EMBEDDED SYSTEM SOFTWARE AND HARDWARE BASICS
HOME TASK

Q1: Some application is implemented to run on a Windows 32 platform. You get a request to port it on some mobile device. What characteristics should you know about the mobile device platform? What parts of the initial application will be changed/added/removed? What challenges could you meet during porting?
A1: Характеристики: архитектура (Arm, x86, x64, MIPS), набор команд, операционная система (Android, IOS, Symbian), размер RAM, ROM и HDD, little-endian или big-endian. Будет удалено: интерфейс и все функции, связанные с Windows, все связанное с x32, если это будет теперь не она. Добавлено: работа с новой архитектурой, новым набором команд, новый интерфейс. Исправлено: endian, если нужно. Проблемы: странная система, странный набор команд, странный язык программирования, странный язык комментариев, типы переменных имеют разное число байт, проблема кастомизации устройства – нет root прав или нельзя установить приложение не из магазина (типа IOS), отсутствие нужных шнуров, отсутствие datasheet'а и еще много проблем.
A1-2: разрядность системы, наличие компиляторов, языки программирования, характеристики процессора

Q2: You get a request to develop software for some part of the embedded system that could be ported later to another device. What should you take into account while developing the software? What subsystem, in general, will be in your application?
A2: условия кросплатформенности: использование кросплатформенного ЯП (C++, Java, Си, Assembler), задание констант, а не точных цифр в коде, использование стандартизированных кросплатформенных библиотек (Qt, STL), написание легко переносимого интерфейса (лучше всего консольного или простого). Минимальные затраты памяти (любой), использование минимального числа регистров и т.д.
A2-2: универсальность кода, возможно - реализация фич/поддержки различных систем, процессоров разбитие системы на подмодули для упрощения замены

Q3: Tabulate the advantages and disadvantages of using programming languages as follows: Machine coding, Assembly coding, C, C++, Java
A3:

ЯП	+	-
Machine coding	+Очень высокая производительность +Легко управлять ресурсами	- Громоздкость (количество) кода - Огромная трудоёмкость
Assembly coding	+ Очень высокая скорость работы + Легко управлять ресурсами	- Сложно для чтения и понимания структуры - Требование высококачественных знаний - Большая трудоёмкость - Невозможно портировать
C	+ Простота изучения + Высокая скорость разработки + Возможность управления ресурсами + Нет необходимости глубоко изучать данные железа + Проще портировать программы + Наличие библиотек	- Сложность углубленного изучения
C++	+ Использование ООП + Масштабируемость	- Немного медленнее C
Java	+ Не нужно много знать об используемом процессоре и ОС. + Простота переноса на другую систему	- Низкая производительность - Большой размер кода - Мало низкоуровневого доступа

Q4: Make data and functions prototype in C for interface that handles packet received from remote side. Packet is received in little-endian. Interface maybe compiled to run on big-endian or little-endian CPU. Prototype shows what data structure is used to store received data and how it is filled

A4: I don't know, how to do it. But I can wrote big-endian and little-endian functions:

```cpp
#pragma pack(push,1)
template<typename T>
struct LittleEndian
{
    unsigned char bytes[sizeof(T)];

    LittleEndian(T t = T())
    {
        operator =(t);
    }

    LittleEndian(const LittleEndian<T> & t)
    {
        for (unsigned i = 0; i < sizeof(T); i++)
            bytes[i] = t.bytes[i];
    }

    LittleEndian(const BigEndian<T> & t)
    {
        for (unsigned i = 0; i < sizeof(T); i++)
            bytes[i] = t.bytes[sizeof(T)-1-i];
    }

    operator const T() const
    {
        T t = T();
        for (unsigned i = 0; i < sizeof(T); i++)
            t |= bytes[i] << (i << 3);
        return t;
    }

    const T operator = (const T & t)
    {
        for (unsigned i = 0; i < sizeof(T); i++)
            bytes[i] = t >> (i << 3);
        return t;
    }
};
#pragma pack(pop)
```

```cpp
#pragma pack(push,1)
template<typename T>
struct BigEndian
{
    unsigned char bytes[sizeof(T)];

    BigEndian(T t = T())
    {
        operator =(t);
    }

    BigEndian(const BigEndian<T> & t)
    {
        for (unsigned i = 0; i < sizeof(T); i++)
            bytes[i] = t.bytes[i];
    }

    BigEndian(const LittleEndian<T> & t)
    {
        for (unsigned i = 0; i < sizeof(T); i++)
            bytes[i] = t.bytes[sizeof(T)-1-i];
    }

    operator const T() const
    {
        T t = T();
        for (unsigned i = 0; i < sizeof(T); i++)
            t |= bytes[sizeof(T) - 1 - i] << (i << 3);
        return t;
    }

    const T operator = (const T t)
    {
        for (unsigned i = 0; i < sizeof(T); i++)
            bytes[sizeof(T) - 1 - i] = t >> (i << 3);
        return t;
    }
};
#pragma pack(pop)
```

HW BASICS
HOME TASK

Q1: What peripherals does M16C provide? What SFR are used for every peripheral device?

Standard on-chip peripherals include 16-bit Multifunction Timers (incl. 3-phase inverter motor control function), UART/Clock Synchronous Serial Interface, 10-bit A/D Converter, DMACs, Watchdog Timer, Oscillation Stop Detection Function

S**IC (несколько на приём-отправку) (UART)
T**IC (контроль прерываний) и T** (регистры) (A0-A4 B0-B2) (таймеры)
AD* (0-7) A/D register

Q2: What exceptions does M16C provide? When are they generated?

CPU exception, Task Exception
Ну, или если провести аналогию с ARM, то, логично предположить, что должны быть что-то типа Exception: Reset, Data Access Memory Abort, Software Interrupt, Undefined Instruction.

Q3: What interrupts does M16C provide? Compare list of interrupts with M16C peripherals and SFRs.

Interrupt source	Vector address. Addr (L) to addr (H)	Peripheral	SFR Address
CAN0 wakeup(3)	+4 to +7 (000416 to 000716)	Controller Area Network	0041
CAN0 receive completion	+8 to +11 (000816 to 000B16)		0042
CAN0 transmit completion	+12 to +15 (000C16 to 000F16)		0043
INT3	+16 to +19 (001016 to 001316)	INT3	0044
UART2 transmit, NACK2 (8)	+60 to +63 (003C16 to 003F16)	UART	004F
UART2 receive, ACK2 (8)	+64 to +67 (004016 to 004316)		0050
UART0 transmit	+68 to +71 (004416 to 004716)		0051
UART0 receive	+72 to +75 (004816 to 004B16)		0052
UART1 transmit	+76 to +79 (004C16 to 004F16)		0053
UART1 receive	+80 to +83 (005016 to 005316)		0054
Timer A0	+84 to +87 (005416 to 005716)	Timer	0055
Timer A1	+88 to +91 (005816 to 005B16)		0056
Timer A2	+92 to +95 (005C16 to 005F16)		0057
Timer A3	+96 to +99 (006016 to 006316)		0058
Timer A4	+100 to +103 (006416 to 006716)		0059
Timer B0	+104 to +107 (0068 16 to 006B16)		005A
Timer B1	+108 to +111 (006C 16 to 006F16)		005B
Timer B2	+112 to +115 (007016 to 007316)		005C

Q4: Implement algorithm for sending/receiving data buffer via UART for M16C. Try two options: polling-driven and interrupt-driven.

Просто какой-то алгоритм для работы с COM портом (Он же UART)

```
procedure TForm1.sendstring();
var s:string;
   i:integer;
begin
if sEdit7.Text<>'' then begin
 if not sCheckBox2.Checked then begin
   if sCheckBox1.Checked then begin
     CommPortDriver1.SendString(sEdit7.Text+char($0D)+char($0A));
   end
   else CommPortDriver1.SendString(sEdit7.Text);
 end
 else begin
   s:='';
   sEdit7.Text :=  StringReplace(sEdit7.Text,' ','',[rfReplaceAll]);
     for i := 1 to length(sEdit7.Text) div 2 do begin
       s :=  s + Char(StrToInt('$'+Copy(sEdit7.Text,(i-1)*2+1,2)));
     end;
     if sCheckBox1.Checked then begin
       CommPortDriver1.SendString(s+char($0D)+char($0A));
     end
```

```
    else begin
      CommPortDriver1.SendString(s);
    end
  end;
  if sListBox1.Items[0]<> sEdit7.Text then begin
    sListBox1.Items.Insert(0,sEdit7.Text);
    if sListBox1.Items.Count-1=150 then begin
      sListBox1.Items.Delete(sListBox1.Items.Count-1);
    end;
  end;
end
else begin
  Beep();
  sEdit7.SetFocus;
end;
end;
```

Принимаем данные с ком порта

Код Delphi

```
1   procedure TForm1.CommPortDriver1ReceiveData(Sender: TObject;
2     DataPtr: Pointer; DataSize: Cardinal);
3   var i : Integer;
4       s : string;
5   begin
6     i:=  0;
7   For i:=0 to (DataSize) - 1 do begin
8     if not sCheckBox2.Checked then
9       s:=s+(PChar(DataPtr)[i])//Если надо в виде ASCII
10    else
11      s:=s+' '+StringtoHex(PChar(DataPtr)[i]);//Если надо в виде HEX
12  end;
13  sMemo2.Lines.Add(s);
14  end;
```

Отправка данных в ком порт :

Код Delphi

```
1  ...
2  sEdit7.Text :=  StringReplace(sEdit7.Text,' ','',[rfReplaceAll]);
3  for i := 1 to length(sEdit7.Text) div 2 do begin
4   s :=  s + Char(StrToInt('$'+Copy(sEdit7.Text,(i-1)*2+1,2)));
5  end;
6   CommPortDriver1.SendString(s+char($0D)+char($0A));
7  ....
```

Development BASICS
HOME TASK

---

1. Imagine you are a RTOS developer. Your task is to implement preemptive OS. What hardware peripherals for task switching algorithm shall you take into account? What information of running process/task should be saved? Explain your choice.
   Планировщик — часть операционной системы, которая отвечает за (псевдо)параллельное выполнения задач, потоков, процессов. Планировщик выделяет потокам процессорное время, память, стек и прочие ресурсы. Планировщик может принудительно забирать управление у

потока (например по таймеру или при появлении потока с бо**л**ьшим приоритетом), либо просто ожидать пока поток сам явно(вызовом некой системной процедуры) или неявно(по завершении) отдаст управление планировщику.
Первый вариант работы планировщика называется реальным или вытесняющим(preemptive), второй, соответственно, не вытесняющим (non-preemptive). Надо, например, сделать Watchdog'и, и, если необходимо, перезапускать процессы.

2. Select some RTOS or OS-less variant for task:
1. Cell-phone billing system
2. Radio-based home automation system

Prove your selection, what is the reason and important features?
Для Cell-phone billing system выбрать OS-less, так как надо просто отправлять некоторые данные через н-ный момент времени (ничего сложного, а грузить ОС на прибор для такой задачи – нецелесообразно, много лишних ресурсов hardware и software). А вот для какой-то автоматизированной системы лучше RTOS, так как ей надо делать какие-то минимальные вычисления и задачи (посложнее).

3. Short overview for RTOS QNX Neutrino, VxWorks, WinCE, mITRON4.0, Micro/OS-II

- Architecture
- Supported platforms
- Scheduling
- Kernel features
- Supported standards
- Development tools

Prepare a comparison table

	QNX Neutrino	VxWorks	WinCE	mITRON4.0	Micro/OS-II
Architecture	Клиент-сервер, микроядро и взаимодействующие процессы	Клиент-сервер, микроядро	Гибридное ядро	manufacturer's microprocessor	Microkernel
Supported platforms	x86, MIPS, PowerPC, SH-4, ARM, StrongARM и xScale.	x86 (including Intel Quark), x86-64, MIPS, PowerPC, SH-4, ARM	x86, MIPS, 32-bit ARM, (SuperH[4] up to 6.0 R2)	8-bit, 16-bit, or 32-bit	ARM Cortex-M3, ARM Cortex-M4F, ARM ARM7TDMI, Atmel AVR
Scheduling	FIFO scheduling round-robin scheduling sporadic scheduling.	• **preemptive priority-based scheduling (default)** • **round-robin scheduling**	Real-time deterministic task scheduling	round-robin scheduling	round robin scheduling.
Kernel features	Successive QNX microkernels have seen a reduction in the code required to implement a given kernel	The OS kernel is separate from middleware, applications and other packages,[9] which enables easier bug fixes and testing of	kernel which supports 32,768 processes, up from the 32 process limit of prior versions. Each process receives 2 GB of	The multi-level queue scheduler is required for strict µ ITRON conformance and it queues tasks in FIFO order, so requests to create an object with priority queueing of tasks (pk_cxxxxxxatr = TA_TPRI) are rejected with E_RSATR. Additional undefined bits in	It is a very small real-time kernel. Memory footprint is about 20KB for a fully functional kernel. Source code is

	call. The object definitions at the lowest layer in the kernel code have become more specific, allowing greater code reuse (such as folding various forms of POSIX signals, realtime signals, and QNX pulses into common data structures and code to manipulate those structures). At its lowest level, the microkernel contains a few fundamental objects and the highly tuned routines that manipulate them. The OS is built from this foundation.	new features.[15] An implementation of a layered source build system allows multiple versions of any stack to be installed at the same time so developers can select which version of any feature set should go into the VxWorks kernel libraries.	virtual address space, up from 32 MB.	the attributes fields must be zero. configuration option *CYGNUM_KERNEL_SYNCH_MBOX_QUEUE_SIZE;* therefore the buffer count field is not supported and is not in fact defined in type pk_cmbx. Queueing of messages is FIFO ordered only, so TA_MPRI (in pk_cmbx →mbxatr) is not supported.	written mostly in ANSI C. Highly portable, ROMable, very scalable, preemptive real-time, deterministic, multitasking kernel. It can manage up to 64 tasks (56 user tasks available). It has connectivity with µC/GUI and µC/FS (GUI and File Systems for µC/OS II). It is ported to more than 100 microprocessors and microcontrollers . It is simple to use and simple to implement but very effective compared to the price/performance ratio. It supports all type of processors from 8-bit to 64-bit.
Supported standards	compiler, linker, libraries and other QNX Neutrino components, precompiled for all CPU architectures that QNX Neutrino supports	standard board support package (BSP) interface between all its supported hardware and the OS. Wind River's BPS developer kit provides a common application programming interface (API) and a stable environment for real-time	Windows CE supports standard Windows-based desktop functions for serial communications. Use these functions and structures to do the following: Open, close, and manipulate serial ports Transmit and receive data	"Standard functionality" (level S), plus many "Extended" (level E) functions.	C, C++ (100%)

		operating system development.	Manage the connection		
Development tools	QNX Momentics IDE для Windows, Solaris, QNX4, QNX6 (до версии Neutrino 6.3.2 включительно), Linux	Tornado, Workbench	Visual Studio Free Pascal and Lazarus	High performance embedded workshop	The Nios II Software Build *Tools* (SBT), Freescale's Software *Development Kit* for Kinetis MCU

ITRON
HOME TASK

- **µITRON variable types**

  UB Unsigned 8-bit integer
  UH Unsigned 16-bit integer
  UW Unsigned 32-bit integer
  UD Unsigned 64-bit integer
  VB 8-bit value with unknown data type
  VH 16-bit value with unknown data type
  VW 32-bit value with unknown data type
  VD 64-bit value with unknown data type
  VP Pointer to an unknown data type
  FP Program start address (pointer)
  INT Signed integer whose size is suitable for the processor
  UINT Unsigned integer whose size is suitable for the processor
  BOOL Boolean value (TRUE or FALSE)

- **Make examples of deadlock and priority inversion using ITRON synchronization functions**

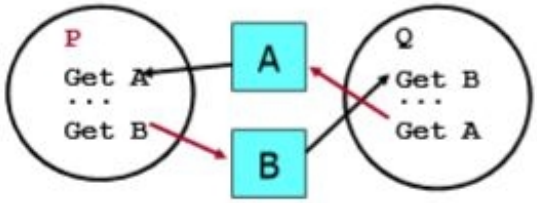
Тупик(deadlock) — это ситуация при которой 2 таска ожидают друг друга т.к. каждый должен отдать "жетон» нужный другому. Для примера:

1. Task A выполняется и захватывает «жетон» X.
2. Выполнение работы Task A прерывается Task B.
3. Task B захватывает «жетон» Y, перед этим пытаясь захватить «жетон» X, который сейчас используется Task A. В результате Task B впадает в ожидание Task A.
4. Task A продолжает выполняться, и хочет захватить «жетон» Y, который используется Task B и в результате тоже впадает в ожидание.

Для того чтобы избежать «тупиков» необходимо чтобы все таски получали «жетон» в определенном порядке, например, сначала таск A, потом B, и т.д.

Deadlock

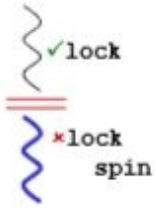
- Textbook definition: Set of threads blocked waiting for event that can only be caused by another thread in the same set
- Classic example:



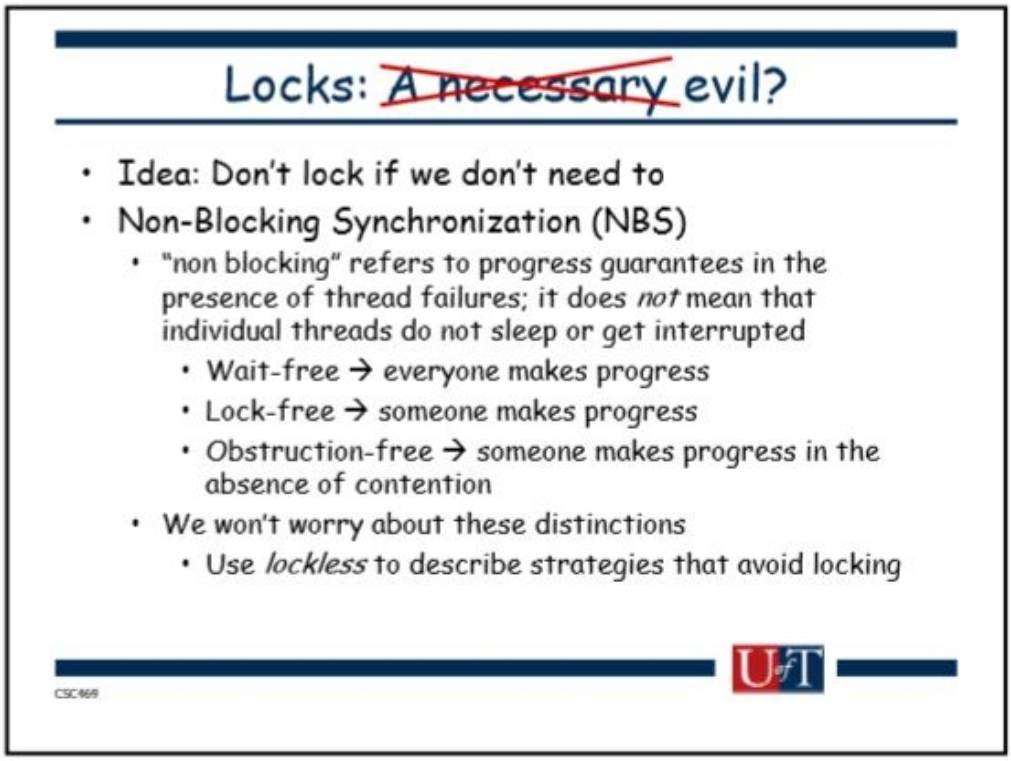
- Self-deadlock also a big issue
  - Thread holds lock on shared data structure and is interrupted
  - Interrupt handler needs same lock!
    - Solutions exist (e.g., disable interrupts while holding lock), but add complexity

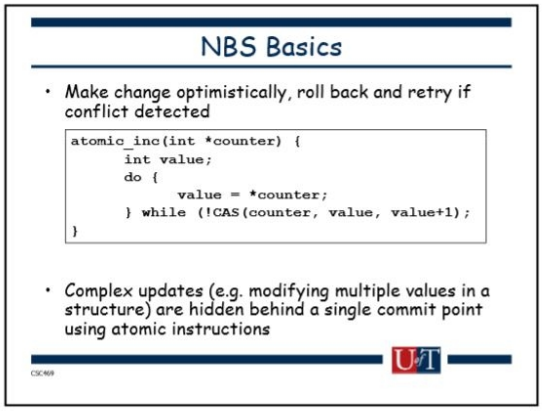
Priority Inversion

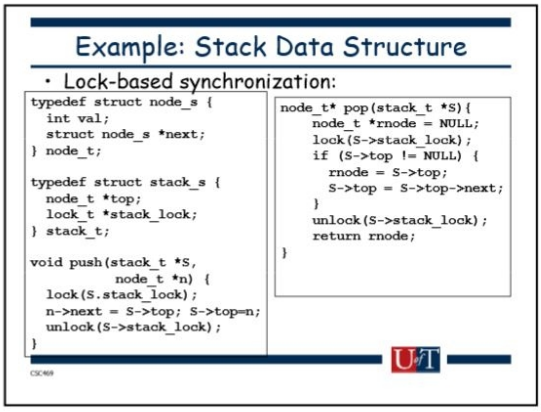
- Lower priority thread gets spinlock
- Higher priority thread becomes runnable and preempts it
  - needs lock, starts spinning
  - Lock holder can't run and release lock
    - May get to run on another CPU

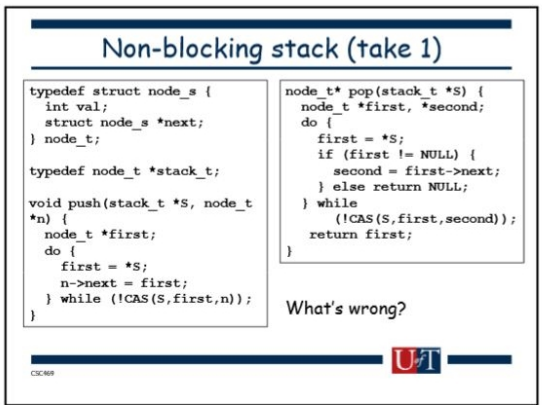


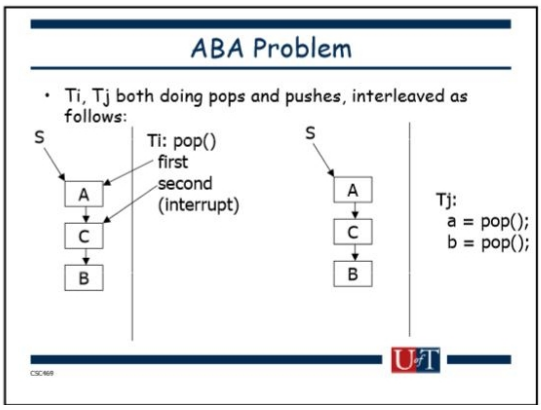
- Solutions exist (e.g. disable preemption while holding spinlock, implement priority inheritance, etc.), but add complexity











- **Mutex object – priority inversion avoidance in mutex objects**

отличие от регулярных мьютексов, мьютексы реального времени обеспечивают наследование приоритетов (priority inheritance, PI), что является одним из нескольких (немногих) известных способов, препятствующих возникновению инверсии приоритетов (priority inversion). Если RT мьютекс захвачен процессом A, и его пытается захватить процесс B (более высокого приоритета), то:

- процесс B блокируется и помещается в очередь ожидающих освобождения процессов wait_list (в описании структуры rt_mutex);
- при необходимости, этот список ожидающих процессов переупорядочивается в порядке приоритетов ожидающих процессов;
- приоритет владельца мьютекса (текущего выполняющегося процесса) B повышается до приоритета ожидающего процесса A (максимального приоритета из ожидающих в очереди процессов);
- это и обеспечивает избежание потенциальной инверсии приоритетов.

Priority inversion commonly happens in poorly designed real-time embedded applications. *Priority inversion* occurs when a higher priority task is blocked and is waiting for a resource being used by a lower priority task, which has itself been preempted by an unrelated medium-priority task. In this situation, the higher priority task's priority level has effectively been inverted to the lower priority task's level.

Enabling certain protocols that are typically built into mutexes can help avoid priority inversion. Two common protocols used for avoiding priority inversion include:

- **priority inheritance protocol**—ensures that the priority level of the lower priority task that has acquired the mutex is raised to that of the higher priority task that has requested the mutex when inversion happens. The priority of the raised task is lowered to its original value after the task releases the mutex that the higher priority task requires.
- **ceiling priority protocol**—ensures that the priority level of the task that acquires the mutex is automatically set to the highest priority of all possible tasks that might request that mutex when it is first acquired until it is released.

When the mutex is released, the priority of the task is lowered to its original value.