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

ИНСТИТУТ ИНФОРМАТИКИ, МАТЕМАТИКИ И ЭЛЕКТРОНИКИ  
Факультет информатики  
Кафедра программных систем

Дисциплина

**Технологии промышленного программирования**

**ОТЧЁТ**  
по лабораторной работе  
**Работа с именованной памятью и**  
**службой реального времени**

Студент: В.А. Артамонов

Группа: № 6231-020402D

Преподаватель: Баландин А.В.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2022

# Задание

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 O p 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.

ПРВ, осуществляющее мониторинг, реализуется в программной системе в виде процесса Р2(М2), запускаемого на базе модуля М2:

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 - <p, t>. 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

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

Вариант 7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Порядок загрузки и запуска  программной системы | Вид функции *F***(***t***)** | Единица временной шкалы Δt (сек) | Единица временной шкалы Δt(сек) | Значение Т (сек) |
| Процесс Р2 запускается стартовым процессом Р0, который запускается «вручную». Процесс Р2 запускает процесс Р1. Именованную память создает процесс Р0. | a=1,5; b = 0,71 | 0.05,  уведомление  сигналом | 0.2,  уведомление  импульсом | 57 |

# Порядок выполнения приложения

Процесс P0 запускается в терминале целевой системы QNX, установленной на виртуальной машине. В этом процессе создается именованная память при помощи функции shm\_open(), устанавливается размер при помощи функции ftruncate() и отображается в адресное пространство процесса при помощи функции mmap().

После этого в процессе P0 в именованную память сохраняется информация о времени работы приложения: частота реагирования процессов P1 и P2, номер текущего тика, общее время работы приложения и длительность тика в наносекундах.

Затем создаем барьеры для синхронизации процессов, определенные в именованной памяти, в настройках устанавливаем свойство глобальности, чтобы барьеры работали и с нитями других процессов.

Вызываем процесс P2. Создаем канал процесса P0, таймер окончания работы приложения, таймер тика, который с заданной периодичностью будет отправлять сигнал процессу P1 и импульс процессу P2. После этого процесс P0 останавливается у барьера, ожидая остальные процессы.

Процесс P2 при запуске, устанавливает соединение с именованной памятью. После этого создается канал и в именованную память записывается id канала. Затем открывается файл для записи тренда. После этого вызывается процесс P1 и создается таймер завершения процесса. Затем процесс P2 останавливается у барьера.

При запуске, процесс P1 подключается к именованной памяти. После этого устанавливаются конфигурации приема сигнала от процесса P0 и создается таймер завершения процесса. Процесс P1 останавливается у барьера, ожидая остальные процессы.

В процессе P0, после преодоления 1го барьера, происходит подключение к каналу связи процесса P2. Процесс P0 останавливается у 2го барьера. Процессы P1 и P2 просто останавливаются у 2го барьера.

После преодоления 2го барьера, в процессах запускаются таймеры.

Процесс P0 в цикле в заданный интервал времени посылает сигнал процессу P1 и импульс процессу P2.

Процесс P1 в цикле принимает сигналы от процесса P0 и если номер текущего тика соответствует частоте срабатыванию процесса P1, то в процессе вычисляется значение параметра p, и записывается в именованную память.

Процесс P2 в цикле принимает импульсы от процесса P0 и если номер текущего тика соответствует частоте срабатыванию процесса P2, то в этом процессе считывается параметр p из именованной памяти и заносится в файл тренда с указанием текущего времени.

Для того, чтобы чтение и запись в именованную память совершалась корректно, в процессах P1 и P2 установлены барьеры.

Процессы завершают работу, когда срабатывает таймер окончания работы процессов. После срабатывания таймеров выполняются действия, прописанные в таймере в случае его срабатывания.

На рисунке ниже приведен график зависимости параметра P от времени.

Рисунок 1 – График зависимости P от времени

# Код программы

**Процесс P0**

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

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

**#include** <time.h>

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

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

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

**#include** <pthread.h>

// Имя именованной памяти

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

// Длительность работы приложения в секундах

**#define** END\_TIME 57 //время работы приложения в сек

// Длительность одного тика в наносекундах

**#define** TICK\_NANO\_DURATION 50000000// нс - длительность тика в наносекундах

// Номер сигнала наступления нового тика

**#define** TICK\_SIGNAL\_NUM SIGUSR1 //номер сигнала уведомления

// Структура данных с информацией о течении времени приложения

**struct** time\_info {

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

**int** currentTick; // Номер текущего тика

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

};

// Структура данных, хранящаяся в именованной памяти

**struct** namedMemory {

**double** p; // характеристика объекта

pthread\_barrier\_t configBarrier; // барьер завершения установки конфигураций процессов

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

pthread\_barrier\_t tickBarrier; // барьер тика

time\_info timeInfo; // информация о течении времени приложения

**int** pidP1; // ID процесса P1

**int** chidP2; // ID канала процесса P2

};

**struct** namedMemory \*namedMemoryPtr;//указатель именованной памяти

**int** P2TickCoid; //соединение с chidP2 для приёма импульсов тика от таймера

//Функции

**struct** namedMemory \***createNamedMemory**(**const** **char**\* name);//создать именов\_память

**void** **setPeriodicTimer**(timer\_t\* periodicTimer, **struct** itimerspec\* periodicTimerStruct, **int** chid);//таймер для импульса

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **int** chid);//таймер завершения ПРВ

**void** **deadHandler**(**int** signo);//обработчик сигналов от таймера

**void** **error**(**const** **char** \*msg);//сообщение об ошибке и завершение процесса

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

std::cout << "Процесс P0: стартовал" << std::endl;

// Присоединение именованной памяти

namedMemoryPtr = createNamedMemory(NAMED\_MEMORY\_NAME);

// Установка параметров времени приложения

namedMemoryPtr->timeInfo.tickNanoDuration = TICK\_NANO\_DURATION;//величина тика

namedMemoryPtr->timeInfo.endTime = END\_TIME;//момент окончания приложения в сек

namedMemoryPtr->timeInfo.currentTick = -1;//показание часов ПРВ в тиках, -1 - часы не запущены

// Барьер контроля окончания запуска процессов ПРВ

pthread\_barrierattr\_t configAttr;

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

**pthread\_barrierattr\_setpshared**(&configAttr, PTHREAD\_PROCESS\_SHARED);//разделяемый барьер

**pthread\_barrier\_init**(&(namedMemoryPtr->configBarrier), &configAttr, 3);//3 места

// Создаём барьер для синхронизации старта таймеров в процессах

pthread\_barrierattr\_t startAttr;

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

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

**pthread\_barrier\_init**(&(namedMemoryPtr->startBarrier), &startAttr, 3);//3 места

// Создаём барьер для синхронизации процессов P1 и P2 во время 0-го тика

pthread\_barrierattr\_t tickAttr;

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

**pthread\_barrierattr\_setpshared**(&tickAttr, PTHREAD\_PROCESS\_SHARED);//разделяемый барьер

**pthread\_barrier\_init**(&(namedMemoryPtr->tickBarrier), &tickAttr, 2);//2 места

/\* %%%%%%%%%%%%%%%%%% запуск процесса P2 регистрации тренда данных %%%%%%%%%%%%%%%%%\*/

std::cout << "Р0: запуск процесса P2" << std::endl;

**int** pidP2 = **spawnl**(P\_NOWAIT, "/home/LR3\_P2","/home/LR3\_P2", NULL );

**if** (pidP2<0){

std::cout << "Р0: ошибка запуска процесса P2, Р0 завершён" << std::endl;

exit(EXIT\_FAILURE);

}

// std::cout << "Р0: процесс P2 запущен" << std::endl;

// Создать канал

**int** chid = **ChannelCreate**(0);//канал для создания соединения для отправки уведомлений таймера импульсом

// Создание периодического таймера тиков часов ПРВ

timer\_t periodicTimer;

**struct** itimerspec periodicTimerStruct;//интервал срабатывания относительного таймера

setPeriodicTimer(&periodicTimer, &periodicTimerStruct, chid);//инициация работы таймера

// создание однократного таймера завершения работы процесса

timer\_t stopTimer;

**struct** itimerspec stopPeriod;

setTimerStop(&stopTimer, &stopPeriod,chid);//уведомление импульсом

// Ожидаем готовности остальных процессов

std::cout << "P0: подходит к config barrier wait" << std::endl;

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

std::cout << "P0 прошёл барьер config barrier wait" << std::endl;

// Подключение к каналу процесса P2

P2TickCoid = **ConnectAttach**(0, namedMemoryPtr->pidP1, namedMemoryPtr->chidP2, 0, 0);

// Ожидаем готовности к запуску таймеров остальных процессов

std::cout << "P0: pthread\_barrier\_wait - подходит к барьеру startBarrier" << std::endl;

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

std::cout << "P0: прошёл барьер startBarrier" << std::endl;

// Запускаем таймеры

**timer\_settime**(stopTimer, 0, &stopPeriod, NULL);//запуск однократного таймера окончания работы приложения

**timer\_settime**(periodicTimer, 0, &periodicTimerStruct, NULL);//запуск периодического таймера тика часов ПРВ

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

**MsgReceivePulse**(chid, NULL, 0, NULL);//ждём импульс истечения тика часов ПРВ

namedMemoryPtr->timeInfo.currentTick++; //Увеличиваем номер текущего тика

**kill**(pidP2, TICK\_SIGNAL\_NUM); //Отправляем сигнал SIGUSR1 процессу P2

**MsgSendPulse**(P2TickCoid, 10, 10, 10); //Отправляем импульс процессу P2: приоритет - 10, код - 10, значение - 10

}

**return** EXIT\_SUCCESS;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ФУНКЦИИ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Функция подключения к именованной памяти

**struct** namedMemory\* **createNamedMemory**(**const** **char**\* name) {

**struct** namedMemory \*namedMemoryPtr;

**int** fd; //дескриптор именованной памяти

// Создание именованной памяти

std::cout << "P0: shm\_open - открытие именованной памяти" << std::endl;

**if** ((fd = **shm\_open**(name, O\_RDWR | O\_CREAT, 0777)) == -1)

error("P0: Ошибка shm\_open");

// Установка размера именованной памяти

**if** (**ftruncate**(fd, 0) == -1 || **ftruncate**(fd, **sizeof**(**struct** namedMemory)) == -1)

error("P0: Ошибка ftruncate");

//Отображение разделяемой именованной памяти в адресное пространство процесса

**if** ((namedMemoryPtr = (namedMemory\*) **mmap**(NULL, **sizeof**(**struct** namedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0)) == MAP\_FAILED)

error("P0: Ошибка mmap");

std::cout << "P0: shm\_open - именованная память открыта и присоединена" << std::endl;

**return** namedMemoryPtr;

}

// Функция запуска таймера для завершения работы приложения по сигналу SIGUSR2

**void** **setTimerStop**(timer\_t \*stopTimer, **struct** itimerspec \*stopPeriod, **int** chid){

**struct** sigevent event;

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

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

std::cout << "P0:setTimerStop - установка однократного таймера посылки сигнала завершения приложения" << std::endl;

// установить время срабатывания однократного таймера

stopPeriod->it\_value.tv\_sec = END\_TIME;//момент времени завершения приложения

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

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

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

// Установка обработчика сигнала SIGUSR2 закрытия приложения

**struct** sigaction act;//структура установки обработчика

sigset\_t set;//набор маски сигналов

**sigemptyset**(&set);//очистить маску сигналов

**sigaddset**(&set, SIGUSR2);//установить маску сигнала SIGUSR2

act.sa\_flags = 0;

act.sa\_mask = set;

act.\_\_sa\_un.\_sa\_handler = &deadHandler;//установка обработчика

**sigaction**(SIGUSR2, &act, NULL);//установить обработчик SIGUSR2

std::cout << "P0:setTimerStop - установлен однократный таймер уведомления о завершении приложения" << std::endl;

}

// Функция создания таймера тика с уведомлением импульсом

**void** **setPeriodicTimer**(timer\_t\* periodicTimer, **struct** itimerspec\* periodicTimerStruct, **int** chid){

**struct** sigevent event;

**int** coid = **ConnectAttach**(0, 0,chid, 0, \_NTO\_COF\_CLOEXEC);//соединение для импульсов уведомления

**if**(coid ==-1)error("P0: функция setPeriodicTimer-ConnectAttach");

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

**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\_NANO\_DURATION;

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

periodicTimerStruct->it\_interval.tv\_nsec = TICK\_NANO\_DURATION;

}

// Обработчик сигнала уведомления о завершении приложения

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

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

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

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

**ConnectDetach**(P2TickCoid);

std::cout << "P0: deadHandler-обработчик завершил процесс" << std::endl;

exit(1);

}

}

// Вывод ошибки и завершение работы процесса

**void** **error**(**const** **char** \*msg) {

perror(msg);

exit(EXIT\_FAILURE);

}

**Процесс P1**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Процесс P1\*\*\*\*\*\*\*\*\*\*/

**#include** <iostream>

**#include** <fcntl.h>

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

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

**#include** <time.h>

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

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

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

**#include** <pthread.h>

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

// Имя именованной памяти

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

// Частота срабатывания процесса P1 в тиках

**#define** P1\_FREQUENCY 5

// Структура данных с информацией о течении времени приложения

**struct** time\_info {

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

**int** currentTick; // Номер текущего тика

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

};

// Структура данных, хранящаяся в именованной памяти.

**struct** namedMemory {

**double** p; // характеристика объекта

pthread\_barrier\_t configBarrier; // барьер завершения установки конфигураций процессов

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

pthread\_barrier\_t tickBarrier; // барьер тика

time\_info timeInfo; // информация о течении времени приложения

**int** pidP1; // ID процесса P1

**int** chidP2; // ID канала процесса P2

};

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

**void** **setTimerStop**(timer\_t\* deadTimer, **struct** itimerspec\* deadPeriod, **long** endTime);

**double** **F**(**double** t);

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

**void** **error**(**const** **char** \*msg);

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

std::cout << "P1: процесс запущен" << std::endl;

// Подключение к именованной памяти

**struct** namedMemory \*namedMemoryPtr = connectToNamedMemory(NAMED\_MEMORY\_NAME);

//создать канал

**int** chid = **ChannelCreate**(0);

namedMemoryPtr->chidP2 = chid;

// Создание таймера завершения работы процесса

timer\_t stopTimer;

**struct** itimerspec stopPeriod;

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

// Ожидаем готовности остальных процессов

std::cout << "P1: процесс подошёл к барьеру configBarrier" << std::endl;

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

std::cout << "P1: процесс прошёл барьер configBarrier" << std::endl;

// Ожидаем готовности к запуску таймеров остальных процессов

std::cout << "P1: подошёл к барьеру startBarrier" << std::endl;

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

std::cout << "P1: процесс прошёл барьер startBarrier" << std::endl;

**timer\_settime**(stopTimer, 0, &stopPeriod, NULL);//запуск таймера

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

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

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

**if**(namedMemoryPtr->timeInfo.currentTick % P1\_FREQUENCY == 0){

**const** **int** currentTick = namedMemoryPtr->timeInfo.currentTick;

**const** **double** tickDuration = (namedMemoryPtr->timeInfo.tickNanoDuration / (1000. \* 1000. \* 1000.));

**const** **double** t = currentTick \* tickDuration;//перевод тиков в системное время

namedMemoryPtr->p = F(t);//значение параметра в момент системного времени

std::cout << "P1: параметр p: " << namedMemoryPtr->p << " t: " << t << std::endl;

}

// Ожидаем прихода сигнала для выхода из бесконечного цикла и завершения процесса

}

**return** EXIT\_SUCCESS;

}//конец main

// Производит подключение к именованной памяти.

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

**struct** namedMemory \*namedMemoryPtr;

**int** fd;

// Открыть именованную память

**if** ((fd = **shm\_open**(name, O\_RDWR, 0777)) == -1)

error("P1:ошибка shm\_open, процесс завершён!");

//Отображение разделяемой именованной памяти в адресное пространство процесса

**if** ((namedMemoryPtr = (namedMemory\*) **mmap**(NULL, **sizeof**(**struct** namedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0)) == MAP\_FAILED){

error("P1: ошибка mmap, процесс завершён!");

}

**return** namedMemoryPtr;

}

//Изменение параметра p от t

**double** **F**(**double** t) {

**const** **double** a = 1.5;

**const** **double** b = 4.8;

**double** p = a\*(b/t-(log(a\*t))/(b\*b));

**return** p;}

// Настраивает таймер для уведомления о необходимости завершения работы приложения.

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

**struct** sigevent event;

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

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

// установить время срабатывания таймера

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

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

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

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

// Добавляем обработчик для корректного закрытия всех используемых ресурсов

**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);

}

// Закрывает все используемые ресурсы при завершении работы приложения

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

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

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

exit(1);

}

}

// Вывод ошибки и завершение работы

**void** **error**(**const** **char** \*msg) {

perror(msg);

exit(EXIT\_FAILURE);

}

**Процесс P2**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Процесс LR3\_P2\*\*\*\*\*\*\*\*\*\*/

**#include** <iostream>

**#include** <fcntl.h>

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

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

**#include** <time.h>

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

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

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

**#include** <pthread.h>

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

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

// Имя именованной памяти

**#define** NAMED\_MEMORY\_NAME "/namedMemory" //имя именованной памяти

// Частота срабатывания процесса P2 в тиках

**#define** P2\_FREQUENCY 20 //интервал репрезентативности параметра в тиках часов ПРВ

// Номер сигнала наступления нового тика.

**#define** TICK\_SIGNAL\_NUM SIGUSR1 //номер сигнала уведомления

// Структура данных с информацией о течении времени приложения

**struct** time\_info {

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

**int** currentTick; // Номер текущего тика

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

};

// Структура данных, хранящаяся в именованной памяти

**struct** namedMemory {

**double** p; // характеристика объекта

pthread\_barrier\_t configBarrier; // барьер завершения установки конфигураций процессов

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

pthread\_barrier\_t tickBarrier; // барьер тика

time\_info timeInfo; // информация о течении времени приложения

**int** pidP1; // ID процесса P1

**int** chidP2; // ID канала процесса P2

};

**struct** namedMemory \***connectToNamedMemory**(**const** **char**\* name);//присоединение именованной памяти

**void** **setTimerStop**(timer\_t\* stopTimer, **struct** itimerspec\* stopPeriod, **long** endTime);//однократный таймер завершения ПРВ

**void** **deadHandler**(**int** signo);//обработчик сигнала завершения работы

**void** **error**(**const** **char** \*msg);

FILE \*trendFile;// Файл с трендом параметра p

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

std::cout << "Р2: процесс запущен" << std::endl;

// подключение к именованной памяти

**struct** namedMemory \*namedMemoryPtr = connectToNamedMemory(NAMED\_MEMORY\_NAME);

// открываем файл для записи тренда

**if**((trendFile = fopen("/home/trend.txt", "w"))==NULL){

error("P2:Ошибка открытия файла для запси тренда, Р2 завершён! ");

}

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

// Конфигурация приёма сигнала

sigset\_t set;

**sigemptyset**(&set);

**sigaddset**(&set, TICK\_SIGNAL\_NUM);

//Запуск дочернего процесса Р1

**int** pidP1;//id процесса P1

pidP1 = **spawnl**( P\_NOWAIT, "/home/LR3\_P1","/home/LR3\_P1", NULL );

**if** (pidP1<0){

error("Р2: ошибка запуска процесса P1, Р2 завершён! ");

}

namedMemoryPtr->pidP1 = pidP1;

std::cout << "P2: процесс Р1 запущен" << std::endl;

// создание таймера завершения работы процесса

timer\_t stopTimer;

**struct** itimerspec stopPeriod;

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

// Ожидаем готовности остальных процессов

std::cout << "P2: процесс подошёл к барьеру configbarrier" << std::endl;

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

std::cout << "P2: процесс прошёл барьер configBarrier" << std::endl;

// Ожидаем готовности к запуску таймеров остальных процессов

std::cout << "P2: процесс подошёл к барьеру startBarrier" << std::endl;

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

std::cout << "P2: процесс прошёл барьер startBarrier" << std::endl;

**timer\_settime**(stopTimer, 0, &stopPeriod, NULL);//запуск таймера

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

**if** (**SignalWaitinfo**(&set, NULL) == TICK\_SIGNAL\_NUM) {

**if** (namedMemoryPtr->timeInfo.currentTick % P2\_FREQUENCY == 0){

**const** **int** currentTick = namedMemoryPtr->timeInfo.currentTick;

**const** **double** tickDuration = (namedMemoryPtr->timeInfo.tickNanoDuration / (1000. \* 1000. \* 1000.));

**const** **double** t = currentTick \* tickDuration;

fprintf(trendFile, "%f\t%f\n", namedMemoryPtr->p, t);

std::cout << "P2: параметр p: " << namedMemoryPtr->p << " t: " << t << std::endl;

}

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

}

//выход из бесконечного цикла в обработчике сигнала

}

**return** EXIT\_SUCCESS;

}//конец main

// Функция присоединения к процессу именованной памяти

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

**struct** namedMemory \*namedMemoryPtr;

**int** fd;

// Открыть именованную память

**if** ((fd = **shm\_open**(name, O\_RDWR, 0777)) == -1)

error("P2:ошибка shm\_open, Р2 завершён!");

//Отображение разделяемой именованной памяти в адресное пространство процесса

**if** ((namedMemoryPtr = (namedMemory\*) **mmap**(NULL, **sizeof**(**struct** namedMemory), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0)) == MAP\_FAILED)

error("P2:ошибка mmap, Р2 завершён!");

**return** namedMemoryPtr;

}

// Настраивает таймер для уведомления о необходимости завершения работы приложения.

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

**struct** sigevent event;

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

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

// установить время срабатывания таймера

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

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

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

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

// Добавляем обработчик для корректного закрытия всех используемых ресурсов.

**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);

}

// Закрывает все используемые ресурсы при завершении работы приложения.

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

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

fclose(trendFile);

std::cout << "P2: процесс завершён сигналом" << std::endl;

exit(1);

}

}

// Вывод ошибки и завершение работы

**void** **error**(**const** **char** \*msg) {

perror(msg);

exit(EXIT\_FAILURE);

}