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
Дисциплина Технологии промышленного программирования

ОТЧЁТ

к лабораторной работе

«Работа с именованной памятью и службой реального времени»

Обучающийся группы 6231-020302D _____ Гижевская В.Д.

Преподаватель_____ Баландин А.В.

Самара 2023

СОДЕРЖАНИЕ

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 $t \in [0,T]$. 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 T программная система должна завершить свою работу.

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

Объект Op в программной системе моделируется процессом P1(M1). Программный модуль M1 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 P2(M2), запускаемого на базе модуля M2 (порядок запуска процессов P1 и P2 определяется в варианте задания):

Процесс P1, начиная с t=0, периодически с заданной частотой обновляет текущее значение параметра p в именованной памяти.

Процесс P2, 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 P2 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 P1 и P2 должны быть синхронизированы по моменту времени t=0. (процесс P2 должен получить первое значение параметра p в момент t=0). Метод синхронизации выбрать самостоятельно.

При наступлении момента t=T 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.

Номер варианта	Порядок загрузки и запуска программной системы	Вид функции $F(t)$	Единица временной шкалы 1t (сек)	Единица временной шкалы Δt (сек)	Значение T (сек)
9.	Процесс P1 запускается стартовым процессом P0, который запускается «вручную». Процесс P1 запускает процесс P2. Именованную память создает процесс P0.	$F(t) = at(1 + ae^{-t})$ $a = 3{,}5$	0.03, уведомление сигналом	0.2, уведомление импульсом	83

2 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.

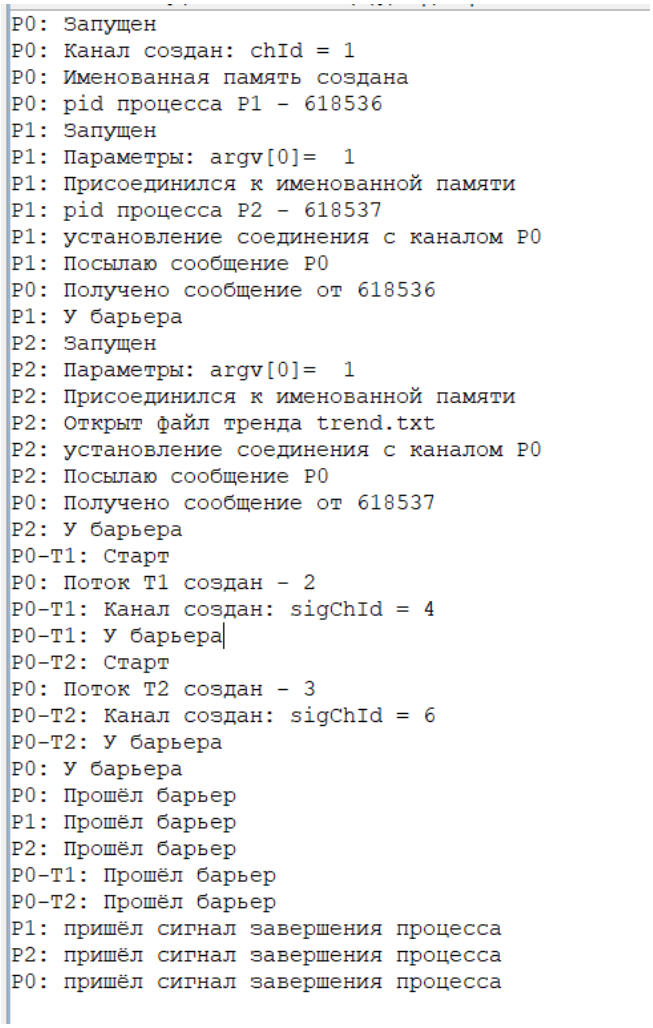

Рисунок 1 – Результат выполнения программы в консоли

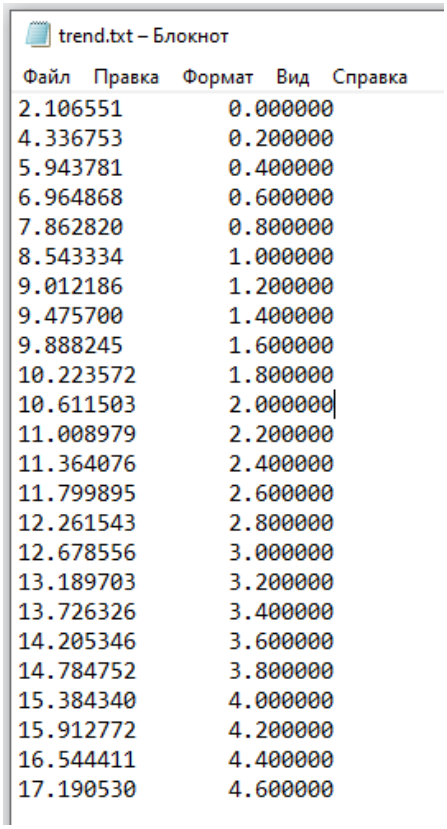

Рисунок 2 – Результат записанные в файл

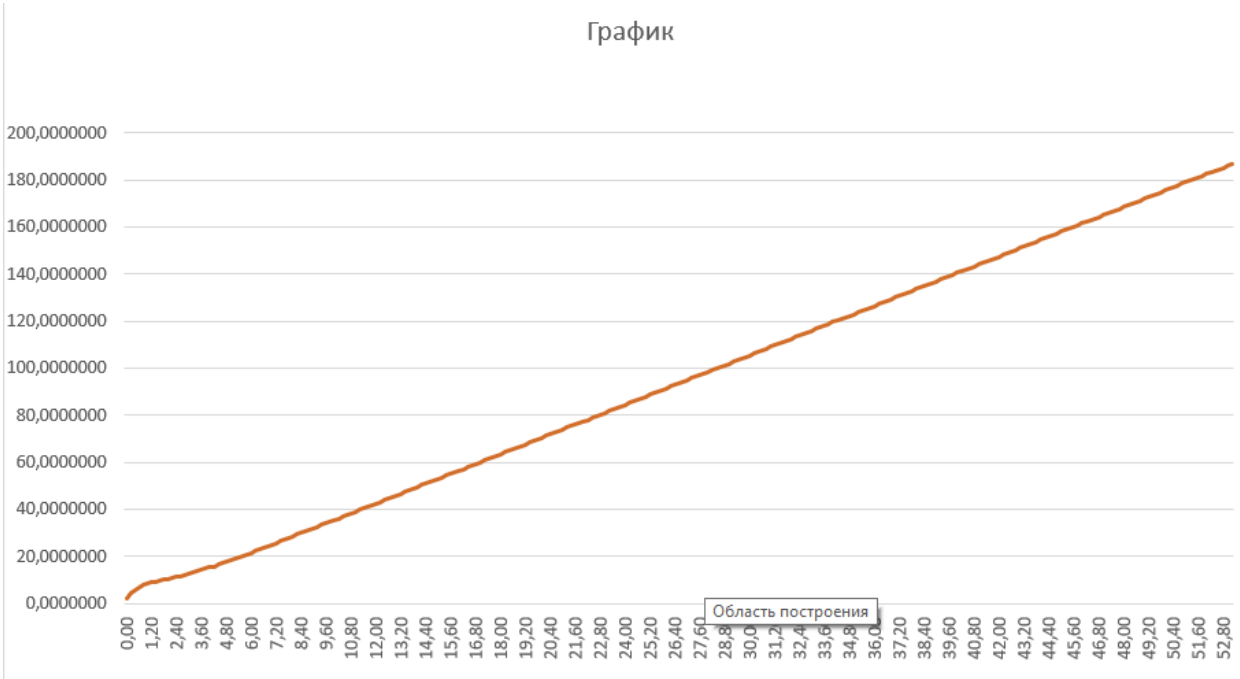

Рисунок 3 – Результат в виде диаграммы excel

ПРИЛОЖЕНИЕ А

Листинг модуля М1

```
#include <cstdlib>
#include <iostream>
#include <string.h>
#include <process.h>
#include <stdlib.h>
#include <sys/neutrino.h>
#include <pthread.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <errno.h>

#include <unistd.h>
#define GetCurrentDir getcwd

using std::cout;
using std::endl;

// Длительность тика 1t (уведомление импульсом - 0,03 с) (наносекунды)
#define DUR_TICK_T 30000000
// Длительность тика dt (уведомление сигналом - 0,2 с) (наносекунды)
#define DUR_TICK_DT 200000000
// Время работы приложения (сек)
#define END_TIME 5 //83 5
// Номер сигнала наступления нового тика (уведомления)
#define TICK_SIGUSR_P1 SIGUSR1

// Имя именованной памятиЫ
#define NAMED_MEMORY "/9/namedMemory"

// 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 pidP0;					// ID процесса P0
	int pidP2;					// ID процесса P2
	int pidP1;					// ID процесса P1
	int signChIdP2;				// ID канала процесса P1
	int tickSigusrP1;			// 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
};
```

```
// Создание именованной памяти
NamedMemory *createNamedMemory(const char* name);
// Установка переодического таймера для отправки импульсов
void setPeriodicTimer(timer_t* periodicTimer, struct itimerspec*
periodicTimerStruct, int sigChId, long tick);
// Установка 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;

// Указатель именованной памяти
struct NamedMemory *namedMemoryPtr;

int main(int argc, char *argv[]) {
	cout << "P0: Запущен" << endl;

	//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;

	// Присоединение именованной памяти
	namedMemoryPtr = createNamedMemory(NAMED_MEMORY);
	cout << "P0: Именованная память создана" << endl;

	// 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->tickSigusrP1 = TICK_SIGUSR_P1;

	// 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 << "P0: 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 << "P0: 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 << "P0: Ошибка pthread_mutex_init: " << strerror(errno) <<
endl;
            return EXIT_FAILURE;
      }

      //вызов дочернего процесса P1
      int pidP1 = spawnl( P_NOWAIT, "/home/host/Lab3/P1/x86/o/P1", chIdStr,
NULL);
      if (pidP1 < 0){
            cout << "P0: Ошибка запуска процесса P1 " << strerror(pidP1) <<
endl;
            exit(EXIT_FAILURE);
      }
      cout << "P0: pid процесса P1 - " << pidP1 <<  endl;

      namedMemoryPtr->pidP1 = pidP1;

      namedMemoryPtr->pidP0 = getpid();

      int count = 0;
      while(count < 2){
            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 " << info.pid <<  endl;
          MsgReply(rcvid, NULL, msg, sizeof(msg));
          count++;
      }

      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;

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

      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" << strerror(res)<< endl;
	}

	while(true){ }

	return EXIT_SUCCESS;
}

// Функция потока T1
void* funcT1(void* args) {
	cout << "P0-T1: Старт" << endl;

    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.durTickT);

	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.countTickT++;

		// Отправляем сишнал P1
		kill(namedMemoryPtr->pidP1, namedMemoryPtr->tickSigusrP1);
	}
}

// Функция потока T2
void* funcT2(void* args) {
	cout << "P0-T2: Старт" << endl;

    int sigChId = ChannelCreate(_NTO_CHF_SENDER_LEN);
    cout << "P0-T2: Канал создан: sigChId = " << sigChId << endl;

	int p1TickCoid = ConnectAttach(0, namedMemoryPtr->pidP2, namedMemoryPtr-
>signChIdP2, _NTO_SIDE_CHANNEL, 0);
	if(p1TickCoid < 0){
		cout << "P0-T2: Ошибка установки соединения с P2 - " <<
strerror(p1TickCoid) <<  endl;
	    exit(EXIT_FAILURE);
	}

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

	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.countTickDt++;;

	    // Отправляем импульс тика процессу P1: приоритет - 10, код - 10,
значение - 10
		MsgSendPulse(p1TickCoid, 10, 10, 10);
	}
}

// Установка 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 установки 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;
}

// 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 << "P0: 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 << "P0: 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 << "P0: Ошибка сопоставления вир. адр. пространства" <<
endl;
          exit(EXIT_FAILURE);
      }

      return namedMemoryPtr;
}

// Установка 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; //учитывать последнюю инициацию сигнала
      act.sa_mask = set;
      act.__sa_un._sa_handler = &deadHandler; //Используется старый тип
обработчика
      sigaction(SIGUSR2, &act, NULL);
}

// Обработка сигнала завершения работы
void deadHandler(int signo) {
      if (signo == SIGUSR2) {
            cout << "P0: пришёл сигнал завершения процесса" <<  endl;
            pthread_barrier_destroy(&(namedMemoryPtr->startBarrier));
            pthread_abort(threadT1);
            pthread_abort(threadT2);
            exit(EXIT_SUCCESS);
      }
}

}
```

ПРИЛОЖЕНИЕ В

Листинг модуля М2

```
#include <cstdlib>
#include <iostream>
#include <pthread.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <sys/neutrino.h>

#include <unistd.h>
#define GetCurrentDir getcwd

using std::cout;
using std::endl;
using std::cos;
using std::log;

// Имя именованной памяти
#define NAMED_MEMORY "/9/namedMemory"

// 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 pidP0;					// ID процесса P0
	int pidP2;					// ID процесса P2
	int pidP1;					// ID процесса P1
	int signChIdP2;				// ID канала процесса P1
	int tickSigusrP1;			// 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
};

// Присоединение именованной памяти
struct NamedMemory *connectToNamedMemory(const char* name);
// Выполнение расчёта функции в единицу времени
double func(double t);
// Установка таймера завершения работы
```

```
void setTimerStop(timer_t* stopTimer, struct itimerspec* stopPeriod, long
endTime);
// Обработка сигнала завершения работы
void deadHandler(int signo);
// Отправляет P0 сообщение о готовности к продолжению работы
void sendReadyMessageToP0(char *chIdP0Str);

int main(int argc, char *argv[]) {
	cout << "P1: Запущен" << endl;
	cout << "P1: Параметры: " << "argv[0]=  " << argv[0] <<  endl;

	struct NamedMemory *namedMemoryPtr = connectToNamedMemory(NAMED_MEMORY);
	cout << "P1: Присоединился к именованной памяти" << endl;

	//вызов дочернего процесса P2
	int pidP2 = spawnl( P_NOWAIT,
"/home/host/Lab3/P2/x86/o/P2",(char*)argv[0], NULL);
	if (pidP2 < 0){
		cout << "P1: Ошибка запуска процесса P2 " << strerror(pidP2) <<
endl;
		exit(EXIT_FAILURE);
	}
	cout << "P1: pid процесса P2 - " << pidP2 <<  endl;

	namedMemoryPtr->pidP2 = pidP2;

	sigset_t set;
	sigemptyset(&set);
	sigaddset(&set, namedMemoryPtr->tickSigusrP1);

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

	sendReadyMessageToP0(argv[0]);

	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 180000000 нсек -> 0,03 сек
	const double tickSecDuration = namedMemoryPtr->timeInfo.durTickT /
1000000000.;

	while(true){
		// Ожидание сигнала
		int sig = SignalWaitinfo(&set, NULL);
		if(sig == namedMemoryPtr->tickSigusrP1){

			double time = namedMemoryPtr->timeInfo.countTickT *
tickSecDuration;
			double value = func(time);

			pthread_mutex_lock(&(namedMemoryPtr->mutex));
			namedMemoryPtr->p = value;
			pthread_mutex_unlock(&(namedMemoryPtr->mutex));
		}
```

```
	}
	return EXIT_SUCCESS;
}

// Выполнение расчёта функции в единицу времени
double func(double t) {
	const double a = 3.5;
	double sc1 = pow(M_E, -1 * t);
	double sc2 = a *sc1;
	double sc3 = 1 + sc2;
	double result = a * t * sc3;
	return result;
}

// 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 << "P1: 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 << "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);
}

// Обработка сигнала завершения работы
void deadHandler(int signo) {
```

```
	if (signo == SIGUSR2) {
		cout << "P1: пришёл сигнал завершения процесса" <<  endl;
		exit(EXIT_SUCCESS);
	}
}

// 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 << "P1: Посылаю сообщение P0" <<  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);
	}
}
```

ПРИЛОЖЕНИЕ С

Листинг модуля М3

```
#include <cstdlib>
#include <iostream>
#include <string.h>
#include <process.h>
#include <stdlib.h>
#include <sys/neutrino.h>
#include <pthread.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>

#include <unistd.h>
#define GetCurrentDir getcwd

using std::cout;
using std::endl;
using std::FILE;

// Имя именованной памяти
#define NAMED_MEMORY "/9/namedMemory"
// Файл для записи трендов
#define TREND_FILE "/home/host/Lab3/Trend/trend.txt"

// 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 pidP0;                                      // ID процесса P0
	int pidP2;                                      // ID процесса P2
	int pidP1;                                      // ID процесса P1
	int signChIdP2;                                 // ID канала процесса P1
	int tickSigusrP1;                          // 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
};

// Присоединение именованной памяти
struct NamedMemory *connectToNamedMemory(const char* name);
// Установка таймера завершения работы
void setTimerStop(timer_t* stopTimer, struct itimerspec* stopPeriod, long
endTime);
// Обработка сигнала завершения работы
```

```
void deadHandler(int signo);
// Отправляет P0 сообщение о готовности к продолжению работы
void sendReadyMessageToP0(char *chIdP0Str, int pidP0);

FILE* trendFile;

int main(int argc, char *argv[]) {
	cout << "P2: Запущен" << endl;
	cout << "P2: Параметры: " << "argv[0]=  " << argv[0] <<  endl;

	struct NamedMemory *namedMemoryPtr = connectToNamedMemory(NAMED_MEMORY);
	cout << "P2: Присоединился к именованной памяти" << endl;

	trendFile = fopen(TREND_FILE, "w");
	if(trendFile == NULL){
		cout << "P2: Ошибка открытия файла для записи тренда" << endl;
		exit(EXIT_FAILURE);
	}
	cout << "P2: Открыт файл тренда trend.txt" << endl;

	//создание канала
	int signChIdP2 = ChannelCreate(_NTO_CHF_SENDER_LEN);
	namedMemoryPtr->signChIdP2 = signChIdP2;

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

	sendReadyMessageToP0(argv[0], namedMemoryPtr->pidP0);

	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 400000000 нсек -> 0,2 сек
	const double tickSecDuration = namedMemoryPtr->timeInfo.durTickDt /
1000000000.;
	//cout << "P2: tickSecDuration - " << tickSecDuration << endl;

	while(true){
		// Ожидаем импульс
		MsgReceivePulse(signChIdP2, NULL, 0, NULL);

		pthread_mutex_lock(&(namedMemoryPtr->mutex));
		double value = namedMemoryPtr->p;
		pthread_mutex_unlock(&(namedMemoryPtr->mutex));

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

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

	return 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;
}

// 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);
	}
}

// Отправляет P0 сообщение о готовности к продолжению работы
void sendReadyMessageToP0(char *chIdP0Str, int pidP0){
	char rmsg[20];
	int chIdP0 = atoi(chIdP0Str);
	cout << "P2: установление соединения с каналом P0" <<  endl;
	int coidP0 = ConnectAttach(0, pidP0, chIdP0, _NTO_SIDE_CHANNEL, 0);
	if(coidP0 == -1){
		cout << "P2: Ошибка соединения с каналом P0 - " <<
strerror(coidP0) <<  endl;
```

```
            exit(EXIT_FAILURE);
        }
        cout << "P2: Посылаю сообщение P0" <<  endl;
        char *smsg1 = (char *)"P1";
        int sendRes = MsgSend(coidP0, smsg1, strlen(smsg1) + 1, rmsg,
sizeof(rmsg));
        if(sendRes == -1){
              cout << "P2: Ошибка MsgSend при отправки в P0 - " <<
strerror(sendRes) <<  endl;
              exit(EXIT_FAILURE);
        }
}
```