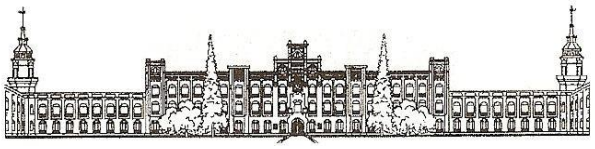

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Лабораторна робота №2

з дисципліни «Комп'ютерні системи»

«МОДЕЛЮВАННЯ КОНВЕЄРНОЇ ОБЧИСЛЮВАЛЬНОЇ СИСТЕМИ»

Виконав студент IV курсу

групи: КВ-11

ПІБ: Терентьєв Іван Дмитрович

Перевірив: _____

Київ 2024

Мета лабораторної роботи

Вивчення особливостей функціонування конвеєрних обчислювальних систем. Ознайомлення з основними видами стратегії керування статичним і динамічним конвеєром. Знаходження оптимальної стратегії керування. Моделювання роботи конвеєрної обчислювальної системи.

Завдання для лабораторної роботи

1. Ознайомитися з описом лабораторної роботи.

2. Отримати варіант завдання у викладача.

3. Побудувати для таблиці зайнятості, що відповідає варіанту, повну та модифіковану діаграму станів, виписати допустимі послідовності латентностей, вибрати оптимальну. Порівняти оптимальну та жадібну стратегії.

4. Написати програму, що моделює:

- роботу статичного конвеєра, що реалізує відповідну варіанту таблицю зайнятості, з оптимальною стратегією управління;

- роботу динамічного конвеєра, що виконує випадкову рівноймовірну суміш двох таблиць зайнятості. Друга таблиця зайнятості відповідає варіанту, який слідує за вашим. (для останнього варіанту друга таблиця зайнятості вибирається відповідно до першого варіанту).

Варіант 22

Відповідно варіант №2

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	*	*				*	*	
2			*					
3				*	*			*

Відповідно варіант №3

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
1	*				*			*
2		*				*		
3			*	*			*	

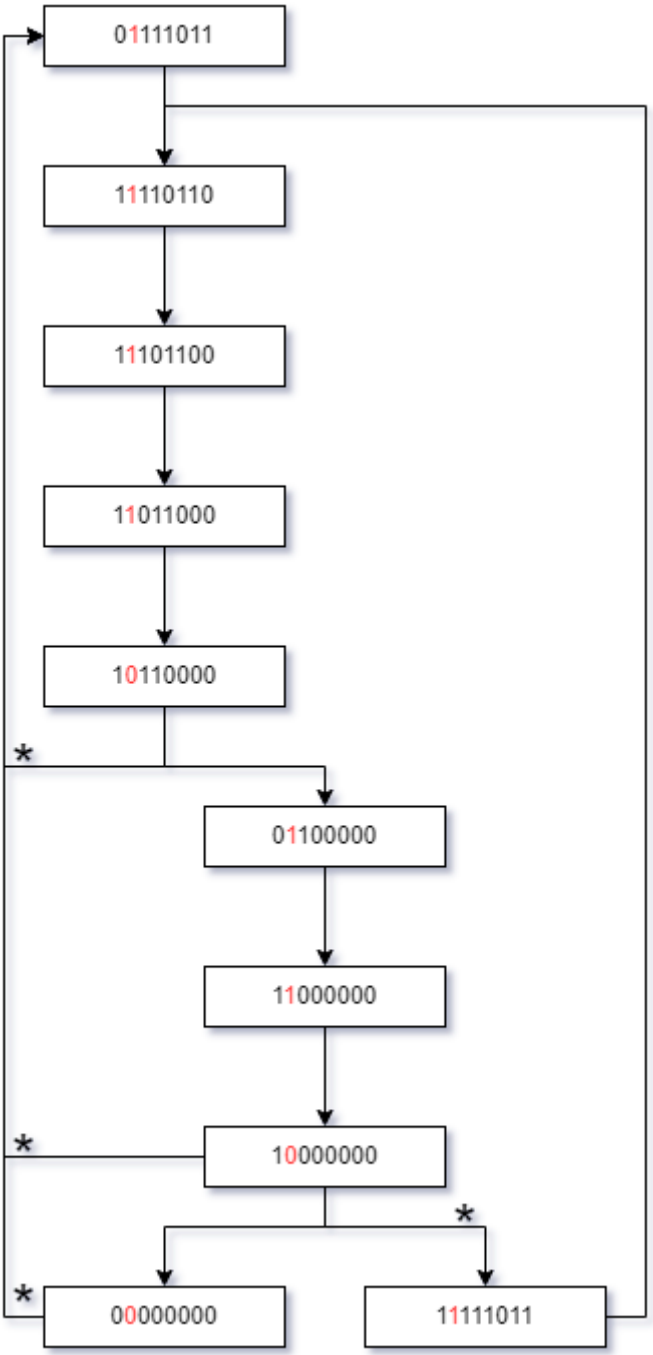

Рис. 1 – Повна діаграма станів

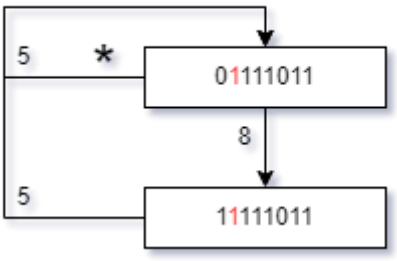

Рис. 2 – Модифікована діаграма станів

Список допустимих латентностей для аналізованої таблиці зайнятості, порівняння жадібної та оптимальної стратегії

Для аналізованої таблиці зайнятості **допустимими латентностями є 5 і 8 тактів**, що дозволяє уникнути конфліктів під час запуску нових ітерацій в конвеєрі. Латентність 5 дозволяє частіше ініціювати нові процеси, але може призводити до неповного використання ресурсів конвеєра. Латентність 8 забезпечує більшу паузу між ітераціями, що може бути корисним для уникнення зіткнень.

Жадібна стратегія використовує латентність 5 для всіх запусків, що забезпечує високу частоту ітерацій, але може призвести до часткового простою етапів конвеєра. Це швидка стратегія, проте не завжди оптимальна.

Оптимальна стратегія чергує латентності 5 і 8, що дозволяє досягти більш ефективного використання ресурсів конвеєра. Таке чергування дозволяє підтримувати циклічний баланс завантаження конвеєра та забезпечує максимальну пропускну здатність системи.

Текст програми

```
=== ./Program.cs ===
using comp_sys_lab2;
using Task = comp_sys_lab2.Task;

static (List<Task> tasks, List<Processor> processors) Process(List<Task> tasks,
List<Processor> processors, List<int> table)
{
    foreach (var task in tasks)
    {
        int curStage = Convert.ToInt32(task.Stage) - 1;
        if (curStage != -1)
        {
            if (((task.ID > 0 && tasks[Convert.ToInt32(task.ID) - 1].Stage < 3)
|| task.ID == 0)
                && processors[table[curStage]].Target.Stub)
            {
                processors[table[curStage]].Target = task;
                task.Stage -= processors[table[curStage]].Performance;
            }
        }
    }
    return (tasks, processors);
}

// Set params
const int processorCount = 3;
int taskCount = 0;
uint tick = 0;
bool twoTables = false;
List<List<int>> tables = [];
// Variant 2
tables.Add([0, 0, 1, 2, 2, 0, 0, 2]);
// Variant 3
tables.Add([0, 1, 2, 2, 0, 1, 2, 0]);
Random random = new();

Console.Write("\nInput task count( > 0): ");
do
{
    taskCount = Convert.ToInt32(Console.ReadLine());
    if (taskCount <= 0)
        Console.WriteLine("Incorrect task count value, try again");
} while (taskCount <= 0);


ConsoleKey info = ConsoleKey.Help;
do
{
    Console.Write("\nUse two tables(y/n): ");
```

```
    info = Console.ReadKey().Key;
    if (info is not (ConsoleKey.N or ConsoleKey.Y))
        Console.WriteLine("Try again...");
} while (info is not (ConsoleKey.N or ConsoleKey.Y));
if (info == ConsoleKey.Y)
    twoTables = true;
Console.WriteLine();


// Create Processors
Task StubTask = new(0, 0, 0, true);
List<Processor> processors = [];
for (uint i = 0; i < processorCount; i++)
    processors.Add(new Processor(i, 1, StubTask));

//Create Tasks
List<Task> tasks = [];
for (uint i = 0; i < taskCount; i++)
    tasks.Add(new Task(i, 8, 8, false));


// Work-on
while (tasks.Count > 0 && !tasks.All(item => item.Stage == 0))
{
    (List<Task> tasks, List<Processor> processors) result;
    result = twoTables ? Process(tasks, processors, tables[random.Next(0,
tables.Count)]) : Process(tasks, processors, tables[0]);

    Console.Write($" {tick} |");
    foreach (Processor processor in processors)
    {
        Console.Write("\t" + processor.ToString());
        processor.Target = StubTask;
    }
    Console.WriteLine();
    tick++;
}
Console.WriteLine($"Count of ticks: {tick}");

// Exit
do
    Console.WriteLine("\nPress Q to exit");
while (Console.ReadKey().Key != ConsoleKey.Q);

namespace comp_sys_lab2
{
    class Task(uint ID, uint Complexity, uint Stage, bool Stub)
    {
        public uint ID = ID;
        public uint Complexity = Complexity;
        public uint Stage = Stage;
```

```
        public bool Stub = Stub;
    }
    class Processor(uint ID, uint Performance, Task Target)
    {
        public uint ID = ID;
        public uint Performance = Performance;
        public Task Target = Target;

        public override string ToString() => !Target.Stub ?
$"[{ID}|{Target.Stage}]" : $"[{ID}|-]";
    }
}
```

Скріншоти роботи програми

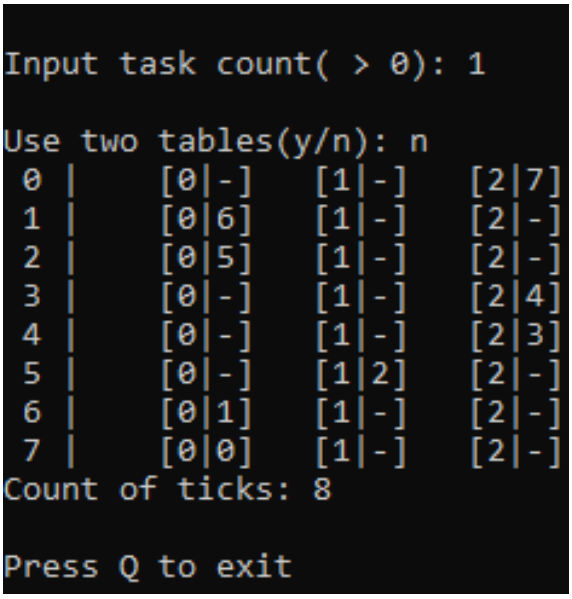

Рис. 3 – Статичний конвеєр, 1 задача

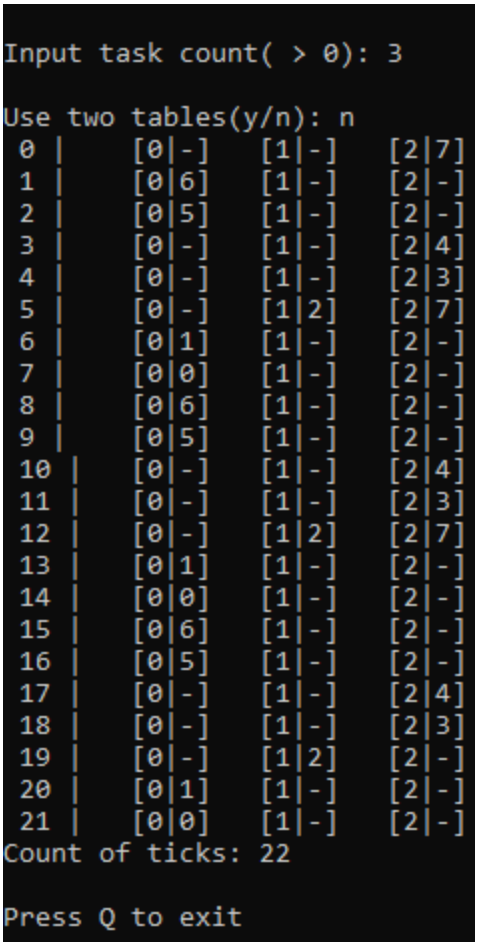

Рис. 4 – Статичний конвеєр, 3 задачі

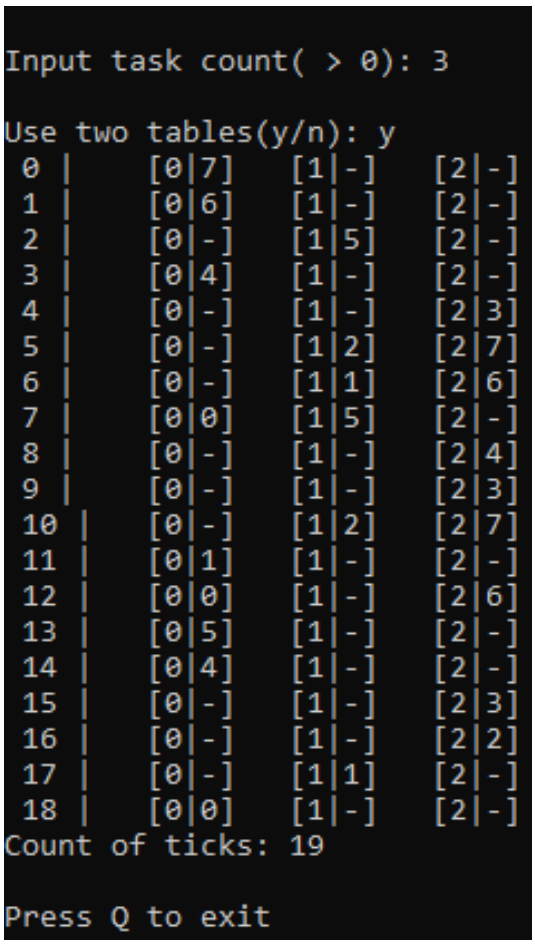

Рис. 5 – Динамічний конвеєр, 3 задачі

Висновки

У результаті виконання лабораторної роботи було досліджено функціонування конвеєрної обчислювальної системи, зокрема стратегії управління, що дозволяють уникнути конфліктів при запуску нових ітерацій. Було виявлено, що чергування латентностей 5 і 8 тактів є оптимальною стратегією для даної таблиці зайнятості, оскільки це забезпечує більш ефективне використання ресурсів у порівнянні з жадібною стратегією, що використовує тільки мінімальну латентність.

Жадібна стратегія, хоча й забезпечує високу частоту запусків ітерацій, може призводити до часткового простою етапів конвеєра через недооцінку потенційних конфліктів. Оптимальна стратегія, яка чергує латентності, дозволяє збалансувати навантаження та досягти максимального рівня пропускної здатності системи.

Отримані результати демонструють важливість ретельного вибору стратегії диспетчеризації для підвищення продуктивності конвеєрних обчислювальних систем, особливо у випадках зі складними конфігураціями етапів і можливими конфліктами між ними.