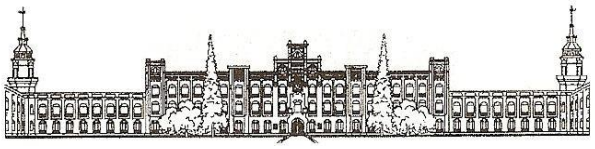

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Лабораторна робота №1

з дисципліни «Комп'ютерні системи»

«ПЛАНУВАННЯ ЗАДАЧ У БАГАТОПРОЦЕСОРНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ»

Виконав студент IV курсу

групи: КВ-11

ПІБ: Терентьєв Іван Дмитрович

Перевірив: _____

Київ 2024

Завдання для лабораторної роботи

1. Ознайомитись з описом лабораторної роботи.

2. Потрібно опрацювати (надати 4 набора параметрів роботи системи):

1) за алгоритмом FIFO;

2) за алгоритмом з окремим процесором планувальником (коли найслабкіший з точки зору продуктивності процесорний елемент є планувальником);

3) за алгоритмом, по якому функції планування покладені на найбільш потужний процесорний елемент, який періодично перериває обчислення для управління чергою. Цей процесор є планувальником, але й приймає безпосередньо участь у обчислювальному процесі. Визначити час роботи процесора над задачами як 20 мс, час на планування – 4 мс;

4) те саме, що і у попередньому пункті із найпотужнішим процесором у якості планувальника, але визначити час роботи над задачами самостійно, виходячи з оптимальної швидкодії системи в цілому.

Відповіддю у лабораторній роботі є кількість реалізованих задач (виконаних системою операцій) за 10 с.

Треба вказати співвідношення кількості виконаних системою операцій до максимально можливої кількості операцій (своєрідний ККД системи). Максимально можлива кількість операцій – це сума продуктивностей працюючих процесорів за 10 с (слід враховувати, що у пп. 3) та 4) найпотужніший процесор працює не весь час, перериваючись на функції планувальника.

Програмний інтерфейс бажано зробити таким, щоб він надавав можливість задавати швидкодію усіх процесорів системи, імовірність виникнення задач, границі складності задач.

Блок-схеми алгоритмів планування

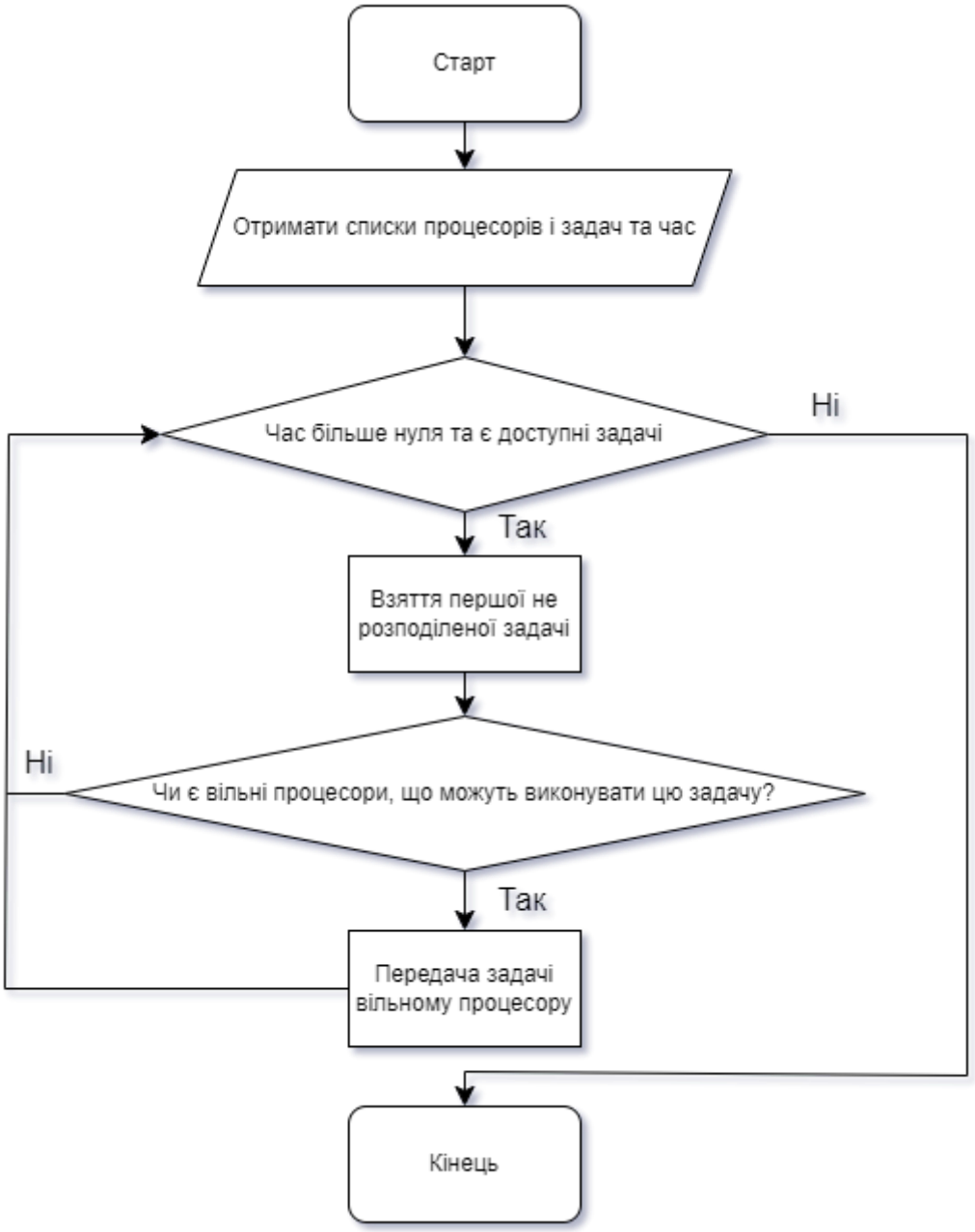

Рис. 1 – Блок схема алгоритму FIFO

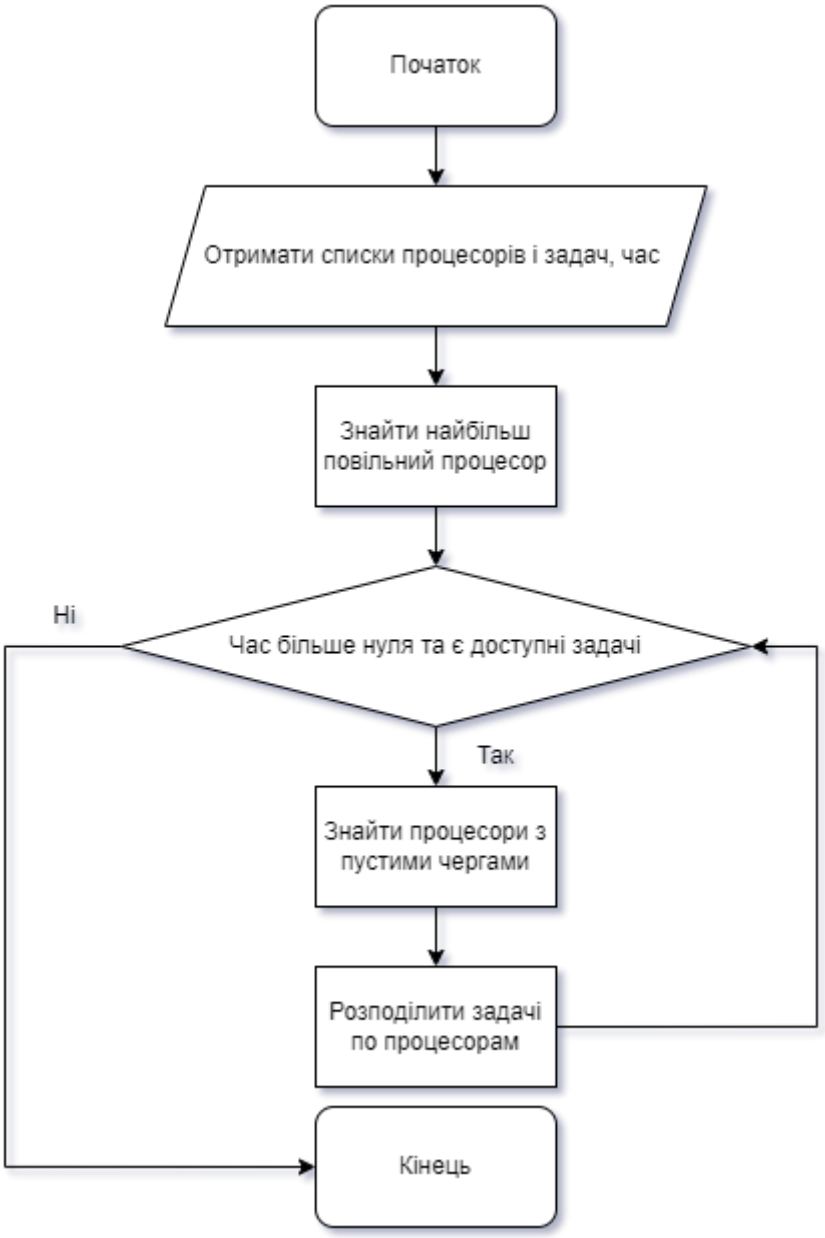

Рис. 2 – Блок схема алгоритму, де функції планування покладені на найбільш повільний процесорний елемент

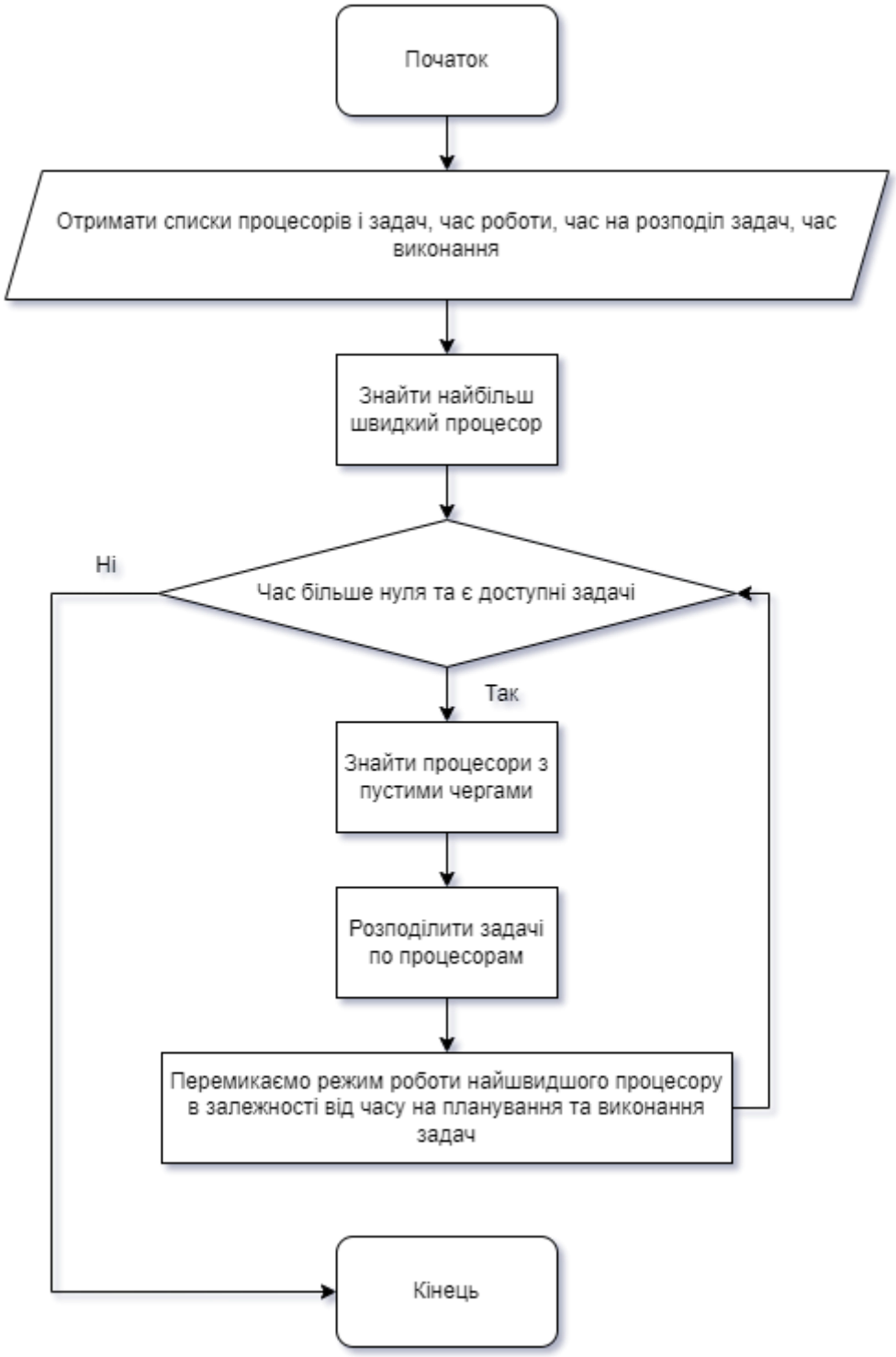

Рис. 3 – Блок схема алгоритму, де функції планування покладені на найбільш швидкий процесорний елемент

Текст програми

```
=== ./ProcessorUnit.cs ===
namespace comp_sys_lab1
{
    class ProcessorUnit
    {


        private uint myNumber = 0;
        private uint performanceClass = 1;
        private uint completedTasksCount = 0;
        private uint completedOperationsCount = 0;
        private uint elapsedOnScheduler = 0;
        public bool workedAsScheduler = false;
        public bool workingAsScheduler = false;
        public uint workingInSchedulerModeTime = 0;
        public uint workingInDefaultModeTime = 0;
        public uint lastLoadTime = 0;
        private List<Task> queue = [];

        public ProcessorUnit(uint myNumber) { this.myNumber = myNumber; }
        public ProcessorUnit() { }
        public void SetPerformance(uint performanceClass) =>
this.performanceClass = performanceClass;
        public uint GetPerformance() => performanceClass;

        public uint GetCompletedTasksCount() => completedTasksCount;

        // CTC - completed tasks count
        public void IncCTC() => completedTasksCount++;

        public uint GetCompletedOperationsCount() => completedOperationsCount;

        // COC - completed operations count
        public void IncCOC() => completedOperationsCount += performanceClass;

        public void Clear()
        {
            completedOperationsCount = 0;
            completedTasksCount = 0;
            elapsedOnScheduler = 0;
            queue = [];
            workedAsScheduler = false;
            workingAsScheduler = false;
            workingInSchedulerModeTime = 0;
            workingInDefaultModeTime = 0;
        }

        public void IncSchedulTime() => elapsedOnScheduler++;
```

```
        public uint GetTimeElapsedOnScheduler() => elapsedOnScheduler;

        public void AddTask(Task task)
        {
            queue.Add(task);
            lastLoadTime = task.complexity;
        }

        private void RemTask(Task task) => queue.Remove(task);

        public void Tick()
        {
            if (workingAsScheduler)
            {
                IncSchedulTime();
                IncSchModeTime();
            }
            else
            {
                if (queue.Count > 0)
                {
                    IncCOC();
                    IncDefModeTime();
                    if (queue.First().Tick(performanceClass))
                    {
                        completedOperationsCount -= queue.First().getBalance();
                        queue.Remove(queue.First());
                        IncCTC();
                    }
                }
            }
        }

        public bool IsQueueEmpty() => queue.Count == 0;
        public int QueueCount() => queue.Count;

        public bool CanWorkWithTask(Task task) =>
task.availableProcessors.Contains(myNumber);

        public void EnableSchedulerMode()
        {
            workedAsScheduler = true;
            workingAsScheduler = true;
            workingInDefaultModeTime = 0;
        }

        public void DisableSchedulerMode()
        {
            workingAsScheduler = false;
            workingInSchedulerModeTime = 0;
        }
```

```
        public void IncSchModeTime() => workingInSchedulerModeTime++;
        public void IncDefModeTime() => workingInDefaultModeTime++;

        public uint getMyNumber() => myNumber;
    }
}


=== ./Program.cs ===
using System.Text;
namespace comp_sys_lab1
{
    class Program
    {
        static readonly uint taskCount = 20000;
        static readonly uint processorsCount = 5;
        static readonly uint elapsedTime = 10000;
        static readonly uint minimalL = 10;
        static readonly uint maximumH = 200;
        static public void SetProcessorPerfomance(ProcessorUnit processorUnit,
uint i)
        {
            Console.WriteLine($"Input processor unit #{i} performance(from 1 to
10): ");
            uint perf;
            do
            {
                perf = Convert.ToUInt32(Console.ReadLine());
                if (perf < 1 || perf > 10)
                    Console.WriteLine("Incorrect perfomance value, try again");
            } while (perf < 1 || perf > 10);
            processorUnit.SetPerformance(perf);
        }
        static public uint SetChance()
        {
            uint probability = 80;
            Console.WriteLine("Set chance? Y?");
            ConsoleKey k = Console.ReadKey().Key;
            Console.WriteLine("\n");
            if (k == ConsoleKey.Y)
            {
                Console.WriteLine("\nInput probability value(from 1 to 100): ");
                do
                {
                    probability = Convert.ToUInt32(Console.ReadLine());
                    if (probability < 1 || probability > 100)
                        Console.WriteLine("Incorrect probability value, try
again");
                } while (probability < 1 || probability > 100);
            }
```

```
            return probability;
        }
        static public Tuple<uint, uint> SetBorders(uint slowestPerf)
        {
            Console.WriteLine("Set borders? Y?");
            ConsoleKey k = Console.ReadKey().Key;
            Console.WriteLine("\n");
            uint hres;
            uint lres;
            if (k == ConsoleKey.Y)
            {
                Console.WriteLine($"Input minimal complexity(>= {slowestPerf *
minimalL} AND <= {slowestPerf * maximumH})");
                do
                {
                    lres = Convert.ToUInt32(Console.ReadLine());
                    if (slowestPerf * minimalL > lres)
                        Console.WriteLine("Incorrect minimal complexity, try
again");
                } while (slowestPerf * minimalL > lres);
                Console.WriteLine($"\nInput maximum complexity(<= {slowestPerf *
maximumH} AND >= {lres})");
                do
                {
                    hres = Convert.ToUInt32(Console.ReadLine());
                    if (hres > slowestPerf * maximumH || hres < lres)
                        Console.WriteLine("Incorrect maximum complexity, try
again");
                } while (hres > slowestPerf * maximumH || hres < lres);
            }
            else
            {
                lres = slowestPerf * minimalL;
                hres = slowestPerf * maximumH;
            }
            return new Tuple<uint, uint>(lres, hres);
        }

        static public void OutTable(List<ProcessorUnit> processorUnits, uint
elapsedTime)
        {
            // Calculating statistics
            ulong completedTasksCount = 0;
            foreach (var processorUnit in processorUnits)
                completedTasksCount += processorUnit.GetCompletedTasksCount();

            ulong potentialOperationsCount = 0;
            foreach (var processorUnit in processorUnits)
                potentialOperationsCount += (processorUnit.GetPerformance() *
elapsedTime) - processorUnit.GetTimeElapsedOnScheduler();
```

```
            ulong resultOperationsCount = 0;
            foreach (var processorUnit in processorUnits)
                resultOperationsCount +=
processorUnit.GetCompletedOperationsCount();

            decimal efficiency = Math.Round((decimal)resultOperationsCount /
potentialOperationsCount * 100, 2);

            // Out statistics
            Console.WriteLine($"Completed tasks: {completedTasksCount}");
            for (int i = 0; i < processorUnits.Count; i++)
                Console.WriteLine($"Processor unit #{i + 1} have done
{processorUnits[i].GetCompletedTasksCount()} tasks");

            Console.WriteLine($"Theoretical count of completed operations:
{potentialOperationsCount}");
            Console.WriteLine($"Real count of completed operations:
{resultOperationsCount}");
            Console.WriteLine($"Efficiency: {efficiency}");
        }
        static public void cleanProcessorUnits(List<ProcessorUnit>
processorUnits)
        {
            foreach (var processorUnit in processorUnits)
                processorUnit.Clear();
        }
        static public void OutAndClean(List<ProcessorUnit> processorUnits, uint
elapsedTime)
        {
            OutTable(processorUnits, elapsedTime);
            cleanProcessorUnits(processorUnits);
        }

        static void Main()
        {
            // 0. Set UTF-8
            Console.OutputEncoding = Encoding.UTF8;

            // 1. Set processor performance
            List<ProcessorUnit> processorUnits = [];
            for (uint i = 0; i < processorsCount; i++)
            {
                processorUnits.Add(new ProcessorUnit(i));
                SetProcessorPerfomance(processorUnits.Last(),
Convert.ToUInt32(processorUnits.Count));
            }
            // 2. Set chance
            uint chance = SetChance();
            // 3. Set complexity borders
            ProcessorUnit slowestOne;
            slowestOne = processorUnits.MinBy(x => x.GetPerformance())!;
```

```
            Tuple<uint, uint> borders = SetBorders(slowestOne.GetPerformance());
            // 4. Generate processes
            TaskGenerator taskGenerator = new(borders, chance, processorsCount);
            List<Task> tasks = taskGenerator.GetTasks(taskCount);
            // 5. Set time
            Console.WriteLine($"Elapsed time: {elapsedTime} ms");
            // 6. First algo
            Console.WriteLine("===FIFO===");
            SchedulerFIFO schedulerFIFO = new(processorUnits, tasks,
elapsedTime);
            OutAndClean(processorUnits, elapsedTime);
            // 7. Second algo
            Console.WriteLine("===Slowest processor unit as scheduler===");
            SchedulerSPUaS schedulerSPUaS = new(processorUnits, tasks,
elapsedTime);
            OutAndClean(processorUnits, elapsedTime);
            // 8. Third algo
            Console.WriteLine("===Fastest processor unit as scheduler by
interrupt===");
            SchedulerFPUaS schedulerFPUaS = new(processorUnits,tasks,
elapsedTime,20,4,false);
            OutAndClean(processorUnits, elapsedTime);
            // 9. Third algo with best settings
            Console.WriteLine("===Fastest processor unit as scheduler by
interrupt(best settings)===");
            SchedulerFPUaS schedulerFPUaSBestSettings = new(processorUnits,
tasks, elapsedTime, 240, 1,true);
            OutAndClean(processorUnits, elapsedTime);
            // 10. Exit
            do
            {
                Console.WriteLine("\nPress Q to exit");
            } while (Console.ReadKey().Key != ConsoleKey.Q);
        }
    }
}

=== ./SchedulerFIFO.cs ===
namespace comp_sys_lab1
{
    class SchedulerFIFO
    {

        public SchedulerFIFO(List<ProcessorUnit> processorUnits, List<Task>
tasks, uint time)
        {
            while (tasks.Count > 0 && time > 0)
            {
                while (TryAddTask(processorUnits, tasks)) ;
                foreach (ProcessorUnit unit in processorUnits)
                    unit.Tick();
```

```
                time--;
            }
        }

        private bool TryAddTask(List<ProcessorUnit> processorUnits, List<Task>
tasks)
        {
            Task current = tasks.First();
            ProcessorUnit unit =
processorUnits.Find(x=>x.CanWorkWithTask(current) && x.IsQueueEmpty())!;
            if (unit != null)
            {
                unit.AddTask(current);
                tasks.Remove(current);
                return true;
            }
            return false;

        }
    }
}


=== ./SchedulerFPUaS.cs ===
namespace comp_sys_lab1
{
    class SchedulerFPUaS
    {
        public SchedulerFPUaS(List<ProcessorUnit> processorUnits, List<Task>
tasks, uint time, uint workTime, uint planTime, bool intellectual)
        {
            processorUnits.MaxBy(x => x.GetPerformance())!.EnableSchedulerMode();
            while (tasks.Count > 0 && time > 0)
            {

                List<ProcessorUnit> emptyQueueUnits = processorUnits.FindAll(x =>
x.IsQueueEmpty());
                foreach (ProcessorUnit processorUnit in emptyQueueUnits)
                {
                    Task desiredTask = tasks
        .Where(t => t.availableProcessors.Contains(processorUnit.getMyNumber()))
        .OrderBy(t => t.availableProcessors.Count)
        .FirstOrDefault()!;
                    if (desiredTask != null)
                    {
                        processorUnit.AddTask(desiredTask);
                        tasks.Remove(desiredTask);
                    }
                }
                foreach (ProcessorUnit unit in processorUnits)
```

```
                    unit.Tick();

                time--;
                if (processorUnits.MaxBy(x =>
x.GetPerformance())!.workingInSchedulerModeTime >= planTime)
                {
                    processorUnits.MaxBy(x =>
x.GetPerformance())!.DisableSchedulerMode();
                }
                if(processorUnits.MaxBy(x =>
x.GetPerformance())!.workingInDefaultModeTime >= workTime)
                {
                    processorUnits.MaxBy(x =>
x.GetPerformance())!.EnableSchedulerMode();
                    if (intellectual)
                    {
                        workTime = (workTime + processorUnits.MaxBy(x =>
x.GetPerformance())!.lastLoadTime)/2;
                    }
                }

            }
        }

    }
}


=== ./SchedulerSPUaS.cs ===
namespace comp_sys_lab1
{
    class SchedulerSPUaS
    {
        public SchedulerSPUaS(List<ProcessorUnit> processorUnits, List<Task>
tasks, uint time)
        {
            while (tasks.Count > 0 && time > 0)
            {
                processorUnits.MinBy(x =>
x.GetPerformance())!.EnableSchedulerMode();
                List<ProcessorUnit> emptyQueueUnits = processorUnits.FindAll(x =>
x.IsQueueEmpty() && !x.workingAsScheduler);
                foreach (ProcessorUnit processorUnit in emptyQueueUnits)
                {
                    Task desiredTask = tasks
        .Where(t => t.availableProcessors.Contains(processorUnit.getMyNumber()))
        .OrderBy(t => t.availableProcessors.Count)
        .FirstOrDefault()!;
                    if (desiredTask != null)
                    {
                        processorUnit.AddTask(desiredTask);
```

```
                        tasks.Remove(desiredTask);
                    }
                }
                foreach (ProcessorUnit unit in processorUnits)
                    unit.Tick();

                time--;
            }
        }

    }
}


=== ./Task.cs ===
namespace comp_sys_lab1
{
    class Task(List<uint> availableProcessors, uint complexity)
    {
        public List<uint> availableProcessors = availableProcessors;
        public uint complexity = complexity;
        public bool status = false; // false - need work, true - done
        public uint balance = 0;
        public bool Tick(uint ticks)
        {
            uint saved = complexity;
            complexity -= ticks;
            if(saved < complexity || complexity == 0)
            {
                status = true;
                balance = saved;
                complexity = 0;
            }
            return status;
        }
        public uint getBalance() => balance;
    }
}


=== ./TaskGenerator.cs ===
namespace comp_sys_lab1
{
    class TaskGenerator(Tuple<uint, uint> borders, uint probability, uint count)
    {
        private readonly List<List<uint>> processorCombo =
GetCombinations(count);
        private readonly double probability = Convert.ToDouble(probability) /
100.0;
        private readonly Random r = new();
```

```
        private Task GenerateTask() => new(processorCombo[r.Next(1,
processorCombo.Count - 1)],
            Convert.ToUInt32(r.Next(Convert.ToInt32(borders.Item1),
Convert.ToInt32(borders.Item2))));

        public List<Task> GetTasks(uint count)
        {
            List<Task> tasks = [];
            for (uint i = 0; i < count; i++)
                if (r.NextDouble() < probability)
                {
                    Task gen = GenerateTask();
                    tasks.Add(gen);
                }
            return tasks;
        }

        private static List<List<uint>> GetCombinations(uint count)
        {
            List<uint> elements = [];
            for (uint i = 0; i < count; i++)
                elements.Add(i);

            List<List<uint>> result = [];
            for (int subsetSize = 0; subsetSize <= elements.Count; subsetSize++)
                result.AddRange(GetCombinations(elements, subsetSize));

            return result;
        }

        private static List<List<uint>> GetCombinations(List<uint> elements, int
subsetSize)
        {
            if (subsetSize == 0)
                return [[]];

            if (elements.Count == 0)
                return [];

            var firstElement = elements[0];
            var restElements = elements.Skip(1).ToList();
            var combinationsWithFirst = GetCombinations(restElements, subsetSize
- 1);

            foreach (var combination in combinationsWithFirst)
                combination.Insert(0, firstElement);

            var combinationsWithoutFirst = GetCombinations(restElements,
subsetSize);

            return [.. combinationsWithFirst, .. combinationsWithoutFirst];
```

```
        }
    }
}
```

Скріншоти, які демонструють процес роботи програми

```
Input processor unit #1 performance(from 1 to 10):
1
Input processor unit #2 performance(from 1 to 10):
2
Input processor unit #3 performance(from 1 to 10):
3
Input processor unit #4 performance(from 1 to 10):
4
Input processor unit #5 performance(from 1 to 10):
5
Set chance? Y?
n

Set borders? Y?
n

Elapsed time: 10000 ms
===FIFO===
Completed tasks: 903
Processor unit #1 have done 84 tasks
Processor unit #2 have done 157 tasks
Processor unit #3 have done 195 tasks
Processor unit #4 have done 237 tasks
Processor unit #5 have done 230 tasks
Theoretical count of completed operations: 150000
Real count of completed operations: 93511
Efficiency: 62,34
===Slowest processor unit as scheduler===
Completed tasks: 1321
Processor unit #1 have done 0 tasks
Processor unit #2 have done 190 tasks
Processor unit #3 have done 287 tasks
Processor unit #4 have done 387 tasks
Processor unit #5 have done 457 tasks
Theoretical count of completed operations: 140000
Real count of completed operations: 136823
Efficiency: 97,73
===Fastest processor unit as scheduler by interrupt===
Completed tasks: 1360
Processor unit #1 have done 91 tasks
Processor unit #2 have done 202 tasks
Processor unit #3 have done 281 tasks
Processor unit #4 have done 383 tasks
Processor unit #5 have done 403 tasks
Theoretical count of completed operations: 148332
Real count of completed operations: 138596
Efficiency: 93,44
===Fastest processor unit as scheduler by interrupt(best settings)===
Completed tasks: 1453
Processor unit #1 have done 112 tasks
Processor unit #2 have done 190 tasks
Processor unit #3 have done 289 tasks
Processor unit #4 have done 390 tasks
Processor unit #5 have done 472 tasks
Theoretical count of completed operations: 149925
Real count of completed operations: 146276
Efficiency: 97,57

Press Q to exit
```

Рис. 4 – Приклад роботи програми на стандартних налаштуваннях

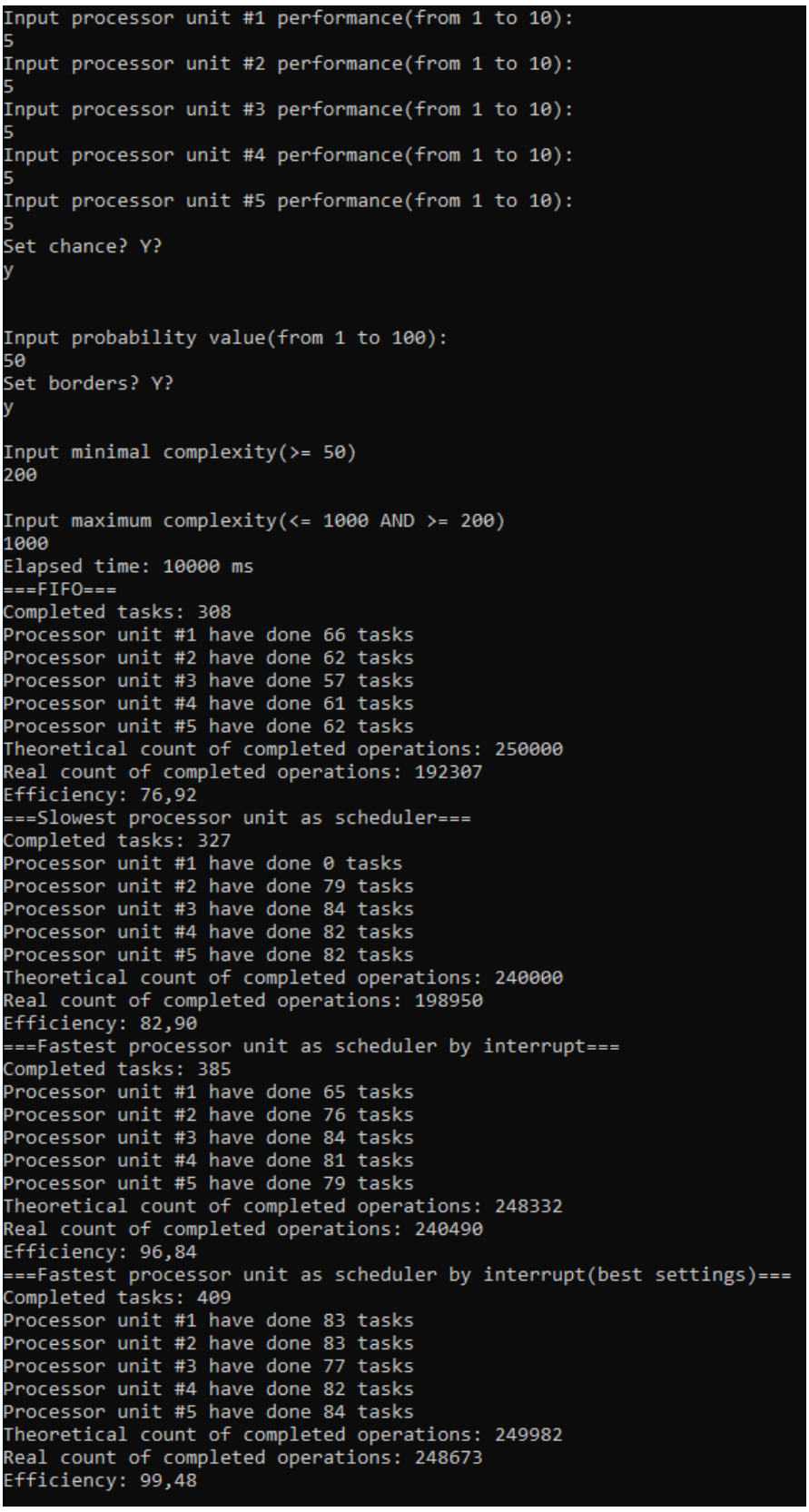

Рис. 5 – Приклад роботи програми з однаковими процесорами, шансом 50% та складністю задач від 200 до 1000(складні)

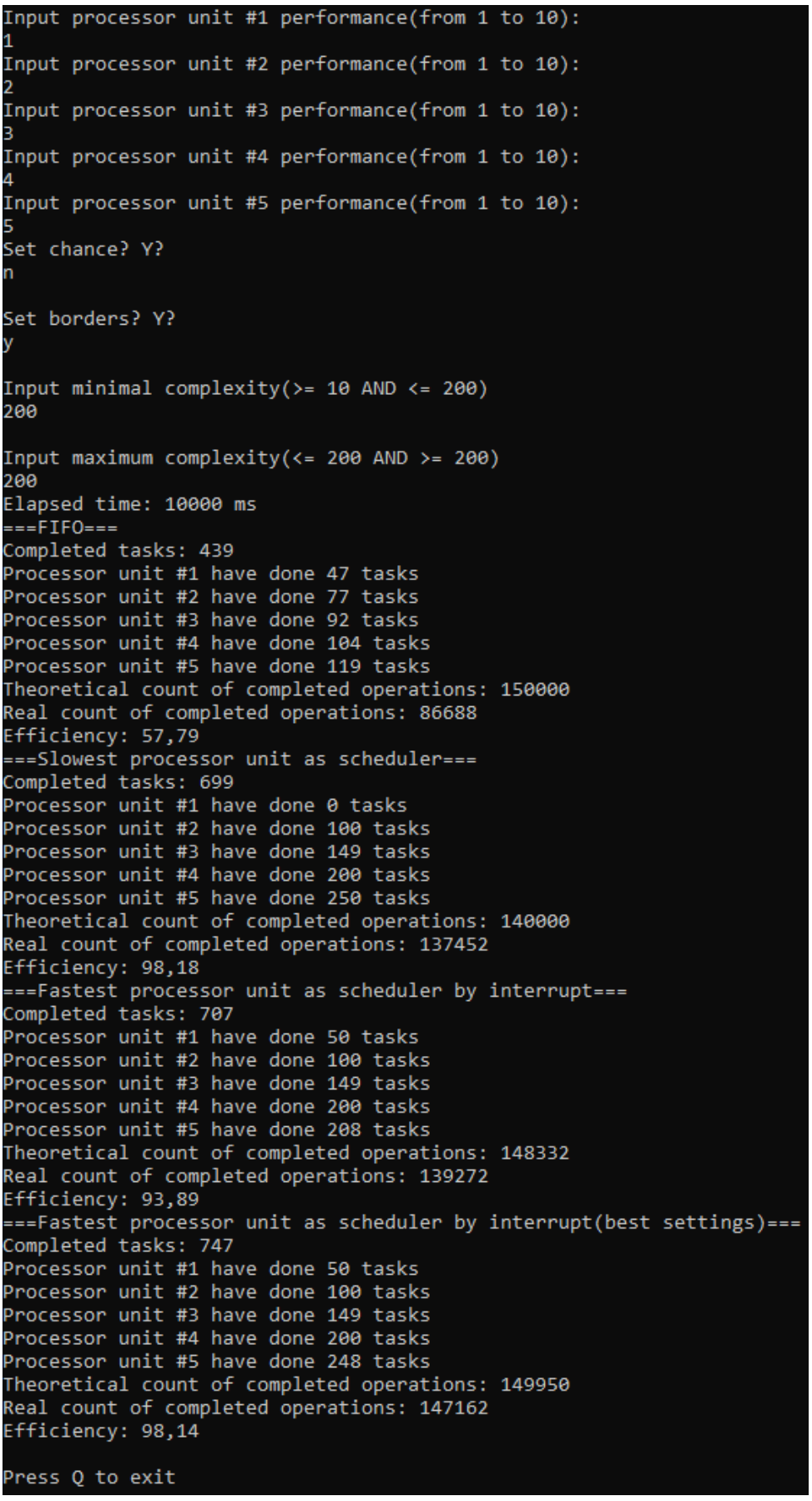

Рис. 6 – Приклад роботи програми на стандартних налаштуваннях з складними задачами

Висновок

У ході виконання лабораторної роботи було досліджено три підходи до планування задач у багатопроцесорних комп’ютерних системах, кожен з яких показав різні результати залежно від умов.

Алгоритм FIFO виявився найпростішим, але продемонстрував низьку ефективність через можливі простої, що виникають при виконанні задач на одному процесорі, тоді як інші процесори можуть простоювати.

Використання алгоритму з окремим планувальником на найслабшому процесорі трохи покращило ситуацію, проте спричинило втрату одного процесорного елемента для виконання обчислювальних задач, що знизило загальну продуктивність системи.

Найефективнішим виявився підхід, де функції планування були покладені на найпотужніший процесор, який періодично переривав свою роботу для управління чергою задач. Це дозволило оптимізувати процес розподілу задач між процесорами та забезпечити кращу продуктивність.

Додавання інтелектуального механізму розподілу часу між плануванням та виконанням задач дозволило досягти максимального коефіцієнта корисної дії, що свідчить про ефективність такого підходу.

У результаті проведених досліджень було зроблено висновок, що правильний вибір алгоритму планування має суттєвий вплив на продуктивність багатопроцесорної системи.