų laukiant.
   * Tai gali būti naudinga, jei yra galimybė efektyviai valdyti atsakymų laukimo laiką, pavyzdžiui, naudojant tokias technologijas kaip futures ar promises.
4. **Panaudojimas gijų grupei (thread pool):**
   * Sukurk gijų grupę, kurioje yra tiek thread'ų, kiek tavo sistema gali efektyviai tvarkyti.
   * Gijų grupė leis optimaliau valdyti išteklius ir užtikrinti efektyvų lygiagrečių operacijų vykdymą.
5. **Prioritizavimas:**
   * Suteikite aukštą prioritetą sunkioms užduotims, taip užtikrindami, kad jos būtų vykdomos kuo greičiau.
   * Mažesnio prioriteto užduotims leiskite laukti, kol resursai bus prieinami.

Svarbu atkreipti dėmesį į tai, kad veiksmingas lygiagrečios programavimo modelio pasirinkimas priklauso nuo konkrečių užduočių, kurias vykdys thread'ai, savybių. Be to, reikėtų atkreipti dėmesį į galimus sinchronizacijos ir gijų saugumo klausimus, kad būtų išvengta konfliktų tarp thread'ų.