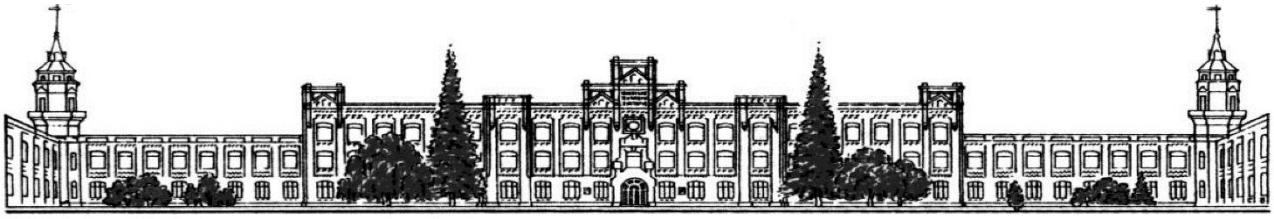

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

# КУРСОВИЙ ПРОЕКТ

з дисципліни «Системне програмне забезпечення»
на тему: «Просторовий планувальник для розподілу завдань у багатопроцесорній системі з топологією "Коло процесорів"»

Студентки 4-го курсу групи ІВ-71
напряму підготовки 123 Комп'ютерна інженерія
спеціальності: комп'ютерні системи та мережі
Молчанової Варвари Сергіївни

Керівник доктор т.н., професор Сімоненко Валерій Павлович

Національна оцінка _____
Кількість балів:_____ Оцінка: ECTS_____

_____ _____
(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

Київ – 2020 р.

# ОПИС АЛЬБОМУ

| № рядка | Формат | Позначення | Найменування | Кількість | Примітка |
|---|---|---|---|---|---|
| 1 | | | Документація | | |
| 2 | | | | | |
| 3 | А4 | ІАЛЦ.007110.001 ОА | Опис альбому | 1 | |
| 4 | | | | | |
| 5 | А4 | ІАЛЦ.007110.002 ТЗ | Технічне завдання | 2 | |
| 6 | | | | | |
| 7 | А4 | ІАЛЦ.007110.003 ПЗ | Пояснювальна записка | 7 | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | | | | | |
| 22 | | | | | |
| 23 | | | | | |
| 24 | | | | | |
| 25 | | | | | |
| 26 | | | | | |
| 27 | | | | | |

| | | | | | *ІАЛЦ.007110.001.ОА* | | |
|---|---|---|---|---|---|---|---|
| Зм. | Арк. | № докум | Підпис | Дата | | | |
| Розробила | | Молчанова В.С. | | | Опис альбому | Літера | Аркуш 1 / Аркушів 1 |
| Перевірив | | Сімоненко В.П. | | | | | |
| Реценз. | | | | | | НТУУ «КПІ» ФІОТ<br>Група ІВ-71 | |
| Н. контр. | | | | | | | |
| Затверд. | | | | | | | |

# ТЕХНИЧНЕ ЗАВДАННЯ

# ЗМІСТ



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | *ІАЛЦ.007110.002.ТЗ* | | | |
| Зм. | Арк. | № докум | Підпис | Дата | | | | |
| Розробила | | Молчанова В.С. | | | Опис альбому | Літера | Аркуш | Аркушів |
| Перевірив | | Сімоненко В.П. | | | | | 1 | 2 |
| Реценз. | | | | | | НТУУ «КПІ» ФІОТ<br>Група ІВ-71 | | |
| Н. контр. | | | | | | | | |
| Затверд. | | | | | | | | |

## 1. Область застосування

В даному курсовому проекті реалізовано алгоритм планування для обчислювальної системи з топологією «Коло процесорів». Розроблена програма моделює роботу системи для заданого користувачем ациклічного орієнтованого графа задачі.

## 2. Підстави для розробки

Підставою для розробки є індивідуальне завдання на курсовий проект.

## 3. Мета і призначення розробки

Метою даної роботи є закріплення знань про паралельні обчислювальні системи та алгоритми планування, які були отримані при вивченні курсу «Системне програмне забезпечення». Розроблений проект призначений для демонстрації роботи планувальника в системі з топологією «Коло процесорів».

## 4. Перелік документації

- Опис альбому
- Технічне завдання
- Пояснювальна записка

## 5. Етапи розробки

- Узгодження технічного завдання
- Аналіз існуючих рішень
- Розробка програмної моделі
- Тестування програмної моделі
- Захист курсової роботи

# ПОЯСНЮВАЛЬНА ЗАПИСКА

# ЗМІСТ



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | *ІАЛЦ.007110.003.ПЗ* | | | |
| Зм. | Арк. | № докум | Підпис | Дата | | | | |
| Розробила | | Молчанова В.С. | | | Опис альбому | Літера | Аркуш | Аркушів |
| Перевірив | | Сімоненко В.П. | | | | | 1 | 7 |
| Реценз. | | | | | | НТУУ «КПІ» ФІОТ<br>Група ІВ-71 | | |
| Н. контр. | | | | | | | | |
| Затверд. | | | | | | | | |

## Вступ

В даний час паралельна обробка даних є пріоритетним напрямком розвитку комп'ютерних систем. Однією з основних проблем в цій галузі є завдання розподілу пов'язаних підзадач по процесорам.

Дане завдання не має алгоритму для отримання гарантовано оптимального рішення. Проте, створено безліч стратегій планування, які досить добре справляються із завданням розподілу. Існують, як пристосовані для будь-яких завдань, так і спеціалізовані стратегії, що враховують особливості завдання (наприклад, ступінь зв'язності завдань, топологію графа завдань, топологію комп'ютерної системи і т.д.). Кожна стратегія має свої переваги і недоліки і не може давати однаково хороші рішення для всіх типів завдань.

Є кілька пріоритетних параметрів при плануванні, таких як: оптимізація часу рішення, оптимізація кількості використовуваних ресурсів, зменшення кількості пересилань. Необхідно шукати компроміс в значеннях цих параметрів, так як вони залежать один від одного, а ми всюди прагнемо до мінімуму. Отримане планування буде лежати між крайніми випадками: всі завдання на одному процесорі і всі завдання на окремих процесорах.

Не останню роль у розподілі грає і облік особливості топології використовуваної комп'ютерної системи. Тобто, вибір стратегії розподілу повинен ґрунтуватися на особливостях, як завдання, так і топології системи

В даному курсовому проекті досліджується і вирішується завдання планування виконання завдань у системі з топологією «Коло процесорів». Результатом виконання є запропонований метод рішення і реалізація середовища моделювання з використанням цього методу.

# 1. Постановка задачі

## 1.1. Вхідні дані

На вхід планувальник приймає спрямований ациклічний граф завдання. Кожен вузол графа має вагу, який визначає час виконання завдання, представленої вузлом. Кожне ребро в графі направлено у напрямку передачі даних і має вагу, що описує обсяг (а отже і тривалість) пересилання.

## 1.2. Топологія системи

Топологія обчислювальної системи - «коло процесорів». Ця топологія є двонаправленою, отже кожен з процесорів має по два порти читання та запису, які підключені до двох сусідів. Топологія однорідна, тому всі процесори в ній однакові.

# 2. Проектування алгоритму планування

## 2.1. Існуючі методи вирішення задач оптимізації

*Генетичний алгоритм*. Суть алгоритму полягає в наступному: Беремо базове рішення і змінюючи параметри знаходимо інше рішення. При цьому дивимося отримане рішення краще базового чи ні. Недоліком такого методу оптимізації є те, що оптимальний варіант отримуємо за велику кількість кроків. З цієї причини даний метод практично не використовується.

*Метод оціночних функцій*. Ціль методу – розбиття на зони. Ми оцінюємо чи потрапило рішення в якусь зону. Виводимо перше завдання на процесор і повинні визначити яку задачу потрібно занурити на той же процесор. Причому кожному рішенню присвоюємо вагу. Чим більше вага тим більше ступінь претендування.

*Метод покрокового конструювання*. Цей метод ще називають «Метод спрямованого пошуку». За алгоритмом спочатку потрібно піти у область прийнятних значень, тобто потрібно спробувати виключити свідомо невірні варіанти. Потім описуємо оптимальний варіант, а після нього прийнятний.

### 2.2. Опис алгоритму

На кожному такті планувальник виконує наступні дії:

1. Шукає задачі, усі батьки яких уже виконані. За допомогою функції getDestProcessor він обирає процесор, у якому буде виконуватися ця задача та додає процеси пересилок даних цієї задачі до черги. Функція getDestProcessor повертає процесор, в якому знаходиться дані найбільшої кількості батьків. Це дозволяє мінімізувати кількість пересилок та втрати часу на них.
2. Обирає з черги ті процеси пересилок, які можуть бути виконані на даному такті та ставить їх на виконання процесорам.
3. Шукає задачі, яким вже був призначений процесор, і якщо в ньому є достатньо даних та він вільний, починає їх виконувати.

## 3. Приклад роботи алгоритму

Створимо довільний тестовий граф, який зображений на рисунку 1:

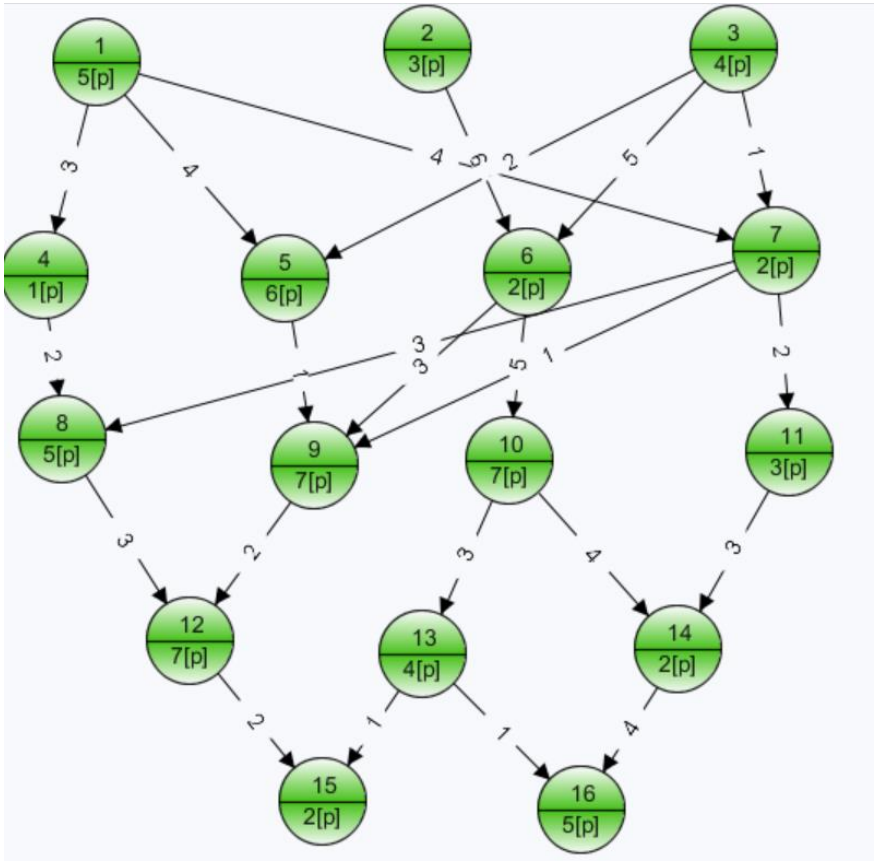


Рисунок 1. Граф завдання

Задамо цей граф програмно у вигляді матриці ребер та списку ваг вершин:

```
val GRAPH_MATRIX = listOf(
            //     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
            listOf(0, 0, 0, 3, 4, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 1
            listOf(0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 2
            listOf(0, 0, 0, 0, 2, 5, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 3
            listOf(0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0), // 4
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0), // 5
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 3, 5, 0, 0, 0, 0, 0, 0), // 6
            listOf(0, 0, 0, 0, 0, 0, 0, 3, 1, 0, 2, 0, 0, 0, 0, 0), // 7
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0), // 8
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0), // 9
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0), // 10
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0), // 11
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0), // 12
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1), // 13
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4), // 14
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 15
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)  // 16
        )
val TASKS_WEIGHTS = listOf(
         // 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
            5, 3, 4, 1, 6, 2, 2, 5, 7, 7, 3, 7, 4, 2, 2, 5)
```

Програма статичного планування, лістинг якої наведено у додатку А, виводить результат потактової часової симуляції алгоритму. Умовні позначення: у верхньому рядку кожного такту записані числа, що відповідають задачі, що зараз виконується, у нижньому - операції читання або запису певної задачі, «-» означає простій процесора. Незважаючи на те, що ширина графу дорівнює 4, найбільш ефективний результат було отримано за умови симуляції на 3-х процесорах. Результат наведено у таблиці 1. Оцінка алгоритму показує, що для даної задачі планувальник видав коефіцієнт прискорення Кп = 1.33 та коефіцієнт ефективності Ке = 0.38

| t | P1 | P2 | P3 | t | P1 | P2 | P3 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 26 | 9 | - | - |
| | - | - | - | | - | - | - |
| 2 | 1 | 2 | 3 | 27 | 9 | - | - |
| | - | - | - | | - | - | - |
| 3 | 1 | 2 | 3 | 28 | 9 | - | - |
| | - | - | - | | - | - | - |
| 4 | 1 | - | 3 | 29 | 9 | - | - |
| | - | - | - | | - | - | - |
| 5 | 1 | - | - | 30 | 9 | - | - |
| | - | r[3] | w[3] | | - | - | - |
| 6 | 4 | - | - | 31 | 9 | - | - |
| | - | r[3] | w[3] | | - | - | - |
| 7 | - | - | - | 32 | 11 | - | - |
| | - | r[3] | w[3] | | - | - | - |
| 8 | - | - | - | 33 | 11 | - | - |
| | - | r[3] | w[3] | | - | - | - |
| 9 | - | - | - | 34 | 11 | - | - |
| | - | r[3] | w[3] | | - | - | - |
| 10 | - | 6 | - | 35 | 12 | - | - |
| | r[3] | - | w[3] | | r[10] | w[10] | - |
| 11 | - | 6 | - | 36 | 12 | - | - |
| | r[3] | - | w[3] | | r[10] | w[10] | - |
| 12 | 5 | 10 | - | 37 | 12 | - | - |
| | r[3] | - | w[3] | | r[10] | w[10] | - |
| 13 | 5 | 10 | - | 38 | 12 | - | - |
| | - | - | - | | r[10] | w[10] | - |
| 14 | 5 | 10 | - | 39 | 12 | - | - |
| | - | - | - | | - | - | - |
| 15 | 5 | 10 | - | 40 | 12 | - | - |
| | - | - | - | | - | - | - |
| 16 | 5 | 10 | - | 41 | 12 | - | - |
| | - | - | - | | - | - | - |
| 17 | 5 | 10 | - | 42 | 14 | - | - |
| | - | - | - | | r[13] | w[13] | - |
| 18 | 7 | 10 | - | 43 | 14 | - | - |
| | - | - | - | | - | - | - |
| 19 | 7 | 13 | - | 44 | 15 | - | - |
| | - | - | - | | r[13] | w[13] | - |
| 20 | 8 | 13 | - | 45 | 15 | - | - |
| | r[6] | w[6] | - | | - | - | - |
| 21 | 8 | 13 | - | 46 | 16 | - | - |
| | r[6] | w[6] | - | | - | - | - |
| 22 | 8 | 13 | - | 47 | 16 | - | - |
| | r[6] | w[6] | - | | - | - | - |
| 23 | 8 | - | - | 48 | 16 | - | - |
| | - | - | - | | - | - | - |
| 24 | 8 | - | - | 49 | 16 | - | - |
| | - | - | - | | - | - | - |
| 25 | 9 | - | - | 50 | 16 | - | - |
| | - | - | - | | - | - | - |

Таблиця 1. Результат виконання

## Висновок

За задачею цього проекту було побудовано багатопроцесорний планувальник для топології “коло процесорів”.

Для трьох процесорів було проведено планування за допомогою розробленої програми та проаналізовано результати.

Результуючий коефіцієнт прискорення за умови використання цього алгоритму дорівнював 1.33, а коефіцієнт ефективності використання процесорів системи був 0.38. Затримки в більшій мірі були викликані очікуванням результатів попередніх задач на інших процесорах, а також очікуванням переходу маркера до процесора, що мав пересилати дані

# ДОДАТОК А

## Лістинг програми

```
class Data(val sourceTask: Int, val destTask: Int, val weight: Int)

abstract class Process(val duration: Int) {
    var remainedTacts = duration

    val isExecuted: Boolean
       get() {return remainedTacts == 0}

    fun execute() {
        remainedTacts--
    }
}

class Task(val id: Int, duration: Int): Process(duration) {
    val children = mutableMapOf<Int, Int>() // taskId -> connectionWeight
    val parents = mutableMapOf<Int, Int>() // taskId -> connectionWeight

    override fun toString(): String {
        return (id+1).toString()
    }
}

class DataSendingProcess(val sourceProcessor: Int, val destProcessor: Int,
val data: Data)
    : Process(data.weight) {
    override fun toString(): String {
        return "${sourceProcessor+1} -> ${destProcessor+1}
(${data.sourceTask})"
    }

    fun copy(): DataSendingProcess {return
DataSendingProcess(sourceProcessor, destProcessor, data)}
}

class Processor(val id: Int) {
    private val processData = mutableListOf<Data>()

    var currentProcess: DataSendingProcess? = null
    var currentTask: Task? = null

    var isBusyWithReading: Boolean = false
        get() {return currentProcess != null && !currentProcess!!.isExecuted}

    var isBusyWithTask: Boolean = false
        get() {return currentTask != null && !currentTask!!.isExecuted}

    fun executeTask(){
        currentTask!!.execute()
        if (currentTask!!.isExecuted) {
            for (childTask in currentTask!!.children) {
                processData.add(Data(currentTask!!.id, childTask.key,
childTask.value))
            }
        }
```

```
    }

    fun executeProcess(){
        currentProcess!!.execute()
        if (currentProcess!!.isExecuted) {
            if (currentProcess!!.destProcessor == id)
                processData.add(currentProcess!!.data)
            currentProcess = null
        }
    }

    fun containsData(sourceTask: Int, destTask: Int): Boolean {
        return processData.any { it.sourceTask == sourceTask && it.destTask
== destTask }
    }

    fun containsAllTaskData(task: Task): Boolean {return task.parents.all
{parent -> containsData(parent.key, task.id) }}

    fun logProcess(): String {
        if (currentProcess != null) {
            return "${if (currentProcess!!.sourceProcessor == id) "w" else
"r" }[${currentProcess!!.data.sourceTask+1}]"
        }
        return "-"
    }

    fun logTask(): String {
        return if (currentTask != null) currentTask.toString() else "-"
    }
}

class ProcessorRing(val processorCount: Int) {
    val processors = List(processorCount) { Processor(it) }
    val dataSendingQueue = mutableListOf<DataSendingProcess>()
    private val plannedTasks = mutableListOf<Task>()

    fun planDataSending(task: Task, destProcessor: Int) {
        if(plannedTasks.contains(task))
            return
        for (parent in task.parents) {
            val (parentId, weight) = parent
            var sourceProcessor = processors.first{it.containsData(parentId,
task.id)}.id
            if (sourceProcessor != destProcessor)
            {
                val path = getPath(sourceProcessor, destProcessor)
                path.removeAt(0)
                for (nextProcessor in path) {
                    dataSendingQueue.add(DataSendingProcess(sourceProcessor,
nextProcessor, Data(parentId, task.id, weight)))
                    sourceProcessor = nextProcessor
                }
            }
        }
        plannedTasks.add(task)
```

```
    }

    fun getPath(from: Int, to: Int): MutableList<Int> {
        val plus = mutableListOf(from)
        val minus = mutableListOf(from)
        var currentPlus = from
        var currentMinus = from
        while (currentPlus != to && currentMinus != to) {
            currentPlus = getNext(currentPlus)
            currentMinus = getPrev(currentMinus)
            plus.add(currentPlus)
            minus.add(currentMinus)
        }
        return if (currentPlus == to) plus else minus
    }

    operator fun get(processorIdx: Int): Processor {
        return processors[processorIdx]
    }

    private fun getNext(i: Int): Int {return (i + 1) % processorCount}

    private fun getPrev(i: Int): Int {return (processorCount + i - 1) %
processorCount}
}

class Planner(private val processorCount: Int, graphMatrix: List<List<Int>>,
tasksWeights: List<Int>) {
    private val ring = ProcessorRing(processorCount)
    private val taskCount = graphMatrix.size
    private val tasks: List<Task> = convertToTasks(graphMatrix, tasksWeights)
    private val executingTasks = mutableListOf<Int>()
    private val doneTasks = mutableListOf<Int>()
    private var tact = 1

    fun run() {
        logHeader()
        while (doneTasks.size < taskCount) {

            //try add data sending to the queue
            for (task in notStartedTasks().filter { processIsReady(it) }) {
                val destProcessorId = chooseDestProcessor(task)
                ring.planDataSending(task, destProcessorId)
            }

            //try assign data sending as processor current task
            val processesToDelete = mutableListOf<DataSendingProcess>()
            for (process in ring.dataSendingQueue) {
                if (!ring[process.sourceProcessor].isBusyWithReading
                && !ring[process.destProcessor].isBusyWithReading
                && processIsReady(process)) {
                    ring[process.sourceProcessor].currentProcess =
process.copy()
                    ring[process.destProcessor].currentProcess =
process.copy()
```

```
                    processesToDelete.add(process)
                }
            }
            ring.dataSendingQueue.removeAll(processesToDelete)

            // assign tasks that are ready to the free processors
            for (processor in ring.processors.filter { !it.isBusyWithTask })
{
                val nextTask = notStartedTasks().firstOrNull{t ->
processIsReady(t)
                            && processor.containsAllTaskData(t)}
                if (nextTask != null) {
                    executingTasks.add(nextTask.id)
                }
                processor.currentTask = nextTask
            }

            logTact()

            //execute tasks in processors
            for (processor in ring.processors) {
                if (processor.isBusyWithTask) {
                    processor.executeTask()
                    if (processor.currentTask!!.isExecuted) {
                        val taskId = processor.currentTask!!.id
                        executingTasks.remove(taskId)
                        doneTasks.add(taskId)
                    }
                }
                if (processor.isBusyWithReading) {
                    processor.executeProcess()
                }
            }

            tact++
        }
    }

    private fun chooseDestProcessor(task: Task): Int {
        // count how much parent tasks each processor has and return id of
the one which has the most
        val maxCount = ring.processors.map { proc -> task.parents.count
{parent -> proc.containsData(parent.key, task.id) } }
        return maxCount.indexOf(maxCount.max())
    }

    private fun convertToTasks(graphMatrix: List<List<Int>>, tasksWeights:
List<Int>): List<Task> {
        val tasks = List(taskCount) {idx -> Task(idx, tasksWeights[idx])}

        for (currentTaskId in 0 until taskCount) {
            for (relatedTaskId in 0 until taskCount) {
                val weight = graphMatrix[currentTaskId][relatedTaskId]
                if (weight != 0) {
                    tasks[currentTaskId].children[relatedTaskId] = weight
                    tasks[relatedTaskId].parents[currentTaskId] = weight
```

```
                }
            }
        }

        return tasks
    }

    private fun logHeader() {
        var tactLine = ""
        tactLine += String.format("%3s", "t")
        for (i in 0 until processorCount) {
            tactLine += String.format("%10s", "P${i+1}")
        }
        println(tactLine)
    }

    private fun logTact() {
        var taskLine = ""
        taskLine += String.format("%s\t", "$tact")
        for (i in 0 until processorCount) {
            val processor = ring[i]
            taskLine += String.format("%s\t", processor.logTask())
        }
        println(taskLine)
        var procLine = String.format("%s", "\t")
        for (i in 0 until processorCount) {
            val processor = ring[i]
            procLine += String.format("%s\t", processor.logProcess())
        }
        println(procLine)
    }

    private fun processIsReady(process: Process): Boolean {
        when (process) {
            is Task ->
                return process.parents.keys.map { tasks[it] }.all {
it.isExecuted }
            is DataSendingProcess ->
                return
ring[process.sourceProcessor].containsData(process.data.sourceTask,
process.data.destTask)
        }
        return false
    }

    private fun notStartedTasks(): List<Task> {
        return tasks.filter{t -> !doneTasks.contains(t.id) &&
!executingTasks.contains(t.id)}
    }
}

object Test {

    @JvmStatic
    fun main(args: Array<String>) {
        val GRAPH_MATRIX = listOf(
```

```
            //          1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
            listOf(0, 0, 0, 3, 4, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 1
            listOf(0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 2
            listOf(0, 0, 0, 0, 2, 5, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 3
            listOf(0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0), // 4
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0), // 5
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 3, 5, 0, 0, 0, 0, 0, 0), // 6
            listOf(0, 0, 0, 0, 0, 0, 0, 3, 1, 0, 2, 0, 0, 0, 0, 0), // 7
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0), // 8
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0), // 9
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 4, 0, 0), // 10
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0), // 11
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0), // 12
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1), // 13
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4), // 14
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), // 15
            listOf(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)  // 16
        )
        val TASKS_WEIGHTS = listOf(
         // 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
            5, 3, 4, 1, 6, 2, 2, 5, 7, 7, 3, 7, 4, 2, 2, 5)

        val planner = Planner(3, GRAPH_MATRIX, TASKS_WEIGHTS)
        planner.run()
    }
}
```