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:

**«Многонитиевые процессы»**

**Вариант №9**

Выполнил Проценко В.И.

Группа М618

Преподаватель Баландин А.В.

Самара 2012

Оглавление

[Цель и постановка задания 3](#_Toc336436136)

[Описание программы 4](#_Toc336436137)

[Результат работы программы 5](#_Toc336436138)

[Приложения 6](#_Toc336436139)

[Исходный код 6](#_Toc336436140)

[Lab\_2\_Multithread.cc 6](#_Toc336436141)

[Lab\_2\_Multithread.h 7](#_Toc336436142)

[std.h 7](#_Toc336436143)

[T1.cc 8](#_Toc336436144)

[T1.h 8](#_Toc336436145)

[T2.cc 9](#_Toc336436146)

[T2.h 9](#_Toc336436147)

# Цель и постановка задания

**Тема лабораторной работы**: многонитиевые процессы.

**Цель задания**: освоение функций для запуска и синхронизации нитей.

**Задание**:

Разработать приложение, состоящее из одного процесса с тремя запущенными нитями: М, Т1, Т2.

Нить М - это функция main(). Нити Т1(F1) и Т2(F2) запускаются нитью М на базе разных функций (F1 и F2). Задача совместной работы нитей сформировать в текстовом буфере процесса некий текст. В исходном состоянии буфер содержит пустую строку. Порядок работы приложения следующий.

Нить М запускает нити Т1(F1) и Т2(F2) (в произвольном порядке), передавая им в качестве параметра указатель буфера, и *только после этого* первой заносит в пустой буфер текст - "Main started\n", и ожидает завершения дозаписи в буфер своих частей текста обеими нитями – Т1 и Т2.

Нить Т1 должна дозаписат в буфер после нити M свою часть текста - "Thread #1 written down\n" и, после завершения нити Т2, терминируется, выводит на экран сообщение - "Thread #1 terminated\n".

Нить Т2 должна дозаписат в буфер после нити Т1 свою часть текста - "Thread #2 written down\n", после чего нить Т2 выводит на экран сообщение - "Thread #2 terminated\n" и терминируется.

В итоге нить М выводит на экран сформированное содержимое буфера, выводит на экран сообщение - "Main terminated\n" и завершается.

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

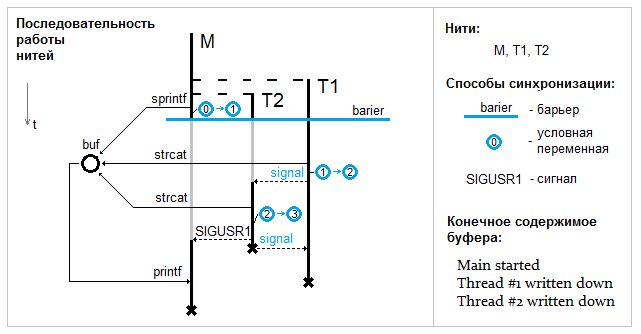
1. Описание разработанного процесса с обоснованием использования выбранных из базового набора методов синхронизации в контексте решаемой задачи.
2. Проверка теоретических знаний по средствам запуска и синхронизации нитей.
3. Демонстрация на компьютере работы многонитиевого процесса в соответствии с заданием.

**ВАРИАНТ №9**

**Используемый базовый набор методов синхронизации:**

1. Условная переменная.
2. Барьер.
3. Сигнал.

# Описание программы



После запуска процесса главная нить **M** инициализирует и создаёт нити T1 и T2. При создании нитям передаётся номер их нити: T1 – 1, T2 – 2, а также указатель на буфер. Главная нить записывает в буфер строку «Main started\n». Все нити синхронизируются перед дальнейшим выполнением по барьеру “ barrier\_start ”.

Для единовременного доступа к общему ресурсу buf используется мьютекс «mutex\_cond\_proc». При синхронизации выполнения нитей T1 и T2 используется переменная «cur\_proc», используемая при проверке на ожидание и условная переменная «cond\_proc», оповещающая нити о событии изменения «cur\_proc». С помощью «cur\_proc» задаётся порядок выполнения: T1, T2. Нить T1 также использует «cur\_proc» для завершения работы после уничтожения нити T2, не завершит работу пока не будут выполнены последвоательно два условия «cur\_proc» == 1, «cur\_proc» == 3.

После барьера независимо от T1 и T2, нить M входит в бесконечный цикл ожидания, предварительно подписав на обработку сигнала «SIGUSR1» функцию «handler». Перед завершением работы нить посылает текущему процессу сигнал «SIGUSR1».

Текущему процессу приходит сигнал «SIGUSR1», запускается обработчик. Обработчик печатает содержимое буфера «buf» и завершает работу процесса.

# C:\QNX_Shared\Lab_2_Multithread\Lab_2_Multithread_Snapshot.pngРезультат работы программы

# Приложения

### Исходный код

### Lab\_2\_Multithread.cc

#include <std.h>

#include <Lab\_2\_Multithread.h>

#include <T1.h>

#include <T2.h>

namespace M\_thread {

pthread\_barrier\_t barrier\_start;

pthread\_mutex\_t mutex\_cond\_proc;

pthread\_cond\_t cond\_proc;

short cur\_proc;

char \* buf;

}

/\*printf("Value %i\n", \*(short\*)(\*(int\*)((int)arg)));

printf("Value %s\n", (char\*)(\*(int\*)((int)arg + 4)));\*/

void handler(int signo) {

printf("Buffer: %s", M\_thread::buf);

exit(EXIT\_SUCCESS);

}

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

int \*arg\_1 = (int\*)malloc(2\*sizeof(int));

int \*arg\_2 = (int\*)malloc(2\*sizeof(int));

M\_thread::buf = (char\*)malloc(30\*sizeof(char));

// T1 - DETACHED. T2 - JOINABLE. T1 waits T2 terminated.

pthread\_attr\_t attr\_T1 = M\_thread::thread\_T1\_init();

pthread\_attr\_t attr\_T2 = M\_thread::thread\_T2\_init();

struct sigaction act;

sigset\_t set;

pthread\_barrier\_init(&M\_thread::barrier\_start, NULL, 3);

pthread\_mutex\_init(&M\_thread::mutex\_cond\_proc, NULL);

pthread\_cond\_init(&M\_thread::cond\_proc, NULL);

sigemptyset(&set);

sigaddset(&set, SIGUSR1);

act.sa\_flags = 0;

act.sa\_mask = set;

act.sa\_handler = &handler;

sigaction(SIGUSR1, &act, NULL);

\*(int\*)((int)arg\_1) = (int)&M\_thread::THREAD\_1;

\*(int\*)((int)arg\_1 + 4) = (int)M\_thread::buf;

\*(int\*)((int)arg\_2) = (int)&M\_thread::THREAD\_2;

\*(int\*)((int)arg\_2 + 4) = (int)M\_thread::buf;

pthread\_create(NULL, &attr\_T1, &M\_thread::thread\_T1, (void\*)(arg\_1));

pthread\_create(NULL, &attr\_T2, &M\_thread::thread\_T2, (void\*)(arg\_2));

// All processes would stop at barrier. M would write to buf first

// before barrier.

// Var cur\_proc would be switched to next process - THREAD\_1.

// Each thread would lock mutex associated with con\_var and cur\_proc

// and check on signal of cond\_var if it his turn.

// After completing T1 would switch cur\_proc to next process - THREAD\_2.

sprintf(M\_thread::buf, "Main started\n");

M\_thread::cur\_proc = M\_thread::THREAD\_1;

pthread\_barrier\_wait(&M\_thread::barrier\_start);

// M is waiting for SIGUSR1, from T2. Exit point from program is in handler.

while(1) {

pause();

}

return EXIT\_SUCCESS;

}

### Lab\_2\_Multithread.h

/\*

\* Lab\_2\_Multithread.h

\*

\* Created on: 25.09.2012

\* Author:

\*/

#ifndef LAB\_2\_MULTITHREAD\_H\_

#define LAB\_2\_MULTITHREAD\_H\_

namespace M\_thread {

extern pthread\_barrier\_t barrier\_start;

extern pthread\_mutex\_t mutex\_cond\_proc;

extern pthread\_cond\_t cond\_proc;

extern short cur\_proc;

const short THREAD\_1 = 1;

const short THREAD\_2 = 2;

}

#endif /\* LAB\_2\_MULTITHREAD\_H\_ \*/

### std.h

/\*

\* std.h

\*

\* Created on: 25.09.2012

\* Author:

\*/

#ifndef STD\_H\_

#define STD\_H\_

#include <stdio.h>

#include <stdlib.h> //std::cout

#include <cstdlib>

#include <iostream>

#include <pthread.h>//barrier

#include <string.h> //strcat

#include <signal.h> //kill

#include <process.h>//getpid()

#include <unistd.h> //pause()

#endif /\* STD\_H\_ \*/

### T1.cc

#include <std.h>

#include <T1.h>

#include <Lab\_2\_Multithread.h>

namespace M\_thread {

pthread\_attr\_t thread\_T1\_init()

{

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_DETACHED);

return attr;

}

void\* thread\_T1(void\* arg)

{

short this\_proc = \*(short\*)(\*(int\*)((int)arg));

char \*buf = (char\*)(\*(int\*)((int)arg + 4));

pthread\_barrier\_wait(&barrier\_start);

pthread\_mutex\_lock(&mutex\_cond\_proc);

while(cur\_proc != THREAD\_1) {

pthread\_cond\_wait(&cond\_proc, &mutex\_cond\_proc);

}

strcat(buf, "Thread#1 written down\n");

cur\_proc = this\_proc + 1;

pthread\_cond\_signal(&cond\_proc);

pthread\_mutex\_unlock(&mutex\_cond\_proc);

pthread\_mutex\_lock(&mutex\_cond\_proc);

while(cur\_proc != THREAD\_2 + 1) {

pthread\_cond\_wait(&cond\_proc, &mutex\_cond\_proc);

}

pthread\_mutex\_unlock(&mutex\_cond\_proc);

std::cout << "T1 terminated" << std::endl;

return 0;

}

}

### T1.h

/\*

\* T1.h

\*

\* Created on: 25.09.2012

\* Author:

\*/

#ifndef T1\_H\_

#define T1\_H\_

#endif /\* T1\_H\_ \*/

namespace M\_thread {

pthread\_attr\_t thread\_T1\_init();

void\* thread\_T1(void\* arg);

}

### T2.cc

#include <std.h>

#include <T2.h>

#include <Lab\_2\_Multithread.h>

namespace M\_thread {

pthread\_attr\_t thread\_T2\_init()

{

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_DETACHED);

return attr;

}

void\* thread\_T2(void\* arg)

{

short this\_proc = \*(short\*)(\*(int\*)((int)arg));

char \*buf = (char\*)(\*(int\*)((int)arg + 4));

pthread\_barrier\_wait(&barrier\_start);

pthread\_mutex\_lock(&mutex\_cond\_proc);

while(cur\_proc != THREAD\_2) {

pthread\_cond\_wait(&cond\_proc, &mutex\_cond\_proc);

}

strcat(buf, "Thread#2 written down\n");

cur\_proc = this\_proc + 1;

pthread\_cond\_signal(&cond\_proc);

kill(getpid(), SIGUSR1);

pthread\_mutex\_unlock(&mutex\_cond\_proc);

std::cout << "T2 terminated" << std::endl;

return 0;

}

}

### T2.h

/\*

\* T2.h

\*

\* Created on: 25.09.2012

\* Author:

\*/

#ifndef T2\_H\_

#define T2\_H\_

#endif /\* T2\_H\_ \*/

namespace M\_thread {

pthread\_attr\_t thread\_T2\_init();

void\* thread\_T2(void\* arg);

}