Реализация алгоритма планирования в порядке очереди «First-Come, First-Served» (FCFS) без вытеснения

1. Общая постановка задачи

Необходимо согласно из приведенного ниже списка заданий, провести исследование, создать новый компонент для заданной архитектуры (x86, AVR, ARM, MIPS, RISC-V) и платформы Eco OS, Windows OS, Linux OS. Исследовательская часть заключается, в изучении аппаратной составляющей или в поиске описания алгоритма, необходимых для реализации согласно выбранной задачи

2. Алгоритм First‑Come, First‑Served

Алгоритм FCFS (First‑Come, First‑Served) относится к простейшим дисциплинам планирования.

Основные характеристики:

- Порядок обслуживания: задачи выполняются строго в том порядке, в котором они «пришли» в систему. Для каждой задачи задаётся время прихода t_{in}. Планировщик всегда выбирает задачу с минимальным t_{in} среди еще не выполненных.
- Без вытеснения (non‑preemptive): если задача начала выполняться, она работает до завершения, не может быть прервана другой задачей по таймеру или приоритету.

Преимущества:

- простота реализации;
- предсказуемый порядок выполнения;
- подходит для систем, где важен порядок поступления запросов.

Недостатки:

- не учитывает длительность задач: длинная задача в начале очереди может сильно задерживать короткие;
- возможен эффект конвоя (convoy effect): множество коротких задач ждут завершения одной долгой;
- нет приоритизации.

3. Реализация

3.1. Структура компонента

Компонент CEcoTaskScheduler1Lab реализует алгоритм планирования задач FCFS (First-Come, First-Served) без вытеснения для платформы Eco OS. Компонент состоит из двух основных частей:

Планировщик задач (CEcoTaskScheduler1Lab) - основной компонент, управляющий очередью задач

Задача (CEcoTask1Lab) - структура, представляющая отдельную задачу в системе

Структура планировщика

```
typedef struct CEcoTaskScheduler1Lab_C761620F {
    /* Таблица функций интерфейса IEcoTaskScheduler1 */
    IEcoTaskScheduler1VTbl* m_pVTblIScheduler;

    /* Счетчик ссылок */
    uint32_t m_cRef;

    /* Интерфейсная Шина */
    IEcoInterfaceBus1* m_pIBus;

    /* Интерфейс для работы с памятью */
    IEcoMemoryAllocator1* m_pIMem;

    /* Системный интерфейс */
    IEcoSystem1* m_pISys;

    /* Данные экземпляра */
    IEcoTimer1Ptr_t m_pIArmTimer;
    CEcoTask1Lab_C761620F* m_pTaskList;
} CEcoTaskScheduler1Lab_C761620F;
```

Структура задачи

```
typedef struct CEcoTask1Lab_C761620F {
    /* Таблица функций интерфейса IEcoTask1 */
    IEcoTask1VTbl* m_pVTblITask;

    /* Счетчик ссылок */
    uint32_t m_cRef;

    /* Указатель на стек задачи */
    volatile byte_t* m_sp;

    /* Указатель на функцию задачи */
    void (*pfunc) (void);

    /* Параметр задачи - время входа для алгоритма FCFS */
    uint16_t timeIn;
} CEcoTask1Lab_C761620F;
```

3.2. Интерфейс планировщика

Интерфейс IEcoTaskScheduler1 содержит следующие методы:

```
typedef struct IEcoTaskScheduler1VTbl {
    /* IEcoUnknown */
    int16_t (ECOCALLMETHOD *QueryInterface)(/* in */ IEcoTaskScheduler1Ptr_t me,
                                            /* in */ const UGUID* riid,
                                            /* out */ voidptr_t* ppv);
    uint32_t (ECOCALLMETHOD *AddRef)(/* in */ IEcoTaskScheduler1Ptr_t me);
    uint32_t (ECOCALLMETHOD *Release)(/* in */ IEcoTaskScheduler1Ptr_t me);

    /* IEcoTaskScheduler1 */
    int16_t (ECOCALLMETHOD *Init)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                  /*in*/ IEcoInterfaceBus1Ptr_t pIBus);
    int16_t (ECOCALLMETHOD *InitWith)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                      /*in*/ IEcoInterfaceBus1Ptr_t pIBus,
                                      /*in*/ voidptr_t heapStartAddress,
                                      /*in*/ uint32_t size);
    int16_t (ECOCALLMETHOD *NewTask)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                     /*in*/ voidptr_t address,
                                     /*in*/ voidptr_t data,
                                     /*in*/ uint32_t stackSize,
                                     /* out */ IEcoTask1** ppITask);
    int16_t (ECOCALLMETHOD *DeleteTask)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                        /*in*/ uint16_t taskId);
    int16_t (ECOCALLMETHOD *SuspendTask)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                         /*in*/ uint16_t taskId);
    int16_t (ECOCALLMETHOD *ResumeTask)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                        /*in*/ uint16_t taskId);
    int16_t (ECOCALLMETHOD *RegisterInterrupt)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                               /*in*/ uint16_t number,
                                               /*in*/ voidptr_t handlerAddress,
                                               /*in*/ int32_t flag);
    int16_t (ECOCALLMETHOD *UnRegisterInterrupt)(/*in*/ IEcoTaskScheduler1Ptr_t me,
                                                 /*in*/ uint16_t number);
    int16_t (ECOCALLMETHOD *Start)(/*in*/ IEcoTaskScheduler1Ptr_t me);
    int16_t (ECOCALLMETHOD *Stop)(/*in*/ IEcoTaskScheduler1Ptr_t me);
} IEcoTaskScheduler1VTbl;
```

Интерфейс задачи

Интерфейс IEcoTask1 расширен методами для работы с временем входа:

```
typedef struct IEcoTask1VTbl {
    /* IEcoUnknown */
    int16_t (ECOCALLMETHOD *QueryInterface)(/* in */ IEcoTask1Ptr_t me,
                                            /* in */ const UGUID* riid,
                                            /* out */ voidptr_t* ppv);
    uint32_t (ECOCALLMETHOD *AddRef)(/* in */ IEcoTask1Ptr_t me);
    uint32_t (ECOCALLMETHOD *Release)(/* in */ IEcoTask1Ptr_t me);

    /* IEcoTask1 */
    int16_t (ECOCALLMETHOD *Delay)(/*in*/ IEcoTask1Ptr_t me,
                                   /*in*/ double_t milliseconds);
    int16_t (ECOCALLMETHOD *Yield)(/*in*/ IEcoTask1Ptr_t me);

    /* Методы для работы с временем входа (FCFS) */
    int16_t (ECOCALLMETHOD *SetTimeIn)(/*in*/ IEcoTask1Ptr_t me,
                                       uint16_t time);
    int16_t (ECOCALLMETHOD *GetTimeIn)(/*in*/ IEcoTask1Ptr_t me);
} IEcoTask1VTbl;
```

3.3. Ключевые функции реализации

Функция создания задачи (NewTask)

```
int16_t ECOCALLMETHOD CEcoTaskScheduler1Lab_C761620F_NewTask(
    /*in*/ IEcoTaskScheduler1Ptr_t me,
    /*in*/ voidptr_t address,
    /*in*/ voidptr_t data,
    /*in*/ uint32_t stackSize,
    /* out */ IEcoTask1** ppITask) {

    int32_t indx = 0;

    /* Проверка указателей */
    if (me == 0) {
        return -1;
    }

    /* Поиск свободного слота в массиве задач */
    for (indx = 0; indx < MAX_STATIC_TASK_COUNT; indx++) {
        if (g_xCEcoTask1List_C761620F[indx].pfunc == 0) {
            /* Инициализация задачи */
            g_xCEcoTask1List_C761620F[indx].pfunc = address;
            g_xCEcoTask1List_C761620F[indx].m_cRef = 1;
            g_xCEcoTask1List_C761620F[indx].m_sp = (byte_t*)&g_xCEcoStackTask1List_C761620F[indx*4096];
            g_xCEcoTask1List_C761620F[indx].timeIn = 0; /* Инициализация времени входа */

            /* Настройка стека задачи */
            /* ... инициализация стека ... */

            g_xCEcoTask1List_C761620F[indx].m_pVTblITask = &g_x81A466F4C27540B1B33D0661E5470F1BVTbl_C761620F;
            *ppITask = (IEcoTask1*)&g_xCEcoTask1List_C761620F[indx];
            return 0;
        }
    }
    return -1; /* Нет свободных слотов */
}
```

Функция установки времени входа (SetTimeIn)

```
uint16_t ECOCALLMETHOD CEcoTask1Lab_C761620F_SetTimeIn(
    IEcoTask1Ptr_t me,
    uint16_t time) {

    CEcoTask1Lab_C761620F* pCMe = (CEcoTask1Lab_C761620F*)me;

    if (me == 0) {
        return -1;
    }

    pCMe->timeIn = time;
    return 0;
}
```

Функция запуска планировщика (Start) - реализация FCFS

```
int16_t ECOCALLMETHOD CEcoTaskScheduler1Lab_C761620F_Start(
    /*in*/ IEcoTaskScheduler1Ptr_t me) {

    CEcoTaskScheduler1Lab_C761620F* pCMe = (CEcoTaskScheduler1Lab_C761620F*)me;
    int16_t i;
    int16_t earliestIn;

    /* Проверка указателей */
    if (me == 0) {
        return -1;
    }

    while (1) {
        earliestIn = -1;

        /* Поиск задачи с наименьшим временем входа (FCFS) */
        for (i = 0; i < MAX_STATIC_TASK_COUNT; i++) {
            if (pCMe->m_pTaskList[i].pfunc != 0 &&
                (earliestIn == -1 ||
                 pCMe->m_pTaskList[i].timeIn < pCMe->m_pTaskList[earliestIn].timeIn)) {
                earliestIn = i;
            }
        }

        /* Проверка наличия задач для выполнения */
        if (earliestIn == -1) {
            /* Нет задач для выполнения - выход из цикла планирования */
            break;
        }

        /* Выполнение выбранной задачи (FCFS без вытеснения) */
        pCMe->m_pTaskList[earliestIn].pfunc();

        /* Удаление выполненной задачи из очереди */
        pCMe->m_pTaskList[earliestIn].pfunc = 0;
    }

    return 0;
}
```

3.4. Пример использования

```
/* Создание планировщика */
IEcoTaskScheduler1* pIScheduler = 0;
pIBus->pVTbl->QueryComponent(pIBus, &CID_EcoTaskScheduler1Lab, 0,
                             &IID_IEcoTaskScheduler1, (void**) &pIScheduler);

/* Инициализация планировщика */
pIScheduler->pVTbl->InitWith(pIScheduler, pIBus, &__heap_start__+0x090000, 0x080000);

/* Создание задач */
IEcoTask1* pITask1 = 0;
IEcoTask1* pITask2 = 0;
IEcoTask1* pITask3 = 0;
IEcoTask1* pITask4 = 0;

pIScheduler->pVTbl->NewTask(pIScheduler, Task1, 0, 0x100, &pITask1);
pIScheduler->pVTbl->NewTask(pIScheduler, Task2, 0, 0x100, &pITask2);
pIScheduler->pVTbl->NewTask(pIScheduler, Task3, 0, 0x100, &pITask3);
pIScheduler->pVTbl->NewTask(pIScheduler, Task4, 0, 0x100, &pITask4);

/* Установка времени входа для алгоритма FCFS */
pITask1->pVTbl->SetTimeIn(pITask1, 3);  /* Задача #1 пришла первой */
pITask2->pVTbl->SetTimeIn(pITask2, 10); /* Задача #2 пришла второй */
pITask3->pVTbl->SetTimeIn(pITask3, 7);  /* Задача #3 пришла третьей */
pITask4->pVTbl->SetTimeIn(pITask4, 20); /* Задача #4 пришла четвертой */

/* Запуск планировщика */
/* Задачи будут выполнены в порядке: #1 (timeIn=3), #3 (timeIn=7),
   #2 (timeIn=10), #4 (timeIn=20) */
pIScheduler->pVTbl->Start(pIScheduler);
```

4. Пример работы

Проводилось несколько серий запусков с различными значениями timeIn для четырех задач. Пример двух конфигураций:

Конфигурация A:

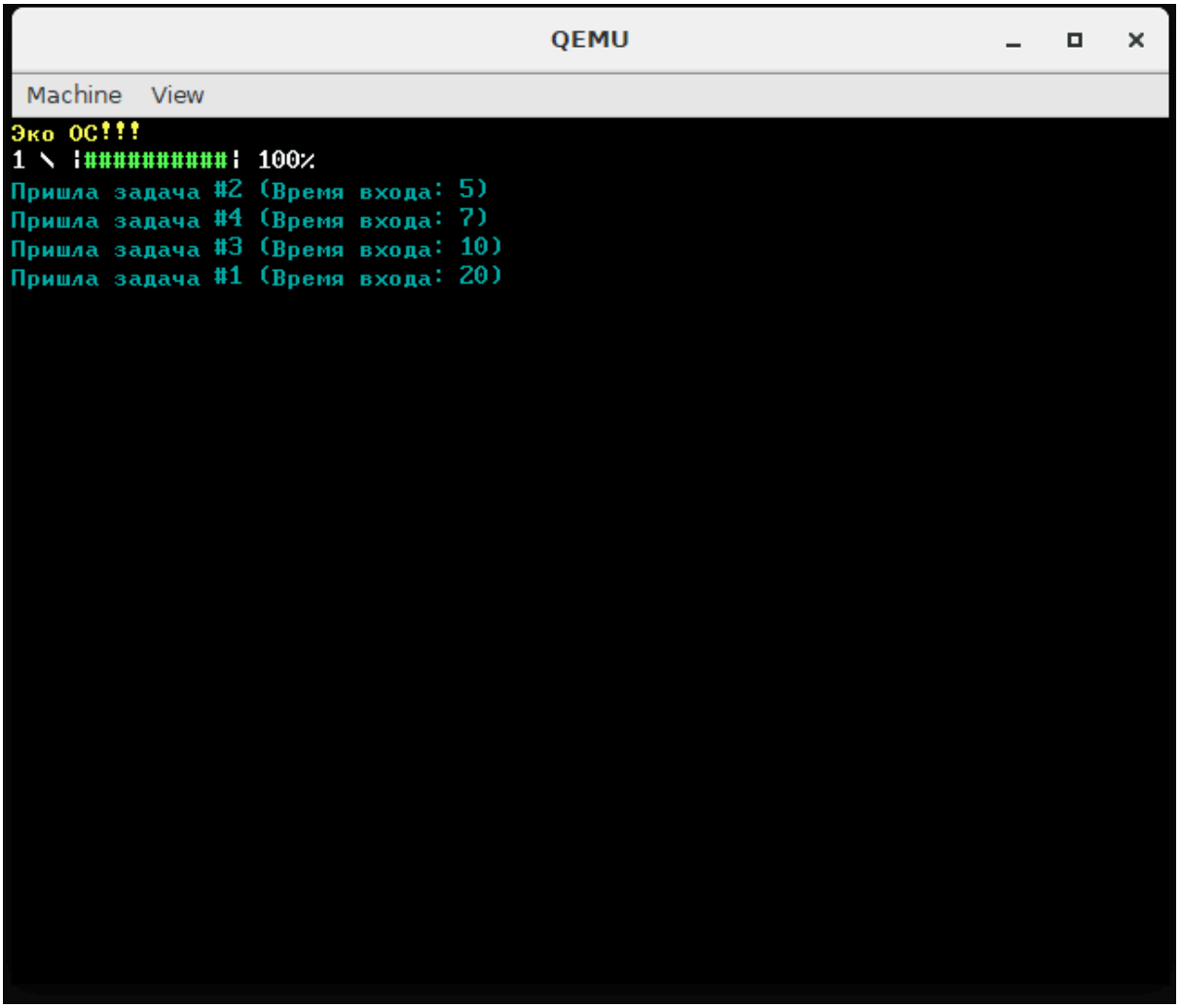
timeIn(T1) = 20

timeIn(T2) = 5

timeIn(T3) = 10

timeIn(T4) = 7

Ожидаемый порядок FCFS: T2 → T4 → T3 → T1.



Конфигурация B:

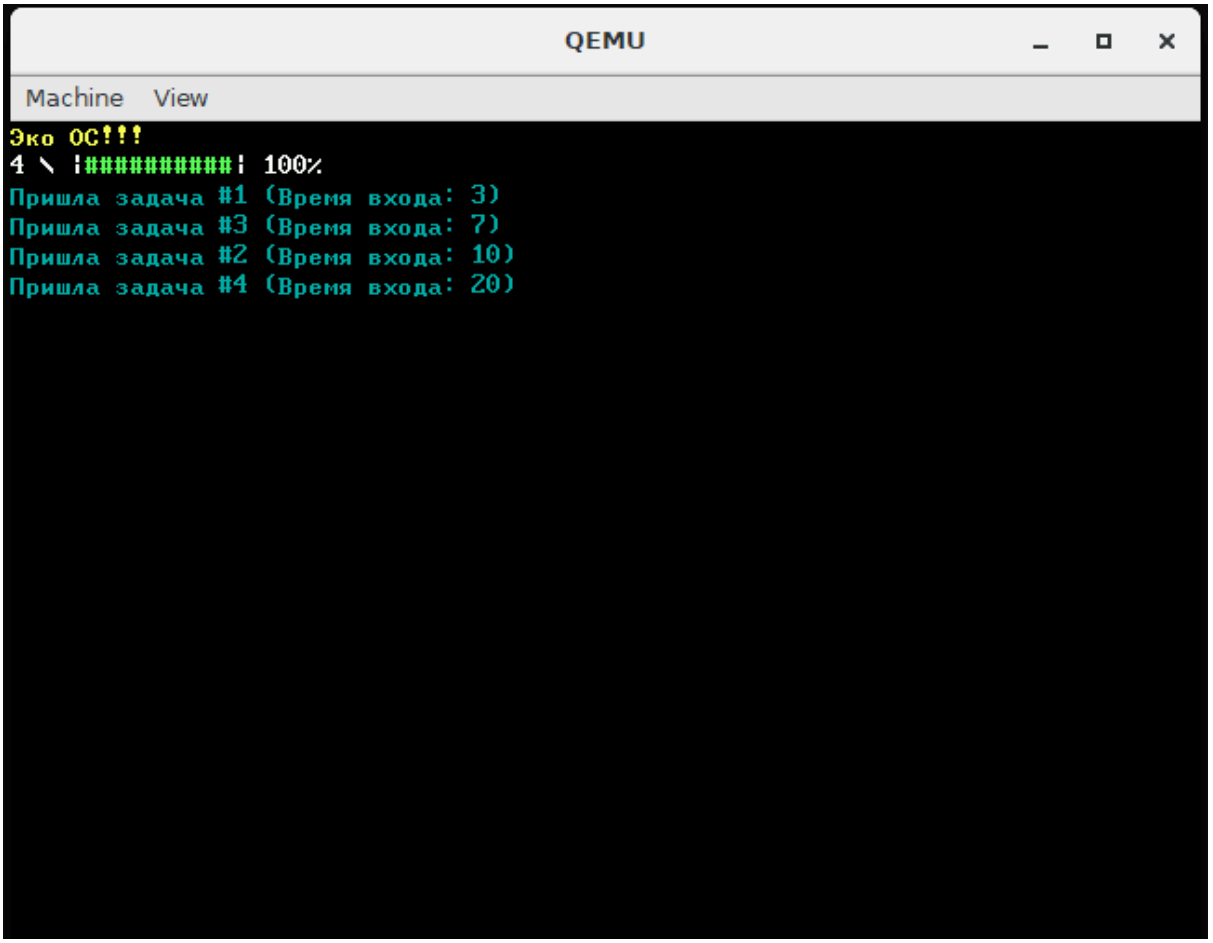
timeIn(T1) = 3

timeIn(T2) = 10

timeIn(T3) = 7

timeIn(T4) = 20

Ожидаемый порядок FCFS: T1 → T3 → T2 → T4.



Во всех случаях система собиралась make-ом и запускалась в QEMU (run.bat).

Сравнение показало:

- при изменении timeIn изменяется порядок появления строк журнала, что подтверждает корректную работу FCFS;
- длительность заполнения прогресс-бара для каждой задачи заметно меняется при увеличении timeIn (задачи с большим временем прихода выполняются медленнее, что удобно для визуальной демонстрации, хотя теоретически FCFS этого не требует).

5. Выводы

В ходе работы:

- Изучен и реализован алгоритм планирования First-Come, First-Served (FCFS) без вытеснения в виде отдельного компонента Eco.TaskScheduler1Lab для архитектуры ARM AArch64 и платформы Eco OS.
- Реализация использует только интерфейсы и компоненты Eco Framework, без стандартных библиотек, что соответствует требованиям bare-metal варианта.
- На примере ОС MySimpleEcoOS показана работа планировщика: при изменении времени входа задач меняется порядок их запуска и связанная с ним визуализация (журнал и скорость прогресса).
- Полученные результаты подтверждают корректность реализации алгоритма FCFS без вытеснения и демонстрируют возможность интеграции нового компонентного планировщика в существующую инфраструктуру Eco OS.