



Quantum[®] Leaps
innovating embedded systems

Application Note **μC/OS-II and** **Turbo C++ 1.01**

Document Version 1.0
April 2005

Quantum Leaps[™], LLC

www.quantum-leaps.com

Copyright © 2002-2005 Quantum Leaps, LLC. All Rights Reserved.

μC/OS-II

1 Introduction

This Application Note describes the port of μC/OS-II to the 80x86 CPU with the Borland Turbo C++ 1.01 compiler. The port is very similar to the original μC/OS-II port to 80x86 published in the book *MicroC/OS-II The Real-Time Kernel* by Jean Labrosse [Labrosse 02]. The book version requires Borland C++ 4.51, which is not free. Borland Turbo C++ 1.01, on the other hand, is available for a free download from the Borland "Museum" (<http://bdn.borland.com/article/0,1410,21751,00.html>). The port has been tested with μC/OS-II v2.52 and 2.60 available on the CD-ROM accompanying the book. No changes in the port are required to work with either version.

2 Installation

The archive `ucos-ii_tcpp101.zip` contains all the elements of the port. The archive is designed to "plug into" the directory structure used in the book distribution of μC/OS-II, and contains the following directories and files:

```
software
|
|--blocks/
|   |--PC/
|   |   |--tcpp101      - utilities for running μC/OS-II on a DOS PC
|   |   |--pc.h        - Borland Turbo C++ 1.01 version of the utilities
|   |   |--pc.c        - PC utilities interface
|   |   |--pc.c        - PC utilities implementation
|   |
|   |--ucos-ii/
|   |   |--EX1_x86L/   - example #1, as described in the "MicroC/OS-II" book
|   |   |--tcpp101/   - Borland Turbo C++ 1.01 version of the example
|   |   |--SOURCE
|   |   |   |--includes.h
|   |   |   |--os_cfg.h
|   |   |   |--test.c
|   |   |   |--test.rsp
|   |   |--TEST
|   |   |   |--MAKETEST.BAT
|   |   |   |--TEST.EXE
|   |   |   |--TEST.MAK
|   |   |   |--TEST.MAP
|   |
|   |--Ix86L/
|   |   |--tcpp101/   - μC/OS-II port to 80x86 CPU, Large memory model
|   |   |   |--os_cpu.h
|   |   |   |--os_cpu_a.asm
|   |   |   |--OS_CPU_A.OBJ
|   |   |   |--os_cpu_c.c
```

Listing 1 Contents of the port ZIP file

NOTE: this port does not contain the µC/OS-II source files or even the example source files. These files are available from the CD-ROM accompanying the µC/OS-II book [Labrosse 02].

All the files in the port have standard names for the µC/OS-II distribution and have been described in the book. To install the port you need to unzip the archive in the root directory on the drive where you've installed µC/OS-II. For example, if you installed µC/OS-II on drive C: then you need to unzip the uc-os-ii_tcpp101.zip port at the root of drive C:.

2.1 Downloading and Installing Borland Turbo C++ 1.01

The legacy Borland Turbo C++ 1.01 compiler is available for free downloads from the Borland "Museum" (<http://bdn.borland.com/article/0,1410,21751,00.html>). In addition, Borland provides a scanned image of the original "Turbo C++ User's Guide" documentation in PDF format (<http://bdn.borland.com/cbuilder/tsuite>). The User's Guide is for Borland C++ v3.0, but still largely applies to version 1.01.

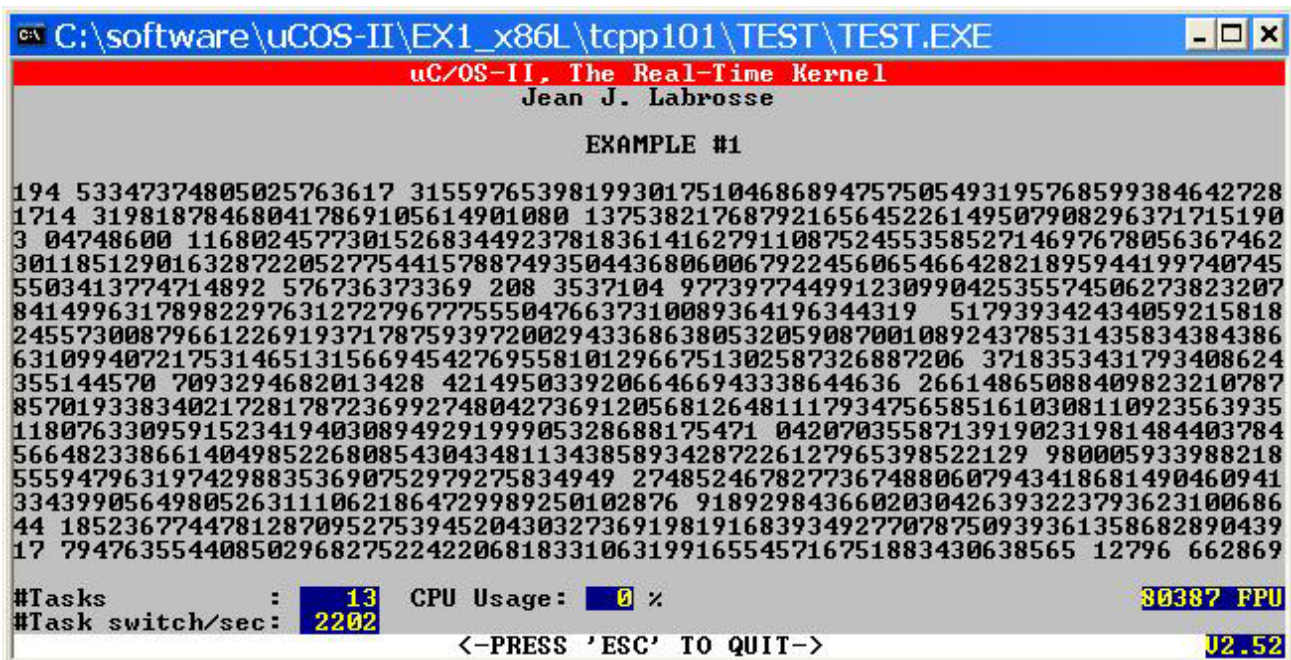
To install Borland Turbo C++ 1.01, unzip the TCPP101.ZIP archive from the Borland "Museum" onto your hard drive. Run the INSTALL.EXE program and follow the installation instructions to install the software. In this document I assume that Turbo C++ 1.01 has been installed in the directory C:\tools\tcpp101. If you choose a different directory, you'll need to modify the make file and the linker response file provided in this port.

2.2 Building the Example

To compile the example (EXAMPLE1) you need to follow exactly the instructions from Chapter 1 of the "MicroC/OS-II" book, except that you work in directory \software\EX1_x86L\tcpp101\test.

```
C:\software\uCOS-II\EX1_x86L\tcpp101\TEST>MAKETEST.BAT
```

Among others, this batch file creates the executable TEST.EXE in the TEST directory. You can run this executable on any DOS machine or in a DOS console on any Windows PC.



```
C:\software\uCOS-II\EX1_x86L\tcpp101\TEST\TEST.EXE
uC/OS-II, The Real-Time Kernel
Jean J. Labrosse

EXAMPLE #1

194 53347374805025763617 3155976539819930175104686894757505493195768599384642728
1714 319818784680417869105614901080 13753821768792165645226149507908296371715190
3 04748600 116802457730152683449237818361416279110875245535852714697678056367462
30118512901632872205277544157887493504436806006792245606546642821895944199740745
5503413774714892 576736373369 208 3537104 97739774499123099042535574506273823207
84149963178982297631272796775550476637310089364196344319 517939342434059215818
24557300879661226919371787593972002943368638053205908700108924378531435834384386
631099407217531465131566945427695581012966751302587326887206 3718353431793408624
355144570 7093294682013428 42149503392066466943338644636 26614865088409823210787
85701933834021728178723699274804273691205681264811179347565851610308110923563935
11807633095915234194030894929199905328688175471 04207035587139190231981484403784
5664823386614049852268085430434811343858934287226127965398522129 980005933988218
55594796319742988353690752979275834949 27485246782773674880607943418681490460941
334399056498052631110621864729989250102876 9189298436602030426393223793623100686
44 18523677447812870952753945204303273691981916839349277078750939361358682890439
17 7947635544085029682752242206818331063199165545716751883430638565 12796 662869

#Tasks : 13 CPU Usage: 0 % 80387 FPU
#Task switch/sec: 2202
<-PRESS 'ESC' TO QUIT-> U2.52
```

3 About The Port

One of the limitations of Turbo C++ 1.01 is the lack of the Turbo Assembler in the free distribution. To work around this shortcoming, this port avoids using in-line assembly, because inlining assembly in Turbo C++ 1.01 causes compilation via assembly, which cannot complete without the Turbo Assembler.

The only assembly language part of the port (OS_CPU_A.ASM) is available in source form and is discussed here, but this module has been pre-compiled with Turbo Assembler 3.0 and is included as an object file in this port.

3.1 OS_CPU.H Header File

Listing 2 shows the choice of the critical section method (OS_CRITICAL_METHOD), and task-to-task context switching method (OS_TASK_SW()).

```
#define OS_CRITICAL_METHOD 3
. . .
#if OS_CRITICAL_METHOD == 3
OS_CPU_SR OSCPUsaveSR(void);
void OSCPUrestoreSR(OS_CPU_SR cpu_sr);

#define OS_ENTER_CRITICAL() (cpu_sr = OSCPUsaveSR()) /* Disable interrupts */
#define OS_EXIT_CRITICAL() (OSCPUrestoreSR(cpu_sr)) /* Enable interrupts */
#endif
. . .
#define uCOS 0x80 /* Interrupt vector # used for context switch */
#define OS_TASK_SW() geninterrupt(uCOS)
. . .
```

Listing 2 Critical Section Method and Task-to-Task Context Switching.

This port demonstrates the most advanced Critical Section Method 3. Also, to avoid inlining assembly, the OS_TASK_SW() macro has been defined as the geninterrupt(uCOS) function call.

3.2 OS_CPU_A.ASM Assembly Language Module

Listing 3 shows the assembly implementation of the functions referred in OS_CPU.H header file. The file provides also Critical Section method 1 functions (OSCPUenable/ OSCPUdisable) as well as the Critical Section #3 functions (OSCPUsaveSR()/OSCPUrestoreSR). Turbo C++ 1.01 would allow for Critical Section #2 implementation, which in fact would be more efficient, but for pedagogical reasons I wanted to demonstrate the use of Critical Section #3.

```
. . .
PUBLIC _OSCPUtaskSw
PUBLIC _OSCPUdisable
PUBLIC _OSCPUenable
PUBLIC _OSCPUsaveSR
PUBLIC _OSCPUrestoreSR
. . .
;*****
;
```

```

; void OSCPUisable(void)
;
;*****
_OSCPUisable PROC FAR
    CLI                ; clear the I flag
    RET                ; return to the caller
_OSCPUisable ENDP

;*****
;
; void OSCPUisable(void)
;
;*****

_OSCPUenable PROC FAR
    STI                ; set the I flag
    RET                ; return to the caller
_OSCPUenable ENDP

;*****
;
; int OSCPUsaveSR(void)
;
;*****

_OSCPUsaveSR PROC FAR
    PUSHF              ; push the flags
    POP AX             ; pop the flags into the return value AX
    CLI                ; clear the I flag
    RET                ; return to the caller
_OSCPUsaveSR ENDP

;*****
;
; void OSCPUrestoreSR(int key)
;
;*****

_OSCPUrestoreSR PROC FAR
    PUSH BP
    MOV BP,SP          ; establish frame pointer
    MOV AX, WORD PTR[BP+6] ; get the CPU_SR into AX
    PUSH AX            ; push the CPU_SR on the stack
    POPF               ; restore the CPU_SR from the stack
    POP BP             ; cleanup the frame pointer
    RET                ; return to the caller
_OSCPUrestoreSR ENDP
. . .

```

Listing 3 Added assembly functions to OS_CPU_A.ASM

4 References

[Labrosse 02]	Labrosse, Jean, J., "MicroC/OS-II The Real-Time Kernel, Second Edition", CMP Books 2002.
[Borland 92]	Borland, Inc. "Turbo C++ 3.0 User's Guide"