Контакты

Сидельников Виктор Викторович vvs_home@list.ru
Крестелев Данила Вадимович krestelev@gmail.com
В контакте Группа СРВ, vk.com/rtsys

Программное обеспечение

QNX – Real-Time OS
http://www.qnx.com/products/evaluation
Othher license keys Academic -> Single user license

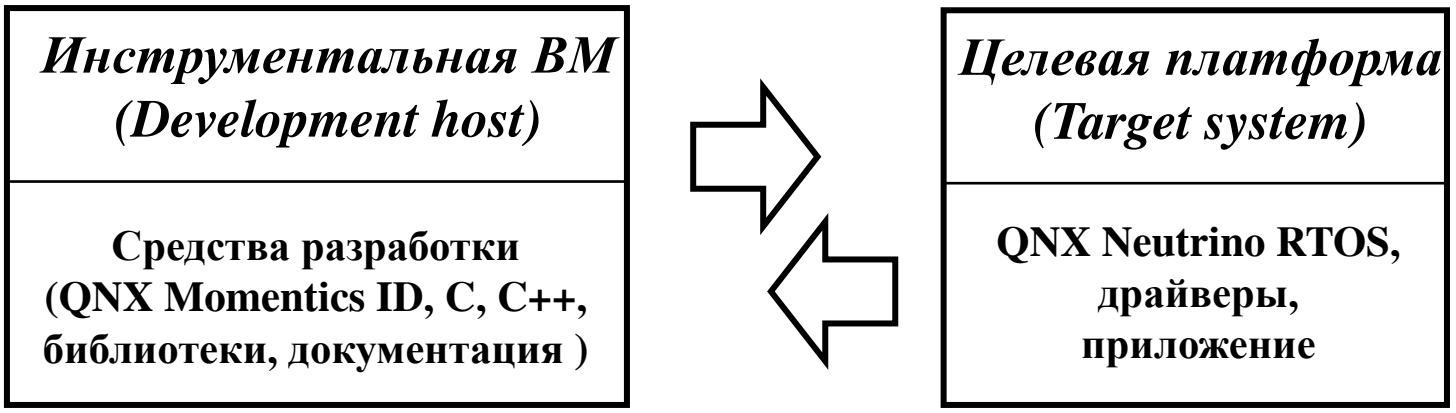
QNX® Software Development Platform 6.6.0 – Windows Hosts
QNX SDP 6.6.0 runtime ISO for VMware [or virtual machine]

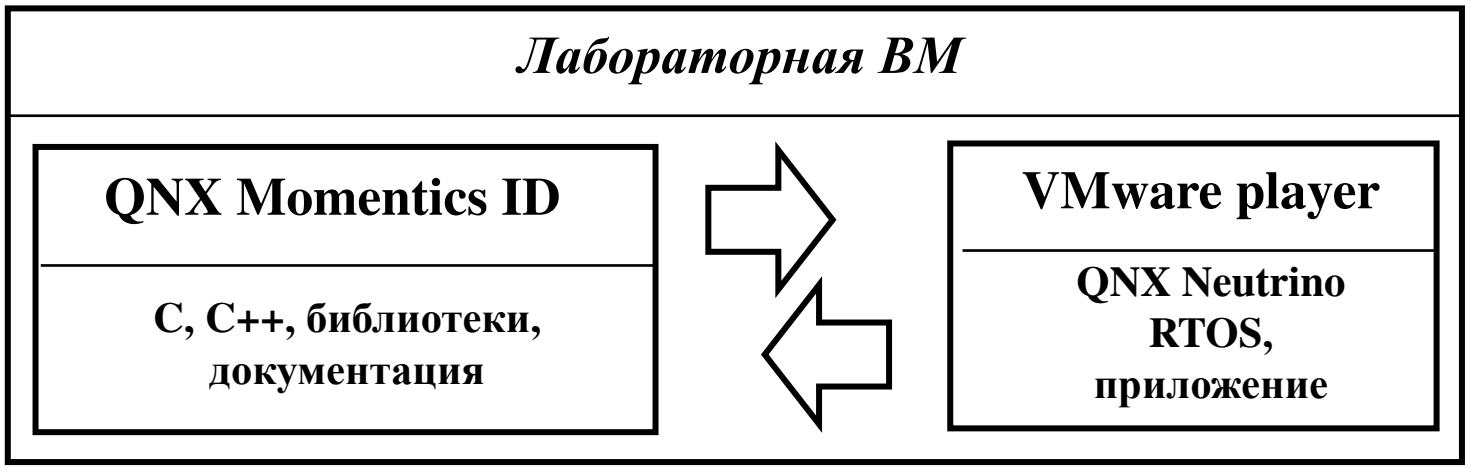
Литература

Цилюрик О., Горшко Е. Анатомия параллелизма QNX/UNIX
Кертен Р. Введение в QNX Neutrino 2. Руководство для разработчиков приложений реального времени

Средства разработки

(*A_Quickstart_Guide.pdf*)





Функции для работы с нитями

```
#include <pthread.h>
int pthread_create( pthread_t* thread, const pthread_attr_t* attr,
                    void* (*start_routine)(void* ), void* arg );
```

thread – идентификатор нити (ID), устанавливается при создании;
attr – атрибутная запись, при значении NULL устанавливается по умолчанию;
void* (*start_routine)(void*) – функция, код которой выполняется в потоке;
arg – аргумент, передаваемый в функцию потока
Простой вызов - **pthread_create(&t, NULL, &func, NULL)**

pthread_t pthread_self(void); - возвращает ID потока;

int pthread_t thread, void value_ptr);** - возвращает результат выполнения (0 –успешное)
Простой вызов - **pthread_join(t, NULL);**

```
#include <unistd.h>
unsigned int sleep( unsigned int seconds );
```

Создание нити, синхронизация завершения

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

void* test_t(void* arg) {
    printf("Thread %d started \n", pthread_self());
    int i;
    for(i=0; i<=10; i++) {
        printf("Thread %d is working %d\n", pthread_self(), i);
        sleep(1);
    }
    printf("Thread %d stop\n”, pthread_self());
    return EXIT_SUCCESS;
}

int main(int argc, char *argv[]) {
    pthread_t thread_id;
    pthread_create(&thread_id, NULL, &test_t, NULL);
    printf("Main thread stop\n");
    pthread_join(thread_id, NULL);
    return EXIT_SUCCESS;
}
```

Семафоры, мьютексы, условные переменные

```
#include <semaphore.h>
int sem_init( sem_t * sem,  int pshared,  unsigned value );
int sem_wait( sem_t * sem );
int sem_post( sem_t * sem );
int sem_trywait( sem_t * sem );
int sem_getvalue(sem_t* sem,  int* value)

#include <pthread.h>
int pthread_mutex_init( pthread_mutex_t* mutex, const pthread_mutexattr_t* attr
);
int pthread_mutex_lock( pthread_mutex_t* mutex );
int pthread_mutex_trylock( pthread_mutex_t* mutex );
int pthread_mutex_timedlock( pthread_mutex_t * mutex,  const struct timespec *
                             abs_timeout );
int pthread_mutex_unlock( pthread_mutex_t* mutex );

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_init( pthread_cond_t* cond,   pthread_condattr_t* attr );
int pthread_cond_wait( pthread_cond_t* cond,  pthread_mutex_t* mutex );
int pthread_cond_signal( pthread_cond_t* cond );
int pthread_cond_broadcast( pthread_cond_t* cond );
```

Вызовы функций семафора

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>

void f(pthread_t id) {
   printf("Thread %d called this function\n", id);
}

sem_t s;

void* thread_2(void* arg) {
    pthread_t id = pthread_self();
    for(;;) {
        printf("Tread %d is working\n", id );
        sem_wait(&s);
        f(id);
        sem_post(&s);
        usleep(300);
    }
    return 0;
}
```

```
int main(int argc, char *argv[]) {
    pthread_t thr_2;
    sem_init(&s, NULL, 10);
    pthread_t id = pthread_self();
    pthread_create(&thr_2, NULL, &thread_2, NULL);
    for(;;) {
        printf("Tread %d is working\n", id );
        sem_wait(&s);
        f(id);
        sem_post(&s);
        usleep(300);
    }
    printf("Main thread stop\n");
    pthread_join(thr_2, NULL);
    return EXIT_SUCCESS;
}
```

Вызовы функций мьютекса

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

pthread_mutex_t mutex
int count = 0;

void* thr_1( void* arg )
{
    int tmp = 0;
    while( 1 ) {
        pthread_mutex_lock( &mutex );
        tmp = count++;
        pthread_mutex_unlock( &mutex );
        printf( "Count is %d\n", tmp );
        sleep( 1 );
    }
    return 0;
}
```

```
void* thr_2( void* arg )
{
    int tmp = 0;
    while( 1 ) {
        pthread_mutex_lock( &mutex );
        tmp = count--;
        pthread_mutex_unlock( &mutex );
        printf( "** Count is %d\n", tmp );
        sleep( 2 );
    }
    return 0;
}

int main( void )
{
    pthread_create( NULL, NULL, &thr_1, NULL );
    pthread_create( NULL, NULL, &thr_2, NULL );
    sleep( 10 );
    return 0;
}
```

Использование условных переменных

```
pthread_cond_t cond;
pthread_mutex_t mutex;

void* thread_1 (void* args){
    while(1){
        pthread_mutex_lock(&mutex);
        if (!condition_1)
            pthread_cond_wait(&cond, &mutex);
        Do_Action_1;
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mutex);
        sleep(s1)
    };
    return EXIT_SUCCESS;
}

void* thread_2 (void* args){
    while(1){
        pthread_mutex_lock(&mutex);
        if (!condition_2)
            pthread_cond_wait(&cond, &mutex);
        Do_Action_2;
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mutex);
        sleep (s2);
    };
    return EXIT_SUCCESS;
}
```

```
int main(int argc, char *argv[]) {
    pthread_t thr_1, thr_2;
    pthread_cond_init(&cond, NULL);
    pthread_mutex_init(&mutex, NULL);
    pthread_create(&thr_1, NULL, &thread_1, NULL);
    pthread_create(&thr_2, NULL, &thread_2, NULL);
    return EXIT_SUCCESS;
}
```