Примеры многопоточных программ (потоки в ОС и RTL)

Востокин Сергей Владимирович

План

- Пример 0: применение Taskbag
- Модель потоков
- Пример 1: Реализация потоков в Windows
- Пример 2: Реализация потоков в Unix
- Пример 3: Реализация потоков в C++

Пример 0: применение Taskbag

Описание примера

- Цель: демонстрация применения шаблона Taskbag для распараллеливания сортировки
- Исходный алгоритм: алгоритм быстрой сортировки Хоара
- Реализации шаблона Taskbag
  - Потоковый пул Windows API
  - Потоковый пул POSIX Threads

Исходный алгоритм

```
void qSort(int* a, int size)
{
    long i = 0, j = size-1;
    int temp, p;

    p = a[ size>>1 ];
    do {
        while ( a[i] < p ) i++;
        while ( a[j] > p ) j--;

        if (i <= j) {
            temp = a[i]; a[i] = a[j]; a[j] = temp;
            i++; j--;
        }
    } while ( i<=j );

    if ( j > 0 ) qSort(a, j+1);
    if ( size > i ) qSort(a+i, size-i);
}
```

Структура кода, к которой требуется привести исходный алгоритм

```
struct task{/*1*/};
struct bag{/*2*/};
void proc(task*){/*3*/}
bool if_job(bag*){/*4*/return false;}
void put(task*,bag*){/*5*/}
void get(task*,bag*){/*6*/}

int _tmain(int argc, _TCHAR* argv[])
{
    task t; bag b;
    while(if_job(&b)){
        get(&t,&b); proc(&t); put(&t,&b);
    }
    return 0;
}
```

Принцип решения

- Задача (task) – сортировка подмассива
- Алгоритм разделяется на 2 стадии:
  - Формирование списка задач
  - Обработка списка
- Задачи формируются за счёт остановки рекурсии, когда разделяемый массив достигает некоторого заданного размера

Структуры данных

```
struct task{
        int*a; int size;
};

struct bag{
        queue<task> taskQueue;
};
```

Функции (1/2)

```
void proc(task*t){
    qSort(t->a,t->size);
}

bool if_job(bag*b){
    return !b->taskQueue.empty();
}
```

Функции (2/2)

```
void put(task*,bag*){
        // код не требуется
}

void get(task*t,bag*b){
        *t=b->taskQueue.front();
        b->taskQueue.pop();
}
```

Формирование списка задач

```
void qSort0(bag*b, int*a, int size)
{
    long i = 0, j = size-1;
    int temp, p;

    p = a[ size>>1 ];
    do {
        while ( a[i] < p ) i++;
        while ( a[j] > p ) j--;

        if (i <= j) {
            temp = a[i]; a[i] = a[j]; a[j] = temp;
            i++; j--;
        }
    } while ( i<=j );

    if ( j > 0 )    if(j+1 < T)    b->taskQueue.push(task(a,j+1));     else qSort0(b, a, j+1);
    if ( size > i ) if(size-i < T) b->taskQueue.push(task(a+i,size-i));else qSort0(b, a+i, size-i);
}
```

Запуск алгоритма

```
int _tmain(int argc, _TCHAR* argv[])
{
    task t; bag b;

    for(int i=0;i<N;i++) arr[i]=rand()%N;

    qSort0(&b, arr, N);

    while(if_job(&b)){
        get(&t,&b); proc(&t); put(&t,&b);
    }

    for(int i=0;i<N;i++) cout<<arr[i]<<' ';

    return 0;
}
```

Перенос блоков кода в шаблон Taskbag (1/4)

```
class TaskBag:public TEMPLET::TBag{
public:
    class TaskBagTask:public TBag::Task{
    public:
        TaskBagTask():TBag::Task(){}
        virtual~TaskBagTask(){}

        void send_task()  {}
        void recv_task()  {}
        void send_result(){}
        void recv_result(){}

        task t;//<----------------
    };
```

Перенос блоков кода в шаблон Taskbag (2/4)

```
public:
    TaskBag(int num_prc,int argc, char* argv[]):TBag(num_prc,argc,argv){
        for(int i=0;i<N;i++) arr[i]=rand()%N; //<----------
        qSort0(&b, arr, N);                   //<----------
    }
    virtual ~TaskBag(){}
    TBag::Task* createTask(){return new TaskBagTask;}
```

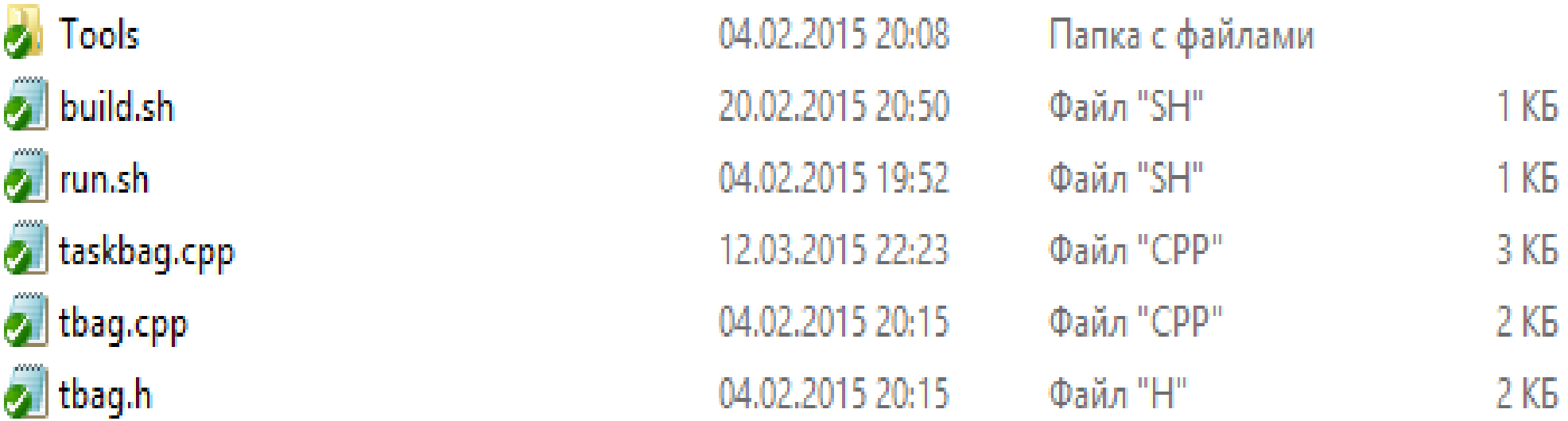
Перенос блоков кода в шаблон Taskbag (3/4)

```
queue<task> taskQueue;//<-------------

bool if_job(){return !taskQueue.empty();}//<---------
void put(Task*t){}//<-------------
void get(Task*t){TaskBagTask* mt=(TaskBagTask*)t;
    tm->t=taskQueue.front(); //<-------------
    taskQueue.pop(); //<-------------
}
```

Перенос блоков кода в шаблон Taskbag (4/4)

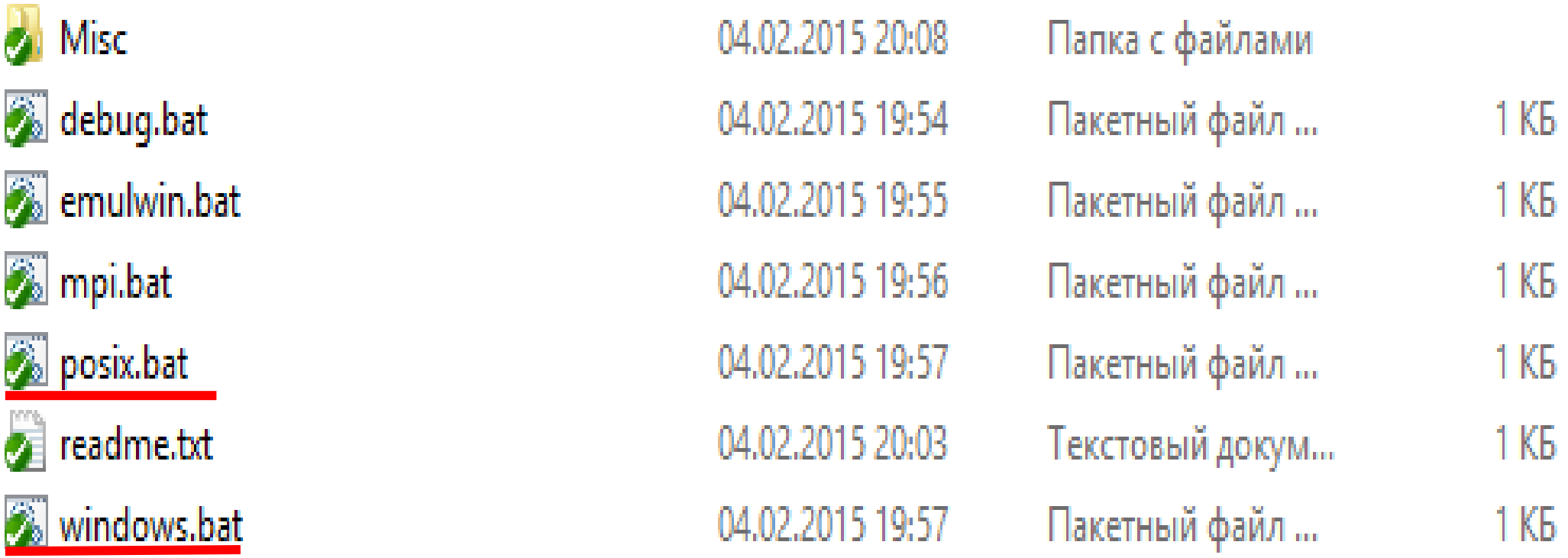
```
void proc(Task*t){TaskBagTask* mt=(TaskBagTask*)t;
    qSort(mt->t.a,mt->t.size); //<------
}
```

Выбор способа выполнения (1/2)



Tools	04.02.2015 20:08	Папка с файлами	
build.sh	20.02.2015 20:50	Файл "SH"	1 КБ
run.sh	04.02.2015 19:52	Файл "SH"	1 КБ
taskbag.cpp	12.03.2015 22:23	Файл "CPP"	3 КБ
tbag.cpp	04.02.2015 20:15	Файл "CPP"	2 КБ
tbag.h	04.02.2015 20:15	Файл "H"	2 КБ

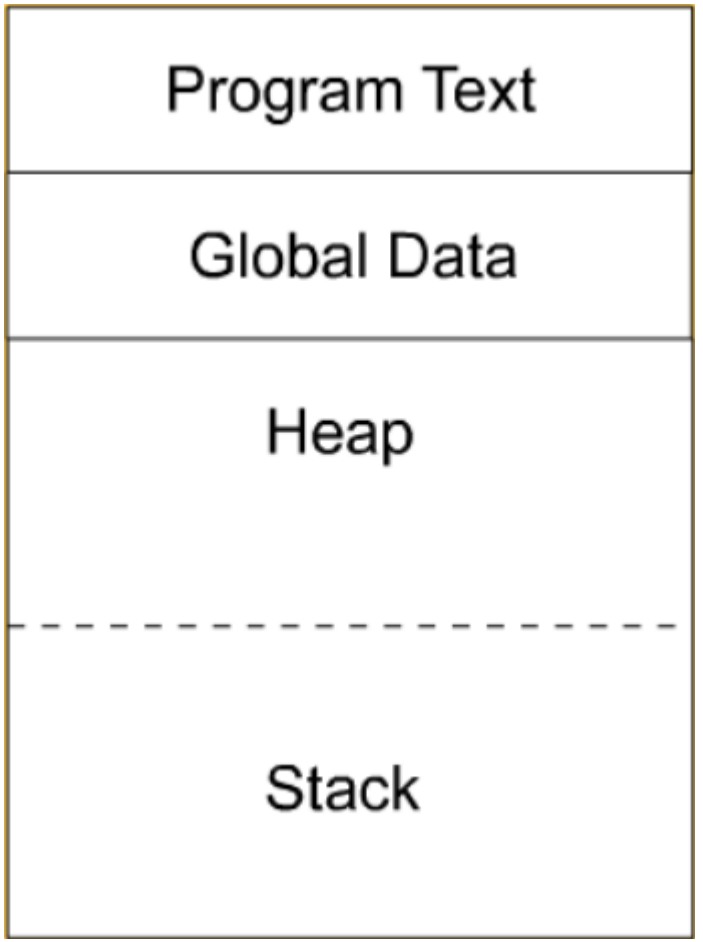
Выбор способа выполнения (2/2)



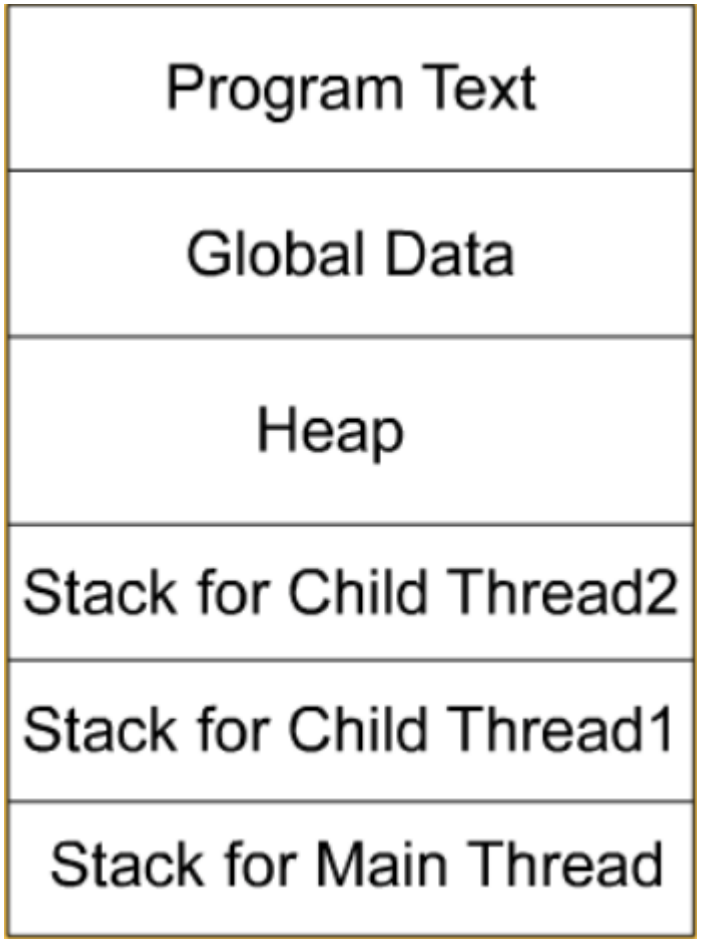
Misc	04.02.2015 20:08	Папка с файлами	
debug.bat	04.02.2015 19:54	Пакетный файл ...	1 КБ
emulwin.bat	04.02.2015 19:55	Пакетный файл ...	1 КБ
mpi.bat	04.02.2015 19:56	Пакетный файл ...	1 КБ
posix.bat	04.02.2015 19:57	Пакетный файл ...	1 КБ
readme.txt	04.02.2015 20:03	Текстовый докум...	1 КБ
windows.bat	04.02.2015 19:57	Пакетный файл ...	1 КБ

Модель потоков

Структура памяти обычного процесса



Структура памяти многопоточного процесса

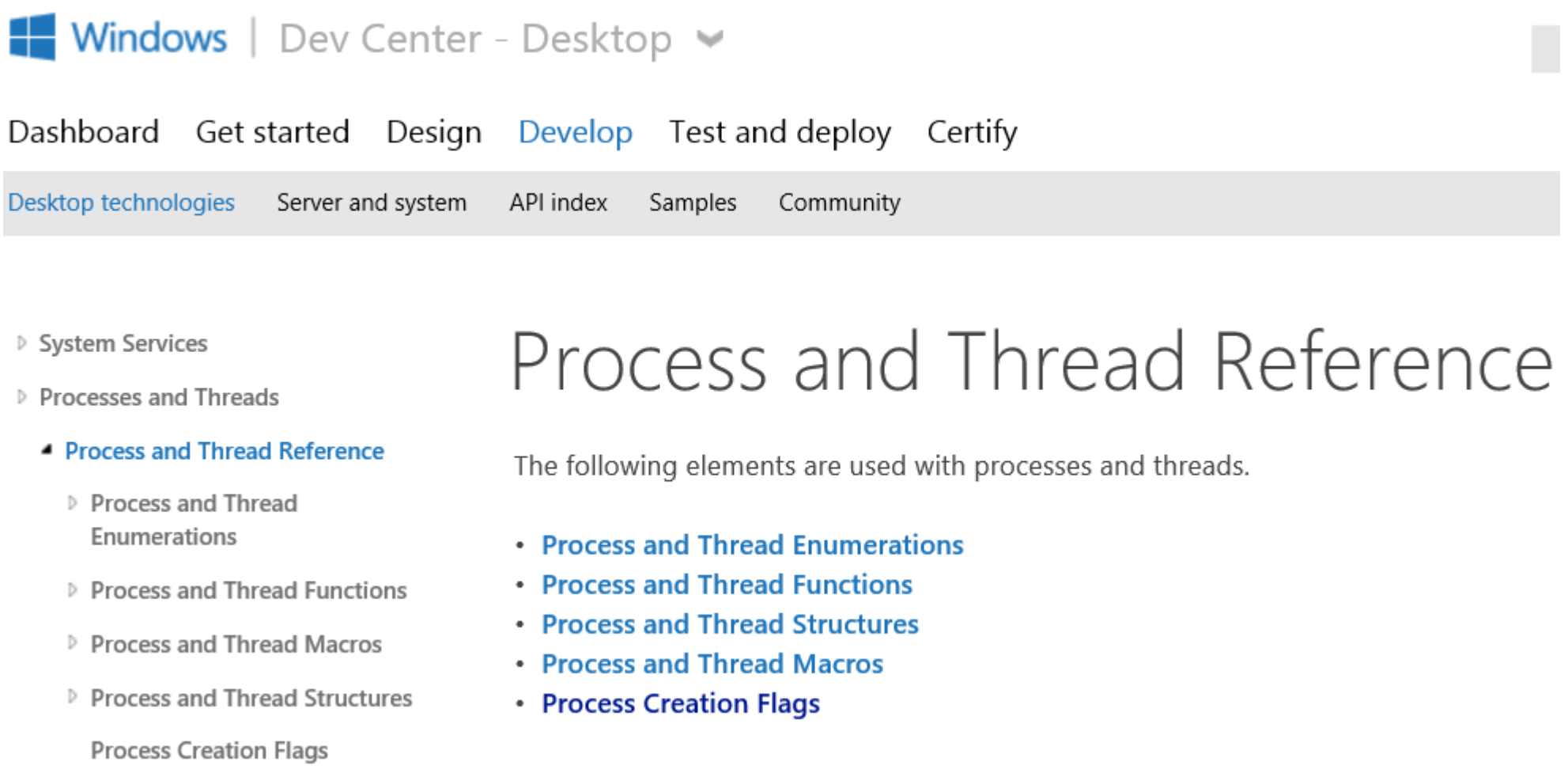


Псевдокод API управления потоками

```
struct thread{void(*tfunc)(thread*);};

struct mutex{};     //конкурентная синхронизация
void lock(mutex*);
void unlock(mutex*);

struct event{};     //условная синхронизация
void wait(event*,mutex*);
void notify(event*);
```

Реализация потоков в Windows

Документация (1/3)



Документация (2/3)

Process and Thread Functions

This topic describes the process and thread functions.

- Process Functions
- Process Enumeration Functions
- Thread Functions
- Process and Thread Extended Attribute Functions
- WOW64 Functions
- Job Object Functions
- Thread Pool Functions
- Thread Ordering Service Functions
- Multimedia Class Scheduler Service Functions
- Fiber Functions
- NUMA Support Functions
- Processor Functions
- User-Mode Scheduling Functions
- Obsolete Functions

Документация (3/3)

Synchronization Functions

The following functions are used in synchronization.

- Asynchronous functions
- Condition variable and SRW lock functions
- Critical section functions
- Event functions
- One-time initialization functions
- Interlocked Functions
- Mutex functions
- Private namespace functions
- Semaphore functions
- Singly-linked list functions
- Synchronization barrier functions
- Timer-queue timer functions
- Wait functions
- Waitable-timer functions

Пример программы (1/5)

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define NUM_THREADS     5

HANDLE cv;
HANDLE mtx;
bool thread4done = false;
```

Пример программы (2/5)

```
DWORD WINAPI task_code(PVOID argument)
{
    int tid;

    tid = *((int *) argument);

    WaitForSingleObject(mtx,INFINITE);

    // let thread 4 executes first
    while (tid != 4 && !thread4done){
         ReleaseMutex(mtx);
         WaitForSingleObject(cv,INFINITE);ResetEvent(cv);
         WaitForSingleObject(mtx,INFINITE);
    }

    printf("Hello World! It's me, thread %d!\n", tid);

    if (tid == 4) { thread4done = true;  SetEvent(cv);}

    ReleaseMutex(mtx);

    /* optionally: insert more useful stuff here */

    return NULL;
}
```

Пример программы (3/5)

```
int main(void)
{
    HANDLE threads[NUM_THREADS];
    int thread_args[NUM_THREADS];
    DWORD id,ret;
    int i;

    mtx = CreateMutex(NULL,FALSE,NULL);
    cv  = CreateEvent(NULL,TRUE,FALSE,NULL);

    printf("Hello from Windows threads app!\n");

    // getting processors configuration
    SYSTEM_INFO si;
    GetSystemInfo(&si);
    printf(" dwNumberOfProcessors=%ld\n", si.dwNumberOfProcessors);
```

Пример программы (4/5)

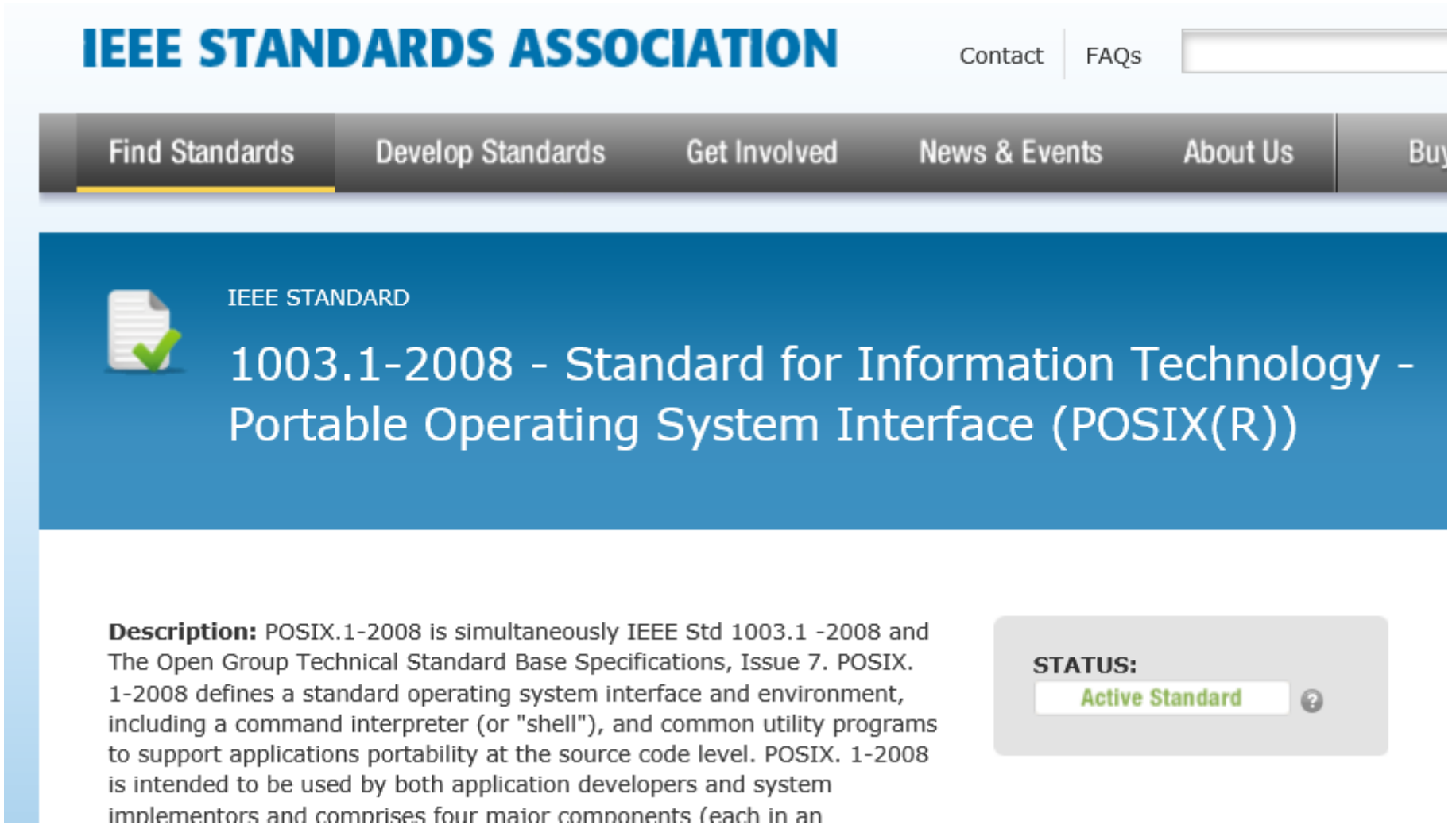
```
// create all threads one by one
for (i=0; i<NUM_THREADS; ++i) {
   thread_args[i] = i;
   printf("In main: creating thread %d\n", i);
   threads[i] = CreateThread(NULL,0,task_code,&thread_args[i],0,&id);
   assert(threads[i] != NULL);
}

// wait for all threads to complete
ret = WaitForMultipleObjects(NUM_THREADS,threads,TRUE,INFINITE);
assert(ret != WAIT_FAILED);
```

Пример программы (5/5)

```
    printf("In main: All threads completed successfully\n");

    for (i=0; i<NUM_THREADS; ++i) CloseHandle(threads[i]);
    CloseHandle(mtx);
    CloseHandle(cv);

    return EXIT_SUCCESS;
}
```

Реализация потоков в POSIX

Документация (1/2)



Документация (2/2)

The Open Group Base Specifications Issue 7
IEEE Std 1003.1, 2013 Edition

NAME

pthread.h - threads

SYNOPSIS

```
#include <pthread.h>
```

DESCRIPTION

The <*pthread.h*> header shall define the following symbolic constants:

PTHREAD_BARRIER_SERIAL_THREAD
PTHREAD_CANCEL_ASYNCHRONOUS
PTHREAD_CANCEL_ENABLE
PTHREAD_CANCEL_DEFERRED
PTHREAD_CANCEL_DISABLE
PTHREAD_CANCELED
PTHREAD_CREATE_DETACHED

Пример программы (1/5)

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#ifndef _WIN32
#include <sys/sysinfo.h>
#include <unistd.h>
#endif // _WIN32

#define NUM_THREADS     5

pthread_cond_t cv;
pthread_mutex_t mtx;
bool thread4done = false;
```

Пример программы (2/5)

```
void *task_code(void *argument)
{
    int tid;

    tid = *((int *) argument);

    pthread_mutex_lock(&mtx);

    // let thread 4 executes first
    while (tid != 4 && !thread4done) pthread_cond_wait(&cv, &mtx);

    printf("Hello World! It's me, thread %d!\n", tid);

    if (tid == 4) { thread4done = true; pthread_cond_broadcast(&cv);

    pthread_mutex_unlock(&mtx);

    /* optionally: insert more useful stuff here */

    return NULL;
}
```

Пример программы (3/5)

```
int main(void)
{
    pthread_t threads[NUM_THREADS];
    int thread_args[NUM_THREADS];
    int rc, i;

    pthread_mutex_init(&mtx, NULL);
    pthread_cond_init(&cv, NULL);

    printf("Hello from POSIX threads app!\n");

    // getting processors configuration
#ifndef _WIN32
    printf(" sysconf_configured=%ld\n", sysconf(_SC_NPROCESSORS_CONF));
    printf(" sysconf_online=%ld\n", sysconf(_SC_NPROCESSORS_ONLN));
    printf(" get_nprocs_conf=%d\n", get_nprocs_conf());
    printf(" get_nprocs=%d\n", get_nprocs());
#endif // _WIN32
```

Пример программы (4/5)

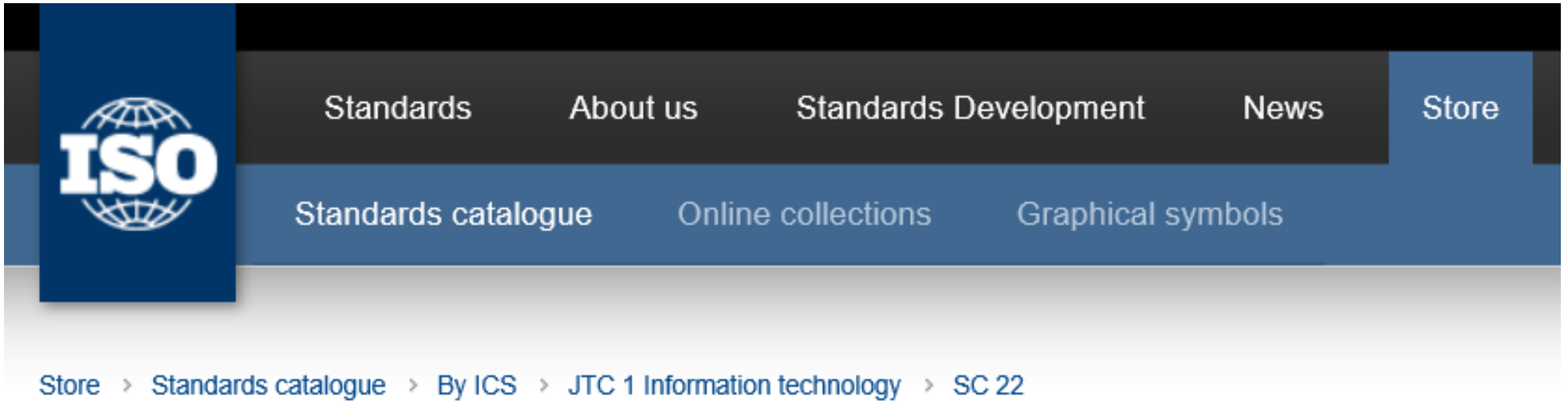
```
// create all threads one by one
for (i=0; i<NUM_THREADS; ++i) {
   thread_args[i] = i;
   printf("In main: creating thread %d\n", i);
   rc = pthread_create(&threads[i], NULL, task_code, (void *) &thread_args[i]);
   assert(0 == rc);
}
```

Пример программы (5/5)

```
    // wait for each thread to complete
    for (i=0; i<NUM_THREADS; ++i) {
        // block until thread i completes
        rc = pthread_join(threads[i], NULL);
        printf("In main: thread %d is complete\n", i);
        assert(0 == rc);
    }

    printf("In main: All threads completed successfully\n");

    pthread_mutex_destroy(&mtx);
    pthread_cond_destroy(&cv);

    return EXIT_SUCCESS;
}
```

Реализация потоков в C++

Документация (1/2)



ISO/IEC 14882:2014

Information technology -- Programming languages -- C++

Abstract

Preview ISO/IEC 14882:2014

Документация (2/2)

Пример программы (1/5)

```
#include <mutex>
#include <condition_variable>
#include <thread>

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define NUM_THREADS     5

std::condition_variable cv;
std::mutex mtx;
bool thread4done = false;
```

Пример программы (2/5)

```
void task_code(int tid)
{
     std::unique_lock<std::mutex> lck(mtx);

    // let thread 4 executes first
    while (tid != 4 && !thread4done)  cv.wait(lck);

    printf("Hello World! It's me, thread %d!\n", tid);

    if (tid == 4) { thread4done = true; cv.notify_all(); }

    /* optionally: insert more useful stuff here */

}
```

Пример программы (3/5)

```
int main(void)
{
    std::thread threads[NUM_THREADS];
    int rc, i;

    printf("Hello from C++ threads app!\n");

    // getting processors configuration

    printf(" hardware_concurrency=%ld\n", std::thread::hardware_concurrency());
```

Пример программы (4/5)

```
// create all threads one by one
for (i=0; i<NUM_THREADS; ++i) {
    printf("In main: creating thread %d\n", i);
    threads[i] = std::thread(task_code, i);
}
```

Пример программы (5/5)

```
    // wait for each thread to complete
    for (i=0; i<NUM_THREADS; ++i) {
        // block until thread i completes
        threads[i].join();
        printf("In main: thread %d is complete\n", i);
    }

    printf("In main: All threads completed successfully\n");

    return EXIT_SUCCESS;
}
```

Заключение

- Для простых задач подходит паттерн «потоковый пул» (farm, bag-of-tasks, master-worker, master-slave)
- Паттерн «потоковый пул» обычно реализован в среде программирования или в библиотеке
- Низкоуровневая поддержка потоков имеется практически во всех современных средах программирования
- Если требуется оптимальное по производительности многопоточное приложение следует использовать API операционной системы и/или RTL языка программирования