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

ОТЧЕТ

по лабораторной работе

Работа с именованной памятью и службой реального времени

Вариант № 13

Студент: Лазарев М.Ю

Группа: 6232-020402D

Преподаватель: Баландин А.В.

Самара 2025

1 Задание

Цель работы - освоение функций ОСРВ для работы с именованной памятью и синхронизации нитей с реальным временем.

Разработать приложение реального времени (ПРВ), осуществляющее мониторинг состояния абстрактного физического объекта O_p, p – изменяющийся во времени параметр объекта. Мониторинг объекта O_p 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 O_p в программной системе моделируется процессом P1(M1). Программный модуль M1 реализует вычисление функции p=F(t) и размещение полученного текущего значения параметра в именованную память, предварительно созданную при загрузке ПРВ.

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 - <p, t>. 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.

При выполнении лабораторной работы обязательно использовать указанную последовательность запуска процессов (нитей) и создания именованной памяти.

Порядок загрузки и запуска программной системы:

Процессы P1 и P2 запускаются «вручную» независимо друг от друга. Именованную память создает процесс P2.

Вид функции F(t):

$F(t) = \ln(t^2 + 2t + 10)$

Единица временной шкалы 1t (сек):

0.09, уведомление сигналом

Единица временной шкалы Δt (сек):

0.3, уведомление импульсом

Значение T (сек):

101

Для синхронизации ПРВ с реальным временем можно использовать вспомогательные нити.

2 Результаты работы

```
//Процесс P0
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <signal.h>
#include <sys/siginfo.h>
#include <sys/neutrino.h>
#include <pthread.h>
#include <errno.h>
#include <sys/wait.h>
#define NAMED_MEMORY "/namedMemory" // имя именованной памяти
#define END_TIME 101 // время работы приложения 101 сек
#define TICK 90000000 // длительность тика 0,09с в наносекундах
#define TICK_SIGNAL_SIGUSR1 SIGUSR1 // номер сигнала уведомления истечения тика
// структура таймера работы приложения
struct Clock {
    long tick; // длительность тика в наносекундах
    int Time; // номер текущего тика
    long endTime; // время работы приложения в секундах
};
// структура именованной памяти
struct namedMemory {
    double p; // параметр объекта
    pthread_barrier_t startBarrier; // барьер синхронизации старта процессов
    Clock timeInfo; // информация о таймере
    int pidP1; // идентификатор процесса P1
    int chidP1; // идентификатор канала процесса P1
    int pidP2; // идентификатор процесса P2
```

```
    pthread_mutexattr_t MutexAttr; // атрибуты мутекса
    pthread_mutex_t Mutex; // мутекс для синхронизации доступа к памяти
};
struct namedMemory *namedMemoryPtr; // указатель на именованную память
// глобальные переменные для ресурсов P0
int p0_chid = -1; // идентификатор канала P0
int p0_coidP1 = -1; // идентификатор соединения с P1
timer_t p0_periodicTimer, p0_stopTimer; // дескрипторы таймеров
// функция создания именованной памяти
struct namedMemory* createNamedMemory(const char* name) {
    int fd; // дескриптор файла именованной памяти
    // очистка старой памяти перед созданием новой
    shm_unlink(name);
    // создание новой именованной памяти
    if ((fd = shm_open(name, O_RDWR | O_CREAT | O_EXCL, 0666)) == -1) {
        perror("P0: Ошибка shm_open");
        exit(EXIT_FAILURE);
    }
    // установка размера именованной памяти
    if (ftruncate(fd, sizeof(struct namedMemory)) == -1) {
        perror("P0: Ошибка ftruncate");
        close(fd);
        exit(EXIT_FAILURE);
    }
    // отображение именованной памяти в адресное пространство процесса
    struct namedMemory *ptr;
    if ((ptr = (namedMemory*) mmap(NULL, sizeof(struct namedMemory),
                        PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)) ==
MAP_FAILED) {
        perror("P0: Ошибка mmap");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd); // закрытие файлового дескриптора после отображения
```

```
    return ptr;
}
// функция установки однократного таймера завершения
void setTimerStop(timer_t *stopTimer, struct itimerspec *stopPeriod) {
    struct sigevent event;
    // настройка уведомления сигналом SIGUSR2
    SIGEV_SIGNAL_INIT(&event, SIGUSR2);
    if (timer_create(CLOCK_REALTIME, &event, stopTimer) == -1) {
        perror("P0: Ошибка создания таймера завершения");
        exit(EXIT_FAILURE);
    }
    // установка времени срабатывания однократного таймера
    stopPeriod->it_value.tv_sec = END_TIME; // через END_TIME секунд
    stopPeriod->it_value.tv_nsec = 0;
    stopPeriod->it_interval.tv_sec = 0;
    stopPeriod->it_interval.tv_nsec = 0;
}
// функция установки периодического таймера тиков
void setPeriodicTimer(timer_t* periodicTimer, struct itimerspec* periodicTimerStruct, int chid) {
    struct sigevent event;
    // установка соединения для уведомлений импульсами
    int coid = ConnectAttach(0, 0, chid, 0, 0);
    if (coid == -1) {
        perror("P0: Ошибка ConnectAttach для таймера");
        exit(EXIT_FAILURE);
    }
    // настройка уведомления импульсами
    SIGEV_PULSE_INIT(&event, coid, SIGEV_PULSE_PRIO_INHERIT, 1, 0);
    if (timer_create(CLOCK_REALTIME, &event, periodicTimer) == -1) {
        perror("P0: Ошибка создания периодического таймера");
        exit(EXIT_FAILURE);
    }
    // установка интервала срабатывания периодического таймера
```

```
    periodicTimerStruct->it_value.tv_sec = 0;
    periodicTimerStruct->it_value.tv_nsec = TICK; // первый тик через TICK наносекунд
    periodicTimerStruct->it_interval.tv_sec = 0;
    periodicTimerStruct->it_interval.tv_nsec = TICK; // повтор каждые TICK наносекунд
}
// обработчик сигнала завершения работы
void deadHandler(int signo) {
    if (signo == SIGUSR2) {
        std::cout << "P0: Получен сигнал завершения" << std::endl;
        // отправка сигналов завершения дочерним процессам
        if (namedMemoryPtr->pidP1 > 0) {
            kill(namedMemoryPtr->pidP1, SIGUSR2);
            std::cout << "P0: Отправлен SIGUSR2 процессу P1" << std::endl;
        }
        if (namedMemoryPtr->pidP2 > 0) {
            kill(namedMemoryPtr->pidP2, SIGUSR2);
            std::cout << "P0: Отправлен SIGUSR2 процессу P2" << std::endl;
        }
        // ожидание завершения дочерних процессов без блокировки
        int status;
        pid_t result;
        int timeout_counter = 0;
        const int max_timeout = 5; // максимальное количество проверок
        // циклическая проверка завершения процессов с короткими паузами
        while (timeout_counter < max_timeout &&
               (namedMemoryPtr->pidP1 > 0 || namedMemoryPtr->pidP2 > 0)) {
            // проверка завершения P1
            if (namedMemoryPtr->pidP1 > 0) {
                result = waitpid(namedMemoryPtr->pidP1, &status, WNOHANG);
                if (result == namedMemoryPtr->pidP1) {
                    std::cout << "P0: Процесс P1 завершился" << std::endl;
                    namedMemoryPtr->pidP1 = 0;
                } else if (result == -1) {
```

```
            perror("P0: Ошибка ожидания P1");
            namedMemoryPtr->pidP1 = 0;
        }
    }
    // проверка завершения P2
    if (namedMemoryPtr->pidP2 > 0) {
        result = waitpid(namedMemoryPtr->pidP2, &status, WNOHANG);
        if (result == namedMemoryPtr->pidP2) {
            std::cout << "P0: Процесс P2 завершился" << std::endl;
            namedMemoryPtr->pidP2 = 0;
        } else if (result == -1) {
            perror("P0: Ошибка ожидания P2");
            namedMemoryPtr->pidP2 = 0;
        }
    }
    // короткая пауза между проверками
    if (namedMemoryPtr->pidP1 > 0 || namedMemoryPtr->pidP2 > 0) {
        usleep(100000); // 100ms пауза
        timeout_counter++;
    }
}
// принудительное завершение процессов, если они еще работают
if (namedMemoryPtr->pidP1 > 0) {
    kill(namedMemoryPtr->pidP1, SIGKILL);
    std::cout << "P0: Процесс P1 принудительно завершен" << std::endl;
}
if (namedMemoryPtr->pidP2 > 0) {
    kill(namedMemoryPtr->pidP2, SIGKILL);
    std::cout << "P0: Процесс P2 принудительно завершен" << std::endl;
}
// освобождение ресурсов P0
if (p0_coidP1 != -1) {
    ConnectDetach(p0_coidP1);
```

```
            std::cout << "P0: Соединение с P1 отключено" << std::endl;
        }
        if (p0_chid != -1) {
            ChannelDestroy(p0_chid);
            std::cout << "P0: Канал уничтожен" << std::endl;
        }
        // уничтожение таймеров
        timer_delete(p0_periodicTimer);
        timer_delete(p0_stopTimer);
        std::cout << "P0: Таймеры удалены" << std::endl;
        // уничтожение именованной памяти
        shm_unlink(NAMED_MEMORY);
        std::cout << "P0: Именованная память уничтожена" << std::endl;
        std::cout << "P0: Завершение работы" << std::endl;
        exit(EXIT_SUCCESS);
    }
}
// функция обработки ошибок
void error(const char *msg) {
    perror(msg);
    exit(EXIT_FAILURE);
}
int main() {
    std::cout << "P0: Запуск" << std::endl;
    // установка обработчика сигнала завершения
    struct sigaction act;
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGUSR2);
    act.sa_flags = 0;
    act.sa_mask = set;
    act.sa_handler = &deadHandler;
    sigaction(SIGUSR2, &act, NULL);
```

```
    // создание именованной памяти
    namedMemoryPtr = createNamedMemory(NAMED_MEMORY);
    // инициализация структуры таймера
    namedMemoryPtr->timeInfo.tick = TICK;
    namedMemoryPtr->timeInfo.endTime = END_TIME;
    namedMemoryPtr->timeInfo.Time = -1; // -1 означает, что таймер не запущен
    // инициализация барьера синхронизации
    pthread_barrierattr_t barrierAttr;
    pthread_barrierattr_init(&barrierAttr);
    pthread_barrierattr_setpshared(&barrierAttr, PTHREAD_PROCESS_SHARED);
    if (pthread_barrier_init(&namedMemoryPtr->startBarrier, &barrierAttr, 3) != 0)
        error("P0: Ошибка инициализации барьера");
    // инициализация мьютекса
    pthread_mutexattr_init(&namedMemoryPtr->MutexAttr);
    pthread_mutexattr_setpshared(&namedMemoryPtr->MutexAttr,
PTHREAD_PROCESS_SHARED);
    pthread_mutex_init(&namedMemoryPtr->Mutex, &namedMemoryPtr->MutexAttr);
    // запуск дочерних процессов P1 и P2
    namedMemoryPtr->pidP1 = spawnl(P_NOWAIT, "/home/P1", "/home/P1", NULL);
    if (namedMemoryPtr->pidP1 < 0)
        error("P0: Ошибка запуска P1");
    namedMemoryPtr->pidP2 = spawnl(P_NOWAIT, "/home/P2", "/home/P2", NULL);
    if (namedMemoryPtr->pidP2 < 0)
        error("P0: Ошибка запуска P2");
    std::cout << "P0: Процессы P1 и P2 запущены" << std::endl;
    // создание канала для приема импульсов от таймера
    p0_chid = ChannelCreate(0);
    namedMemoryPtr->chidP1 = p0_chid; // сохранение chid для P1
    // настройка таймеров
    struct itimerspec periodicSpec, stopSpec;
    setPeriodicTimer(&p0_periodicTimer, &periodicSpec, p0_chid);
    setTimerStop(&p0_stopTimer, &stopSpec);
    // ожидание готовности всех процессов (синхронизация через барьер)
    pthread_barrier_wait(&namedMemoryPtr->startBarrier);
```

```
    // запуск таймеров
    timer_settime(p0_stopTimer, 0, &stopSpec, NULL);
    timer_settime(p0_periodicTimer, 0, &periodicSpec, NULL);
    // подключение к каналу P1 для отправки импульсов
    p0_coidP1 = ConnectAttach(0, namedMemoryPtr->pidP1, namedMemoryPtr->chidP1, _NTO_SIDE_CHANNEL, 0);
    if (p0_coidP1 < 0)
        error("P0: Ошибка подключения к P1");
    std::cout << "P0: Основной цикл начат" << std::endl;
    // основной цикл работы P0
    while (true) {
        // ожидание импульса от периодического таймера
        MsgReceivePulse(p0_chid, NULL, 0, NULL);
        // обновление времени ПРВ
        namedMemoryPtr->timeInfo.Time++;
        // уведомление дочерних процессов о наступлении тика
        kill(namedMemoryPtr->pidP2, TICK_SIGNAL_SIGUSR1); // P2 получает сигнал
        MsgSendPulse(p0_coidP1, 10, 1, namedMemoryPtr->timeInfo.Time); // P1 получает импульс
    }
    return EXIT_SUCCESS;
}

//Процесс P1
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <signal.h>
#include <sys/siginfo.h>
#include <sys/neutrino.h>
```

```
#include <pthread.h>
#include <errno.h>
#define NAMED_MEMORY_NAME "/namedMemory"
struct Clock {
    long tick;
    int Time;
    long endTime;
};
struct namedMemory {
    double p;
    pthread_barrier_t startBarrier;
    Clock timeInfo;
    int pidP1;
    int chidP1;
    int pidP2;
    pthread_mutexattr_t MutexAttr;
    pthread_mutex_t Mutex;
};
// вычисление параметра p
double calculateParameter(double t) {
    return log(t*t + 2*t + 10);
}
// глобальная переменная для chid
int p1_chid = -1;
// обработчик завершения
void deadHandler(int signo) {
    if (signo == SIGUSR2) {
        std::cout << "P1: Получен сигнал завершения" << std::endl;
        // освобождение ресурсов
        if (p1_chid != -1) {
            ChannelDestroy(p1_chid);
            std::cout << "P1: Канал уничтожен" << std::endl;
        }
```

```
        std::cout << "P1: Завершение работы" << std::endl;
        exit(EXIT_SUCCESS);
    }
}
void error(const char *msg) {
    perror(msg);
    exit(EXIT_FAILURE);
}
int main() {
    std::cout << "P1: Запуск" << std::endl;
    // установка обработчика сигнала
    struct sigaction act;//структура обработчика
    sigset_t set;//маска
    sigemptyset(&set);//очистка маски
    sigaddset(&set, SIGUSR2);//установка маски SIGUSR2
    act.sa_flags = 0;
    act.sa_mask = set;
    act.sa_handler = &deadHandler;//установка обработчика
    sigaction(SIGUSR2, &act, NULL);//для SIGUSR2
    // подключение к именованной памяти
    int fd = shm_open(NAMED_MEMORY_NAME, O_RDWR, 0666);
    if (fd == -1)
        error("P1: Ошибка открытия именованной памяти");
    struct namedMemory *namedMemoryPtr;
    namedMemoryPtr = (namedMemory*) mmap(NULL, sizeof(struct namedMemory),
                                PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (namedMemoryPtr == MAP_FAILED)
        error("P1: Ошибка отображения памяти");
    close(fd);
    // создание канала для импульсов
    p1_chid = ChannelCreate(0);
    namedMemoryPtr->chidP1 = p1_chid; // Сообщаем P0 наш chid
    std::cout << "P1: Канал создан, chid = " << p1_chid << std::endl;
```

```
    // ожидание синхронизации
    pthread_barrier_wait(&namedMemoryPtr->startBarrier);
    const double tickToSec = namedMemoryPtr->timeInfo.tick / 1e9;
    std::cout << "P1: Основной цикл начат" << std::endl;
    // основной цикл
    while (true) {
        // ожидание импульса
        MsgReceivePulse(p1_chid, NULL, 0, NULL);
        // вычисление времени в секундах
        double currentTime = namedMemoryPtr->timeInfo.Time * tickToSec;
        // обновление параметра
        pthread_mutex_lock(&namedMemoryPtr->Mutex);
        namedMemoryPtr->p = calculateParameter(currentTime);
        pthread_mutex_unlock(&namedMemoryPtr->Mutex);
        std::cout << "P1: Время " << currentTime << "c, параметр p = " << namedMemoryPtr->p
<< std::endl;
    }
    return EXIT_SUCCESS;
}

//Процесс P2
#include <iostream>
#include <fcntl.h>
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/siginfo.h>
#include <sys/neutrino.h>
#include <pthread.h>
#include <errno.h>
#define NAMED_MEMORY_NAME "/namedMemory"
```

```
#define P2_INTERVAL 3 // интервал записи каждые 3 тика
#define TICK_SIGUSR1 SIGUSR1
struct Clock {
    long tick;
    int Time;
    long endTime;
};
struct namedMemory {
    double p;
    pthread_barrier_t startBarrier;
    Clock timeInfo;
    int pidP1;
    int chidP1;
    int pidP2;
    pthread_mutexattr_t MutexAttr;
    pthread_mutex_t Mutex;
};
// глобальная переменная для файла
FILE *trendFile = NULL;
// обработчик завершения
void deadHandler(int signo) {
    if (signo == SIGUSR2) {
        std::cout << "P2: Получен сигнал завершения" << std::endl;
        // закрытие файла
        if (trendFile != NULL) {
            fclose(trendFile);
            trendFile = NULL;
            std::cout << "P2: Файл тренда закрыт" << std::endl;
        }
        std::cout << "P2: Завершение работы" << std::endl;
        exit(EXIT_SUCCESS);
    }
}
```

```
void error(const char *msg) {
    perror(msg);
    exit(EXIT_FAILURE);
}
int main() {
    std::cout << "P2: Запуск" << std::endl;
    // установка обработчиков сигналов
    struct sigaction act;
    sigset_t sigset;
    // обработчик SIGUSR2
    sigemptyset(&sigset);
    sigaddset(&sigset, SIGUSR2);
    act.sa_flags = 0;
    act.sa_mask = sigset;
    act.sa_handler = &deadHandler;
    sigaction(SIGUSR2, &act, NULL);
    // маска для SIGUSR1 (тики)
    sigset_t tickSet;
    sigemptyset(&tickSet);
    sigaddset(&tickSet, TICK_SIGUSR1);
    // подключение к именованной памяти
    int fd = shm_open(NAMED_MEMORY_NAME, O_RDWR, 0666);
    if (fd == -1)
        error("P2: Ошибка открытия именованной памяти");
    struct namedMemory *namedMemoryPtr;
    namedMemoryPtr = (namedMemory*) mmap(NULL, sizeof(struct namedMemory),
                                PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    if (namedMemoryPtr == MAP_FAILED)
        error("P2: Ошибка отображения памяти");
    close(fd);
    // открытие файла для записи тренда
    trendFile = fopen("/home/trend.txt", "w");
    if (!trendFile)
```

```
        error("P2: Ошибка открытия файла тренда");
    std::cout << "P2: Файл тренда открыт" << std::endl;
    // синхронизация
    pthread_barrier_wait(&namedMemoryPtr->startBarrier);
    const double tickToSec = namedMemoryPtr->timeInfo.tick / 1e9;
    std::cout << "P2: Основной цикл начат" << std::endl;
    // основной цикл
    while (true) {
        // ожидание сигнала тика
        if (sigwaitinfo(&tickSet, NULL) == TICK_SIGUSR1) {
            int currentTick = namedMemoryPtr->timeInfo.Time;
            // проверка интервала записи
            if (currentTick % P2_INTERVAL == 0) {
                double currentTime = currentTick * tickToSec;
                // запись в файл
                pthread_mutex_lock(&namedMemoryPtr->Mutex);
                fprintf(trendFile, "%.6f\t%.6f\n", namedMemoryPtr->p, currentTime);
                fflush(trendFile); // сброс буфера в файл
                pthread_mutex_unlock(&namedMemoryPtr->Mutex);
                std::cout << "P2: Записано p = " << namedMemoryPtr->p << " при t = " << currentTime
<< "c" << std::endl;
            }
        }
    }
    // fclose(trendFile);
    return EXIT_SUCCESS;
```

Problems | Target File System Navigator | Target Navigator | Console

<terminated> P0 [C/C++ QNX QConn (IP)] /home/P0 on Neutrino650QNX1 pid 835614

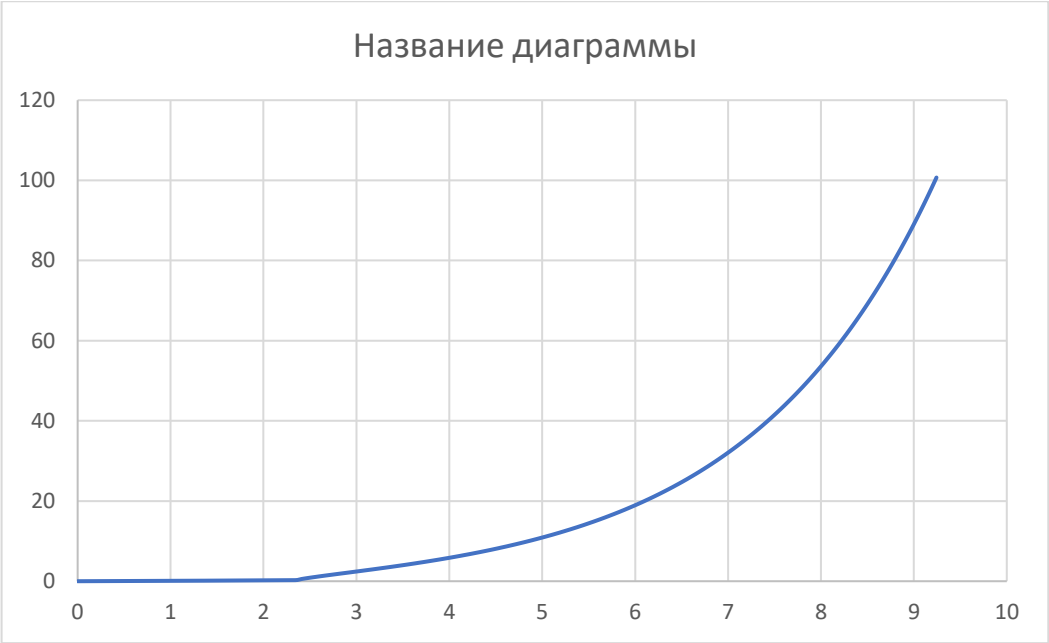
```
P0: Запуск
P0: Процессы P1 и P2 запущены
P1: Запуск
P1: Канал создан, chid = 1
P2: Запуск
P2: Файл тренда открыт
P2: Основной цикл начат
P0: Основной цикл начат
P1: Основной цикл начат
P1: Время 0c, параметр p = 2.30259
P2: Записано p = 0 при t = 0c
P1: Время 0.09c, параметр p = 2.32122
P1: Время 0.18c, параметр p = 2.34107
P2: Записано p = 2.34107 при t = 0.27c
P1: Время 0.27c, параметр p = 2.36207
P1: Время 0.36c, параметр p = 2.38413
P1: Время 0.45c, параметр p = 2.40717
P2: Записано p = 2.40717 при t = 0.54c
P1: Время 0.54c, параметр p = 2.43112
P1: Время 0.63c, параметр p = 2.4559
P1: Время 0.72c, параметр p = 2.48143
P2: Записано p = 2.48143 при t = 0.81c
P1: Время 0.81c, параметр p = 2.50765
P1: Время 0.9c, параметр p = 2.53449
P1: Время 0.99c, параметр p = 2.56188
P2: Записано p = 2.56188 при t = 1.08c
P1: Время 1.08c, параметр p = 2.58975
```

}

Problems | Target File System Navigator | Target Navigator | Console

<terminated> P0 [C/C++ QNX QConn (IP)] /home/P0 on Neutrino650QNX1 pid 835614

```
P1: Время 100.26c, параметр p = 9.23626
P1: Время 100.35c, параметр p = 9.23804
P2: Записано p = 9.23804 при t = 100.44c
P1: Время 100.44c, параметр p = 9.23981
P1: Время 100.53c, параметр p = 9.24158
P1: Время 100.62c, параметр p = 9.24335
P2: Записано p = 9.24335 при t = 100.71c
P1: Время 100.71c, параметр p = 9.24512
P1: Время 100.8c, параметр p = 9.24689
P1: Время 100.89c, параметр p = 9.24865
P0: Получен сигнал завершения
P0: Отправлен SIGUSR2 процессу P1
P1: Получен сигнал завершения
P0: Отправлен SIGUSR2 процессу P2
P1: Канал уничтожен
P2: Получен сигнал завершения
P1: Завершение работы
P2: Файл тренда закрыт
P2: Завершение работы
P0: Процесс P1 завершился
P0: Процесс P2 завершился
P0: Соединение с P1 отключено
P0: Канал уничтожен
P0: Таймеры удалены
P0: Именованная память уничтожена
P0: Завершение работы
```


Название диаграммы
120
100
80
60
40
20
0
0
1
2
3
4
5
6
7
8
9
10