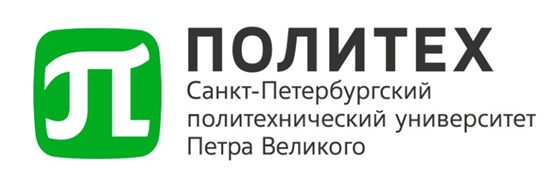
Министерство науки и высшего образования Российской Федерации  
 Санкт-Петербургский политехнический университет Петра Великого  
 Высшая школа программной инженерии



ЛАБОРАТОРНАЯ РАБОТА

ОПИСАНИЕ МОДЕЛИ ОПЕРАЦИОННОЙ СИСТЕМЫ

по дисциплине «Верификация и тестирование программного обеспечения»

Студенты гр. № 5140904/40102 Петунин А.А.

Осокин Д.А.

Руководитель Павлов Е. А.

Санкт-Петербург

2024 г.

Оглавление

[Описание задания 3](#_Toc183743740)

[Описание работы модели/системы: 3](#_Toc183743741)

[Основные задачи 3](#_Toc183743742)

[Расширенные задачи 3](#_Toc183743743)

[Возможные состояния и переходы базовых задач 5](#_Toc183743744)

[Механизм переключения задач 6](#_Toc183743745)

[Приоритеты задач 6](#_Toc183743746)

[Задание 7](#_Toc183743747)

[Ход работы 7](#_Toc183743748)

[Пример работы приложения 8](#_Toc183743749)

[Тестирование 12](#_Toc183743750)

[Тест-кейсы по требованиям к системе 12](#_Toc183743751)

[Модульное тестирование 13](#_Toc183743752)

[Верификация всех инвариантов 15](#_Toc183743753)

[Приложение. Тексты программы 16](#_Toc183743754)

# Описание задания

## Описание работы модели/системы:

Операционная система обеспечивает контролируемое выполнение в реальном времени нескольких процессов и обеспечивает параллельное и асинхронное выполнение задач.

В системе присутствует планировщик, который организует последовательность выполнения задачи.

В системе может быть два типа задач:

* Основные
* Расширенные

## Основные задачи

Основные задачи освобождают процессор если:

* Задача завершилась,
* Произошло переключение на более приоритетную задачу.

Расширенные задачи отличаются от основных тем, что они могут находится в состоянии ожидания(WaitEvent). Состояние ожидания позволяет процессору быть освобожденным и переназначенным на задачу с более низким приоритетом без необходимости завершать выполнение расширенной задачи.

## Расширенные задачи

Расширенные задачи имеют четыре состояния:

* **running** - Выполнение задачи. Только одна задача может находиться в этом состоянии в любой момент времени.
* **ready** – Задача завершила свое выполнение. Планировщик решает, какая готовая задача будет выполнена следующей.
* **waiting** – Задача ожидает выполнения другого события
* **suspended** – Задача находится в подвешенном состоянии и может быть активирована.

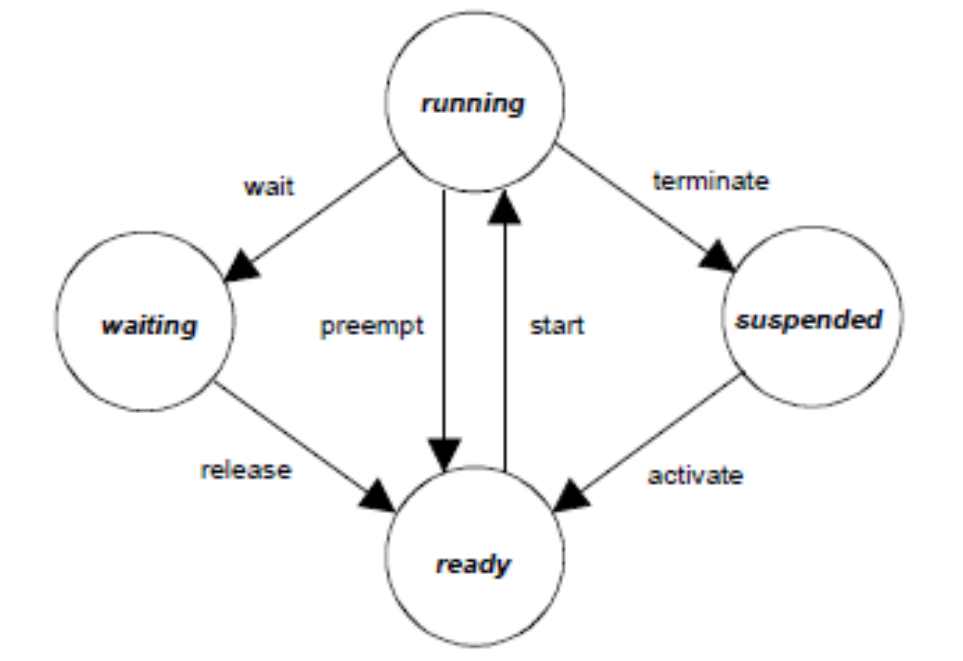


Рисунок 1. Расширенная модель переходов.

Таблица 1. Описание переходов.



**Завершение задачи возможно только в том случае, если она завершается сама**. Это ограничение снижает сложность операционной системы. Прямого перехода из состояния **suspended** в состояние **waiting** не предусмотрено.

## Возможные состояния и переходы базовых задач

Модель состояний базовых задач практически идентична модели состояний расширенных задач. Единственным отличием является то, что основные задачи не имеют состояния ожидания.

* **running** - Выполнение задачи. Только одна задача может находиться в этом состоянии в любой момент времени.
* **ready** - Задача завершила свое выполнение. Планировщик решает, какая готовая задача будет выполнена следующей.
* **suspended** - задача находится в подвешенном состоянии и может быть активирована.

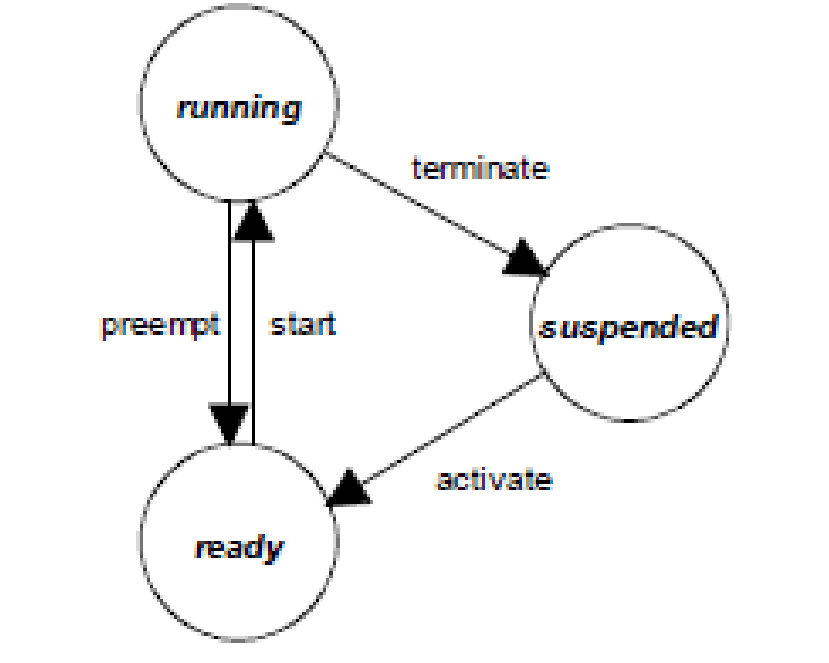


Рисунок 2. Модель переходов состояний базовой задачи.

Таблица 2. Описание переходов базовой задачи.



## Механизм переключения задач

За переключение задач отвечает планировщик. Планировщик – это своего рода ресурс, который захватывается и освобождается задачами. Таким образом, задача может зарезервировать планировщик, чтобы избежать переключения задачи до тех пор, пока она не будет освобождена.

## Приоритеты задач

Планировщик выбирает задачи для выполнения из списка задач, которые находятся в состоянии **ready** основываясь на их приоритете.

Всего существует три уровня приоритета 0-3, где 0 – это наименьший приоритет задачи.

Задачи с одинаковым уровнем приоритета запускаются в зависимости от порядка их активации, при этом расширенные задачи в состоянии ожидания не блокируют запуск последующих задач с одинаковым приоритетом.

Выбор задач происходит по алгоритму FIFO, сначала выполняется первая (самая старая) задача из списка задач с одинаковым приоритетом.

Задача, переходящая из состояния ожидания, рассматривается как последняя (самая новая) задача в очереди **ready**. Планировщик имеет ограничение по количеству задач, которые могут находиться в состоянии **ready**.

На рисунке 3 продемонстрирован пример реализации планировщика, используемого для каждого уровня приоритета. Несколько задач различных приоритетов находятся в состоянии ready, три задачи с приоритетом 3, одна задача с приоритетом 2, одна с приоритетом 1 и две задачи с приоритетом 0. Задача, которая прождала больше всего времени, в зависимости от порядка запроса, находится в нижней части каждой очереди.

После того как процессор обработал и завершил задачу, планировщик выбирает следующую задачу для обработки (приоритет 3, первая очередь). Задачи с приоритетом 2 могут быть обработаны только после того, как все задачи с более высоким приоритетом перейдут из состояний **running** и **ready**.

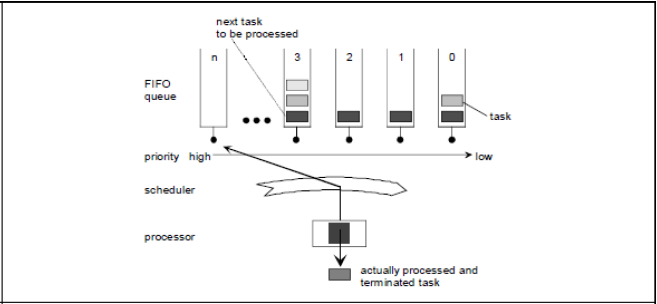


Рисунок 3. Пример работы планировщика, порядок событий.

## Задание

Необходимо реализовать программу по представленному описанию.

Требования:

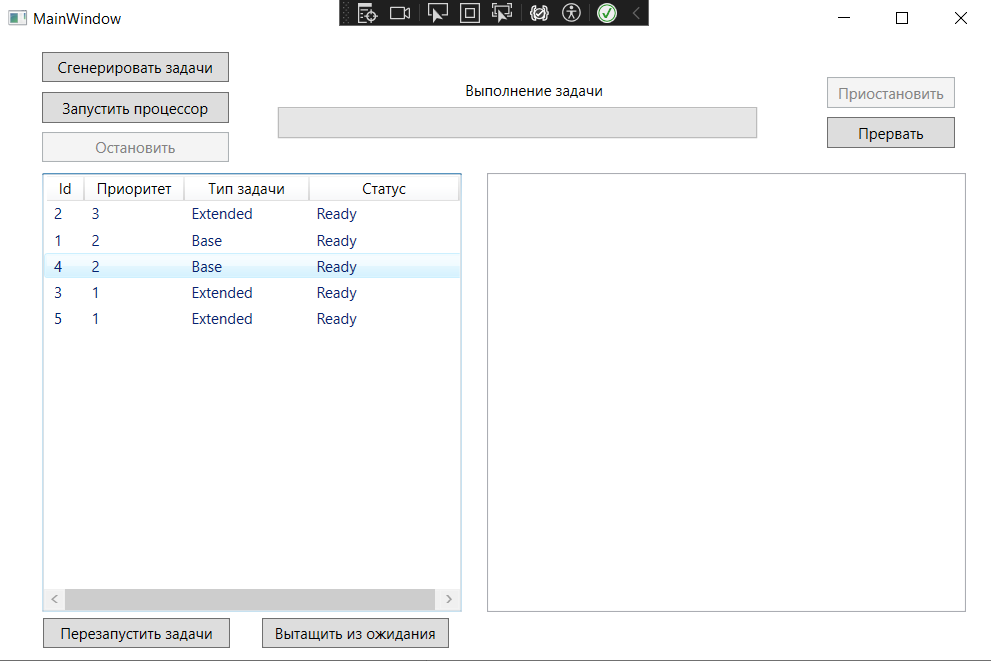
* Язык программирования – любой
* Протестировать написанный код. Покрытия тестами не менее 80% (использовать инструменты оценки тестового покрытия)
* Написать отчет о тестировании
* Обязательно наличия модуля/сервиса для генерации задач.

# Ход работы

В ходе работы мы спроектировали и реализовали приложение на языке C#, которое моделирует работу операционной системы, контролирующей выполнение задач с разными приоритетами. Это приложение включает в себя модуль генерации и инициализации задач и их состояний, модель переходов между этими состояниями и проверку наличия необходимых инвариантов. Также при помощи WPF был реализован пользовательский интерфейс.

## Пример работы приложения

* Начальное состояние

Рисунок 4 - Начальное состояние приложения

1. Процессор запущен

INFO Task processor was started. 2024-11-27 19:27:40.6623

Перед выполнением задачи она удаляется из очереди, отсортированной по приоритету.

1. Запущена задача с id=2.

INFO EXTENDED: id=2 with priority=1 start executing. RUNNING. 2024-11-27 19:27:40.7069

1. Задача c id=2 переведена в очередь ожидания.

INFO EXTENDED: id=2 with priority=1 has been placed in the WAITING queue. (WAIT). 2024-11-27 19:27:43.6551

1. Запущена задача с id=3, в очереди готовых к выполнению задач (ready) освобождается место.

INFO BASE: id=3 with priority=1 start executing. RUNNING. 2024-11-27 19:27:43.6730

1. Свободное место в очереди готовых задач занимает задача из режима ожидания с id=2.

INFO EXTENDED: id=2 with priority=1 has been placed in the READY queue from WAITING. RELEASE 2024-11-27 19:27:43.6852

1. После окончания выполнения задачи с id=3 начинает выполнение задача с id=5.

INFO BASE: id=5 with priority=1 start executing. RUNNING. 2024-11-27 19:27:47.6881

1. Задача с id=5 прервана и перемещена в очередь прерванных задач.

INFO BASE: id=5 with priority=1 has been placed in the SUSPENDED queue. TERMINATE. 2024-11-27 19:27:49.5478

1. Следующей по приоритету является приостановленная задача с id=2, она продолжает выполняться с места ее приостановки.

INFO EXTENDED: id=2 with priority=1 start executing. RUNNING. 2024-11-27 19:27:49.6919

1. После окончания выполнения задачи с id=2 начинает выполнение задача с id=1.

INFO EXTENDED: id=1 with priority=0 start executing. RUNNING. 2024-11-27 19:27:50.7032

1. После окончания выполнения задачи с id=4 начинает выполнение последняя задача в коллекции

INFO EXTENDED: id=4 with priority=0 start executing. RUNNING. 2024-11-27 19:27:54.7113

1. Была нажата кнопка генерации задач, из-за чего в очередь готовых к выполнению задач было добавлено n новых задач, как минимум одна из которых имеет более высокий приоритет. Поэтому задача с id=4 переводится в очередь готовых к выполнению задач.

INFO EXTENDED: id=4 with priority=0 has been placed in the READY queue from RUNNING. PREEMPT. 2024-11-27 19:27:57.8791

1. Начинает выполнение задача с id=10.

INFO EXTENDED: id=10 with priority=3 start executing. RUNNING. 2024-11-27 19:27:57.8934

1. После окончания выполнения задачи с id=10 начинает выполнение задача с id=7.

INFO EXTENDED: id=7 with priority=2 start executing. RUNNING. 2024-11-27 19:28:02.9128

1. Задача с id=7 прервана и перемещена в очередь прерванных задач.

INFO EXTENDED: id=7 with priority=2 has been placed in the SUSPENDED queue. TERMINATE 2024-11-27 19:28:03.0077

1. Начинает выполнение задача с id=6.

INFO EXTENDED: id=6 with priority=1 start executing. RUNNING. 2024-11-27 19:28:03.0077

1. Задача с id=6 прервана и перемещена в очередь прерванных задач.

INFO EXTENDED: id=6 with priority=1 has been placed in the SUSPENDED queue. TERMINATE 2024-11-27 19:28:06.3707

1. Начинает выполнение задача с id=8.

INFO BASE: id=8 with priority=1 start executing. RUNNING. 2024-11-27 19:28:06.3743

1. После окончания выполнения задачи с id=8 начинает выполнение задача с id=9.

INFO EXTENDED: id=9 with priority=1 start executing. RUNNING. 2024-11-27 19:28:11.3817

1. После окончания выполнения задачи с id=9 начинает выполнение задача с id=4.

INFO EXTENDED: id=4 with priority=0 start executing. RUNNING. 2024-11-27 19:28:16.3916

1. Была нажата кнопка «Перезапустить задачи». Задача с id=5 была перемещена из очереди прерванных в очередь готовых к выполнению задач.

INFO BASE: id=5 with priority=1 has been placed in the READY queue from SUSPENDED. ACTIVATE. 2024-11-27 19:28:19.7646

1. Была нажата кнопка «Перезапустить задачи». Задача с id=6 была перемещена из очереди прерванных в очередь готовых к выполнению задач.

INFO EXTENDED: id=6 with priority=1 has been placed in the READY queue from SUSPENDED. ACTIVATE 2024-11-27 19:28:19.7646

1. Начинает выполнение задача с id=5.

INFO BASE: id=5 with priority=1 start executing. RUNNING. 2024-11-27 19:28:22.4009

1. После окончания выполнения задачи с id=5 начинает выполнение задача с id=6.

INFO EXTENDED: id=6 with priority=1 start executing. RUNNING. 2024-11-27 19:28:26.4086

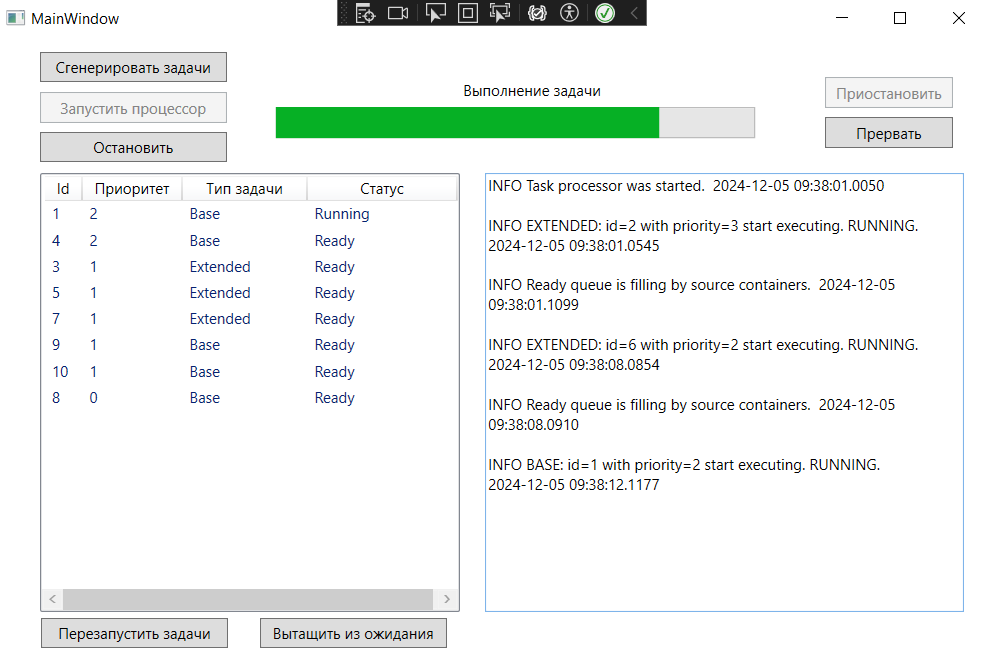


Рисунок 5 - Процесс выполнение задач

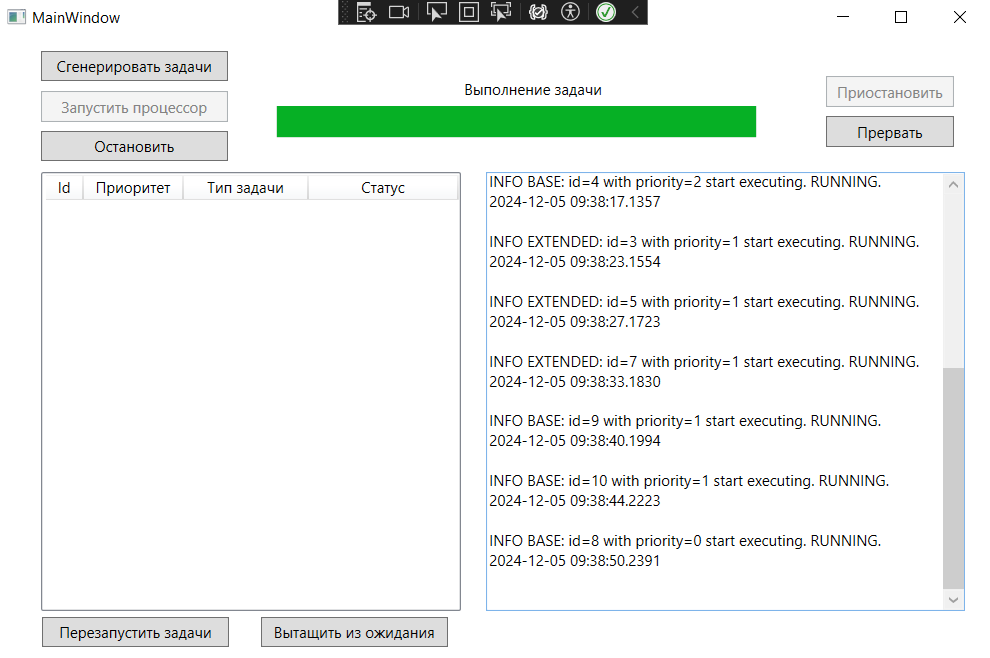


Рисунок 6 – Сгенерированные задачи были выполнены

## Тестирование

## Тест-кейсы по требованиям к системе

Не более N(8) задач может быть в состоянии READY:

1. Начальное состояние. Пустой список задач. Пример на рисунке 7.

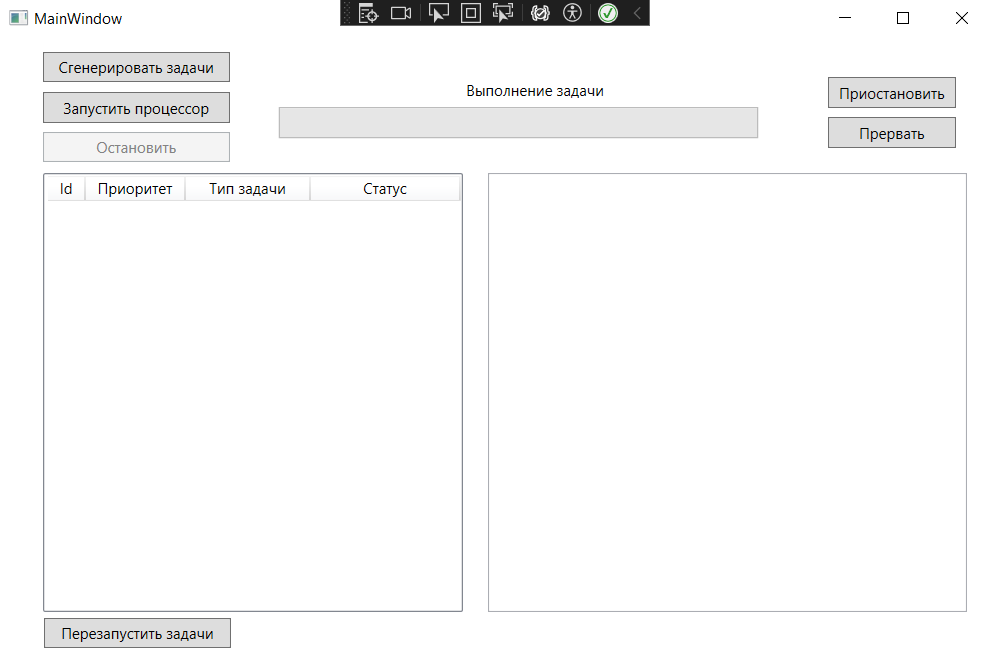


Рисунок 7 - Начальное состояние системы

1. Тройное нажатие на кнопку «Сгенерировать задач». Было сгенерировано 15 задач (одно нажатие генерирует 5 задач). В списке задач готовых к исполнению 8 максимально приоритетных задач. Пример такого поведения на рисунке 8.

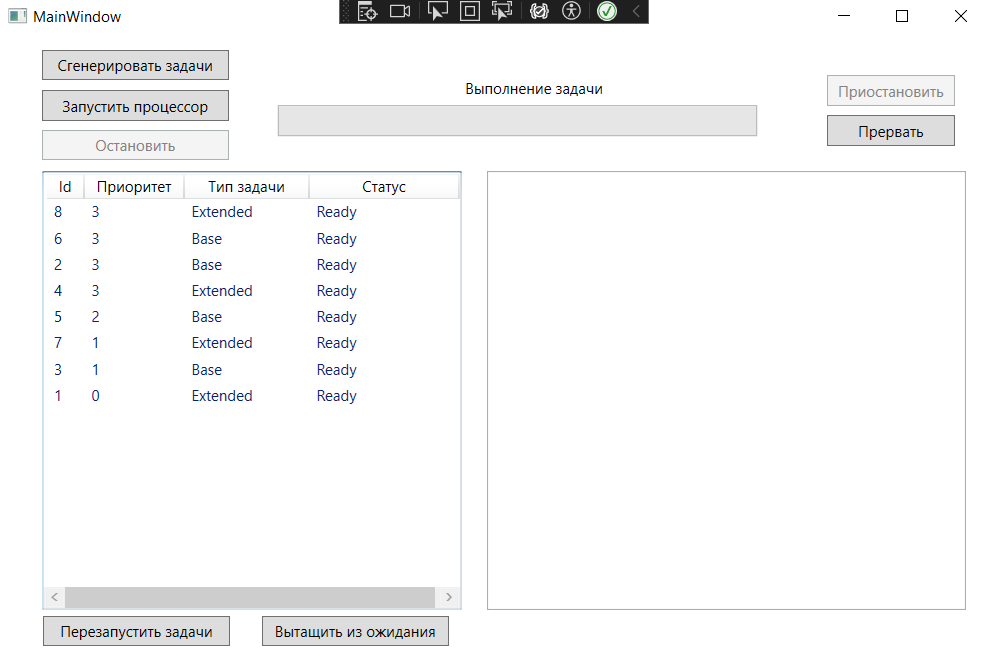


Рисунок 8 - Задачи сгенерированы, а процессор не запущен

1. Нажатие на кнопку «Запустить процессор», после чего список задач готовых к выполнению пополняется из созданных (ожидающих изменения статуса на ready). А также в список задач пользовательского интерфейса добавляется исполняемая задача со статусом running. Пример такого поведения на рисунке 9.

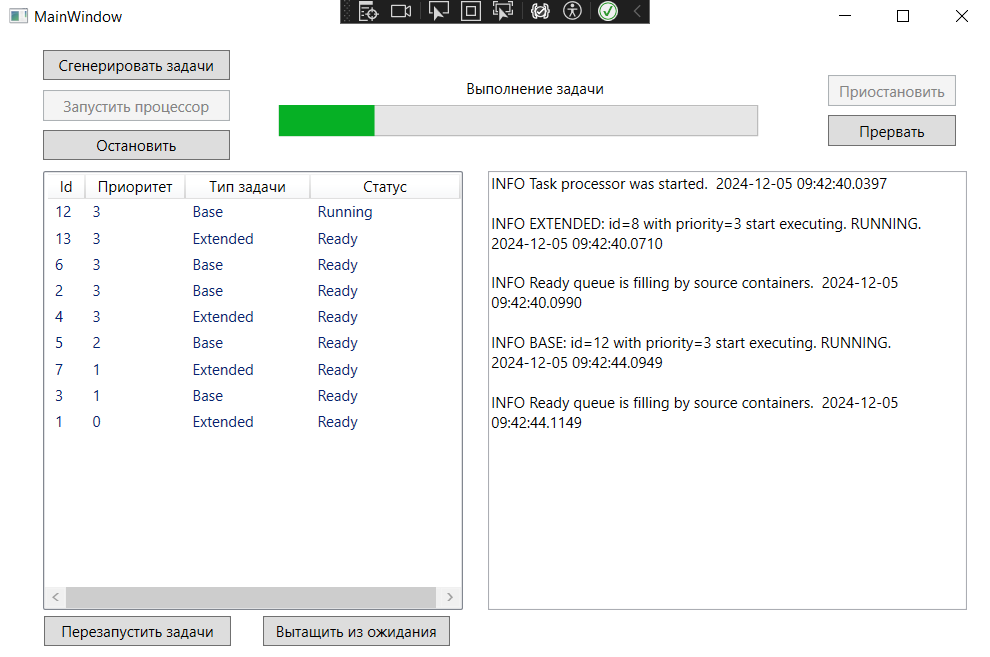


Рисунок 9 - Процессор был запущен

1. Задачи исполняются. Список задач готовых к исполнению имеет постоянную длину равную N до тех пор, пока созданные задачи, не имеющие статус ready, не закончатся. Пример такого поведения на рисунке 10.

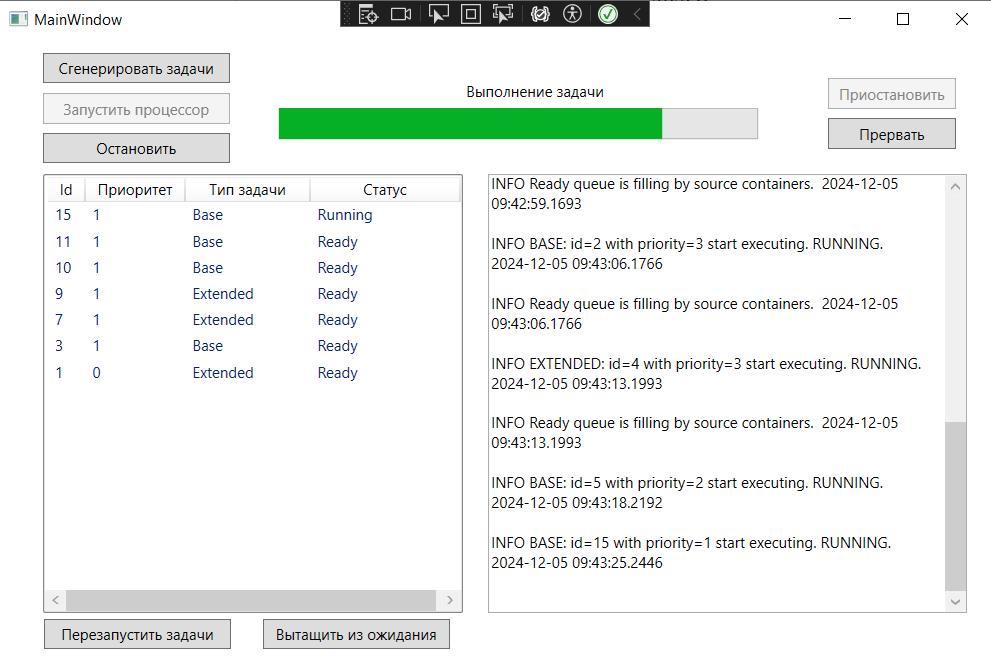


Рисунок 10 - Общее число задач стало меньше N

Тест перехода задач между состояниями:

1. Генерируем 5 задач, запускаем процессор.

INFO Task processor was started. 2024-12-01 19:21:12.8187

1. Прерываем первую выполняющуюся задачу, она переходит в очередь с прерванных задач.

INFO BASE: id=4 with priority=3 start executing. RUNNING. 2024-12-01 19:21:12.8342

INFO BASE: id=4 with priority=3 has been placed in the SUSPENDED queue. TERMINATE. 2024-12-01 19:21:15.4981

1. Следующая задача дожидается выполнения прошлой, после чего запускается. Если она является расширенной приостанавливаем ее, помещая в очередь задач в режиме ожидания.

INFO EXTENDED: id=5 with priority=3 start executing. RUNNING. 2024-12-01 19:21:15.8469

INFO EXTENDED: id=5 with priority=3 has been placed in the WAITING queue. (WAIT). 2024-12-01 19:21:18.7258

1. Автоматически запускается следующая задача с наибольшим приоритетом. Отрабатывает событие запуска новой задачи, сигнализирующее об освободившемся месте в очереди задач, готовых к выполнению. Если N доступных мест в очереди готовых к выполнению задач больше 5, то ожидающая задача переходит в очередь «ready».

INFO EXTENDED: id=3 with priority=2 start executing. RUNNING. 2024-12-01 19:21:18.7325

INFO EXTENDED: id=5 with priority=3 has been placed in the READY queue from WAITING. RELEASE 2024-12-01 19:21:18.7325

1. Следующая задача с наибольшим приоритетом дожидается выполнения прошлой, после чего запускается.

INFO EXTENDED: id=5 with priority=3 start executing. RUNNING. 2024-12-01 19:21:22.7387

1. Следующая задача с наибольшим приоритетом дожидается выполнения прошлой, после чего запускается.

INFO EXTENDED: id=1 with priority=1 start executing. RUNNING. 2024-12-01 19:21:23.7448

1. Прерываем первую выполняющуюся задачу, она переходит в очередь с прерванных задач.

INFO EXTENDED: id=1 with priority=1 has been placed in the SUSPENDED queue. TERMINATE 2024-12-01 19:21:24.9025

1. Автоматически запускается следующая задача с наибольшим приоритетом.

INFO BASE: id=2 with priority=1 start executing. RUNNING. 2024-12-01 19:21:25.7489

1. Нажимаем на кнопку «Перезапустить задачи», чтобы перенести задачи из очереди «suspended» в очередь «ready». В очереди ready есть 2 свободных места для этих задач, поэтому обе задачи переходят в очередь «ready».

INFO BASE: id=4 with priority=3 has been placed in the READY queue from SUSPENDED. ACTIVATE. 2024-12-01 19:21:29.8777

INFO EXTENDED: id=1 with priority=1 has been placed in the READY queue from SUSPENDED. ACTIVATE 2024-12-01 19:21:29.8777

1. Если выполняемая задача имеет приоритет ниже, чем одна из добавленных в очередь «ready» на прошлом шаге, то она добавляется в ready очередь, а задача с большим приоритетом запускается.

INFO BASE: id=2 with priority=1 has been placed in the READY queue from RUNNING. PREEMPT. 2024-12-01 19:21:29.8777

INFO BASE: id=4 with priority=3 start executing. RUNNING. 2024-12-01 19:21:30.7573

1. Следующая задача с наибольшим приоритетом и наиболее ранним временем создания дожидается выполнения прошлой, после чего запускается

INFO BASE: id=2 with priority=1 start executing. RUNNING. 2024-12-01 19:21:35.7644

1. Следующая задача с наибольшим приоритетом дожидается выполнения прошлой, после чего запускается

INFO EXTENDED: id=1 with priority=1 start executing. RUNNING. 2024-12-01 19:21:41.7708

## Модульное тестирование

Покрытие тестами не менее 80% (использовать инструменты оценки тестового покрытия)

Для анализа покрытия кода тестами использовалась утилита coverlet.collector версии 6.0.2.

Результат анализа покрытия изображен на рисунке 11.

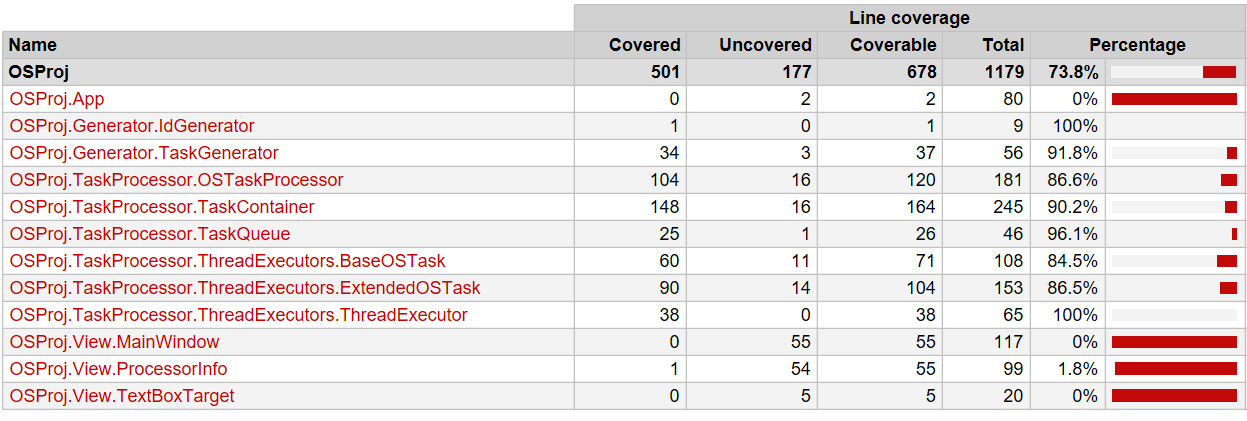


Рисунок 11 - Результат анализа покрытия тестами

На анализе видно, что модули, отвечающие за логику работы приложения покрыты тестами более 80%. Модули, отвечающие за пользовательский интерфейс и классы представления (namespace OSProj.View) не были протестированы.

Пример класса для модульного тестирования задач (классы реализующие интерфейс IOSTask):

namespace OSProjTest.Tests

{

public class IOSTaskTests

{

[Fact]

public void BaseOSTask\_PropTest()

{

int id = 1;

int priority = 3;

uint loopCount = 1;

var taskFunc = () => { id++; };

var baseTask = new BaseOSTask(id, priority, taskFunc, loopCount);

Assert.Equal(id, baseTask.Id);

Assert.Equal(priority, baseTask.Priority);

Assert.Equal(TaskType.Base, baseTask.TaskType);

baseTask.Run();

Thread.Sleep(200);

Assert.Equal(baseTask.Id + 1, id);

var action = () => { Thread.Sleep(1000); };

var baseTask1 = new BaseOSTask(id, priority, action, 2);

OSTaskProcessor.UpdateProgressBarDelegate updateProgress = (double persent) =>

{

if (persent > 0.001)

{

Assert.True(Math.Abs(persent - 0.5) < 0.001);

baseTask1.Cancel();

}

};

baseTask1.SetUpdateProgressBaseDelegate(updateProgress);

baseTask1.Run();

baseTask1.Wait();

baseTask1.Dispose();

Assert.False(baseTask1.IsRunning);

}

[Fact]

public void ExtendedOSTask\_PropTest()

{

int id = 1;

int priority = 3;

uint loopCount = 1;

var taskFunc = () => { id++; };

var extendedTask = new ExtendedOSTask(id, priority, taskFunc, loopCount);

Assert.Equal(id, extendedTask.Id);

Assert.Equal(priority, extendedTask.Priority);

Assert.Equal(TaskType.Extended, extendedTask.TaskType);

extendedTask.Run();

Thread.Sleep(200);

Assert.Equal(extendedTask.Id + 1, id);

var action1 = () => { Thread.Sleep(5000); };

var extendedTask1 = new ExtendedOSTask(id + 1, priority, action1);

extendedTask1.Run();

extendedTask1.Pause();

Assert.True(extendedTask1.Paused);

extendedTask1.Dispose();

var action2 = () => { Thread.Sleep(1000); };

var extendedTask2 = new ExtendedOSTask(id + 2, priority, action2, 2);

OSTaskProcessor.UpdateProgressBarDelegate updateProgress = (double persent) =>

{

if (persent > 0.001)

{

Assert.True(Math.Abs(persent - 0.5) < 0.001);

extendedTask2.Cancel();

}

};

extendedTask2.SetUpdateProgressBaseDelegate(updateProgress);

extendedTask2.Run();

extendedTask2.Wait();

extendedTask2.Dispose();

Assert.False(extendedTask2.IsRunning);

}

}

}

# Приложение. Тексты программы

|  |
| --- |
| using NLog;  using OSProj.Generator;  using OSProj.TaskProcessor.ThreadExecutors;  using System.Threading.Tasks;  namespace OSProj.TaskProcessor  {  public partial class OSTaskProcessor  {  public delegate void UpdateProgressBarDelegate(double percentage);  private UpdateProgressBarDelegate? \_updateProgressBar;  private TaskGenerator \_taskGenerator = new TaskGenerator();  private TaskContainer \_taskContainer = new TaskContainer();  private ThreadExecutor? \_processingThread;  private Logger \_logger;  private volatile IOSTask? \_activeTask = null;  public bool Running { get { return \_processingThread != null ? \_processingThread.IsRunning : false; } }  public delegate bool TaskWaitCheck(IOSTask task);  public event TaskWaitCheck? OnTaskWaitCheck;  public OSTaskProcessor(Logger logger)  {  OnTaskWaitCheck += OnTaskWaitCheckHandler;  \_logger = logger;  }  public void SetGenerator(TaskGenerator generator)  {  \_taskGenerator = generator;  }  public void Generate()  {  \_taskGenerator.Generate();  List<IOSTask> tasks = \_taskGenerator.PopGenerated();  \_taskContainer.AddTasks(tasks);  OnGenerateHandler();  \_taskContainer.UpdateCollectionView();  }  public void Start()  {  if (\_processingThread == null)  {  Action taskPlanner = () =>  {  Thread.CurrentThread.IsBackground = true;  \_activeTask = \_taskContainer.GetTopTaskAsActive();  if (\_activeTask != null)  {  Task.Delay(200);  if (\_updateProgressBar != null)  \_activeTask.SetUpdateProgressBarDelegate(\_updateProgressBar);  \_activeTask.Run();  \_taskContainer.UpdateCollectionView();  if (OnTaskWaitCheck != null && OnTaskWaitCheck.Invoke(\_activeTask))  {  \_activeTask.Wait();  if (\_activeTask.TaskType == TaskType.Base)  \_activeTask.Dispose();  else  {  var extTask = (ExtendedOSTask)\_activeTask;  if (extTask != null && !extTask.Paused)  \_activeTask.Dispose();  }  }  \_taskContainer.UpdateCollectionView();  }  else  {  Thread.Sleep(1000);  \_taskContainer.UpdateCollectionView();  }  };  \_processingThread = new ThreadExecutor(taskPlanner);  \_processingThread.Run();  \_logger.Info("Task processor was started.");  }  }  public void Stop()  {  if (\_processingThread != null)  {  \_processingThread.Cancel();  \_processingThread = null;  }  }  public void SetUpdateProcessorInfoDelegate(TaskContainer.UpdateQueuesInfo updateDelegate)  {  \_taskContainer.SetUpdateQueuesInfo(updateDelegate);  }  public void SetUpdateProgressBarDelegate(UpdateProgressBarDelegate updateProgressBar)  {  \_updateProgressBar = updateProgressBar;  }  public void PauseActiveTask()  {  if (\_activeTask != null && \_activeTask.TaskType == TaskType.Extended)  Wait(\_activeTask);  }  public void TerminateActiveTask()  {  if (\_activeTask != null)  Terminate(\_activeTask);  }  private bool OnGenerateHandler()  {  if (\_taskContainer.GetNextTaskPriority() > \_activeTask?.Priority)  {  Preempt(\_activeTask);  return false;  }  return true;  }  private bool OnTaskWaitCheckHandler(IOSTask task)  {  bool needWait = true;  int waitPriority = \_taskContainer.GetMaxWaitingPriority();  if (\_taskContainer.GetMaxSourceTasksPriority() > waitPriority)  {  \_taskContainer.FillMainContainerFromSource();  \_logger.Info("Ready queue is filling by source containers.");  needWait = OnGenerateHandler();  }  else if (waitPriority > -1)  {  Release();  }  return needWait;  }  private void Preempt(IOSTask task)  {  task.Cancel();  task.SetReadyFromRunning();  \_taskContainer.AddTaskToMain(task);  }  private void Wait(IOSTask task)  {  if (\_taskContainer.AddTaskToWaiting(task))  {  ((ExtendedOSTask)task)?.Pause();  }  }  public void Release()  {  \_taskContainer.FillMainContainerFromWaiting();  }  private void Terminate(IOSTask task)  {  if (\_taskContainer.AddTaskToSuspended(task))  task.Cancel();  }  public void Activate()  {  \_taskContainer.FillMainContainerFromSuspended();  if (\_activeTask != null && \_activeTask.Priority < \_taskContainer.GetNextTaskPriority())  Preempt(\_activeTask);  }  }  }  using OSProj.TaskProcessor.ThreadExecutors;  namespace OSProj.TaskProcessor  {  public class TaskContainer  {  public int MaxMainContainerSize { get; set; } = 8;  private IOSTask? \_activeTask;  private TaskQueue \_mainTasksCollection = new();  private TaskQueue \_priority0Tasks = new();  private TaskQueue \_priority1Tasks = new();  private TaskQueue \_priority2Tasks = new();  private TaskQueue \_priority3Tasks = new();  private TaskQueue \_waitingCollection = new();  private TaskQueue \_suspendedCollection = new();  public delegate void UpdateQueuesInfo(IOSTask? activeTask, TaskQueue mainTasks, TaskQueue waitingTasks, TaskQueue suspendedTasks);  private UpdateQueuesInfo? \_updateDelegate;  public TaskContainer()  {  }  public bool IsMainContainerFull() { return \_mainTasksCollection.Count >= MaxMainContainerSize; }  public void SetUpdateQueuesInfo(UpdateQueuesInfo updateDelegate)  {  \_updateDelegate = updateDelegate;  UpdateCollectionView();  }  public void UpdateCollectionView()  {  if (\_updateDelegate != null)  {  lock (\_mainTasksCollection) lock (\_waitingCollection) lock (\_suspendedCollection)  {  \_updateDelegate(\_activeTask, \_mainTasksCollection, \_waitingCollection, \_suspendedCollection);  }  }  }  public int? GetNextTaskPriority()  {  lock (\_mainTasksCollection)  {  return \_mainTasksCollection.Next?.Priority;  }  }  public IOSTask? GetTopTaskAsActive()  {  \_activeTask = PopMainTask();  return \_activeTask;  }  public IOSTask? PopMainTask()  {  lock (\_mainTasksCollection)  {  return \_mainTasksCollection.Pop();  }  }  public bool AddTaskToMain(IOSTask task)  {  bool result = \_mainTasksCollection.Count < MaxMainContainerSize;  if (result)  {  \_mainTasksCollection.Push(task);  UpdateCollectionView();  }  return result;  }  public bool AddTaskToWaiting(IOSTask task)  {  bool res = task.TaskType == Generator.TaskType.Extended && task.TaskStatus == Generator.OSTaskStatus.Running;  if (res)  {  lock (\_waitingCollection)  {  \_waitingCollection.Push(task);  ((IExtendedTasksStateSetter)task)?.SetWaitingState();  UpdateCollectionView();  }  }  return res;  }  public bool AddTaskToSuspended(IOSTask task)  {  bool res = task.TaskStatus == Generator.OSTaskStatus.Running;  if (res)  {  lock (\_suspendedCollection)  {  task.SetSuspendedState();  \_suspendedCollection.Push(task);  UpdateCollectionView();  }  }  return res;  }  public void AddTask(IOSTask task)  {  switch (task.Priority)  {  case 0:  \_priority0Tasks.Push(task);  break;  case 1:  \_priority1Tasks.Push(task);  break;  case 2:  \_priority2Tasks.Push(task);  break;  case 3:  \_priority3Tasks.Push(task);  break;  }  }  public void AddTasks(List<IOSTask> tasks)  {  foreach (IOSTask task in tasks)  AddTask(task);  FillMainContainerFromSource();  }  public void FillMainContainerFromSource()  {  bool isAllEmpty = false;  while (!IsMainContainerFull() && !isAllEmpty)  {  if (!\_priority3Tasks.Empty)  {  \_mainTasksCollection.Push(\_priority3Tasks.Pop());  }  else if (!\_priority2Tasks.Empty)  {  \_mainTasksCollection.Push(\_priority2Tasks.Pop());  }  else if (!\_priority1Tasks.Empty)  {  \_mainTasksCollection.Push(\_priority1Tasks.Pop());  }  else if (!\_priority0Tasks.Empty)  {  \_mainTasksCollection.Push(\_priority0Tasks.Pop());  }  else  {  isAllEmpty = true;  }  }  if (!isAllEmpty)  UpdateCollectionView();  }  public void FillMainContainerFromWaiting()  {  int emptyPlacesCount = MaxMainContainerSize - \_mainTasksCollection.Count;  if (emptyPlacesCount > 0)  {  emptyPlacesCount = \_waitingCollection.Count < emptyPlacesCount ? \_waitingCollection.Count : emptyPlacesCount;  for (int i = 0; i < emptyPlacesCount; i++)  {  IOSTask? task = \_waitingCollection.Pop();  if (task != null)  {  lock (\_mainTasksCollection)  {  \_mainTasksCollection.Push(task);  ((IExtendedTasksStateSetter)task).SetReadyFromWaiting();  UpdateCollectionView();  }  }  }  }  }  public void FillMainContainerFromSuspended()  {  int emptyPlacesCount = MaxMainContainerSize - \_mainTasksCollection.Count;  if (emptyPlacesCount > 0)  {  emptyPlacesCount = \_suspendedCollection.Count < emptyPlacesCount ? \_suspendedCollection.Count : emptyPlacesCount;  for (int i = 0; i < emptyPlacesCount; i++)  {  IOSTask? task = \_suspendedCollection.Pop();  if (task != null)  {  lock (\_mainTasksCollection)  {  \_mainTasksCollection.Push(task);  task.SetReadyFromSuspended();  UpdateCollectionView();  }  }  }  }  }  public int GetMaxWaitingPriority()  {  int res = -1;  if (\_waitingCollection.Count > 0)  res = \_waitingCollection.Get().Select(x => x.Priority).Max();  return res;  }  public int GetMaxSourceTasksPriority()  {  int res = -1;  if (!\_priority3Tasks.Empty)  res = 3;  else if (!\_priority2Tasks.Empty)  res = 2;  else if (!\_priority1Tasks.Empty)  res = 1;  else if (!\_priority0Tasks.Empty)  res = 0;  return res;  }  private IOSTask? PopWaitingTask()  {  return \_waitingCollection.Pop();  }  private IOSTask? PopSuspendedTask()  {  return \_suspendedCollection.Pop();  }  }  }  using OSProj.TaskProcessor.ThreadExecutors;  namespace OSProj.TaskProcessor  {  public class TaskQueue  {  private LinkedList<IOSTask> \_queue = new();  public bool IsSorted { get; }  public int Count => \_queue.Count;  public IOSTask? Next => \_queue.Count > 0 ? \_queue.First() : null;  public bool Empty => \_queue.Count == 0;  public TaskQueue(bool isSorted = true)  {  IsSorted = isSorted;  }  public IEnumerable<IOSTask> Get()  {  return \_queue;  }  public void Push(IOSTask? item)  {  if (item == null)  return;  if (!IsSorted || \_queue.Count == 0 || \_queue.Select(task => task.Priority).Min() >= item.Priority)  {  \_queue.AddLast(item);  }  else  {  var element = \_queue.First;  while (element != null)  {  if (CompareTasks(element.Value, item))  {  \_queue.AddBefore(element, item);  break;  }  element = element.Next;  }  }  }  public IOSTask? Pop()  {  IOSTask? res = \_queue.First?.Value;  if (res != null)  \_queue.RemoveFirst();  return res;  }  private bool CompareTasks(IOSTask task1, IOSTask task2)  {  return task1.Priority < task2.Priority || task1.Priority == task2.Priority && task1.CreationTime > task2.CreationTime;  }  }  }  using OSProj.Generator;  namespace OSProj.TaskProcessor.ThreadExecutors  {  public class BaseOSTask : ThreadExecutor, IOSTask  {  protected OSTaskProcessor.UpdateProgressBarDelegate? \_updateProgressBar;  public int Id { get; }  public int Priority { get; }  public virtual TaskType TaskType { get { return TaskType.Base; } }  public OSTaskStatus TaskStatus { get; set; } = OSTaskStatus.Ready;  public DateTime CreationTime { get; }  public BaseOSTask(int id, int priority, Action threadFunc, uint loopingCount = 0)  : base(threadFunc, loopingCount)  {  Id = id;  Priority = priority;  CreationTime = DateTime.UtcNow;  }  public override void Run()  {  if (!\_isRunning)  {  CancelTokenSource = new CancellationTokenSource();  Action action;  if (\_loopingCount > 0)  {  action = () =>  {  for (long i = 1; i <= \_loopingCount && !CancelTokenSource.Token.IsCancellationRequested; i++)  {  ThreadFunction();  if (\_updateProgressBar != null)  {  double persentage = (double)i / (\_loopingCount);  \_updateProgressBar.Invoke(persentage);  }  }  };  }  else  {  action = () =>  {  while (!CancelTokenSource.Token.IsCancellationRequested)  ThreadFunction();  };  }  if (\_updateProgressBar != null)  \_updateProgressBar.Invoke(0);  \_task = Task.Run(action, CancelTokenSource.Token);  \_isRunning = true;  }  SetRunningState();  }  public override void Cancel()  {  base.Cancel();  }  public override void Wait()  {  base.Wait();  }  public override void Dispose()  {  base.Dispose();  }  public virtual void SetSuspendedState()  {  TaskStatus = OSTaskStatus.Suspended;  View.ProcessorInfo.logger.Info($"BASE: id={Id} with priority={Priority} has been placed in the SUSPENDED queue. TERMINATE.");  }  public virtual void SetReadyFromSuspended()  {  TaskStatus = OSTaskStatus.Ready;  View.ProcessorInfo.logger.Info($"BASE: id={Id} with priority={Priority} has been placed in the READY queue from SUSPENDED. ACTIVATE.");  }  public virtual void SetReadyFromRunning()  {  TaskStatus = OSTaskStatus.Ready;  View.ProcessorInfo.logger.Info($"BASE: id={Id} with priority={Priority} has been placed in the READY queue from RUNNING. PREEMPT.");  }  public virtual void SetRunningState()  {  TaskStatus = OSTaskStatus.Running;  View.ProcessorInfo.logger.Info($"BASE: id={Id} with priority={Priority} start executing. RUNNING.");  }  public void SetUpdateProgressBarDelegate(OSTaskProcessor.UpdateProgressBarDelegate updateProgressBar)  {  \_updateProgressBar = updateProgressBar;  }  }  }  using OSProj.Generator;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  namespace OSProj.TaskProcessor.ThreadExecutors  {  public class ExtendedOSTask : BaseOSTask, IExtendedTasksStateSetter  {  private ManualResetEventSlim \_processorThreadEvent = new(false);  private ManualResetEventSlim \_taskEvent = new(false);  private bool \_paused = false;  public override TaskType TaskType { get { return TaskType.Extended; } }  public bool Paused { get { return \_paused; } }  public ExtendedOSTask(int id, int priority, Action threadFunc, uint loopingCount = 0)  : base(id, priority, threadFunc, loopingCount)  {  \_processorThreadEvent.Reset();  \_taskEvent.Set();  }  public override void Run()  {  if (!Paused)  {  if (!\_isRunning)  {  CancelTokenSource = new CancellationTokenSource();  Action action = () =>  {  \_processorThreadEvent.Reset();  };  if (\_loopingCount > 0)  {  action += () =>  {  for (long i = 1; i <= \_loopingCount && !CancelTokenSource.Token.IsCancellationRequested; i++)  {  \_taskEvent.Wait();  ThreadFunction();  if (\_updateProgressBar != null)  {  double persentage = (double)i / (\_loopingCount);  \_updateProgressBar.Invoke(persentage);  }  }  };  }  else  {  action += () =>  {  while (!CancelTokenSource.Token.IsCancellationRequested)  {  \_taskEvent.Wait();  ThreadFunction();  }  };  }  action += \_processorThreadEvent.Set;  if (\_updateProgressBar != null)  \_updateProgressBar.Invoke(0);  \_task = Task.Run(action, CancelTokenSource.Token);  \_isRunning = true;  }  }  else  Resume();  SetRunningState();  }  public override void Cancel()  {  base.Cancel();  }  public override void Wait()  {  \_processorThreadEvent.Wait();  }  public override void Dispose()  {  base.Dispose();  }  public void Pause()  {  \_processorThreadEvent.Set();  \_taskEvent.Reset();  \_paused = true;  }  public void Resume()  {  \_taskEvent.Set();  \_processorThreadEvent.Reset();  \_paused = false;  }  public void SetWaitingState()  {  TaskStatus = OSTaskStatus.Waiting;  View.ProcessorInfo.logger.Info($"EXTENDED: id={Id} with priority={Priority} has been placed in the WAITING queue. (WAIT).");  }  public override void SetSuspendedState()  {  TaskStatus = OSTaskStatus.Suspended;  View.ProcessorInfo.logger.Info($"EXTENDED: id={Id} with priority={Priority} has been placed in the SUSPENDED queue. TERMINATE");  }  public override void SetReadyFromSuspended()  {  TaskStatus = OSTaskStatus.Ready;  View.ProcessorInfo.logger.Info($"EXTENDED: id={Id} with priority={Priority} has been placed in the READY queue from SUSPENDED. ACTIVATE");  }  public void SetReadyFromWaiting()  {  TaskStatus = OSTaskStatus.Ready;  View.ProcessorInfo.logger.Info($"EXTENDED: id={Id} with priority={Priority} has been placed in the READY queue from WAITING. RELEASE");  }  public override void SetReadyFromRunning()  {  TaskStatus = OSTaskStatus.Ready;  View.ProcessorInfo.logger.Info($"EXTENDED: id={Id} with priority={Priority} has been placed in the READY queue from RUNNING. PREEMPT.");  }  public override void SetRunningState()  {  TaskStatus = OSTaskStatus.Running;  View.ProcessorInfo.logger.Info($"EXTENDED: id={Id} with priority={Priority} start executing. RUNNING.");  }  }  } |