

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА  
(САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики   
Кафедра Программных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
  
к лабораторной работе №4 по дисциплине «Технологии промышленного программирования»

Обучающийся группы 6232-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Портнов К.А.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баландин А.В.

Самара 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc147860536)

[2 Результаты работы 5](#_Toc147860537)

[ПРИЛОЖЕНИЕ А Листинг модуля M1 7](#_Toc147860538)

[ПРИЛОЖЕНИЕ B Листинг модуля M2 13](#_Toc147860539)

[ПРИЛОЖЕНИЕ C Листинг модуля M3 18](#_Toc147860540)

1. Постановка задачи

Разработать приложение реального времени (ПРВ), осуществляющее мониторинг состояния абстрактного физического объекта Op, p – изменяющийся во времени параметр объекта. Мониторинг объекта Op осуществляется на относительном интервале времени t∈[0,Т]. За 0 принимается момент начала штатной работы ПРВ после её загрузки в вычислительную среду. В процессе мониторинга ПРВ формирует на вешнем носителе файл тренда параметра p. Непосредственно в момент времени Т программная система должна завершить свою работу.

Изменение параметра p во времени моделируется функцией p=F(t), где t∈[0,Т] - момент времени получения текущего значения параметра p, выраженный в секундах.

Объект Op в программной системе моделируется процессом Р1(М1). Программный модуль М1 реализует вычисление функции p=F(t) и размещение полученного текущего значения параметра p в именованную память, предварительно созданную при загрузке ПРВ (порядок создания именованной памяти определяется в варианте задания).

ПРВ, осуществляющее мониторинг, реализуется в программной системе в виде процесса Р2(М2), запускаемого на базе модуля М2 (порядок запуска процессов Р1 и Р2 определяется в варианте задания):

Процесс Р1, начиная с t=0, периодически с заданной частотой обновляет текущее значение параметра p в именованной памяти.

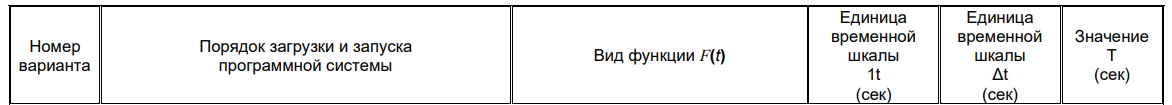
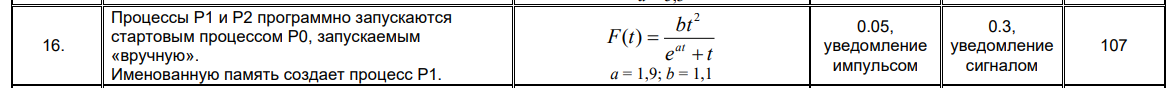
Процесс Р2, начиная с момента времени t=0, периодически с заданным периодом Δt считывает из именованной памяти текущее значение параметра p и формирует датированное значение в виде пары - . Результаты периодического считывания значений параметра p и соответствующей метки времени t используются процессом Р2 для занесения в текстовый файл (тренд параметра p) символьной строки, в которой символьное представление значения параметра p и соответствующего момента времени t разделяются знаком табуляции \t формата, а вся строка завершается управляющим символом \n:

"<p>\t<t>\n "

Процессы Р1 и Р2 должны быть синхронизированы по моменту времени t=0. (процесс Р2 должен получить первое значение параметра p в момент t=0). Метод синхронизации выбрать самостоятельно.

При наступлении момента t=Т работа программной системы должна немедленно завершиться (все процессы терминируются).

Результаты работы ПРВ представить в виде графика тренда параметра p(t), например, загрузив содержимое полученного файла с трендом в MS EXCEL.



1. Результаты работы

Результаты работы представлены в виде вывода на консоль сообщений во время выполнения программы, а также содержимого выходного файла, в который параллельно производимым расчётам записывались снимаемые метрики и их отображение в виде excel графика.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 1 – Результат выполнения программы в консоли

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

Рисунок 2 – Результат записанные в файл

Изображение выглядит как текст, снимок экрана, линия, Параллельный

Автоматически созданное описание

Рисунок 3 – Результат в виде диаграммы excel

ПРИЛОЖЕНИЕ А  
Листинг модуля M1

**#include** <cstdlib>

**#include** <iostream>

**#include** <sys/neutrino.h>

**#include** <pthread.h>

**#include** <string.h>

**#include** <stdlib.h>

**#include** <fcntl.h>

**#include** <sys/mman.h>

**#include** <unistd.h>

**#define** GetCurrentDir getcwd

**using** std::cout;

**using** std::endl;

**#include** <pthread.h>

// Имя именованной памяти

**#define** NAMED\_MEMORY "/16/namedMemory"

// ---- Структуры для именованной памяти ---- //

// Структура данных с информацией о течении времени приложения

**struct** Clock {

**long** durTickT; // Длительность одного тика в наносекундах

**long** durTickDt; // Длительность одного тика в наносекундах

**int** countTickDt; // Номер текущего тика часов ПРВ

**int** countTickT; // Номер текущего тика часов ПРВ

**long** endTime; // Длительность работы приложения в секундах

};

// Структура данных, хранящаяся в именованной памяти NAMED\_MEMORY

**struct** NamedMemory {

**double** p; // Вычисляемый параметр

**int** pidP2; // ID процесса P2

**int** pidP1; // ID процесса P1

**int** signChIdP1; // ID канала процесса P1

**int** tickSigusrP2; // Номер сигнала наступления нового тика (уведомления)

pthread\_mutexattr\_t mutexAttr; // Атрибутная запись мутекса

pthread\_mutex\_t mutex; // Мутекс доступа к именованной памяти

pthread\_barrier\_t startBarrier; // Барьер старта таймеров

Clock timeInfo; // Информация о течении времени ПРВ

};

// ---- Структуры для именованной памяти ---- //

// Отображение текущей директории

**void** **viewWorkDirectory**();

// Присоединение именованной памяти

**struct** NamedMemory \***connectToNamedMemory**(**const** **char**\* name);

// Устанвока переодического таймера для отправки импульсов

**void** **setPeriodicTimer**(timer\_t\* periodicTimer, **struct** itimerspec\* periodicTimerStruct, **int** sigChId, **long** tick);

// Устанвока таймера завершения работы

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime);

// Обработка сигнала завершения работы

**void** **deadHandler**(**int** signo);

// Функция потока T1

**void**\* **funcT1**(**void**\* args);

// Функция потока T2

**void**\* **funcT2**(**void**\* args);

// Дескриптор T1

pthread\_t threadT1;

// Дескриптор T2

pthread\_t threadT2;

**int** **main**(**int** argc, **char** \*argv[]) {

cout << "P0: Запущен" << endl;

viewWorkDirectory();

//создание канала

**int** chId = **ChannelCreate**(\_NTO\_CHF\_SENDER\_LEN);

**char** buffer[20];

**const** **char** \*chIdStr = **itoa**(chId, buffer, 10);

cout << "P0: Канал создан: chId = " << chId << endl;

//вызов дочернего процесса P1

**int** pidP1 = **spawnl**( P\_NOWAIT, "/home/host/lab3p1/x86/o/lab3p1", chIdStr, NULL);

**if** (pidP1 < 0){

cout << "P0: Ошибка запуска процесса P1" << pidP1 << endl;

**return** EXIT\_FAILURE;

}

cout << "P0: pid процесса P1 - " << pidP1 << endl;

**bool** isWait = **true**;

**while**(isWait){

**char** msg[20];

\_msg\_info info;

**int** rcvid = **MsgReceive**(chId, msg, **sizeof**(msg), &info);

**if**(rcvid == -1){

cout << "P0: Ошибка MsgReceive - " << strerror(rcvid) << endl;

}

cout << "P0: Получено сообщение от P1" << endl;

**if** (info.pid == pidP1){

**MsgReply**(rcvid, NULL, msg, **sizeof**(msg));

isWait = **false**;

}

}

**struct** NamedMemory \*namedMemoryPtr = connectToNamedMemory(NAMED\_MEMORY);

timer\_t stopTimer;

**struct** itimerspec stopPeriod;

setTimerStop(&stopTimer, &stopPeriod, namedMemoryPtr->timeInfo.endTime);

**int** pidP2 = **spawnl**( P\_NOWAIT, "/home/host/lab3p2/x86/o/lab3p2", chIdStr, NULL);

**if** (pidP2 < 0){

cout << "P0: Ошибка запуска процесса P2 - " << strerror(pidP2) << endl;

**return** EXIT\_FAILURE;

}

cout << "P0: pid процесса P2 - " << pidP2 << endl;

namedMemoryPtr->pidP1 = pidP1;

namedMemoryPtr->pidP2 = pidP2;

**int** threadT1Res = **pthread\_create**(&threadT1, NULL, funcT1, NULL);

**if**(threadT1Res != 0){

cout << "P0: Ошибка старта T1 " << strerror(threadT1Res) << endl;

**return** EXIT\_FAILURE;

}

cout << "P0: Поток T1 создан - " << threadT1 << endl;

**int** threadT2Res = **pthread\_create**(&threadT2, NULL, funcT2, NULL);

**if**(threadT2Res != 0){

cout << "P0: Ошибка старта T2 " << strerror(threadT2Res) << endl;

**return** EXIT\_FAILURE;

}

cout << "P0: Поток T2 создан - " << threadT2 << endl;

cout << "P0: У барьера" << endl;

**pthread\_barrier\_wait**(&(namedMemoryPtr->startBarrier));

cout << "P0: Прошёл барьер" << endl;

// запуск таймера завершения

**int** res = **timer\_settime**(stopTimer, 0, &stopPeriod, NULL);

**if**(res == -1){

cout << "P0: Ошибка запуска таймера завершения - " << strerror(res)<< endl;

}

**while**(**true**){ }

**return** EXIT\_SUCCESS;

}

// Функция потока T1

**void**\* **funcT1**(**void**\* args) {

cout << "P0-T1: Старт" << endl;

**struct** NamedMemory \*namedMemoryPtr = connectToNamedMemory(NAMED\_MEMORY);

**int** sigChId = **ChannelCreate**(\_NTO\_CHF\_SENDER\_LEN);

cout << "P0-T1: Канал создан: sigChId = " << sigChId << endl;

timer\_t periodicTimer;

**struct** itimerspec periodicTick;

setPeriodicTimer(&periodicTimer, &periodicTick, sigChId, namedMemoryPtr->timeInfo.durTickDt);

cout << "P0-T1: У барьера" << endl;

**pthread\_barrier\_wait**(&(namedMemoryPtr->startBarrier));

cout << "P0-T1: Прошёл барьер" << endl;

**int** res = **timer\_settime**(periodicTimer, 0, &periodicTick, NULL);

**if**(res == -1){

cout << "P0-T1: Ошибка запуска переодического таймера - " << strerror(res)<< endl;

}

**while**(**true**){

**MsgReceivePulse**(sigChId, NULL, 0, NULL);

namedMemoryPtr->timeInfo.countTickDt++;

//cout << "P0-T1: countTickDt " << namedMemoryPtr->timeInfo.countTickDt << endl;

// Отправляем сишнал P2

**kill**(namedMemoryPtr->pidP2, namedMemoryPtr->tickSigusrP2);

}

}

// Функция потока T2

**void**\* **funcT2**(**void**\* args) {

cout << "P0-T2: Старт" << endl;

**struct** NamedMemory \*namedMemoryPtr = connectToNamedMemory(NAMED\_MEMORY);

**int** sigChId = **ChannelCreate**(\_NTO\_CHF\_SENDER\_LEN);

cout << "P0-T2: Канал создан: sigChId = " << sigChId << endl;

**int** p1TickCoid = **ConnectAttach**(0, namedMemoryPtr->pidP1, namedMemoryPtr->signChIdP1, \_NTO\_SIDE\_CHANNEL, 0);

**if**(p1TickCoid < 0){

cout << "P0-T2: Ошибка устанвоки соединения c P1 - " << strerror(p1TickCoid) << endl;

exit(EXIT\_FAILURE);

}

timer\_t periodicTimer;

**struct** itimerspec periodicTick;

setPeriodicTimer(&periodicTimer, &periodicTick, sigChId, namedMemoryPtr->timeInfo.durTickT);

cout << "P0-T2: У барьера" << endl;

**pthread\_barrier\_wait**(&(namedMemoryPtr->startBarrier));

cout << "P0-T2: Прошёл барьер" << endl;

**int** res = **timer\_settime**(periodicTimer, 0, &periodicTick, NULL);

**if**(res == -1){

cout << "P0-T2: Ошибка запуска переодического таймера - " << strerror(res)<< endl;

}

**while**(**true**){

**MsgReceivePulse**(sigChId, NULL, 0, NULL);

namedMemoryPtr->timeInfo.countTickT++;

//cout << "P0-T1: countTickT " << namedMemoryPtr->timeInfo.countTickT << endl;

// Отправляем импульс тика процессу P1: приоритет - 10, код - 10, значение - 10

**MsgSendPulse**(p1TickCoid, 10, 10, 10);

}

}

// Устанвока переодического таймера для отправки импульсов

**void** **setPeriodicTimer**(timer\_t\* periodicTimer, **struct** itimerspec\* periodicTimerStruct, **int** sigChId, **long** tick){

// соединение для импульсов уведомления

**int** coid = **ConnectAttach**(0, 0, sigChId, 0, \_NTO\_COF\_CLOEXEC);

**if**(coid ==-1){

cout << "P0: Ошибка устанвоки соединения канала и процесса - " << strerror(coid) << endl;

exit(EXIT\_FAILURE);

}

// импульсы

**struct** sigevent event;

SIGEV\_PULSE\_INIT(&event, coid, SIGEV\_PULSE\_PRIO\_INHERIT, 1, 0);

**timer\_create**(CLOCK\_REALTIME, &event, periodicTimer);

// установить интервал срабатывания периодического таймера тика в системном времени

periodicTimerStruct->it\_value.tv\_sec = 0;

periodicTimerStruct->it\_value.tv\_nsec = tick;

periodicTimerStruct->it\_interval.tv\_sec = 0;

periodicTimerStruct->it\_interval.tv\_nsec = tick;

}

// Устанвока таймера завершения работы

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime) {

**struct** sigevent event;

SIGEV\_SIGNAL\_INIT(&event, SIGUSR2);

**int** res1 = **timer\_create**(CLOCK\_REALTIME, &event, stopTimer);

**if**(res1 == -1){

cout << "P0: Ошибка создания таймера остановки " << strerror(res1)<< endl;

}

stopPeriod->it\_value.tv\_sec = endTime;

stopPeriod->it\_value.tv\_nsec = 0;

stopPeriod->it\_interval.tv\_sec = 0;

stopPeriod->it\_interval.tv\_nsec = 0;

**struct** sigaction act;

sigset\_t set;

**sigemptyset**(&set);

**sigaddset**(&set, SIGUSR2);

act.sa\_flags = 0;

act.sa\_mask = set;

act.\_\_sa\_un.\_sa\_handler = &deadHandler;

**sigaction**(SIGUSR2, &act, NULL);

}

// Обработка сигнала завершения работы

**void** **deadHandler**(**int** signo) {

**if** (signo == SIGUSR2) {

cout << "P0: пришёл сигнал завершения процесса" << endl;

**pthread\_abort**(threadT1);

**pthread\_abort**(threadT2);

exit(EXIT\_SUCCESS);

}

}

// Отображение текущей директории

**void** **viewWorkDirectory**(){

**char** cCurrentPath[FILENAME\_MAX];

**if** (!GetCurrentDir(cCurrentPath, **sizeof**(cCurrentPath)))

exit(EXIT\_FAILURE);

cCurrentPath[**sizeof**(cCurrentPath) - 1] = '\0';

cout << "P0: текущая рабочая директория - " << cCurrentPath << endl;

}

// Функция присоединения к процессу именованной памяти

**struct** NamedMemory\* **connectToNamedMemory**(**const** **char**\* name) {

**struct** NamedMemory \*namedMemoryPtr;

//дескриптор именованной памяти

**int** fd = **shm\_open**(name, O\_RDWR, 0777);

**if**(fd == -1){

cout << "P0: Ошибка открытия объекта именованной памяти - " << strerror(fd) << endl;

exit(EXIT\_FAILURE);

}

namedMemoryPtr = (NamedMemory\*) **mmap**(NULL, **sizeof**(**struct** NamedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

**if**(namedMemoryPtr == MAP\_FAILED){

cout << "P0: Ошибка сопоставления вир. адр. пространства" << endl;

exit(EXIT\_FAILURE);

}

**return** namedMemoryPtr;

}

ПРИЛОЖЕНИЕ B  
Листинг модуля M2

**#include** <cstdlib>

**#include** <iostream>

**#include** <stdio.h>

**#include** <math.h>

**#include** <string.h>

**#include** <stdlib.h>

**#include** <fcntl.h>

**#include** <sys/mman.h>

**#include** <time.h>

**#include** <sys/neutrino.h>

**#include** <pthread.h>

**#include** <errno.h>

**#include** <unistd.h>

**#define** GetCurrentDir getcwd

**using** std::cout;

**using** std::endl;

// Длительность тика 1t (уведомление импульсом - 0,05 c) (наносекунды)

**#define** DUR\_TICK\_T 50000000

// Длительность тика dt (уведомление сигналом - 0,3 с) (наносекунды)

**#define** DUR\_TICK\_DT 300000000

// Время работы приложения (сек)

**#define** END\_TIME 5 //107

// Номер сигнала наступления нового тика (уведомления)

**#define** TICK\_SIGUSR\_P2 SIGUSR1

// Имя именованной памяти

**#define** NAMED\_MEMORY "/16/namedMemory"

// ---- Структуры для именованной памяти ---- //

// Структура данных с информацией о течении времени приложения

**struct** Clock {

**long** durTickT; // Длительность одного тика в наносекундах

**long** durTickDt; // Длительность одного тика в наносекундах

**int** countTickDt; // Номер текущего тика часов ПРВ

**int** countTickT; // Номер текущего тика часов ПРВ

**long** endTime; // Длительность работы приложения в секундах

};

// Структура данных, хранящаяся в именованной памяти NAMED\_MEMORY

**struct** NamedMemory {

**double** p; // Вычисляемый параметр

**int** pidP2; // ID процесса P2

**int** pidP1; // ID процесса P1

**int** signChIdP1; // ID канала процесса P1

**int** tickSigusrP2; // Номер сигнала наступления нового тика (уведомления)

pthread\_mutexattr\_t mutexAttr; // Атрибутная запись мутекса

pthread\_mutex\_t mutex; // Мутекс доступа к именованной памяти

pthread\_barrier\_t startBarrier; // Барьер старта таймеров

Clock timeInfo; // Информация о течении времени ПРВ

};

// ---- Структуры для именованной памяти ---- //

// Отображение текущей директории

**void** **viewWorkDirectory**();

// Обработка сигнала завершения работы

**void** **deadHandler**(**int** signo);

// Отправляет P0 сообщение о готовности к продолжению работы

**void** **sendReadyMessageToP0**(**char** \*chIdP0Str);

// Выполнение расчёта функции в единицу времени

**double** **func**(**double** t);

// Устанвока таймера завершения работы

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime);

// Создание именованной памяти

NamedMemory \***createNamedMemory**(**const** **char**\* name);

// Указатель именованной памяти

**struct** NamedMemory \*namedMemoryPtr;

**int** **main**(**int** argc, **char** \*argv[]) {

cout << "P1: Запущен" << endl;

viewWorkDirectory();

cout << "P1: Параметры: " << "argv[0]= " << argv[0] << endl;

// Присоединение именованной памяти

namedMemoryPtr = createNamedMemory(NAMED\_MEMORY);

// Установка параметров времени приложения

namedMemoryPtr->timeInfo.durTickT = DUR\_TICK\_T;

namedMemoryPtr->timeInfo.durTickDt = DUR\_TICK\_DT;

namedMemoryPtr->timeInfo.endTime = END\_TIME;

// показание часов ПРВ в тиках, -1 - часы не запущены

namedMemoryPtr->timeInfo.countTickT = -1;

namedMemoryPtr->timeInfo.countTickDt = -1;

namedMemoryPtr->tickSigusrP2 = TICK\_SIGUSR\_P2;

// Барьер для синхронизации старта таймеров в процессах

pthread\_barrierattr\_t startAttr;

**pthread\_barrierattr\_init**(&startAttr);

**pthread\_barrierattr\_setpshared**(&startAttr, PTHREAD\_PROCESS\_SHARED);

**pthread\_barrier\_init**(&(namedMemoryPtr->startBarrier), &startAttr, 5);

// Инициализация атрибутной записи разделяемого мутекса

**int** r1 = **pthread\_mutexattr\_init**(&namedMemoryPtr->mutexAttr);

**if**(r1 != EOK){

cout << "Р1: Ошибка pthread\_mutexattr\_init: " << strerror(errno) << endl;

**return** EXIT\_FAILURE;

}

// Установить в атрибутной записи мутекса свойство "разделяемый"

**int** r2 = **pthread\_mutexattr\_setpshared**(&namedMemoryPtr->mutexAttr, PTHREAD\_PROCESS\_SHARED);

**if**(r2 != EOK){

cout << "Р1: Ошибка pthread\_mutexattr\_setpshared: " << strerror(errno) << endl;

**return** EXIT\_FAILURE;

}

// Инициализация разделяемого мутекса

**int** r3 = **pthread\_mutex\_init**(&namedMemoryPtr->mutex, &namedMemoryPtr->mutexAttr);

**if**(r3 != EOK){

cout << "Р1: Ошибка pthread\_mutex\_init: " << strerror(errno) << endl;

**return** EXIT\_FAILURE;

}

//создание канала

**int** signChIdP1 = **ChannelCreate**(\_NTO\_CHF\_SENDER\_LEN);

namedMemoryPtr->signChIdP1 = signChIdP1;

sendReadyMessageToP0(argv[0]);

timer\_t stopTimer;

**struct** itimerspec stopPeriod;

setTimerStop(&stopTimer, &stopPeriod, namedMemoryPtr->timeInfo.endTime);

cout << "P1: У барьера" << endl;

**pthread\_barrier\_wait**(&(namedMemoryPtr->startBarrier));

cout << "P1: Прошёл барьер" << endl;

// запуск таймера завершения

**int** res = **timer\_settime**(stopTimer, 0, &stopPeriod, NULL);

**if**(res == -1){

cout << "P1: Ошибка запуска таймера" << strerror(res)<< endl;

}

// величина тика из нсек в сек 50000000 нсек -> 0.05 сек

**const** **double** tickSecDuration = namedMemoryPtr->timeInfo.durTickT / 1000000000.;

**while**(**true**){

**MsgReceivePulse**(signChIdP1, NULL, 0, NULL);

**double** time = namedMemoryPtr->timeInfo.countTickT \* tickSecDuration;

**double** value = func(time);

**pthread\_mutex\_lock**(&(namedMemoryPtr->mutex));

namedMemoryPtr->p = value;

**pthread\_mutex\_unlock**(&(namedMemoryPtr->mutex));

cout << "P1: Расчитал" << " value: " << value << " time: " << time << " countTickT: " << namedMemoryPtr->timeInfo.countTickT << endl;

}

**return** EXIT\_SUCCESS;

}

// Создание именованной памяти

NamedMemory \***createNamedMemory**(**const** **char**\* name){

**struct** NamedMemory \*namedMemoryPtr;

//дескриптор именованной памяти

**int** fd = **shm\_open**(name, O\_RDWR | O\_CREAT, 0777);

**if**(fd == -1){

cout << "P1: Ошибка создания/открытия объекта именованной памяти - " << strerror(fd) << endl;

exit(EXIT\_FAILURE);

}

**int** tr1 = **ftruncate**(fd, 0);

**int** tr2 = **ftruncate**(fd, **sizeof**(**struct** NamedMemory));

**if**(tr1 == -1 || tr2 == -1){

cout << "P1: Ошибка ftruncate" << endl;

exit(EXIT\_FAILURE);

}

namedMemoryPtr = (NamedMemory\*) **mmap**(NULL, **sizeof**(**struct** NamedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

**if**(namedMemoryPtr == MAP\_FAILED){

cout << "P1: Ошибка сопоставления вир. адр. пространства" << endl;

exit(EXIT\_FAILURE);

}

**return** namedMemoryPtr;

}

// Устанвока таймера завершения работы

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime) {

**struct** sigevent event;

SIGEV\_SIGNAL\_INIT(&event, SIGUSR2);

**int** res1 = **timer\_create**(CLOCK\_REALTIME, &event, stopTimer);

**if**(res1 == -1){

cout << "P1: Ошибка создания таймера остановки " << strerror(res1)<< endl;

}

stopPeriod->it\_value.tv\_sec = endTime;

stopPeriod->it\_value.tv\_nsec = 0;

stopPeriod->it\_interval.tv\_sec = 0;

stopPeriod->it\_interval.tv\_nsec = 0;

**struct** sigaction act;

sigset\_t set;

**sigemptyset**(&set);

**sigaddset**(&set, SIGUSR2);

act.sa\_flags = 0;

act.sa\_mask = set;

act.\_\_sa\_un.\_sa\_handler = &deadHandler;

**sigaction**(SIGUSR2, &act, NULL);

}

// Выполнение расчёта функции в единицу времени

**double** **func**(**double** t) {

**const** **double** a = 1.9;

**const** **double** b = 1.1;

**double** result = (b \* pow(t, 2))/(**pow**(M\_E, a \* t) + t);

**return** result;

}

// Отправляет P0 сообщение о готовности к продолжению работы

**void** **sendReadyMessageToP0**(**char** \*chIdP0Str){

**char** rmsg[20];

**int** chIdP0 = atoi(chIdP0Str);

cout << "P1: установление соединения с каналом P0" << endl;

**int** coidP0 = **ConnectAttach**(0, **getppid**(), chIdP0, \_NTO\_SIDE\_CHANNEL, 0);

**if**(coidP0 == -1){

cout << "P1: Ошибка соединения с каналом P0 - " << strerror(coidP0) << endl;

exit(EXIT\_FAILURE);

}

cout << "Р1: Посылаю сообщение Р0" << endl;

**char** \*smsg1 = (**char** \*)"P1";

**int** sendRes = MsgSend(coidP0, smsg1, strlen(smsg1) + 1, rmsg, **sizeof**(rmsg));

**if**(sendRes == -1){

cout << "P1: Ошибка MsgSend при отправки в P0 - " << strerror(sendRes) << endl;

exit(EXIT\_FAILURE);

}

}

// Обработка сигнала завершения работы

**void** **deadHandler**(**int** signo) {

**if** (signo == SIGUSR2) {

cout << "P1: пришёл сигнал завершения процесса" << endl;

**pthread\_barrier\_destroy**(&(namedMemoryPtr->startBarrier));

exit(EXIT\_SUCCESS);

}

}

// Отображение текущей директории

**void** **viewWorkDirectory**(){

**char** cCurrentPath[FILENAME\_MAX];

**if** (!GetCurrentDir(cCurrentPath, **sizeof**(cCurrentPath)))

exit(EXIT\_FAILURE);

cCurrentPath[**sizeof**(cCurrentPath) - 1] = '\0';

cout << "P1: текущая рабочая директория - " << cCurrentPath << endl;

}

ПРИЛОЖЕНИЕ C  
Листинг модуля M3

**#include** <cstdlib>

**#include** <iostream>

**#include** <stdio.h>

**#include** <string.h>

**#include** <stdlib.h>

**#include** <sys/neutrino.h>

**#include** <pthread.h>

**#include** <fcntl.h>

**#include** <sys/mman.h>

**#include** <unistd.h>

**#define** GetCurrentDir getcwd

**using** std::cout;

**using** std::endl;

**using** std::FILE;

// Имя именованной памяти

**#define** NAMED\_MEMORY "/16/namedMemory"

// Файл для записи трендов

**#define** TREND\_FILE "/home/host/lab3Trend/trend.txt"

// ---- Структуры для именованной памяти ---- //

// Структура данных с информацией о течении времени приложения

**struct** Clock {

**long** durTickT; // Длительность одного тика в наносекундах

**long** durTickDt; // Длительность одного тика в наносекундах

**int** countTickDt; // Номер текущего тика часов ПРВ

**int** countTickT; // Номер текущего тика часов ПРВ

**long** endTime; // Длительность работы приложения в секундах

};

// Структура данных, хранящаяся в именованной памяти NAMED\_MEMORY

**struct** NamedMemory {

**double** p; // Вычисляемый параметр

**int** pidP2; // ID процесса P2

**int** pidP1; // ID процесса P1

**int** signChIdP1; // ID канала процесса P1

**int** tickSigusrP2; // Номер сигнала наступления нового тика (уведомления)

pthread\_mutexattr\_t mutexAttr; // Атрибутная запись мутекса

pthread\_mutex\_t mutex; // Мутекс доступа к именованной памяти

pthread\_barrier\_t startBarrier; // Барьер старта таймеров

Clock timeInfo; // Информация о течении времени ПРВ

};

// ---- Структуры для именованной памяти ---- //

// Отображение текущей директории

**void** **viewWorkDirectory**();

// Присоединение именованной памяти

**struct** NamedMemory \***connectToNamedMemory**(**const** **char**\* name);

// Устанвока таймера завершения работы

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime);

// Обработка сигнала завершения работы

**void** **deadHandler**(**int** signo);

FILE\* trendFile;

**int** **main**(**int** argc, **char** \*argv[]) {

cout << "P2: Запущен" << endl;

viewWorkDirectory();

**struct** NamedMemory \*namedMemoryPtr = connectToNamedMemory(NAMED\_MEMORY);

trendFile = fopen(TREND\_FILE, "w");

**if**(trendFile == NULL){

cout << "P2: Ошибка открытия файла для запси тренда" << endl;

exit(EXIT\_FAILURE);

}

cout << "Р2: Открыт файл тренда trend.txt" << endl;

sigset\_t set;

**sigemptyset**(&set);

**sigaddset**(&set, namedMemoryPtr->tickSigusrP2);

timer\_t stopTimer;

**struct** itimerspec stopPeriod;

setTimerStop(&stopTimer, &stopPeriod, namedMemoryPtr->timeInfo.endTime);

cout << "P2: У барьера" << endl;

**pthread\_barrier\_wait**(&(namedMemoryPtr->startBarrier));

cout << "P2: Прошёл барьер" << endl;

// запуск таймера завершения

**int** res = **timer\_settime**(stopTimer, 0, &stopPeriod, NULL);

**if**(res == -1){

cout << "P2: Ошибка запуска таймера" << strerror(res)<< endl;

}

// величина тика из нсек в сек 300000000 нсек -> 0.3 сек

**const** **double** tickSecDuration = namedMemoryPtr->timeInfo.durTickDt / 1000000000.;

**while**(**true**){

**int** sig = **SignalWaitinfo**(&set, NULL);

**if**(sig == namedMemoryPtr->tickSigusrP2){

**pthread\_mutex\_lock**(&(namedMemoryPtr->mutex));

**double** value = namedMemoryPtr->p;

**pthread\_mutex\_unlock**(&(namedMemoryPtr->mutex));

**double** time = namedMemoryPtr->timeInfo.countTickDt \* tickSecDuration;

cout << "P2: Записывает " << "value: " << value << " time: " << time << " countTickDt: " << namedMemoryPtr->timeInfo.countTickDt << endl;

fprintf(trendFile, "%f\t%f\n", value, time);

}

}

**return** EXIT\_SUCCESS;

}

// Отображение текущей директории

**void** **viewWorkDirectory**(){

**char** cCurrentPath[FILENAME\_MAX];

**if** (!GetCurrentDir(cCurrentPath, **sizeof**(cCurrentPath)))

exit(EXIT\_FAILURE);

cCurrentPath[**sizeof**(cCurrentPath) - 1] = '\0';

cout << "P2: текущая рабочая директория - " << cCurrentPath << endl;

}

// Устанвока таймера завершения работы

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime) {

**struct** sigevent event;

SIGEV\_SIGNAL\_INIT(&event, SIGUSR2);

**int** res1 = **timer\_create**(CLOCK\_REALTIME, &event, stopTimer);

**if**(res1 == -1){

cout << "P2: Ошибка создания таймера остановки " << strerror(res1)<< endl;

}

stopPeriod->it\_value.tv\_sec = endTime;

stopPeriod->it\_value.tv\_nsec = 0;

stopPeriod->it\_interval.tv\_sec = 0;

stopPeriod->it\_interval.tv\_nsec = 0;

**struct** sigaction act;

sigset\_t set;

**sigemptyset**(&set);

**sigaddset**(&set, SIGUSR2);

act.sa\_flags = 0;

act.sa\_mask = set;

act.\_\_sa\_un.\_sa\_handler = &deadHandler;

**sigaction**(SIGUSR2, &act, NULL);

}

// Обработка сигнала завершения работы

**void** **deadHandler**(**int** signo) {

**if** (signo == SIGUSR2) {

cout << "P2: пришёл сигнал завершения процесса" << endl;

fclose(trendFile);

exit(EXIT\_SUCCESS);

}

}

// Функция присоединения к процессу именованной памяти

**struct** NamedMemory\* **connectToNamedMemory**(**const** **char**\* name) {

**struct** NamedMemory \*namedMemoryPtr;

//дескриптор именованной памяти

**int** fd = **shm\_open**(name, O\_RDWR, 0777);

**if**(fd == -1){

cout << "P2: Ошибка открытия объекта именованной памяти - " << strerror(fd) << endl;

exit(EXIT\_FAILURE);

}

namedMemoryPtr = (NamedMemory\*) **mmap**(NULL, **sizeof**(**struct** NamedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

**if**(namedMemoryPtr == MAP\_FAILED){

cout << "P2: Ошибка сопоставления вир. адр. пространства" << endl;

exit(EXIT\_FAILURE);

}

**return** namedMemoryPtr;

}