



Универзитет у Београду  
Електротехнички факултет

***КОНКУРЕНТНО И ДИСТРИБУИРАНО  
ПРОГРАМИРАЊЕ***

Домаћи задатак

**Дистрибуирана обрада  
математичких израза**

студент:  
Вуковић Данило 61/06

Прва верзија:  
Место одбране:

31. Мај 2009.  
Електротехнички факултет, лабораторија 25

## Садржај

<i>Текст задатка</i>	2
<i>Опис решења</i>	3
<i>Имплементација решења</i>	4
<i>UML дијаграми</i>	5
<i>Упутство</i>	7

## Текст задатка

Пројектовати дистрибуирани рачунарски систем који треба да омогући дистрибуирану обраду и међусобну удаљену синхронизацију временски захтевних послова математичке обраде.

Програм треба да ради у систему који се састоји од више рачунара повезаних у LAN (Local Area Network) или WAN (Wide Area Network).

У систему постоји три типа програма:

1. Централни сервер који служи за праћење рада извршавања дистрибуиране обраде, чување информација о доступним чворовима у мрежи и могућност поновног

стартовања појединих послова.

2. Радна станица која од централног сервера добија послове које треба одрадити.

3. Кориснички програм, који задаје посао који треба урадити и његове параметре.

Процес започиње тако што кориснички програм задаје посао који је потребно извршити. Овај посао представља математички израз који је потребно израчунати. Након тога кориснички

програм контактира централни сервер коме прослеђује тај посао и параметре потребне за његову обраду. Када централни сервер прими посао и његове параметре он га прослеђује једној радној станици, чека резултат обраде и враћа комплетан резултат клијенту.

Када радна станица прими посао, креира нит намењену извршавању тог посла, и у тој нити започиње са његовим извршавањем на основу примљених параметара. Посао се извршава на радној станици тако што покренута нит радне станице позива одговарајућу методу инстанце класе за паралелно процесирање математичких израза (имплементира интерфејс *Equations*). Да би се обављало паралелно процесирање потребно је прво иницијализовати ову инстанцу адресама и портovima преосталих класа за паралелно израчунавање који се налазе на осталим радним станицама (интерфејс *Connector*), програму за паралелно израчунавање је потребно доделити један слободни серверски порт (*setPort*) и покренути их свакој радној станици (метод *connect()*).

Инстанцирање класа које раде математичку обраду и повезивање на радној станици обезбеђује се користећи статичке методе класе *Creator*. Када се заврши израчунавање математичких израза, радна станица прикупља излаз рачунања, пакује га у одговарајућу датотеку коју прослеђује серверу, као и резултате извршавања. Да би се обезбедило прикупљање информација о активним радним станицама централни сервер на сваких *x* секунди проверава да ли је нека радна станица исправна. Уколико сервер утврди да нека радна станица која је

до тог тренутка обављала математичка израчунавања није више исправна, посао који је обављала та радна станица прослеђује некој слободној радној станици. Након слања захтева за обраду кориснички програм може да раскине везу са централним сервером. Беза може бити раскинута гашењем програма или затварањем комуникационог канала. Када се следећи пут повеже кориснички програм може да тражи резултате претходно задате обраде. Треба обезбедити да централни сервер може да у паралели да прима већи број послова које је потребно обрадити. Кориснички програм може од централног сервера да тражи информације о статусу посла, а може да тражи и резултате. Одмах по стартовању радне станице шаљу централном серверу информацију о томе да су стартоване и број послова које могу у паралели да обрађују. Број послова које радне станице могу да приме је аргумент који им се поставља приликом покретања.

Параметри посла које клијент задаје су: математичка операција коју је потребно извршити, датотека у којој се налази корисникова матрица коју је потребно обрадити, датотека са низом вредности које је потребно још применити у операцији (параметар је опциони), име датотеке у коју је потребно сместити резултате операције коју је са радне станице треба пребацити на клијентски рачунар да које представљају резултате. Ови параметри могу да се задају или путем корисничког интерфејса или путем текстуалне датотеке.

Централни сервер у лог уписује време када је пристигао сваки посао, број под којим је посао сачуван, име рачунара коме је посао прослеђен, време када је посао завршен и његов тренутни статус. Статус посла може да буде: *Ready* – приспео на сервер али није никоме прослеђен, *Scheduled* – тренутно се прослеђује радној станици, *Running* – извршавање је у току, *Done* – посао се успешно извршио, *Failed* – посао није могао да се изврши (емитовао је неки изузетак), *Aborted* – корисник је одустао од извршавања посла.

Проблем речити користећи мрежну комуникацију у програмском језику Јава. Решење треба да буде независно од посла који се обавља. За сваки од ова три типа рачунара треба да постоји одговарајући графички кориснички интерфејс (GUI треба да буде развијен користећи Јава **SWING** компоненте). Радна станица треба да има могућност покретања и без корисничког интерфејса.

## Опис решења

успешно завршено. Одмах при завршетку посла, обавештава сервер да је обрада готова. Датотека се шаље серверу, сервер на захтев клијента шаље му резултат.

Постоје три врсте програма: сервер, радна станица и клијент. Сервер је програм који представља везу између клијента и радних станица. Сервер прикупља послове од клијената и распоређује их по радним станицама да се извршавају. Када радне станице обаве извршавање посла, обавештавају сервер, и он резултат шаље клијенту.

Сервер има могућност рада са више клијената и клијенти могу да шаљу произвољан број послова на извршавање. Сваки клијент има свој идентификатор којим се идентификује на серверу. На основу тога сервер води рачуна о власништву послова. Сервер опслужује клијенте на порту који се задаје приликом покретања сервера. Том приликом се задаје и порт преко кога се повезују и радне станице. Радне станице немају идентификаторе, већ им сервер додељује и.д. како се која пријављује. Радна станица шаље серверу и још параметре колико може послова да извршава истовремено као и порт за комуникацију са осталим радним станицама. Портови на којима сервер ослушкује конекције задају се приликом покретања сервера.

При стартовању клијента, прва ствар која треба да се уради јесте да се подеси адреса и порт сервера и идентификатор клијента. Ако је све успешно урађено, појавиће се екран са опцијама за рад са пословима. Послови су математичке операције са матрицама. Постоји неколико расположивих функција за израчунавање. За израчунавање се користи библиотека *KDPMath.jar*. Параметри посла се задају убацивањем параметара у одговарајућа поља или путем датотеке. Клијент може у сваком тренутку да види списак његових послова и њихов тренутни статус. Ако је посао завршен, може да га довуче са сервера. Омогућено је и да ако клијент жели да неки посао прекине, то и уради.

Радне станице се при стартовању иницијализују адресом и портом сервера, портом за комуникацију са осталим радним станицама и максималним бројем послова које истовремено могу да извршавају. Када се повежу, сервер за сваку станицу креира нит путем које на сваких пет секунди проверава исправност радне станице. Радне станице, ако је неки посао захтеван, међу собом деле посао на мање задатке и тим путем се постиже убрзање. Сваки пристигли посао се извршава у паралели. За сваки посао постоји главна радна станица која прикупља резултате од осталих станица и пакује их у датотеку ако је све

## Имплементација решења

Цео пакет је развијен у програмском језику *Java*, уз помоћ програмерског окружења *Eclipse*. За мрежну комуникацију коришћени су асинхрони сокети из пакета *Java NIO*. Главна предност овога решења је што нема блокирања код позива основних операција за комуникацију – *accept()*, *connect()*, *send()*, *receive()*. Због тога, класе и функције за комуникацију из поменутог пакета, су морале да претрпе значајне промене. Остале су, наравно, компатибилне са старијим класама из *Socket* фамилије.

У новом пакету, постоје четири значајне врсте класа:

*Channels* – везе са фајловима, сокетима и другим, омогућавајући им неблокирајуће читање и евентуално упис

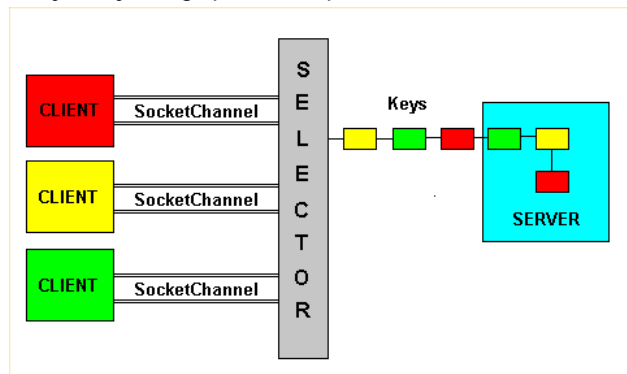
*Buffers* – објекти слични низовима, помћу којих се може директно читати или писати у канал

*Selectors* – обавештава које се операције тренутно могу извршити над каналом

*SelectionKeys* – садрже и одржавају информације о У/И догађајима који се могу извршити над каналом

Због ових карактеристика, сервери базирани на овој технологији погодни су за опслуживање великог броја захтева у секунди. Код класичних сервера са блокирајућим сокетима, при великом броју конекција, сервер више времена троши при промени контекста нити (режијски трошкови веома расту) него на опслуживању клијената. Класичан сервер за сваку конекцију мора да креира бар две нити – за читање и упис. Сто је за неколико стотина конекција велики број. С друге стране, *NIO* сервер неколико стотина конекција може да обради из једне нити! Неки тестови су показали да такви сервери могу да се носе са неколико хиљада конекција у секунди.

Клијент је одрађен, такође, са класама из новог

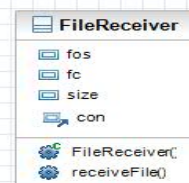
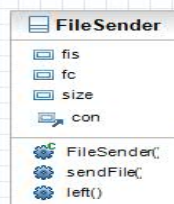
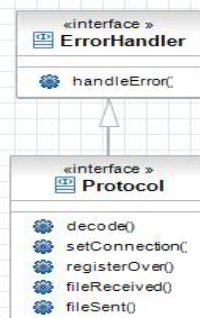
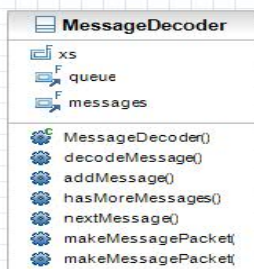
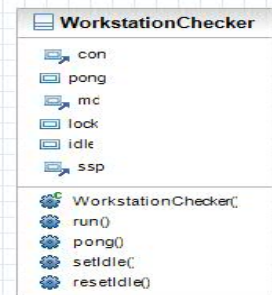
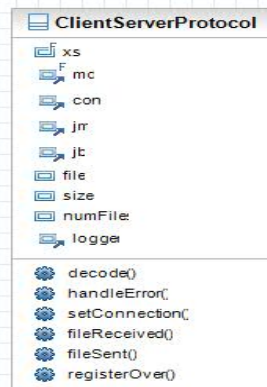
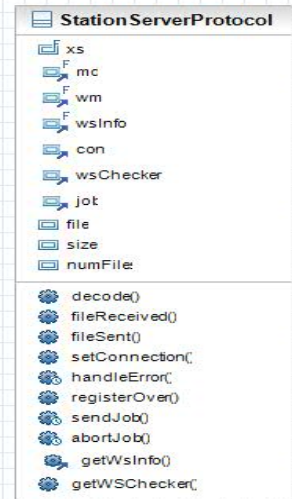
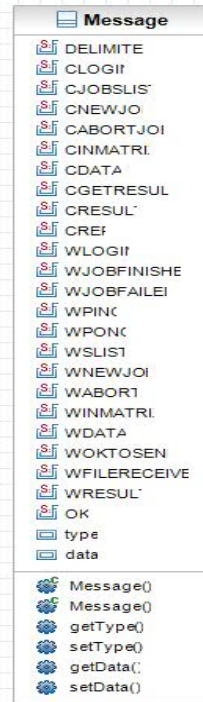


пакета, али није коришћена опција за неблокирајуће позиве. Урађен је у духу блокирајућих сокета. На њему се нећемо задржавати.

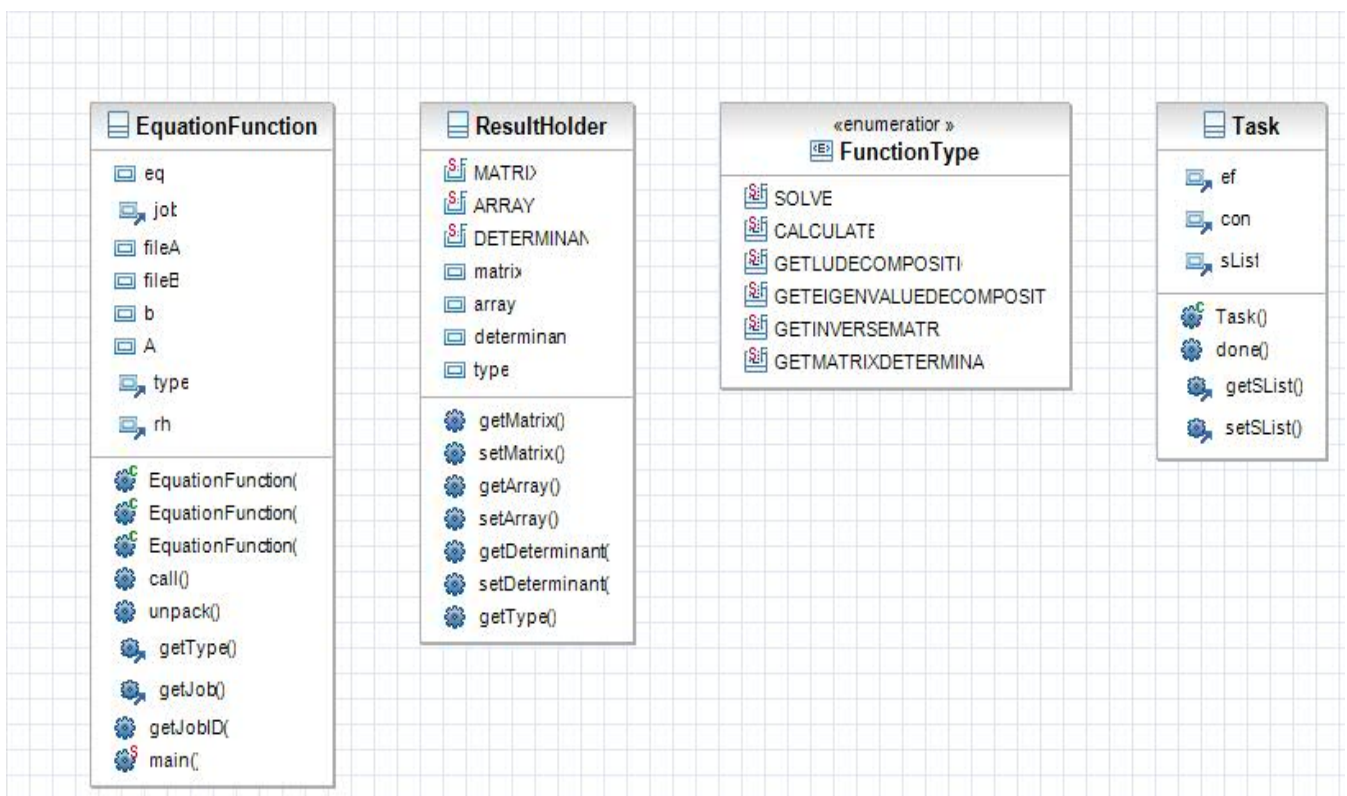
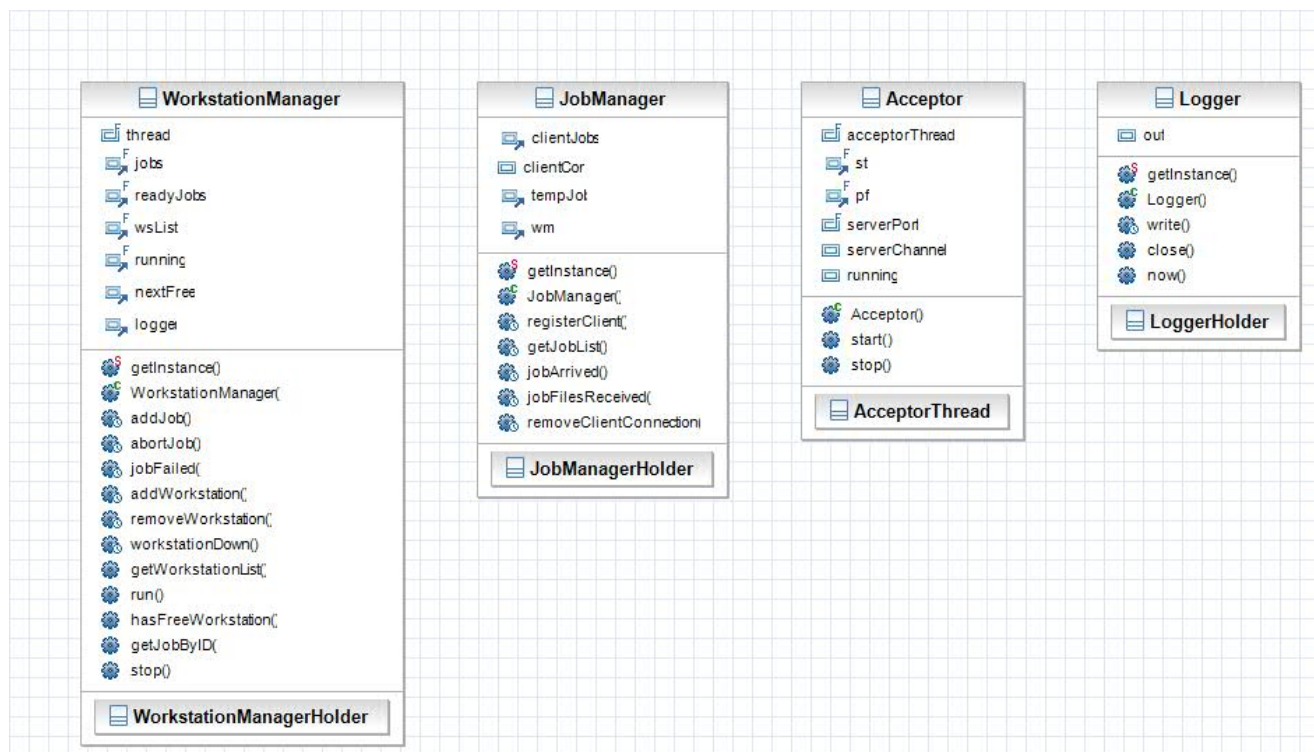
Сервер функционише на следћем принципу:

За прихватање конекција одговорна је класа *Acceptor*. Она представља нит, која сваку нову клијентску конекцију региструје код селектора. Класа *Acceptor* користи блокирајући позив *accept()*. Класа *SelectorThread* врши селекцију операција које се могу извршити над каналом. Када *Acceptor* прими конекцију, он позива *registerChannel()* методу класе *SelectorThread*. Та метода служи да пријавимо канал код селектора, како би могао да нас обавести када можемо да извршимо неку операцију над каналом која нам је од интереса. *SelectorThread* позива методу *select()* од класе *Selector* која се блокира све док се не деси бар један догађај, тј. бар један канал буде спреман за извршавање операције. Она враћа листу кључева, *SelectionKey*, која садржи информације о операцијама које се могу извршити. Класа *Connection* нам служи за обраду свих врста уписа и читања из сокет канала. Она представља и идентификацију саговорника са друге стране жице. Када канал буде спреман, извршавају се методе *handleRead()* и *handleWrite()* у зависности од типа операције. За извршавање се користи специјални базен кешираних нити, који се алоцира позивом *Factory* метода класе *Executors.newCachedThreadPool()*. За сваки тип комуникације постоји посебан протокол, у зависности ко комуницира и од кога иде комуникација. Класа *MessageDecoder* врши обраду бафера у потрази за порукама. Њу користе сви протоколи да би могли из канала да прочитају поруку. Помоћу ње се може објекат *Message* трансформисати у *XML* и даље, помоћу бафера транспортовати кроз канал. За трансформацију објеката у *XML* користи се библиотека *Xstream*. Постоје и класе за пријем и слање фајлова, *FileReceiver* и *FileSender*. *WorkstationManager* управља свим станицама и распоређује послове станицама. Када се нека станица прекине он додели посао другој станици.

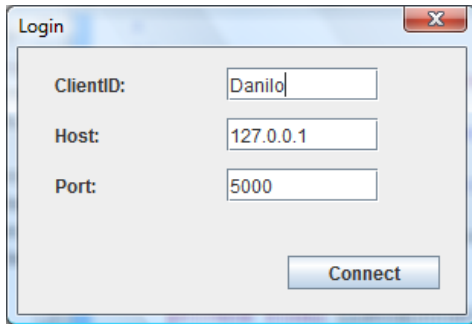
У пакету *workstation.task* налазе се класе који служе за извршавање послова добијених од сервера. Главна класа *Task*, наслеђује класу *FutureTask* која је погодна за извршавање помоћу класе *Executors*. Лако се може отказати посао и манипулисати обрађеним подацима. У класи *EquationFunction* врше се позиви библиотеке за математичка извршавања.



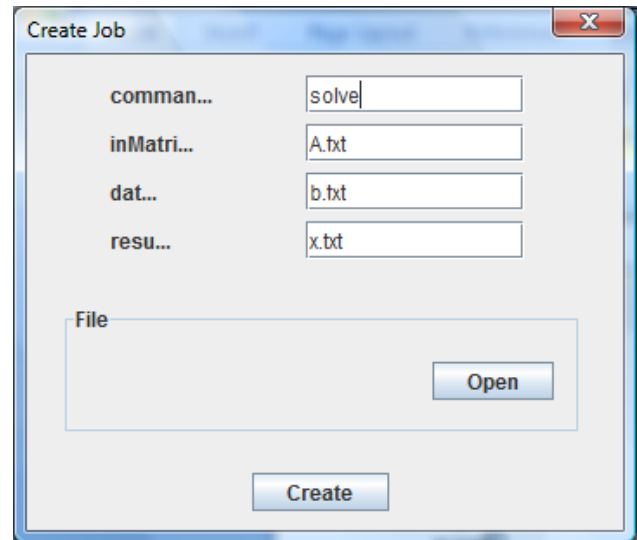




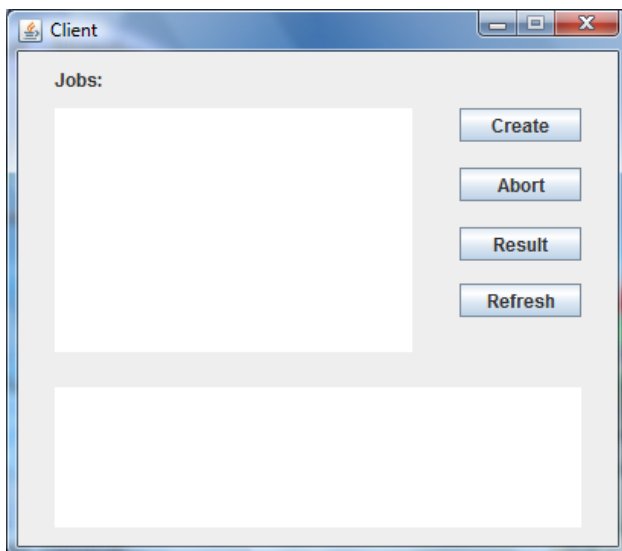
## Упутство



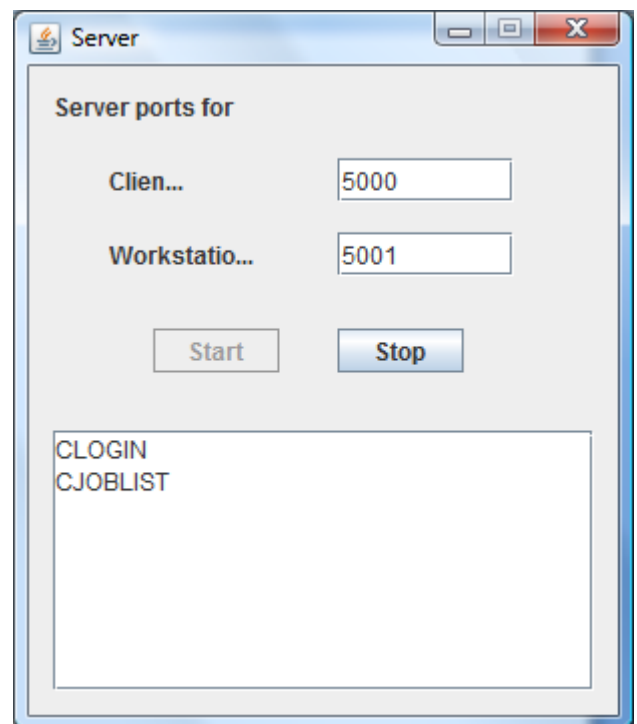
Када се стартује клијент појавиће се дијалог за приступ серверу. За поље *ClientID* треба унети своју идентификацију за преостала два поља адресу и порт сервера.



Када се појави дијалог са слике, у њему можете да подесавате параметре посла. Ако желите, можете посао да учитате из датотеке.

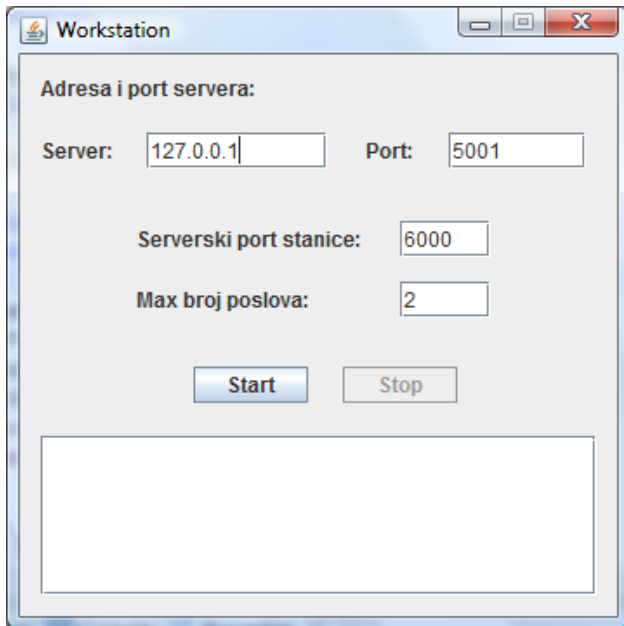


Када се улогујете, појавиће се дијалог за рад са пословима. На дугме *Create* креирате нови посао, а на остала, респективно, поништавате посао, тражите резултат израчунавања и освежавате листу послова.



Једина поља за подешавање сервера су портови на које ће се конектовати клијенти и радне станице. Када то подесите, можете да покренете сервер.





Радној станици подешавајте адресу и порт сервера, број порта са кога ће да комуницира са осталим станицама и максимални број послова које ће извршавати. Када све подесите, можете покренути станицу.