



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА
Институт искусственного интеллекта
Кафедра проблем управления

Лабораторная работа №3
по дисциплине «Операционные системы реального времени»

Тема работы: «Исследование межпроцессного взаимодействия в операционной системе реального времени»

Выполнил студент группы

КРБО-02-21

Набиев Р. Р

Преподаватель:

Смирнов М.Ю.

Работа представлена к защите:

«___» _____ 2023 г.

Москва, 2023 г.

Исследование межпроцессного взаимодействия в операционной системе реального времени

Цель работы

Изучить основные способы передачи информации и посмотреть на работу этих способов в условиях реального времени на основе библиотеки RTKLib.

Задание

1. Изучить программу, имитирующую работу разных способов передачи информации, её элементов.
2. Поменять приоритет TaskClassB на 10, скомпилировать и запустить программу.
3. Поменять приоритет TaskClassB обратно на 11, скомпилировать и запустить программу.
4. Поменять объём буфера (FIFO) на 12, посмотреть на изменения в работе.

Ход работы

1. Изучение программы, имитирующей работу разных способов передачи информации

Для выполнения данной лабораторной работы в среде Automation Studio необходимо было воспользоваться готовым программным проектом «lab3».

В состав данного проекта входит библиотека RTKLib. Она представляет собой специализированную библиотеку компании B&R, позволяющую работать с внутренним устройством реального времени. Также с ее помощью имеется возможность создать задачу с приоритетом выше, чем у базовых программ B&R.

Программа, используемая в лабораторной работе, представляет из себя 2 способа передачи данных:

- через FIFO (cnt_b);
- через численный счётчик (cnt_c).

FIFO передаёт данные через свою внутреннюю очередь, размер которой задаётся в строке (маркировка *). Численный счётчик же работает в тот момент, когда он получает приоритет – возможность работать.

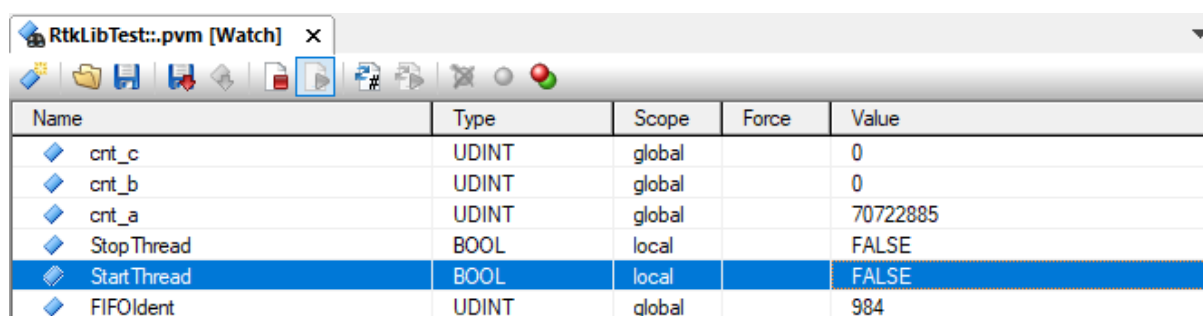
Программа RtkLibTest позволяет передавать данные между процессами, устанавливать приоритетность, а также запускать и останавливать передачу данных. Основными функциями являются incr_a, отвечающая за отправку данных, incr_b, реализующая обработку данных через FIFO и численный счетчик, и CYCLIC, в которой содержится алгоритм начала и окончания работы программы, а также такие параметры задач как приоритет и объем буфера. Исходный код представлен в приложении А.

Для диагностики работы программы использовались инструменты Watch и Trace. Инструмент Watch позволяет просматривать используемые переменных и изменять их. Инструмент Trace позволяет оценить по графикам динамику изменений значений переменных в ходе работы программы.

Для запуска программы необходимо добавить входящие в её состав переменные в интерфейс Watch и установить значение TRUE в переменную StartThread, отвечающую за начало работы исполняемых задач.

2. Изменение приоритета TaskClassB на 10

Значения переменных интерфейса Watch при изменении приоритета задачи осуществления передачи данных TaskClassB на 10 представлены на рисунке 1.



Name	Type	Scope	Force	Value
cnt_c	UDINT	global		0
cnt_b	UDINT	global		0
cnt_a	UDINT	global		70722885
StopThread	BOOL	local		FALSE
StartThread	BOOL	local		FALSE
FIFIdent	UDINT	global		984

Рисунок 1 - Окно Watch при приоритете TaskClassB равном 10

С помощью интерфейса Trace можно отследить скорость передачи данных с помощью FIFO или численного счётчика, однако по рисунку 2 видно, что передача данных осуществляться не будет, так как исполняемая задача имеет слишком низкий приоритет выполнения.

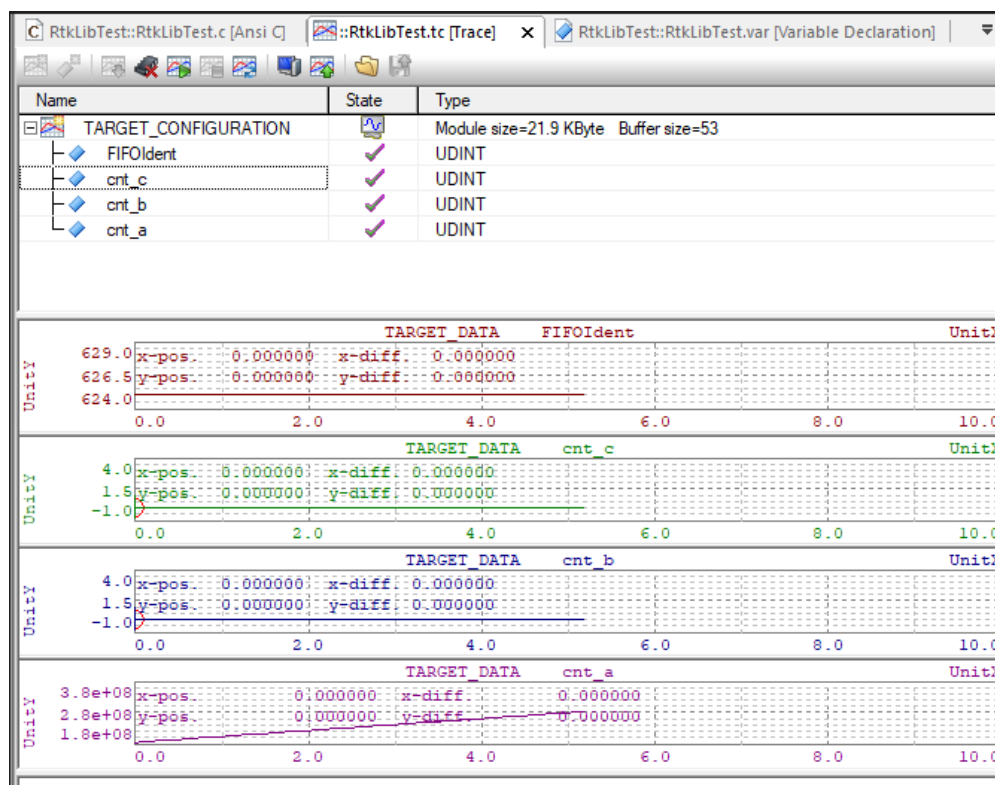


Рисунок 2 - Показания интерфейса Trace при приоритете задачи равном 10

3. Изменение приоритета TaskClassB обратно на 11

После изменения приоритета задачи с 10 на 11 становится возможна передача данных обоими способами (FIFO для cnt_b и численный счетчик для cnt_c). При этом график передачи данных без изменения размера очереди FIFO будет иметь линейную зависимость от времени, а скорость заметно превосходить численный счётчик, что видно из рисунка 3.

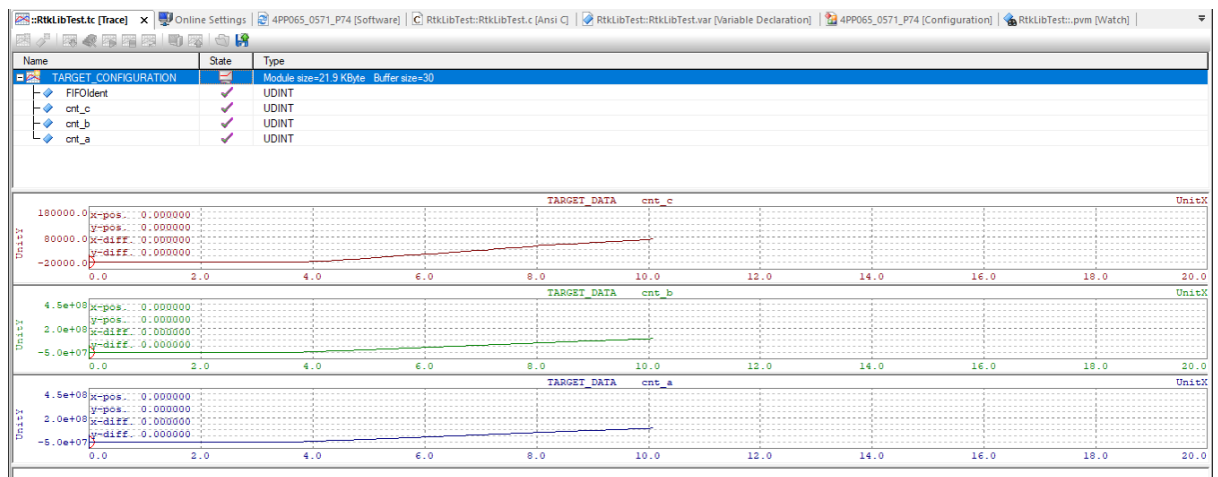


Рисунок 3 - Показания интерфейса Trace при приоритете задачи равном 11

4. Изменение объема буфера (FIFO) на 12

Изменение объема буфера в коде с 1024 на 12 представлено на рисунке 4.

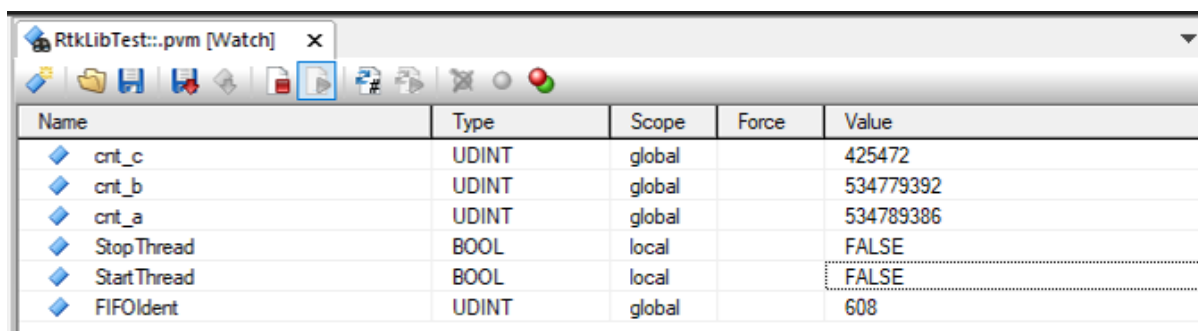
```

- }
- if(StartThread)
- {
-     cnt_a = 0;
-     cnt_b = 0;
-     cnt_c = 0;
-     StartThread = 0;
-     RtkCreatePlainFifo("ABFIFO", 1, 12, &FIFOIdent);
-     statusStartThread = RtkCreateTask(
-         "TestTaskA", // LPSTRING
-         10, // WORD
-         8192, // ULONG
-         8192, // ULONG
-         RTK_TASK_APPLICATION, //
-         (void*) &incr_a, //
-         (ULONG) &a, // ULONG
-         &TaskIdentA // LPULONG
-     );
-     lpulTaskIdent

```

Рисунок 4 - Изменение объема буфера в коде с 1024 на 12

На рисунке 5 представлены значения переменных при приоритете TaskClassB равном 11 и объеме буфера равном 12.



Name	Type	Scope	Force	Value
cnt_c	UDINT	global		425472
cnt_b	UDINT	global		534779392
cnt_a	UDINT	global		534789386
StopThread	BOOL	local		FALSE
StartThread	BOOL	local		FALSE
FIFOLdent	UDINT	global		608

Рисунок 5 - Показания в интерфейсе Watch при объеме буфера равном 12

Из графиков нарастания счетчиков, представленных на рисунке 6, видно, что эффективность передачи данных снизилась – возрастание графика показаний числового счетчика происходит нелинейно, а с уменьшением размеров внутренней очереди FIFO уменьшилась и скорость передачи данных.

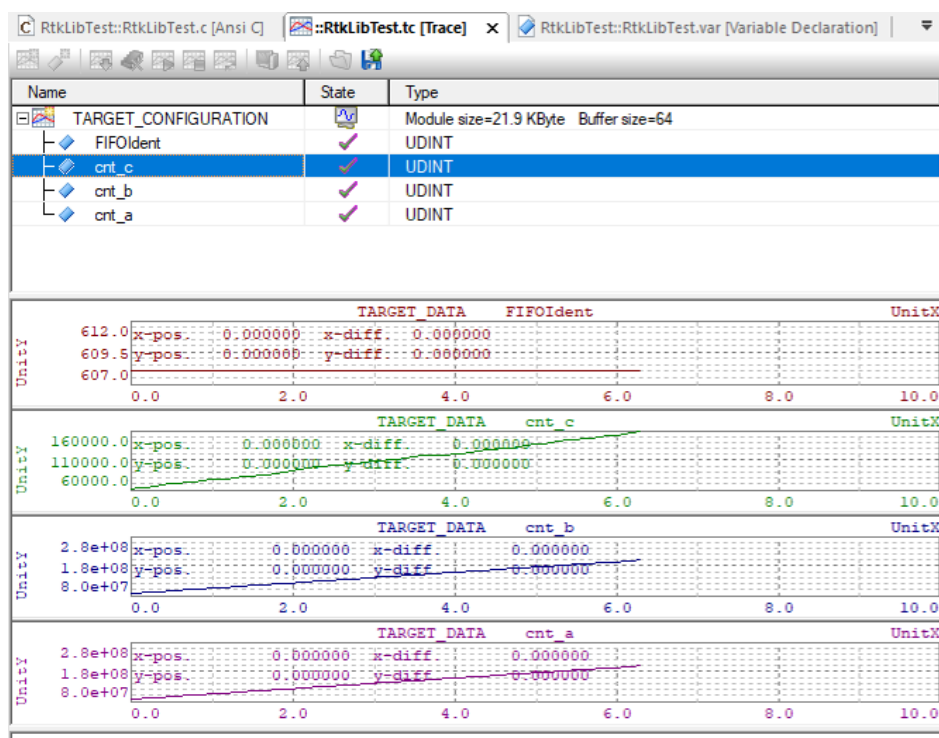


Рисунок 6 - Показания интерфейса Trace при объеме буфера равном 12

Выводы

На основе библиотеки RTKLib были изучены основные способы передачи информации в условиях реального времени. На практике была изучена работа этих способов в условиях реального времени: изменения работы программы при изменении приоритета задачи, изменении объема буфера; была исследована эффективность способа FIFO в зависимости от размеров его внутренней очереди.

Приложение А. Листинг исходного кода программы

```
#include <bur/plctypes.h>

#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif

#include <RtkUser.h>

RTK_ERROR statusStopThread;
RTK_ERROR statusStartThread;

void incr_a(USINT* par)
{
    while(1)
    {
        cnt_a++;
        if((cnt_a&0xFF)==128){
            (*par)=1;
            RtkWritePlainFifo (FIFOIdent, par);
        }else
            (*par)=0;
    }
}

USINT fifo_rd;

void incr_b(USINT* par)
{
    while(1)
    {
        if ((*par)==1)
            cnt_c=cnt_c+256;
        while ( (RtkReadPlainFifo(FIFOIdent,&fifo_rd)==0)) {
            cnt_b=cnt_b+fifo_rd*256;
        }
        RtkSleepTaskUsec(1);
    }
}

void _INIT NewProgramInit(void)
{
}

void _CYCLIC NewProgramCyclic(void)
{
    if(StopThread)
    {
        StopThread = 0;
        statusStopThread = RtkDeleteTask (TaskIdentA);
        statusStopThread = RtkDeleteTask (TaskIdentB);
        RtkDeletePlainFifo ( FIFOIdent);
        cnt_a = 0;
    }
}
```

```

        cnt_b = 0;
        cnt_c = 0;
    }
    if(StartThread)
    {
        cnt_a = 0;
        cnt_b = 0;
        cnt_c = 0;
        StartThread = 0;
        RtkCreatePlainFifo ( "ABFIFO", 1, 1024, &FIFOIdent);
        statusStartThread = RtkCreateTask (

            "TestTaskA",          // LPSTRING lpszTaskName,
                                   10,
            // WORD wTaskPriority,
                                   8192,
            // ULONG ulTaskSupervisorStackSize,
                                   8192,
            // ULONG ulTaskUserStackSize,

            RTK_TASK_APPLICATION, // RTK_TASKFLAG TaskFlags,
                                   (void*)
            incr_a,               // LPRTK_CREATE_TASK_FKT lpTaskFunction,
                                   (ULONG)
            &a,                   // ULONG ulTaskFunctionParameter,

            &TaskIdentA           // LPULONG lpulTaskIdent
                                   );
        statusStartThread = RtkCreateTask (

            "TestTaskB",          // LPSTRING lpszTaskName,
                                   10,
            // WORD wTaskPriority,
                                   8192,
            // ULONG ulTaskSupervisorStackSize,
                                   8192,
            // ULONG ulTaskUserStackSize,

            RTK_TASK_APPLICATION, // RTK_TASKFLAG TaskFlags,
                                   (void*)
            incr_b,               // LPRTK_CREATE_TASK_FKT lpTaskFunction,
                                   (ULONG)
            &a,                   // ULONG ulTaskFunctionParameter

            &TaskIdentB           // LPULONG lpulTaskIdent
                                   );
    }
}

```