Федеральное агентство по образованию

Государственное образовательное учреждение

высшего профессионального образования

**САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. академика С.П. КОРОЛЕВА**

**(национальный исследовательский университет)**

**Факультет информатики**

Отчёт к лабораторной работе

по дисциплине «Параллельные вычислительные системы»

Тема:

**«Использование именованной памяти. Синхронизация нитей с реальным временем»**

**Вариант №9**

Выполнил Проценко В.И.

Группа М618

Преподаватель Баландин А.В.

Самара 2012

Оглавление

[Цель и постановка задания 3](#_Toc338541431)

[Описание программы 5](#_Toc338541432)

[Результат работы программы 6](#_Toc338541433)

[Приложения 8](#_Toc338541434)

[Исходный код 8](#_Toc338541435)

[P0.c 8](#_Toc338541436)

[P1.c 11](#_Toc338541437)

[P2.c 15](#_Toc338541438)

[results.txt 18](#_Toc338541439)

# Цель и постановка задания

**Тема лабораторной работы**: использование именованной памяти, синхронизация нитей с реальным временем.

**Цель задания**: освоение функций ОСРВ для синхронизации нитей с реальным временем.

**Задание**:

Требуется разработать программную систему мониторинга абстрактного физического объекта Op(t), p(t) – изменяющийся во времени параметр. Мониторинг объекта Op(t) осуществляется на относительном интервале времени [0,Т] с заданной единицей шкалы времени - 1t.

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

Программная система должна содержать два процесса Р1 и Р2, запускаемых на базе своих программных модулей (порядок запуска процессов определяется в варианте задания). При необходимости в программной системе могут запускаться дополнительные процессы (вспомогательные).

Процесс Р1, начиная с t=0, моделирует изменяющийся во времени параметр p(t) периодической записью в именованную память вычисленного значения pi, где pi=F(1t•i), i – порядковый номер очередного вычисленного значения параметра p(t) в момент интервального времени t=1t•i ∈[0,Т], i=0,1,2, ….

Процесс Р2, начиная с момента времени t=0, периодически считывает параметр p(t) с заданным периодом Δt и формирует датированное значение в виде пары - <p(t),t>, где t=j•Δt), j – порядковый номер считывания очередного значения параметра p(t) на интервале времени [0,Т], j=0,1,2, …. Результаты периодического считывания значений параметра- <p(t),t> (тренд p(t)), заносятся процессом Р2 в текстовый файл, в котором значения в паре <p(t),t> разделены пробелом, а каждая пара завершается управляющим символом \n.

Процессы Р1 и Р2 должны быть синхронизированы по моменту времени t=0 (должны начать действовать "одновременно", метод синхронизации выбрать самостоятельно).

При t>Т работа программной системы должна завершиться (все процессы терминироваться).

Для тренда параметра p(t) построить график, например, загрузив содержимое файла с трендом в MS EXCEL.

**Форма отчетности**: для отчета по лабораторной работе требуется представить следующее:

1. Демонстрация работы программной системы в соответствии с заданием.
2. Проверка теоретических знаний по средствам запуска процессов (нитей) и синхронизации нитей с реальным временем.
3. Представление оформленного отчета, по результатам выполнения лабораторной работы, с графиком тренда.

**ВАРИАНТ №9**

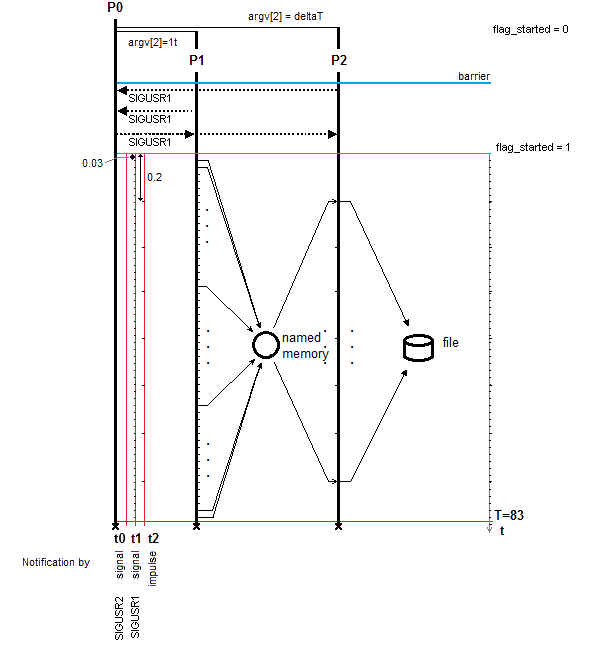
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Порядок запуска процессов Р1 и Р2 | Вид функции **F(t)** | Единица временной шкалы 1t(сек) | Единица временной шкалы Δt (сек) | Значение Т (сек) |
| Процесс Р1 программно запускается стартовым процессом Р0, запускаемым из командной строки shell. Процесс Р1 программно запускает процесс Р2 | FP2= | 0.03, уведомление сигналом | 0.2, уведомление импульсом | 83 |

# Описание программы

Для синхронизации обращения к именованной памяти использовался мьютекс. Мьютекс доступен нескольким процессам через именованную память. Имена именованной памяти передаются как аргументы командной строки создаваемым процессам P1, P2.

Именованная память для значения f(t), а также для мьютекс инициализируется процессом P0. Для синхронизации старта используется обмен сигналами SIGUSR1. Для остановки вычислений в P1, P2 использовался сигнал SIGUSR2.

Процесс P0 ожидает завершения процессов P1, P2, после чего очищает ресурсы и терминируется.



# Результат работы программы

# F:\Reserv\!!!!!!!!!!!!!!!!!!!!\Параллельные_вычислительные_системы\Лабораторные\ASPVS_Lab_3_Realtime\documentation\snapshot.png

Рисунок Работа программы

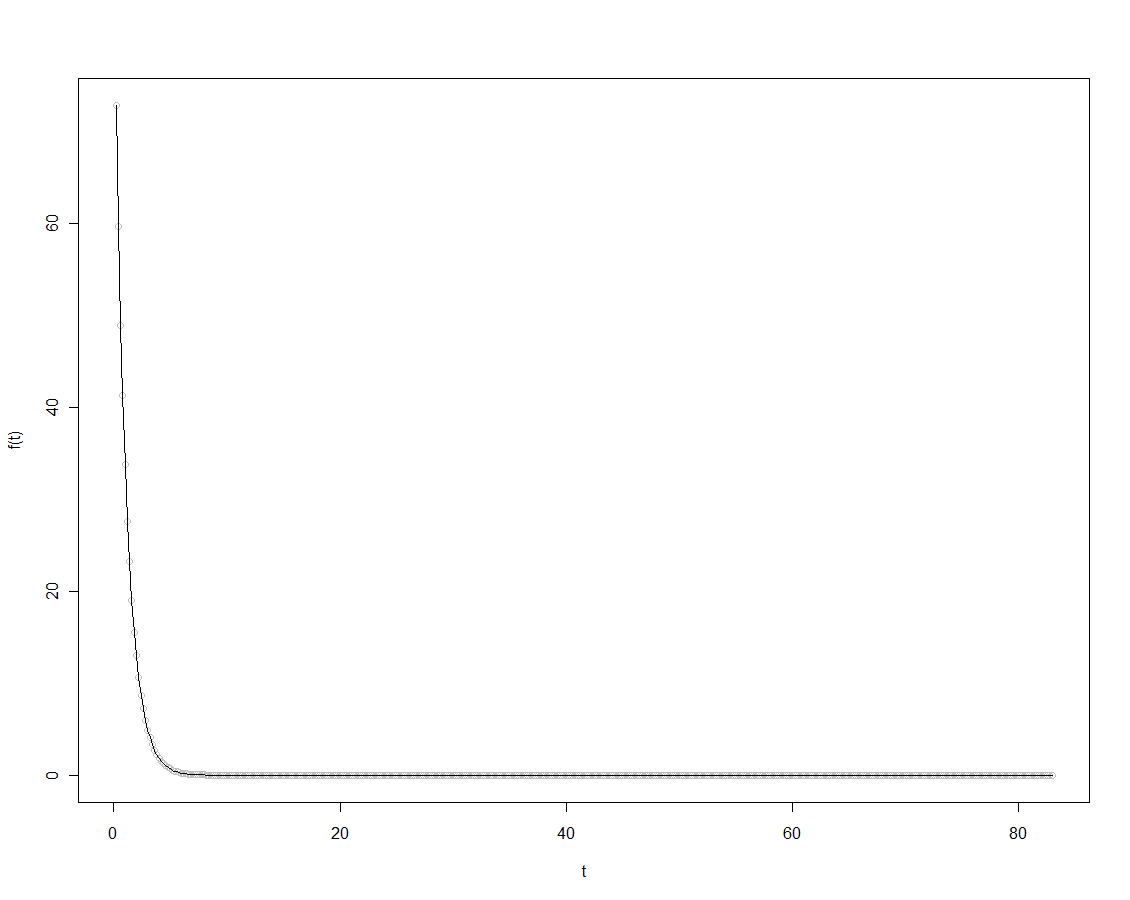


Рисунок График тренда

# Приложения

### Исходный код

### P0.c

#include <stdlib.h>

#include <stdio.h>

#include <pthread.h>

#include <errno.h>

#include <sys/mman.h>

#include <sys/neutrino.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#include <time.h>

#define T 83

#define dt 0.03

#define deltaT 0.2

#define shared\_mem\_name "/memshared"

#define shared\_mutex\_name "/mutexshared"

const char \*PROCNAME = "P0 process";

int bthreads, bthreads\_max;

int flag\_started = 0;

int flag\_terminating = 0;

timer\_t timerid;

struct sigevent event;

struct itimerspec timer;

static void sig\_hndlr(int signo) {

if (signo == SIGUSR1) {

printf("P0 Got SIGUSR1\n");

bthreads++;

if ((!flag\_started) && (bthreads >= bthreads\_max)) {

kill(0, SIGUSR1); //broadcasting start signal

printf("P0 Starting timer. Timer id: %i, Timer interval (sec): %i\n", timerid, timer.it\_value.tv\_sec);

timer\_settime(timerid, 0, &timer, NULL);

flag\_started = 1;

}

}

if (signo == SIGUSR2) {

printf("P0 got SIGUSR2\n");

if (!flag\_terminating) {

kill(0, SIGUSR2);

flag\_terminating = 1;

}

}

}

// Tricky var initialization.

void shared\_mem\_init(const char \*name, double \*\*var, int \*fd) {

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

if(\*fd == -1) {

fprintf(stderr, "%s: Error attaching shared memory '%s': %s\n",

PROCNAME, name, strerror(errno));

exit(EXIT\_FAILURE);

}

int status = ftruncate(\*fd, sizeof(double));

if ( status == -1 ) {

fprintf(stderr, "%s: Error truncating shared memory '%s': status: %i\n%s\n",

PROCNAME, name, status,strerror(errno));

exit(EXIT\_FAILURE);

}

\*var = mmap(0, sizeof(double), PROT\_READ | PROT\_WRITE, MAP\_SHARED, \*fd, 0);

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

fprintf(stderr, "%s: Error shared mem maping. %s\n", PROCNAME, strerror(errno));

exit(EXIT\_FAILURE);

}

\*\*var = 1.0;

close(\*fd);

}

// Tricky var initialization.

void shared\_mutex\_init(const char \*name, pthread\_mutex\_t \*\*mutex, int \*fd) {

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

if(\*fd == -1) {

fprintf(stderr, "%s: Error attaching shared memory '%s': %s\n",

PROCNAME, name, strerror(errno));

exit(EXIT\_FAILURE);

}

int status = ftruncate(\*fd, sizeof(double));

if ( status == -1 ) {

fprintf(stderr, "%s: Error truncating shared memory '%s': status: %i\n%s\n",

PROCNAME, name, status,strerror(errno));

exit(EXIT\_FAILURE);

}

\*mutex = mmap(0, sizeof(pthread\_mutex\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, \*fd, 0);

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

fprintf(stderr, "%s: Error shared mem maping. %s\n", PROCNAME, strerror(errno));

exit(EXIT\_FAILURE);

}

close(\*fd);

}

void timer\_signal\_init(int time) {

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

// Creating timer

if (timer\_create(CLOCK\_REALTIME, &event, &timerid) == -1) {

fprintf(stderr, "%s: timer creation error", PROCNAME);

perror(NULL);

exit(EXIT\_FAILURE);

}

timer.it\_value.tv\_sec = time;

timer.it\_value.tv\_nsec = 0;

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_nsec = 0;

}

void my\_barrier() {

bthreads++;

while (!flag\_started) {

pause();

}

}

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

printf("P0 started. Gid = %i\n", getgid());

int pid = getpid();

printf("P0 flag\_started ptr = %i, val = %i", &flag\_started, flag\_started);

// Initializing barrier

bthreads = 0;

bthreads\_max = 3;

signal(SIGUSR1, sig\_hndlr);

signal(SIGUSR2, sig\_hndlr);

// Initializing shared memory. To return from init\_function pointer on mapped area

// we should use pointer on pointer.

double \*\*shared\_mem = (double\*\*)malloc(sizeof(double));

int \*fd\_shared\_mem = (int\*)malloc(sizeof(int));

shared\_mem\_init(shared\_mem\_name, shared\_mem, fd\_shared\_mem);

// Initializing shared mutex

pthread\_mutex\_t \*\*shared\_mutex = (pthread\_mutex\_t\*\*)malloc(sizeof(pthread\_mutex\_t));

int \*fd\_shared\_mutex = (int\*)malloc(sizeof(int));

shared\_mutex\_init(shared\_mutex\_name, shared\_mutex, fd\_shared\_mutex);

// Initializing arguments for P1 and starting it.

char \*argv0 = (char\*)malloc(sizeof(float));

char \*argv1 = (char\*)malloc(sizeof(float));

char \*argv2 = (char\*)malloc(strlen(shared\_mem\_name) + 1);

char \*argv3 = (char\*)malloc(strlen(shared\_mutex\_name) + 1);

sprintf(argv0, "%i", pid);

sprintf(argv1, "%f", dt);

sprintf(argv2, "%s", shared\_mem\_name);

sprintf(argv3, "%s", shared\_mutex\_name);

int pid\_1 = spawnl(P\_NOWAIT, "/tmp/P1", argv0, argv1, argv2, argv3, NULL);

// Initializing arguments for P2 and starting it.

sprintf(argv0, "%i", pid);

sprintf(argv1, "%f", deltaT);

sprintf(argv2, "%s", shared\_mem\_name);

sprintf(argv3, "%s", shared\_mutex\_name);

int pid\_2 = spawnl(P\_NOWAIT, "/tmp/P2", argv0, argv1, argv2, argv3, NULL);

// Initializing one-shot timer for P1, P2 termination. Notification by signal - SIGUSR2.

timer\_signal\_init(T);

printf("P0 at barrier\n");

my\_barrier();

printf("P0 after barrier\n");

// Separating part of process, where user signals could interrupt waitpid().

while(!flag\_terminating) {

pause();

}

// Waiting for child processes exit.

int st1 = waitpid(pid\_1, NULL, WEXITED);

printf("P0 :P1 termination status %i, errno = %i\n", st1, errno);

int st2 = waitpid(pid\_2, NULL, WEXITED);

printf("P0 :P2 termination status %i, errno = %i\n", st2, errno);

printf("P0 Last value of shared\_mem = %f\n", \*\*shared\_mem);

// Closing shared memory

close(\*fd\_shared\_mem);

shm\_unlink(shared\_mem\_name); // in some examples memory is unlinked just after getting fd value.

printf("Huston, P0 is shutting down.\n");

return EXIT\_SUCCESS;

}

### P1.c

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <sys/siginfo.h>

#include <sys/neutrino.h>

#include <pthread.h>

#include <sys/stat.h>

#include <sys/neutrino.h>

#include <sys/mman.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <math.h>

#include <signal.h>

const char \*PROCNAME = "P1 process";

timer\_t timerid;

struct sigevent event;

struct itimerspec timer;

int \*fd\_shared\_mem;

double \*\*shared\_mem;

int \*fd\_shared\_mutex;

pthread\_mutex\_t \*\*shared\_mutex;

int flag\_started = 0;

double t, dt;

// Tricky var initialization.

void shared\_mem\_open(const char \*name, double \*\*var, int \*fd) {

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

if(\*fd == -1) {

fprintf(stderr, "%s: Error attaching shared memory '%s': %s\n",

PROCNAME, name, strerror(errno));

exit(EXIT\_FAILURE);

}

\*var = mmap(0, sizeof(double), PROT\_READ | PROT\_WRITE, MAP\_SHARED, \*fd, 0);

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

fprintf(stderr, "%s: Error shared mem maping. %s\n", PROCNAME, strerror(errno));

exit(EXIT\_FAILURE);

}

close(\*fd);

}

void shared\_mutex\_open(const char \*name, double \*\*mutex, int \*fd) {

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

if(\*fd == -1) {

fprintf(stderr, "%s: Error attaching shared memory '%s': %s\n",

PROCNAME, name, strerror(errno));

exit(EXIT\_FAILURE);

}

\*mutex = mmap(0, sizeof(double), PROT\_READ | PROT\_WRITE, MAP\_SHARED, \*fd, 0);

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

fprintf(stderr, "%s: Error shared mem maping. %s\n", PROCNAME, strerror(errno));

exit(EXIT\_FAILURE);

}

close(\*fd);

}

void timer\_signal\_init(double dt) {

SIGEV\_SIGNAL\_INIT(&event, SIGUSR1);

// Creating timer

if (timer\_create(CLOCK\_REALTIME, &event, &timerid) == -1) {

fprintf(stderr, "%s: timer creation error", PROCNAME);

perror(NULL);

exit(EXIT\_FAILURE);

}

timer.it\_value.tv\_sec = 0;

timer.it\_value.tv\_nsec = dt \* 1000000000; // 0 in it\_value struct would not lunch timer

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_nsec = dt \* 1000000000;

}

double f(double t) {

return 100 \* exp(-t) \* cos(0.1\*t + 100);

}

void timer\_tick\_handle() {

t += dt;

pthread\_mutex\_lock(\*shared\_mutex);

\*\*shared\_mem = f(t);

pthread\_mutex\_unlock(\*shared\_mutex);

}

static void sig\_hndlr(int signo) {

if (signo == SIGUSR1) {

if (!flag\_started) {

printf(" P1 Starting timer. Timer id: %i, Timer interval: %i\n", timerid, timer.it\_interval.tv\_nsec);

int status = timer\_settime(timerid, 0, &timer, NULL);

flag\_started = 1;

} else

{

timer\_tick\_handle();

}

} else

if (signo == SIGUSR2) {

printf(" P1 on termination. SIGUSR2 received.\n");

if (flag\_started) {

close(\*fd\_shared\_mem);

exit(EXIT\_SUCCESS);

}

}

}

void my\_barrier(int ppid) {

// Sending to P0 signal - "we are ready".

kill(ppid, SIGUSR1);

// Waiting for SIGUSR1 to come. It will switch flag to started.

while(!flag\_started) {

pause();

}

}

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

printf(" P1 started. Gid = %i\n", getgid());

t = 0;

shared\_mem = (double\*\*)malloc(sizeof(double));

fd\_shared\_mem = (int\*)malloc(sizeof(int));

shared\_mutex = (pthread\_mutex\_t\*\*)malloc(sizeof(pthread\_mutex\_t));

fd\_shared\_mutex = (int\*)malloc(sizeof(int));

int ppid = atoi(argv[0]);

dt = atof(argv[1]);

char \*shared\_mem\_name = argv[2];

char \*shared\_mutex\_name = argv[3];

printf(" P1 ppid = %i, dt = %f, shared\_mem\_name = %s\n", ppid, dt, shared\_mem\_name);

// Initializing signal handlers.

// SIGUSR1 - sync signal. SIGUSR2 - terminating signal.

signal(SIGUSR1, sig\_hndlr);

signal(SIGUSR2, sig\_hndlr);

// Connecting to shared memory

shared\_mem\_open(shared\_mem\_name, shared\_mem, fd\_shared\_mem);

shared\_mutex\_open(shared\_mutex\_name, shared\_mutex, fd\_shared\_mutex);

// Initializing timer

timer\_signal\_init(dt);

printf(" P1 before barrier\n");

my\_barrier(ppid);

printf(" P1 after barrier\n");

// Main loop waiting for signals.

while(1) {

pause();

}

return EXIT\_SUCCESS;

}

### P2.c

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <sys/siginfo.h>

#include <sys/neutrino.h>

#include <pthread.h>

#include <sys/stat.h>

#include <sys/neutrino.h>

#include <sys/mman.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#include <math.h>

#include <signal.h>

#define CODE\_TIMER 1

#define RESULTS\_FILE\_FULL\_NAME "/tmp/results.txt"

const char \*PROCNAME = "P2 process.";

timer\_t timerid;

struct sigevent event;

struct itimerspec timer;

int \*fd\_shared\_mem;

double \*\*shared\_mem;

int \*fd\_shared\_mutex;

pthread\_mutex\_t \*\*shared\_mutex;

int flag\_started = 0;

double t, deltaT;

int chid;

FILE \*resultFile;

// Tricky var initialization. Read only.

void shared\_mem\_open(const char \*name, double \*\*var, int \*fd) {

printf(" P2 Opening shared memory: %s\n", name);

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

printf(" P2 Checking for errors after shm\_open.\n");

if(\*fd == -1) {

fprintf(stderr, "%s: Error attaching shared memory '%s': %s\n",

PROCNAME, name, strerror(errno));

exit(EXIT\_FAILURE);

}

printf(" P2 Mapping memory\n");

\*var = mmap(0, sizeof(double), PROT\_READ, MAP\_SHARED, \*fd, 0);

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

fprintf(stderr, "%s: Error shared mem maping. %s\n", PROCNAME, strerror(errno));

exit(EXIT\_FAILURE);

}

printf(" P2 Shared opened\n");

}

void shared\_mutex\_open(const char \*name, double \*\*mutex, int \*fd) {

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

if(\*fd == -1) {

fprintf(stderr, "%s: Error attaching shared memory '%s': %s\n",

PROCNAME, name, strerror(errno));

exit(EXIT\_FAILURE);

}

\*mutex = mmap(0, sizeof(double), PROT\_READ | PROT\_WRITE, MAP\_SHARED, \*fd, 0);

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

fprintf(stderr, "%s: Error shared mem maping. %s\n", PROCNAME, strerror(errno));

exit(EXIT\_FAILURE);

}

close(\*fd);

}

static void sig\_hndlr(int signo) {

if (signo == SIGUSR1) {

if (!flag\_started) {

printf(" P2 Starting timer. Timer id: %i, Timer interval: %i\n", timerid, timer.it\_interval.tv\_nsec);

timer\_settime(timerid, 0, &timer, NULL);

flag\_started = 1;

}

}

if (signo == SIGUSR2) {

printf(" P2 on termination. SIGUSR2 received.\n");

if (flag\_started) {

close(\*fd\_shared\_mem);

ChannelDestroy(chid);

close(resultFile);

exit(EXIT\_SUCCESS);

}

}

}

void timer\_pulse\_init(double dt) {

int coid;

if ((chid = ChannelCreate(0)) == -1) {

fprintf(stderr, "%s: can't create channel!\n", PROCNAME);

perror(NULL);

exit(EXIT\_FAILURE);

}

// Connecting to own channel

coid = ConnectAttach(0, 0, chid, \_NTO\_SIDE\_CHANNEL, 0);

if (coid == -1) {

fprintf(stderr, "%s: connection error", PROCNAME);

perror(NULL);

exit(EXIT\_FAILURE);

}

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

// Creating timer

if (timer\_create(CLOCK\_REALTIME, &event, &timerid) == -1) {

fprintf(stderr, "%s: timer creation error", PROCNAME);

perror(NULL);

exit(EXIT\_FAILURE);

}

timer.it\_value.tv\_sec = 0;

timer.it\_value.tv\_nsec = dt \* 1000000000; // 0 in it\_value struct would not lunch timer

timer.it\_interval.tv\_sec = 0;

timer.it\_interval.tv\_nsec = dt \* 1000000000;

}

void timer\_tick\_handle() {

static double temp\_value = 0;

t += deltaT;

pthread\_mutex\_lock(\*shared\_mutex);

temp\_value = \*\*shared\_mem;

pthread\_mutex\_unlock(\*shared\_mutex);

fprintf(resultFile, "%f %f\n", t, temp\_value);

}

void my\_barrier(int ppid) {

// Sending to P0 signal - "we are ready".

kill(ppid, SIGUSR1);

// Waiting for SIGUSR1 to come. It will switch flag to started.

while(!flag\_started) {

pause();

}

}

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

printf(" P2 started. Gid = %i\n", getgid());

t = 0;

shared\_mem = (double\*\*)malloc(sizeof(double));

fd\_shared\_mem = (int\*)malloc(sizeof(int));

shared\_mutex = (pthread\_mutex\_t\*\*)malloc(sizeof(pthread\_mutex\_t));

fd\_shared\_mutex = (int\*)malloc(sizeof(int));

int ppid = atoi(argv[0]);

deltaT = atof(argv[1]);

char \*shared\_mem\_name = argv[2];

char \*shared\_mutex\_name = argv[3];

printf(" P2 ppid = %i, dt = %f, shared\_mem\_name = %s\n", ppid, deltaT, shared\_mem\_name);

printf(" P2 Flag started ptr = %i, val = %i\n", &flag\_started, flag\_started);

// Initializing signals

signal(SIGUSR1, sig\_hndlr);

signal(SIGUSR2, sig\_hndlr);

// Conneting to shared memory

shared\_mem\_open(shared\_mem\_name, shared\_mem, fd\_shared\_mem);

shared\_mutex\_open(shared\_mutex\_name, shared\_mutex, fd\_shared\_mutex);

// Opening file to write trend

resultFile = fopen(RESULTS\_FILE\_FULL\_NAME, "w+");

// Initializing interval timer.

timer\_pulse\_init(deltaT);

printf(" P2 before barrier\n");

my\_barrier(ppid);

printf(" P2 after barrier\n");

// Main loop waiting for pulse messages.

while(1) {

MsgReceivePulse(chid, NULL, 0, NULL); // No buffer.

timer\_tick\_handle();

}

return EXIT\_SUCCESS;

}

### results.txt

t f(t)

0.200000 72.776531

0.400000 59.676227

0.600000 48.906299

0.800000 41.217737

1.0 33.744547

1.200000 27.611551

1.400000 23.240267

1.600000 18.998016

1.800000 15.522250

2.0 13.048612

2.200000 10.651479

2.400000 8.690485

2.600000 7.296853

2.800000 5.948180

3.0 4.846502

3.200000 4.064617

3.400000 3.308949

3.600000 2.692537

3.800000 2.255621

4.0 1.833886

4.200000 1.490335

4.400000 1.247127

4.600000 1.012658

4.800000 0.821906

5.0 0.687033

5.200000 0.557161

5.400000 0.451640

5.600000 0.377119

5.800000 0.305445

6.0 0.247285

6.200000 0.206259

6.400000 0.166847

6.600000 0.134906

6.800000 0.112401

7.0 0.090807

7.200000 0.073329

7.400000 0.061028

7.600000 0.049240

7.800000 0.039709

8.0 0.033011

8.200000 0.026599

8.400000 0.021421

8.600000 0.017787

8.800000 0.014312

9.0 0.011510

9.200000 0.009546

9.400000 0.007670

9.600000 0.006159

9.800000 0.005101

10.0 0.004092

10.200000 0.003281

10.400000 0.002714

10.600000 0.002174

10.800000 0.001740

11.0 0.001437

11.200000 0.001149

11.400000 0.000918

11.600000 0.000757

11.800000 0.000604

12.0 0.000482

12.200000 0.000397

12.400000 0.000316

12.600000 0.000251

12.800000 0.000207

13.0 0.000164

13.200000 0.000130

13.400000 0.000107

13.600000 0.000085

13.800000 0.000067

14.0 0.000055

14.200000 0.000043

14.400000 0.000034

14.600000 0.000028

14.800000 0.000022

15.0 0.000017

15.200000 0.000014

15.400000 0.000011

15.600000 0.000009

15.800000 0.000007

16.0 0.000005

16.200000 0.000004

16.400000 0.000003

16.600000 0.000003

16.800000 0.000002

17.0 0.000002

17.200000 0.000001

17.400000 0.000001

17.600000 0.000001

17.800000 0.000001

18.0 0.0

18.200000 0.0

18.400000 0.0

18.600000 0.0

18.800000 0.0

19.0 0.0

19.200000 0.0

19.400000 0.0

19.600000 0.0

19.800000 0.0

20.0 0.0

20.200000 0.0

20.400000 0.0

20.600000 0.0

20.800000 0.0

21.0 0.0

21.200000 -0.0

21.400000 -0.0

21.600000 -0.0

21.800000 -0.0

22.0 -0.0

22.200000 -0.0

22.400000 -0.0

22.600000 -0.0

22.800000 -0.0

23.0 -0.0

23.200000 -0.0

23.400000 -0.0

23.600000 -0.0

23.800000 -0.0

24.0 -0.0

24.200000 -0.0

24.400000 -0.0

24.600000 -0.0

24.800000 -0.0

25.0 -0.0

25.200000 -0.0

25.400000 -0.0

25.600000 -0.0

25.800000 -0.0

26.0 -0.0

26.200000 -0.0

26.400000 -0.0

26.600000 -0.0

26.800000 -0.0

27.0 -0.0

27.200000 -0.0

27.400000 -0.0

27.600000 -0.0

27.800000 -0.0

28.0 -0.0

28.200000 -0.0

28.400000 -0.0

28.600000 -0.0

28.800000 -0.0

29.0 -0.0

29.200000 -0.0

29.400000 -0.0

29.600000 -0.0

29.800000 -0.0

30.0 -0.0

30.200000 -0.0

30.400000 -0.0

30.600000 -0.0

30.800000 -0.0

31.0 -0.0

31.200000 -0.0

31.400000 -0.0

31.600000 -0.0

31.800000 -0.0

32.0 -0.0

32.200000 -0.0

32.400000 -0.0

32.600000 -0.0

32.800000 -0.0

33.0 -0.0

33.200000 -0.0

33.400000 -0.0

33.600000 -0.0

33.800000 -0.0

34.0 -0.0

34.200000 -0.0

34.400000 -0.0

34.600000 -0.0

34.800000 -0.0

35.0 -0.0

35.200000 -0.0

35.400000 -0.0

35.600000 -0.0

35.800000 -0.0

36.0 -0.0

36.200000 -0.0

36.400000 -0.0

36.600000 -0.0

36.800000 -0.0

37.0 -0.0

37.200000 -0.0

37.400000 -0.0

37.600000 -0.0

37.800000 -0.0

38.0 -0.0

38.200000 -0.0

38.400000 -0.0

38.600000 -0.0

38.800000 -0.0

39.0 -0.0

39.200000 -0.0

39.400000 -0.0

39.600000 -0.0

39.800000 -0.0

40.0 -0.0

40.200000 -0.0

40.400000 -0.0

40.600000 -0.0

40.800000 -0.0

41.0 -0.0

41.200000 -0.0

41.400000 -0.0

41.600000 -0.0

41.800000 -0.0

42.0 -0.0

42.200000 -0.0

42.400000 -0.0

42.600000 -0.0

42.800000 -0.0

43.0 -0.0

43.200000 -0.0

43.400000 -0.0

43.600000 -0.0

43.800000 -0.0

44.0 -0.0

44.200000 -0.0

44.400000 -0.0

44.600000 -0.0

44.800000 -0.0

45.0 -0.0

45.200000 -0.0

45.400000 -0.0

45.600000 -0.0

45.800000 -0.0

46.0 -0.0

46.200000 -0.0

46.400000 -0.0

46.600000 -0.0

46.800000 -0.0

47.0 -0.0

47.200000 -0.0

47.400000 -0.0

47.600000 -0.0

47.800000 -0.0

48.0 -0.0

48.200000 -0.0

48.400000 -0.0

48.600000 -0.0

48.800000 -0.0

49.0 -0.0

49.200000 -0.0

49.400000 -0.0

49.600000 -0.0

49.800000 -0.0

50.0 -0.0

50.200000 -0.0

50.400000 -0.0

50.600000 -0.0

50.800000 -0.0

51.0 -0.0

51.200000 -0.0

51.400000 -0.0

51.600000 -0.0

51.800000 -0.0

52.0 -0.0

52.200000 -0.0

52.400000 -0.0

52.600000 0.0

52.800000 0.0

53.0 0.0

53.200000 0.0

53.400000 0.0

53.600000 0.0

53.800000 0.0

54.0 0.0

54.200000 0.0

54.400000 0.0

54.600000 0.0

54.800000 0.0

55.0 0.0

55.200000 0.0

55.400000 0.0

55.600000 0.0

55.800000 0.0

56.0 0.0

56.200000 0.0

56.400000 0.0

56.600000 0.0

56.800000 0.0

57.0 0.0

57.200000 0.0

57.400000 0.0

57.600000 0.0

57.800000 0.0

58.0 0.0

58.200000 0.0

58.400000 0.0

58.600000 0.0

58.800000 0.0

59.0 0.0

59.200000 0.0

59.400000 0.0

59.600000 0.0

59.800000 0.0

60.0 0.0

60.200000 0.0

60.400000 0.0

60.600000 0.0

60.800000 0.0

61.0 0.0

61.200000 0.0

61.400000 0.0

61.600000 0.0

61.800000 0.0

62.0 0.0

62.200000 0.0

62.400000 0.0

62.600000 0.0

62.800000 0.0

63.0 0.0

63.200000 0.0

63.400000 0.0

63.600000 0.0

63.800000 0.0

64.0 0.0

64.200000 0.0

64.400000 0.0

64.600000 0.0

64.800000 0.0

65.0 0.0

65.200000 0.0

65.400000 0.0

65.600000 0.0

65.800000 0.0

66.0 0.0

66.200000 0.0

66.400000 0.0

66.600000 0.0

66.800000 0.0

67.0 0.0

67.200000 0.0

67.400000 0.0

67.600000 0.0

67.800000 0.0

68.0 0.0

68.200000 0.0

68.400000 0.0

68.600000 0.0

68.800000 0.0

69.0 0.0

69.200000 0.0

69.400000 0.0

69.600000 0.0

69.800000 0.0

70.0 0.0

70.200000 0.0

70.400000 0.0

70.600000 0.0

70.800000 0.0

71.0 0.0

71.200000 0.0

71.400000 0.0

71.600000 0.0

71.800000 0.0

72.0 0.0

72.200000 0.0

72.400000 0.0

72.600000 0.0

72.800000 0.0

73.0 0.0

73.200000 0.0

73.400000 0.0

73.600000 0.0

73.800000 0.0

74.0 0.0

74.200000 0.0

74.400000 0.0

74.600000 0.0

74.800000 0.0

75.0 0.0

75.200000 0.0

75.400000 0.0

75.600000 0.0

75.800000 0.0

76.0 0.0

76.200000 0.0

76.400000 0.0

76.600000 0.0

76.800000 0.0

77.0 0.0

77.200000 0.0

77.400000 0.0

77.600000 0.0

77.800000 0.0

78.0 0.0

78.200000 0.0

78.400000 0.0

78.600000 0.0

78.800000 0.0

79.0 0.0

79.200000 0.0

79.400000 0.0

79.600000 0.0

79.800000 0.0

80.0 0.0

80.200000 0.0

80.400000 0.0

80.600000 0.0

80.800000 0.0

81.0 0.0

81.200000 0.0

81.400000 0.0

81.600000 0.0

81.800000 0.0

82.0 0.0

82.200000 0.0

82.400000 0.0

82.600000 0.0

82.800000 0.0

83.0 0.0