**Вступ**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*2*

###### *ІАЛЦ.462637.003.ПЗ*

Ефективність роботи в паралельних ВС великою мірою залежить від якості вирішення завдань планування, оптимізації, управління та диспетчеризації завдань.

В даному курсовому проекті досліджується і вирішується завдання планування виконання завдань у системі з топологією «Загальна шина з маркерним доступом». Дана система є досить поширеною, тому дослідження, розробка і реалізація ефективних алгоритмів планування мають велике практичне значення.

Основне завдання розробленого мною алгоритму - правильне занурення задач графа на ресурси системи. Воно повинно враховувати мінімальний трафік по загальній шині і по можливості рівну навантаження на ресурси системи.

Результатом виконання є запропонований метод рішення і реалізація середовища моделювання з використанням цього методу.

1. Огляд існуючих рішень

Алгоритм планування дає змогу планувальнику вибирати з готових до виконання потоків той ,котрий потрібно виконувати наступним залежно від стратегії планування, яку реалізують алгоритми. Їх поділяють на витісняльні та невитісняльні. Витісняльні алгоритми переривають потоки під час їхнього виконання, невитісняльні - не переривають. Деякі алгоритми відповідають лише одній із цих стратегій, інші можуть мати як витісняльний, так і не витісняльний варіанти реалізації.

**1.1.Планування за принципом FIFO.**

Невитісняльний алгоритм, у якому потоки ставлять на виконання в порядку їхньої появи у системі й виконуються до переходу в стан очікування, явної передачі керування або завершення. Чергу готових потоків при цьому організовують за принципом FIFO, тому алгоритм називають алгоритмом FIFO. Як тільки в системі створюється новий потік, його керуючий блок додається у хвіст черги. Коли процесор звільняється, його надають потоку з голови черги. У такого алгоритму багато недоліків: -він за визначенням є невитісняльним - середній час відгуку для нього може бути доволі значним.

**1.2.Кругове планування.**

Найпростішим для розуміння і найсправедливішим витісняльним алгоритмом є алгоритм кругового планування. Кожному потокові виділяють інтервал часу, який називають квантом часу і упродовж якого цьому потокові дозволено виконуватися. Коли потік усе ще виконується після вичерпання кванта часу, його переривають і перемикають процесор на виконання інструкцій іншого потоку. Коли він блокується або закінчує своє виконання до вичерпання кванта часу, процесор теж передають іншому потокові. Довжина кванта часу для всієї системи однакова. Такий алгоритм реалізувати досить легко. Для цього черга готових потоків має бути циклічним списком. Коли потік вичерпав свій квант часу, його переміщують у кінець списку, туди ж додають і нові потоки. Єдиною характеристикою, яка впливає на роботу алгоритму довжина кванта часу. На практиці рекомендують встановлювати довжину кванта в 10-100 мс.

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*3*

###### *ІАЛЦ.462637.003.ПЗ*

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*4*

###### *ІАЛЦ.462637.003.ПЗ*

**1.3.Планування із пріоритетами.**

Планування за принципом кругового чергування припускає, що всі потоки однаково важливі. В іншому разі необхідно застосовувати планування із пріоритетами. Основна ідея проста: кожному потокові надають пріоритет, при цьому на виконання ставитиметься потік із найвищим пріоритетом із черги готових потоків. Пріоритети можуть надаватися потокам статично або динамічно. Одним із підходів до реалізації планування із пріоритетами є алгоритм багаторівневих черг. У цьому випадку організовують кілька черг для груп потоків із різними пріоритетами. Рішення про вибір потоку для виконання приймають таким чином:

-якщо в черзі потоків із найвищим пріоритетом є потоки, для них слід використати якийсь простіший алгоритм планування (наприклад, кругового планування), не звертаючи уваги на потоки в інших чергах.

- якщо в черзі немає жодного потоку, переходять до черги потоків з нижчим пріоритетом.

Для різних черг можна використовувати різні алгоритми планування, крім того, кожній черзі може бути виділена певна частка процесорного часу.

**1.4.Планування на підставі характеристик подальшого виконання.**

Важливим класом алгоритмів планування з пріоритетами є алгоритми, в яких рішення про вибір потоку для виконання приймають на підставі знання або оцінки характеристик подальшого його виконання. Насамперед, слід відзначити алгоритм «перший - із найкоротшим часом виконання» (STCF), коли з кожним потоком пов’язують тривалість наступного інтервалу використання процесора і для виконання щоразу вибирають той потік, у якого цей інтервал найкоротший. У результаті потоки, що захоплюють процесор на коротший час, отримують під час планування перевагу і швидше виходять із системи. Для короткотермінового планування реалізувати його не можливо, а для довготермінового планування його використовують досить часто.

**1.5.Багаторівневі черги зі зворотним зв’язком.**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*5*

###### *ІАЛЦ.462637.003.ПЗ*

Алгоритм багаторівневих черг зі зворотним зв’язком є найуніверсальнішим алгоритмом планування, але при цьому одним із найскладніших у реалізації. З погляду організації структур даних цей алгоритм схожий на звичайний алгоритм багаторівневих черг : є кілька черг готових потоків із різним пріоритетом, при цьому потоки черги із нижчим пріоритетом виконуються, тільки коли всі черги верхнього рівня порожні. Відмінності між двома алгоритмами полягають у тому, що:

-потокам дозволено переходити з рівня на рівень(із черги в чергу)

-потоки в одній черзі поєднуються не за пріоритетом, а за довжиною інтервалу використання процесора, потоки із коротшим інтервалом перебувають у черзі з більшим пріоритетом. Усередині всіх черг, крім найнижчої, використовують кругове планування (у найнижчій працює алгоритм FIFO). Різні черги відповідають різній довжині кванта часу - що вищий пріоритет, то коротший квант (звичайно довжина кванта для сусідніх черг зменшується удвічі). Якщо потік вичерпав свій квант часу, він переміщається у хвіст черги із нижчим пріоритетом (із довшим квантом). Можна також автоматично переміщати потоки, які давно не отримували керування, із черги нижнього рівня на рівень вище.

1. Топологія «Загальна шина з маркерним доступом»

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*6*

###### *ІАЛЦ.462637.003.ПЗ*

Комп’ютери, які входять в «шину» утворюють логічне кільце (Рис. 2.1) і передають по ньому спеціальний пакет – маркер. Якщо станція має дані для передачі, то вона може почати її лише після того як отримає маркер, який передається по черзі по логічному кільцю.

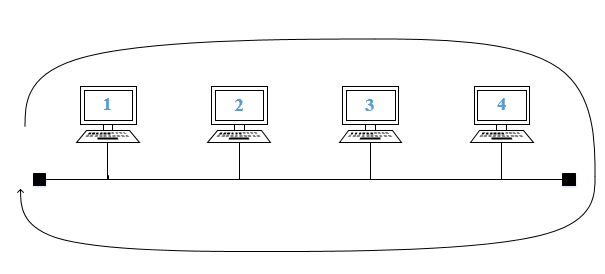


Рис. 2.1 – Топологія «Загальна шина з маркерним доступом»

Маркерні методи відносяться до децентралізованих і детермінованих методів управління обміном в системі. У них немає явно вираженого центру, але існує чітка система пріоритетів, і тому виникнення конфліктних ситуацій виключається.

Топологія шина з маркерним методом доступу має наступні властивості:

1) загальна шина використовується тільки для передачі повідомлень між процесорами;

2) декілька повідомлення не можуть передаватися (так як існує маркер, який захоплює ресурс);

3) посилка повідомлень не може бути перервана;

4) процесор може одночасно передавати дані і виконувати обчислення.

1. **Опис алгоритму планування**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*7*

###### *ІАЛЦ.462637.003.ПЗ*

Для системи з топологією «Загальна шина з маркерним доступом»

необхідно розробити 2 алгоритми планування :

* Алгоритм завантаження задач на процессори.
* Алгоритм передачі данних між процессорами.
  1. **Алгоритм завантаження задач на процессори**
* Підраховуємо кількість завдань , для якої дана задача буде

батьківської. Таким чином проходимо всі вершини. У результаті отримуємо список задач, залежних від заданої вершини.

- Сортуємо за спаданням час передачі даних від батьківської задачі до нащадків.

- Завантажуємо першу задачу на вільний процесор.

- Якщо кожна наступна задача може виконуватися, тобто не залежить від інших завдань, то завантажуємо її на вільний процесесор. При відсутності вільних процесорів завдання очікує звільнення процессора.

- Синхронно передаємо маркер по логічному кільцю. Перехід маркера з процесора на процесор буде займати 1 такт. Передача даних між процесорами можлива тільки при наявності маркера у даного процесора. При передачі процесор захоплює маркер, і залишається захопленим впродовж всього часу передачі даних цим процессором. Після передачі даних процесор звільняє маркер.

- Якщо після виконання завдань у нас виникає наступна можливість виконання залежних задач від даної, то:

1) виконується аналіз даної вершини.

2) продивлюється список залежних вершин.

3) виконується аналіз залежних вершин.

- Якщо після завершення батьківської задачі можливе виконання списку задач-нащадків, то ми за певним алгоритмом (алгоритм передачі даних між процесорами) передаємо дані для виконання задач-нащадків на інші процесори для ефективного завантаження всієї системи.

- Якщо звільнився ресурс, і в списку є задача, яка готова до виконання, то ця задача занурюється на даний ресурс. Вибирається зі списку задача з найбільшою вагою переходу.

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*8*

###### *ІАЛЦ.462637.003.ПЗ*

- Якщо виникла ситуація, коли на одну задачу передаються дані з різних завдань, то залишаємо на виконання на своєму ресурсі задачу, вага у якої максимальна.

* 1. **Алгоритм передачі данних між процессорами**

- Щоб запобігти частому звертанню до шини і ефективну передачу між процесорами використовується маркер, який дозволяє передачу даних від одного процесора до іншого. Тобто якщо процесор вимагає передачу, то він очікує маркер.

- Якщо процесор захопив маркер, то шина зайнята(інші процесори не можуть передавати дані) . При цьому задача на процесорі з маркером, переглядає список завдань, яким потрібно передати данні. Аналізується список нащадків. Якщо нащадок-завдання залежна від якоїсь задачі, і ця задача вже завантажена на один з процесорів, то проводиться аналіз доцільності передачі з батьківської задачі на інший процесор.

Перевірка відбувається по наступному критерію: якщо tпер < часу звільнення процесора, на якому виконується залежна задача, то проводиться передачу.

- При наявності задач, які мають однаковий час пересилки даних, виконується задача з максимальним часом виконання.

Пересилання даних здійснюється за допомогою буферизації даних. Тобто, як тільки задача виконана, то її дані можуть передаватися іншій задачі, які поміщаються в буфер цього завдання і очікують свого використання.

**4.Результати роботи планувальника**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*9*

###### *ІАЛЦ.462637.003.ПЗ*

За завданням було побудовано ациклічний, орієнтований граф на 15 вершин(Рис. 4.1.). Так як максимальна ширина яруса складає 4, то кількість процессорів було обрано відповідно 4. При моделюванні використовувалися дані з таблиці переходів(Табл. 4.1) та таблиці вагів вершин(Табл. 4.2). Результати моделювання потактно заносяться в таблицю (Табл. 4.3 ), створену за допомогою классу JTable на мові програмування Java.

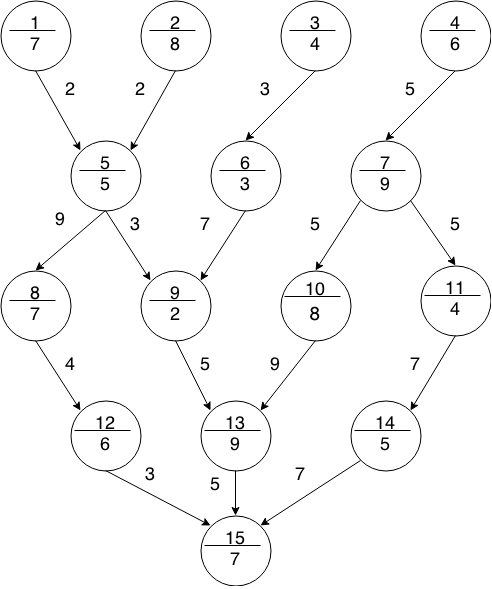


Рис. 4.1 – Ациклічний, орієнтований граф

Табл. 4.1 – Таблиця переходів між вершинами

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*10*

###### *ІАЛЦ.462637.003.ПЗ*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | - | - | - | - | 2 | - | - | - | - | - | - | - | - | - | - |
| 2 | - | - | - | - | 2 | - | - | - | - | - | - | - | - | - | - |
| 3 | - | - | - | - | - | 3 | - | - | - | - | - | - | - | - | - |
| 4 | - | - | - | - | - | - | 5 | - | - | - | - | - | - | - | - |
| 5 | - | - | - | - | - | - | - | 9 | 3 | - | - | - | - | - | - |
| 6 | - | - | - | - | - | - | - | - | 7 | - | - | - | - | - | - |
| 7 | - | - | - | - | - | - | - | - | - | 5 | 5 | - | - | - | - |
| 8 | - | - | - | - | - | - | - | - | - | - | - | 4 | - | - | - |
| 9 | - | - | - | - | - | - | - | - | - | - | - | - | 5 | - | - |
| 10 | - | - | - | - | - | - | - | - | - | - | - | - | 9 | - | - |
| 11 | - | - | - | - | - | - | - | - | - | - | - | - | - | 7 | - |
| 12 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 3 |
| 13 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 5 |
| 14 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 7 |
| 15 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

Табл. 4.2 – Таблиця ваги вершин

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |  |
| 7 | 8 | 4 | 6 | 5 | 3 | 9 | 7 | 2 | 8 | 4 | 6 | 9 | 5 | 7 |  |

Табл. 4.3 – Результат роботи планувальника

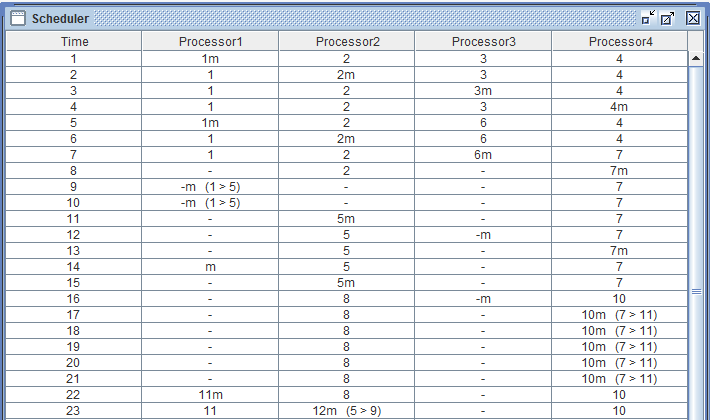
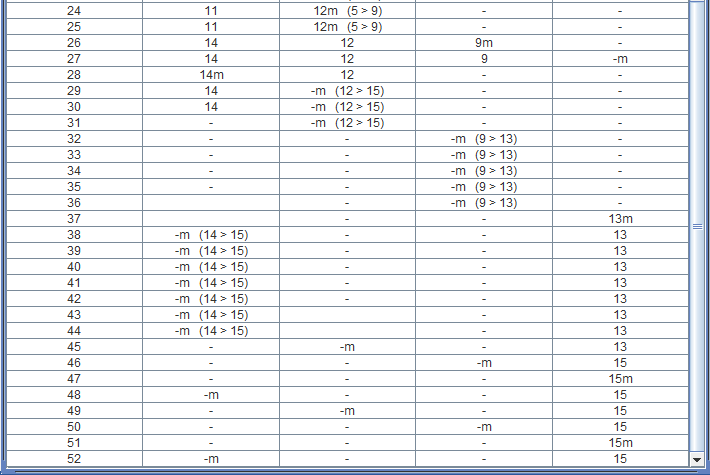


Табл. 4.3 – Результат роботи планувальника. Продовження.



Перевірка ефективності роботи планувальника:

1. Коефіцієнт прискорення (Kn) показує скорочення часу виконання паралельної програми в паралельній системі з P процессорами (Tp) в порівнянні з часом виконання послідовної програми в одно процесорній системі (T1):

Kn=T1/Tp = 90/52 = 1,74

1. Коефіцієнт ефективності (Ке) показує ступінь використання Р процесорів системи :

Kе=T1/ (Tp\*P) = 90/(52\*4) = 0,44

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*11*

###### *ІАЛЦ.462637.003.ПЗ*

**5.Висновки**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*12*

###### *ІАЛЦ.462637.003.ПЗ*

В даному курсовому проекті було розроблено просторовий планувальник для топології «Загальна шина з маркерним доступом». При розробці планувальника було розроблено власний алгоритм, який складається з двох частин : алгоритму завантаження задач на процессори та алгоритму передачі данних між процессорами. За допомогою алгоритма, було розроблено програмний додаток для занурення графа у систему з N процессорами з топологією «Загальна шина з маркерним доступом».

Перед початком занурення було проаналізовано граф, а саме ширину яруса, та обрано оптимальну кількість процессорів для заданого графа - 4.

Результат заноситься в таблицю, в якій відображається кожен такт роботи процессора, яка задача виконується в данний момент часу, який процессор захопив маркер та які пересилки виконуються. Оцінка ефективності системи виконується за двома основними критеріями – коефіцієнтом ефективності та коефіцієнтом прискорення.

Коефіцієнт прискорення дорівнює 1.74, а коефіцієнт ефективності -0,44. Такі результати є невисокими, тому що у системі є простої за рахунок нерівномірного розміщення задач на ярусах, а також за рахунок очікування маркера та пересилки даних між процесорами.

**6. Список використаних джерел**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*13*

###### *ІАЛЦ.462637.003.ПЗ*

1. [Таненбаум Э. С.](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%BD%D0%B5%D0%BD%D0%B1%D0%B0%D1%83%D0%BC,_%D0%AD%D0%BD%D0%B4%D1%80%D1%8E), Вудхалл А. С. Операционные системы. Разработка и реализация = Operating Systems: Design and Implementation. — 3-е изд. — СПб.: [Питер](https://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)), 2007. — 704 с. — [ISBN 978-5-469-01403-4](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785469014034).

2. Жуков І. А., Корочкін О. В. Паралельні та розподілені обчислення. – К.: «Корнійчук», 2014. – 284 с.

3. Шеховцов В. А. Операційні системи. Підручник К.: Видавнича група BHV, 2005. — 576 с.: іл. ISBN 966-552 -157-8

4. Конспект лекцій з курсу «Системне програмне забезпечення»

**Додаток А. Лістинг програми Scheduler**

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*14*

###### *ІАЛЦ.462637.003.ПЗ*

**package** Course\_work;

**public** **class** Link {

**private** Task fromTask;

**private** Task toTask;

**private** **int** weight;

**public** Link(Task fromTask,Task toTask, **int** weight){

**this**.fromTask = fromTask;

**this**.toTask = toTask;

**this**.weight = weight;

}

**public** Task getFromTask() {

**return** fromTask;

}

**public** **void** setFromTask(Task fromTask) {

**this**.fromTask = fromTask;

}

**public** Task getToTask() {

**return** toTask;

}

**public** **void** setToTask(Task toTask) {

**this**.toTask = toTask;

}

**public** **int** getWeight() {

**return** weight;

}

**public** **void** setWeight(**int** weight) {

**this**.weight = weight;

}

}

**package** Course\_work;

**import** java.util.ArrayList;

**public** **class** Processor {

**private** **boolean** isBusy;

**private** **boolean** isMarked;

**private** Task executeTask;

**private** ArrayList<Task> executedTasks = **new** ArrayList<Task>();

**public** Processor(**boolean** busy,**boolean** marked){

**this**.isBusy= busy;

**this**.isMarked=marked;

}

**public** **boolean** isBusy() {

**return** isBusy;

}

**public** **void** setBusy(**boolean** isBusy) {

**this**.isBusy = isBusy;

}

**public** **boolean** isMarked() {

**return** isMarked;

}

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*15*

###### *ІАЛЦ.462637.003.ПЗ*

**public** **void** setMarked(**boolean** isMarked) {

**this**.isMarked = isMarked;

}

**public** Task getExecuteTask() {

**return** executeTask;

}

**public** **void** setExecuteTask(Task executeTask) {

**this**.executeTask = executeTask;

}

**public** ArrayList<Task> getExecutedTasks() {

**return** executedTasks;

}

**public** **void** addExecutedTasks() {

**this**.executedTasks.add(**this**.executeTask);

}

**public** **boolean** pretends(Task task, ArrayList<Task> allExecutedTasks) {

**for** (**int** i = 0; i < executedTasks.size(); i++) {

ArrayList<Task> taskArrayList = executedTasks.get(i).getDependentTasks();

**for** (**int** j = 0; j < allExecutedTasks.size(); j++) {

**if** (taskArrayList.contains(allExecutedTasks.get(j))) {

taskArrayList.remove(allExecutedTasks.get(j));

}

}

**if** (taskArrayList.contains(task)) {

**return** **true**;

}

}

**return** **false**;

}

}

package Course\_work;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

public class Scheduler{

private final int PROCESSOR\_COUNT = 4;

private int marker = 0;

private Processor[] processors;

private Map<Integer, Task> tasks;

private boolean isBusFree = true;

private ArrayList<Link> links;

private ArrayList<Task> readyTasks = new ArrayList<Task>();

private ArrayList<Task> executedTasks = new ArrayList<Task>();

private int numberOfTasks;

private ArrayList<Task> doneATasks = new ArrayList<Task>();

public Scheduler(int[][] graphMatrix,int[] tasksWeights){

numberOfTasks = graphMatrix.length;

setSystemParametres(graphMatrix,tasksWeights);

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*16*

###### *ІАЛЦ.462637.003.ПЗ*

processors = new Processor[PROCESSOR\_COUNT];

for (int i = 0; i < PROCESSOR\_COUNT; i++) {

Processor processor = new Processor(false, false);

processors[i] = processor;

}

}

public void run() {

int takt = 0;

processors[0].setMarked(true);

ArrayList<String> modelingResult = new ArrayList<String>();

String taktLine = "";

String split = ";";

taktLine += "t" + split + " ";

for (int i = 1; i <= PROCESSOR\_COUNT; i++) {

taktLine += "P" + i + split + " ";

}

taktLine += " Bus";

System.out.println(taktLine);

modelingResult.add(taktLine);

while (doneATasks.size() != numberOfTasks) {

takt++;

for (int i = 0; i < processors.length; i++) {

if ((processors[i].isBusy()) && (processors[i].getExecuteTask().getRemainder() == 0)) {

processors[i].setBusy(false);

processors[i].addExecutedTasks();

doneATasks.add(processors[i].getExecuteTask());

executedTasks.remove(processors[i].getExecuteTask());

for (int j = 0; j < numberOfTasks; j++) {

if ((!doneATasks.contains(tasks.get(j))) && (!executedTasks.contains(tasks.get(j)))) {

if (tasks.get(j).getFatherTasksToDo().contains(processors[i].getExecuteTask())) {

tasks.get(j).getFatherTasksToDo().remove(processors[i].getExecuteTask());

}

if ((tasks.get(j).getFatherTasksToDo().size() == 0) && (!readyTasks.contains(tasks.get(j)))) {

readyTasks.add(tasks.get(j));

}

}

}

}

}

for (int i = 0; i < PROCESSOR\_COUNT; i++) {

if (!processors[i].isBusy()) {

if (readyTasks.size() != 0) {

for (int j = 0; j < readyTasks.size() - 1; j++) {

for (int k = +1; k < j; k++) {

if (readyTasks.get(k).getDuration() > readyTasks.get(j).getDuration()) {

readyTasks.add(j, readyTasks.get(k));

readyTasks.remove(k + i);

}

}

}

Task taskForExecute = null;

for (int j = 0; j < readyTasks.size(); j++) {

if (processors[i].pretends(readyTasks.get(j), executedTasks)) {

if (taskForExecute != null) {

if (processors[i].getExecuteTask() != null) {

if (getConnectionWeight(readyTasks.get(j), processors[i].getExecuteTask()) > getConnectionWeight(taskForExecute, processors[i].getExecuteTask())) {

taskForExecute = readyTasks.get(j);

}

}

} else {

taskForExecute = readyTasks.get(j);

}

}

}

if (processors[i].isMarked() & (!isBusFree)) {

isBusFree = false;

}

if (taskForExecute != null) {

processors[i].setExecuteTask(taskForExecute);

executedTasks.add(taskForExecute);

readyTasks.remove(taskForExecute);

} else {

processors[i].setExecuteTask(readyTasks.get(0));

executedTasks.add(readyTasks.get(0));

readyTasks.remove(0);

}

processors[i].setBusy(true);

}

}

}

for (int i = 0; i < processors.length; i++) {

if (processors[i].isBusy()) {

processors[i].getExecuteTask().execute();

}

}

for (int i = 0; i < PROCESSOR\_COUNT; i++) {

if ((processors[i].isMarked()) & (isBusFree)) {

processors[i].setMarked(false);

if (i != PROCESSOR\_COUNT - 1) {

processors[i + 1].setMarked(true);

marker = i + 1;

break;

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*17*

###### *ІАЛЦ.462637.003.ПЗ*

} else {

processors[0].setMarked(true);

marker = 0;

break;

}

}

}

taktLine = takt + split + " ";

for (int i = 0; i < PROCESSOR\_COUNT; i++) {

Task aTask = null;

if (processors[i].isBusy()) {

aTask = processors[i].getExecuteTask();

}

if (aTask != null) {

taktLine += (processors[i].getExecuteTask().getId() + 1);

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*18*

###### *ІАЛЦ.462637.003.ПЗ*

} else {

taktLine += "-";

}

if (processors[i].isMarked()) {

taktLine += "m" + split + " ";

} else {

taktLine += split + " ";

}

}

if (!isBusFree) {

taktLine += links.get(marker).getFromTask().getId() + " > " +

links.get(marker).getToTask().getId() + split;

}

System.out.println(taktLine);

modelingResult.add(taktLine);

}

}

private void setSystemParametres(int[][] graphMatrix, int[] tasksWeights) {

tasks = new HashMap<Integer, Task>();

for (int i = 0; i < numberOfTasks; i++) {

Task Task = new Task(i, tasksWeights[i]);

tasks.put(i, Task);

}

for (int i = 0; i < numberOfTasks; i++) {

ArrayList<Task> dependentTasks = new ArrayList<Task>();

for (int j = 0; j < numberOfTasks; j++) {

if (graphMatrix[i][j] != 0) {

dependentTasks.add(tasks.get(j));

}

}

tasks.get(i).setDependentATasks(dependentTasks);

}

for (int i = 0; i < numberOfTasks; i++) {

ArrayList<Task> fatherTasks = new ArrayList<Task>();

for (int j = 0; j < numberOfTasks; j++) {

if (graphMatrix[j][i] != 0) {

fatherTasks.add(tasks.get(j));

}

}

tasks.get(i).setFatherTasks(fatherTasks);

}

links = new ArrayList<Link>();

for (int i = 0; i < numberOfTasks; i++) {

for (int j = 0; j < numberOfTasks; j++) {

if (graphMatrix[i][j] != 0) {

Link link = new Link(tasks.get(i), tasks.get(j), graphMatrix[i][j]);

links.add(link);

}

}

}

for (int i = 0; i < numberOfTasks; i++) {

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*19*

###### *ІАЛЦ.462637.003.ПЗ*

if (isTopTask(graphMatrix, tasks.get(i))) {

readyTasks.add(tasks.get(i));

}

}

}

private boolean isTopTask(int[][] graphMatrix,Task task) {

for (int i = 0; i < graphMatrix.length; i++) {

if (graphMatrix[i][task.getId()] != 0) {

return false;

}

}

return true;

}

private int getConnectionWeight(Task task1, Task task2) {

for (int i = 0; i < links.size(); i++) {

if ((links.get(i).getFromTask().equals(task1)) && (links.get(i).getToTask().equals(task2))) {

return links.get(i).getWeight();

}

if ((links.get(i).getFromTask().equals(task2)) && (links.get(i).getToTask().equals(task1))) {

return links.get(i).getWeight();

}

}

return -1;

}

}

**package** Course\_work;

**import** java.util.ArrayList;

**public** **class** Task {

**private** **int** id;

**private** **int** duration;

**private** **int** remainder;

**private** ArrayList<Task> dependentTasks;

**private** ArrayList<Task> fatherTasks;

**private** ArrayList<Task> fatherTasksToDo;

**private** **boolean** isPretended = **false**;

**public** Task(**int** id, **int** duration) {

**this**.id = id;

**this**.duration = duration;

**this**.remainder = duration;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** **int** getDuration() {

**return** duration;

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*20*

###### *ІАЛЦ.462637.003.ПЗ*

}

**public** **void** setDuration(**int** duration) {

**this**.duration = duration;

}

**public** **int** getRemainder() {

**return** remainder;

}

**public** **void** setRemainder(**int** remainder) {

**this**.remainder = remainder;

}

**public** ArrayList<Task> getDependentTasks() {

**return** dependentTasks;

}

**public** **void** setDependentATasks(ArrayList<Task> dependentTasks) {

**this**.dependentTasks = dependentTasks;

}

**public** ArrayList<Task> getFatherTasks() {

**return** fatherTasks;

}

**public** **void** setFatherTasks(ArrayList<Task> fatherTasks) {

**this**.fatherTasks = fatherTasks;

**this**.fatherTasksToDo = (ArrayList<Task>) fatherTasks.clone();

}

**public** ArrayList<Task> getFatherTasksToDo() {

**return** fatherTasksToDo;

}

**public** **void** setFatherTasksToDo(ArrayList<Task> fatherTasksToDo) {

**this**.fatherTasksToDo = fatherTasksToDo;

}

**public** **boolean** isPretended() {

**return** isPretended;

}

**public** **void** setPretended(**boolean** isPretended) {

**this**.isPretended = isPretended;

}

**public** **void** execute() {

remainder--;

}

}

private static final long serialVersionUID = 1L;

public static void createGUI() {

JFrame frame = new JFrame("Scheduler");

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*21*

###### *ІАЛЦ.462637.003.ПЗ*

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

String[] columnNames = {

"Time",

"Processor1",

"Processor2",

"Processor3",

"Processor4"

};

JTable table = new JTable(data, columnNames);

table.setDefaultRenderer(table.getColumnClass(1), new DefaultTableCellRenderer(){

private static final long serialVersionUID = 1L;

public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);

super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

return this;

}

});

JScrollPane scrollPane = new JScrollPane(table);

frame.getContentPane().add(scrollPane);

frame.setPreferredSize(new Dimension(450, 200));

frame.pack();

frame.setLocationRelativeTo(null);

frame.setVisible(true);

}

public static void main(String[] args) {

javax.swing.SwingUtilities.invokeLater(new Runnable() {

public void run() {

JFrame.setDefaultLookAndFeelDecorated(true);

createGUI();

}

});

}

**package** Course\_work;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int**[][] GRAPH\_MATRIX = {

{0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 9, 3, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 5, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,7, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

};

**int**[] TASKS\_WEIGHTS = {

7, 8, 4, 6, 5, 3, 9, 7, 2, 8, 4, 6, 9, 5, 7

};

Scheduler scheduler = **new** Scheduler(GRAPH\_MATRIX, TASKS\_WEIGHTS);

scheduler.run();

}

}

**package** Course\_work;

**import** java.util.ArrayList;

**public** **class** Task {

**private** **int** id;

**private** **int** duration;

**private** **int** remainder;

**private** ArrayList<Task> dependentTasks;

**private** ArrayList<Task> fatherTasks;

**private** ArrayList<Task> fatherTasksToDo;

**private** **boolean** isPretended = **false**;

**public** Task(**int** id, **int** duration) {

**this**.id = id;

**this**.duration = duration;

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*22*

###### *ІАЛЦ.462637.003.ПЗ*

**this**.remainder = duration;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** **int** getDuration() {

**return** duration;

}

**public** **void** setDuration(**int** duration) {

**this**.duration = duration;

}

**public** **int** getRemainder() {

**return** remainder;

}

**public** **void** setRemainder(**int** remainder) {

**this**.remainder = remainder;

}

**public** ArrayList<Task> getDependentTasks() {

**return** dependentTasks;

}

**public** **void** setDependentATasks(ArrayList<Task> dependentTasks) {

*Изм.*

*Лист*

*№ докум.*

*Підпис*

*Дата*

*Лист*

*23*

###### *ІАЛЦ.462637.003.ПЗ*

**this**.dependentTasks = dependentTasks;

}

**public** ArrayList<Task> getFatherTasks() {

**return** fatherTasks;

}

**public** **void** setFatherTasks(ArrayList<Task> fatherTasks) {

**this**.fatherTasks = fatherTasks;

**this**.fatherTasksToDo = (ArrayList<Task>) fatherTasks.clone();

}

**public** ArrayList<Task> getFatherTasksToDo() {

**return** fatherTasksToDo;

}

**public** **void** setFatherTasksToDo(ArrayList<Task> fatherTasksToDo) {

**this**.fatherTasksToDo = fatherTasksToDo;

}

**public** **void** execute() { remainder--;

}