**ASSIGNMENT - 3**

**EMBEDDED SYSTEM AND DESIGN**

**PRIYANSHU SHARMA**

**15BCE1282**

**“DEVELOPING AND CONFIGURING THE eCos SYSTEM”**

**1. What is eCos?**

**ANSWER -** The **Embedded Configurable Operating System** (**eCos**) is a [free and open source](https://en.wikipedia.org/wiki/Free_and_open_source_software" \o "Free and open source software) [real-time operating system](https://en.wikipedia.org/wiki/Real-time_operating_system" \o "Real-time operating system) intended for [embedded systems](https://en.wikipedia.org/wiki/Embedded_system" \o "Embedded system) and applications which need only one [process](https://en.wikipedia.org/wiki/Process_(computing)" \o "Process (computing)) with [multiple threads](https://en.wikipedia.org/wiki/Thread_(computing)" \o "Thread (computing)). It is designed to be customizable to precise application requirements of run-time performance and hardware needs. It is implemented in [C](https://en.wikipedia.org/wiki/C_(programming_language)" \o "C (programming language))/[C++](https://en.wikipedia.org/wiki/C++" \o "C++) and has [compatibility layers](https://en.wikipedia.org/wiki/Compatibility_layer" \o "Compatibility layer) and [application programming interfaces](https://en.wikipedia.org/wiki/Application_programming_interface" \o "Application programming interface) for [POSIX](https://en.wikipedia.org/wiki/POSIX" \o "POSIX) and u[ITRON](https://en.wikipedia.org/wiki/ITRON_project" \o "ITRON project).

**DESIGN**

eCos was designed for devices with memory sizes in the range of a few tens or several hundred [kilobytes](https://en.wikipedia.org/wiki/Kilobyte" \o "Kilobyte), or for applications with real-time requirements.

eCos runs on a wide variety of hardware platforms, including [ARM](https://en.wikipedia.org/wiki/ARM_architecture" \o "ARM architecture), [CalmRISC](https://en.wikipedia.org/w/index.php?title=CalmRISC&action=edit&redlink=1" \o "CalmRISC (page does not exist)), [FR-V](https://en.wikipedia.org/wiki/FR-V" \o "FR-V), [Hitachi H8](https://en.wikipedia.org/wiki/Hitachi_H8" \o "Hitachi H8), [IA-32](https://en.wikipedia.org/wiki/IA-32" \o "IA-32), [Motorola 68000](https://en.wikipedia.org/wiki/Motorola_68000" \o "Motorola 68000), [Matsushita AM3x](https://en.wikipedia.org/w/index.php?title=Matsushita_AM3x&action=edit&redlink=1" \o "Matsushita AM3x (page does not exist)), [MIPS](https://en.wikipedia.org/wiki/MIPS_architecture" \o "MIPS architecture), [NEC V8xx](https://en.wikipedia.org/w/index.php?title=NEC_V8xx&action=edit&redlink=1" \o "NEC V8xx (page does not exist)), [Nios II](https://en.wikipedia.org/wiki/Nios_II" \o "Nios II), [PowerPC](https://en.wikipedia.org/wiki/PowerPC" \o "PowerPC), [SPARC](https://en.wikipedia.org/wiki/SPARC" \o "SPARC), and [SuperH](https://en.wikipedia.org/wiki/SuperH" \o "SuperH).

The eCos distribution includes [RedBoot](https://en.wikipedia.org/wiki/RedBoot" \o "RedBoot), an [open source](https://en.wikipedia.org/wiki/Open_source" \o "Open source) [application](https://en.wikipedia.org/wiki/Application_software" \o "Application software) that uses the eCos [hardware abstraction layer](https://en.wikipedia.org/wiki/Hardware_abstraction_layer" \o "Hardware abstraction layer) to provide [bootstrap](https://en.wikipedia.org/wiki/Booting" \o "Booting) [firmware](https://en.wikipedia.org/wiki/Firmware" \o "Firmware) for [embedded systems](https://en.wikipedia.org/wiki/Embedded_system" \o "Embedded system).

**STEPS TO BUILD -**

1. Install Virtual Box and boot Ubuntu 16.04 LTS in it.
2. After setting up the Ubuntu, open the terminal start downloading the packages -

#sudo -i

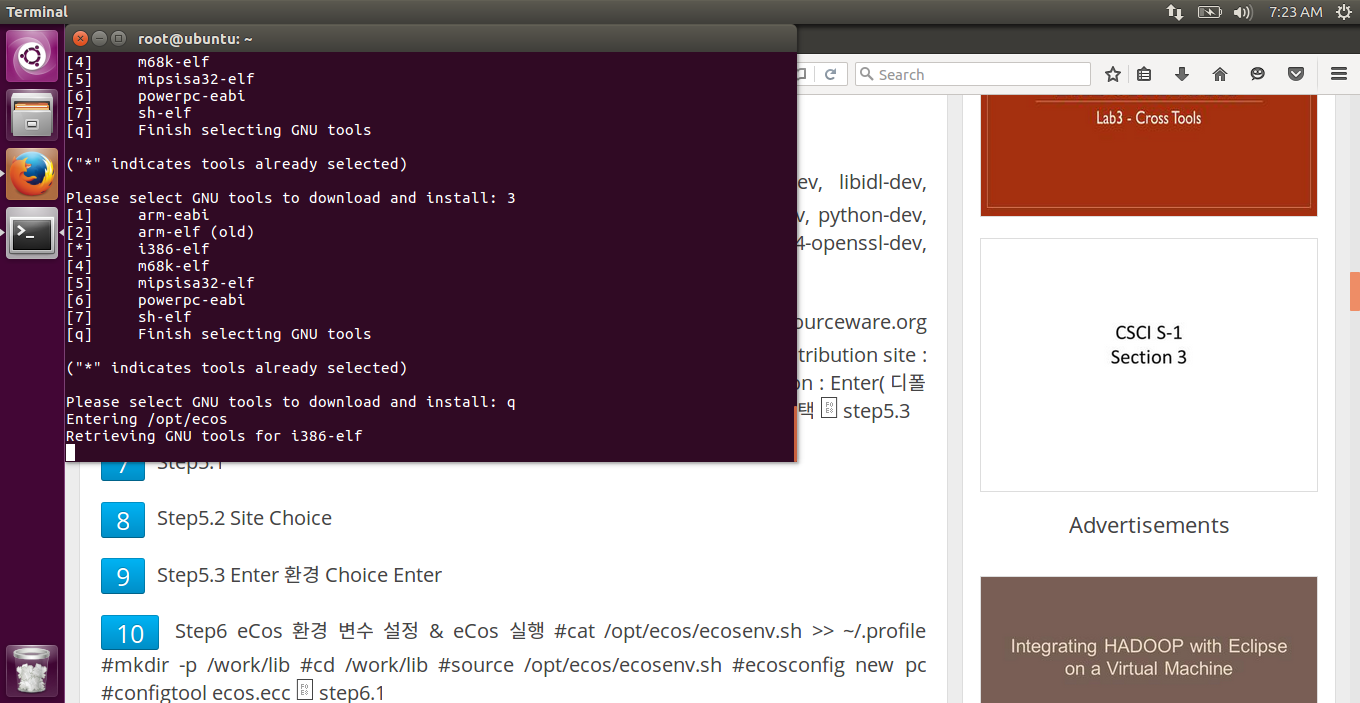
With this you will be having root access.

#sudo apt-get install bcc, iasl, uuid-dev, zlib1g-dev, libidl-dev, libsdl1.2-dev, libxcursor-dev, libstdc++5, libhal-dev, libxml2-dev, libxslt1-dev, python-dev, libqt4-dev, qt4-dev-tools, libcap-dev, libxmu-dev, libpam0g-dev, libcurl4-openssl-dev, libxrandr-dev, libxinerama-dev, makeself, tk texlive-latex-base, grub

1. Download the eCos and its configuration files on the host system.

#wget --passive-ftp ftp://ecos.sourceware.org/pub/ecos/ecos-install.tcl

#sh ecos-install.tcl



After this, Please select a distribution site, please select a directory for installation, please select GNU tools to download and install.

1. Configure the eCos tool

#cat /opt/ecos/ecosenv.sh >> ~/.profile

#mkdir -p /work/lib

#cd /work/lib

#source /opt/ecos/ecosenv.sh

#ecosconfig new pc

#configtool ecos.ecc

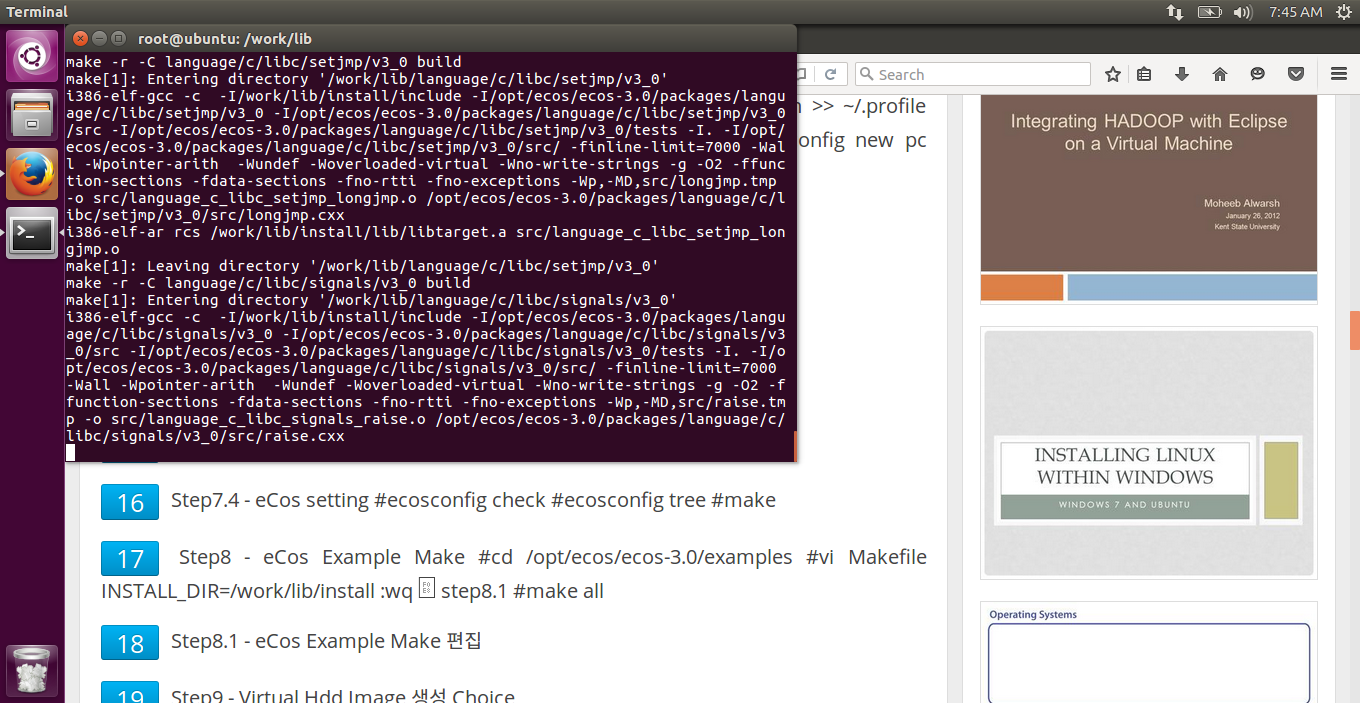
After this dialog box will appear, in that click on Build->Templates->select i386 PC target with RTL8139 ethernet->Choice->GRUB->select channel = 2->save->exit

1. Configuration checking

#ecosconfig check

#ecosconfig tree

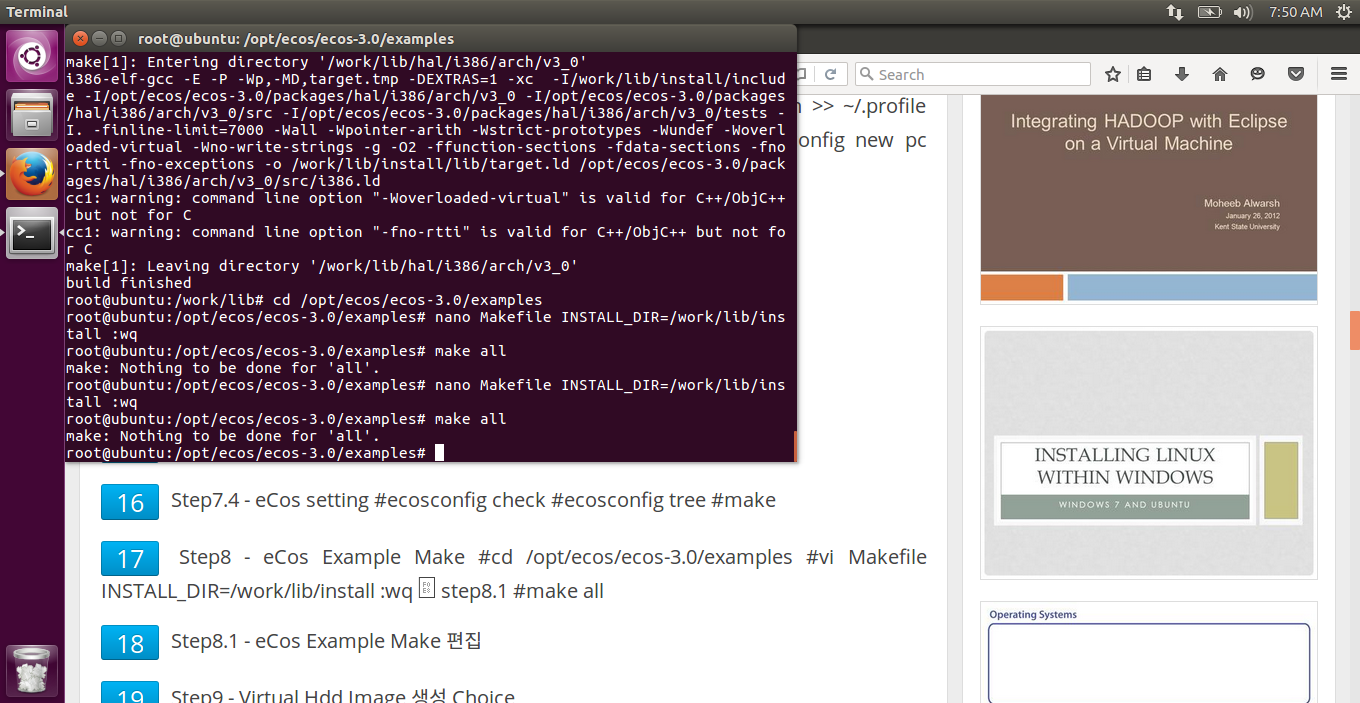
#make



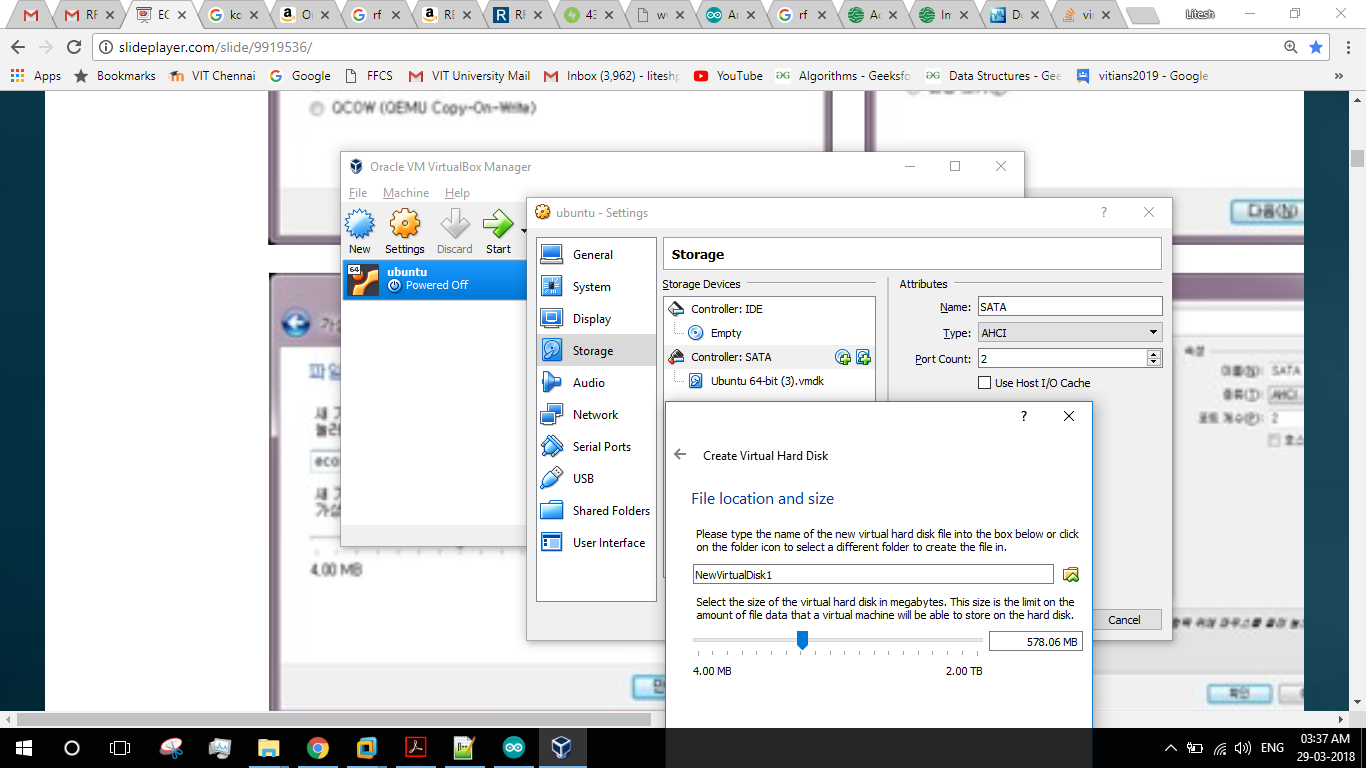
#cd /opt/ecos/ecos-3.0/examples

#nano Makefile INSTALL\_DIR=/work/lib/install :wq 

#make all



1. Shutdown the system and Create Virtual Hdd Image Choice



7. Now boot the Ubuntu again and open terminal to alter the partition table and to mount the newly created virtual partition.

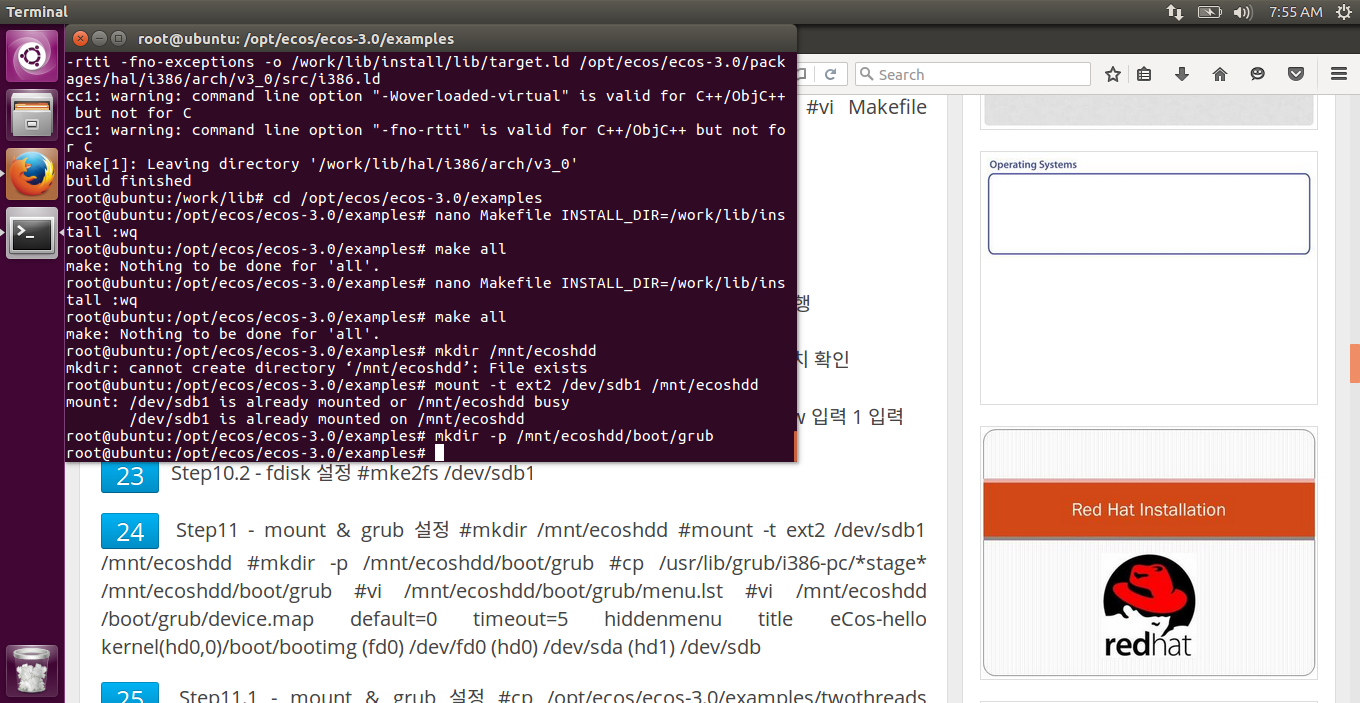
#fdisk/dev/sdb

#mke2fs /dev/sdb1

#mkdir /mnt/ecoshdd

#mount -t ext2 /dev/sdb1 /mnt/ecoshdd

#mkdir -p /mnt/ecoshdd/boot/grub



#cp /usr/lib/grub/x86\_64/ /mnt/ecoshdd/boot/grub

#nano /mnt/ecoshdd/boot/grub/menu.lst

default=0

timeout=5

hiddenmenu

title eCos-hello

kernel (hd0,0)/boot/bootimg

#nano/mnt/ecoshdd/boot/grub/device.map

(fd0) /dev/fd0

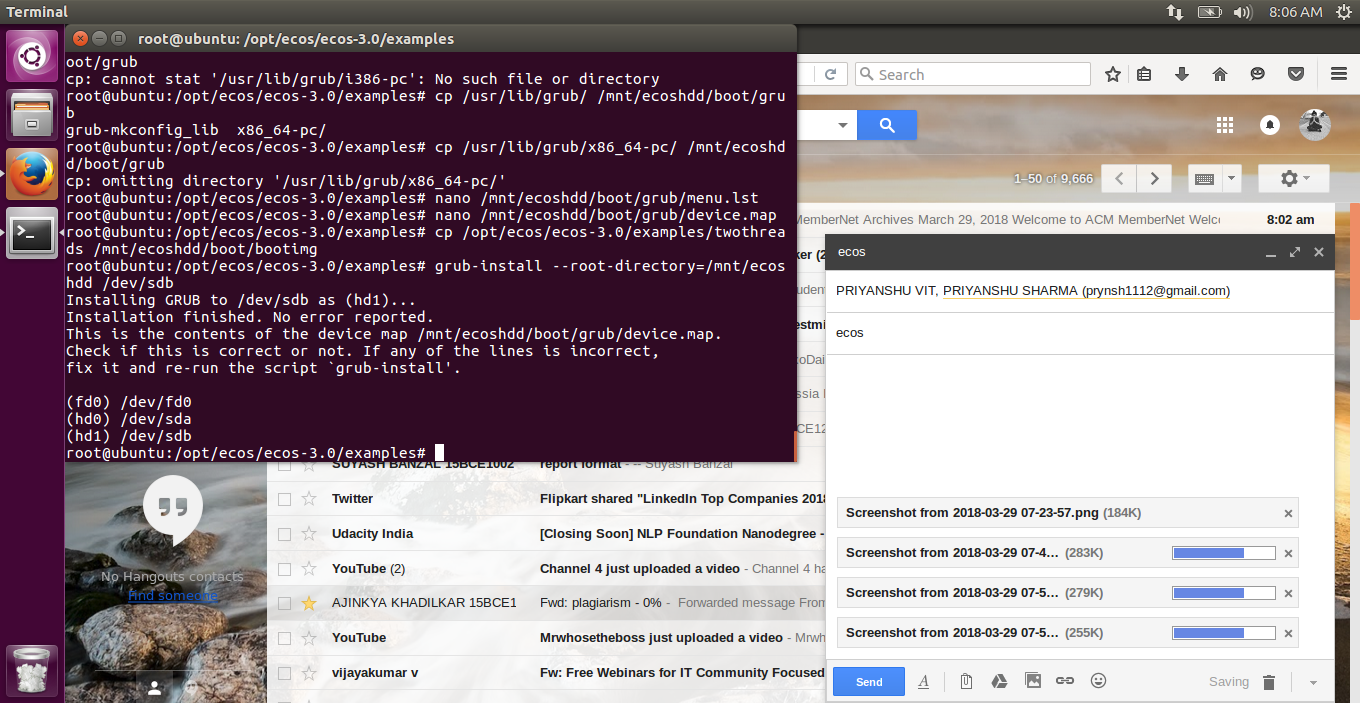
(hd0) /dev/sda

(hd1) /dev/sdb

1. Grub installation

#cp /opt/ecos/ecos-3.0/examples/hello /mnt/ecoshdd/boot/bootimg

#grub-install --root-directory=/mnt/ecoshdd /dev/sdb



1. Now delete the host Ubuntu
2. Reboot the virtual machine

**OUTPUT**

**CODE -**

**Hello Program**

/\* this is a simple hello world program \*/

#include <stdio.h>

int main(void)

{

printf("Hello, eCos world!\n");

printf("Priyanshu Sharma\n");

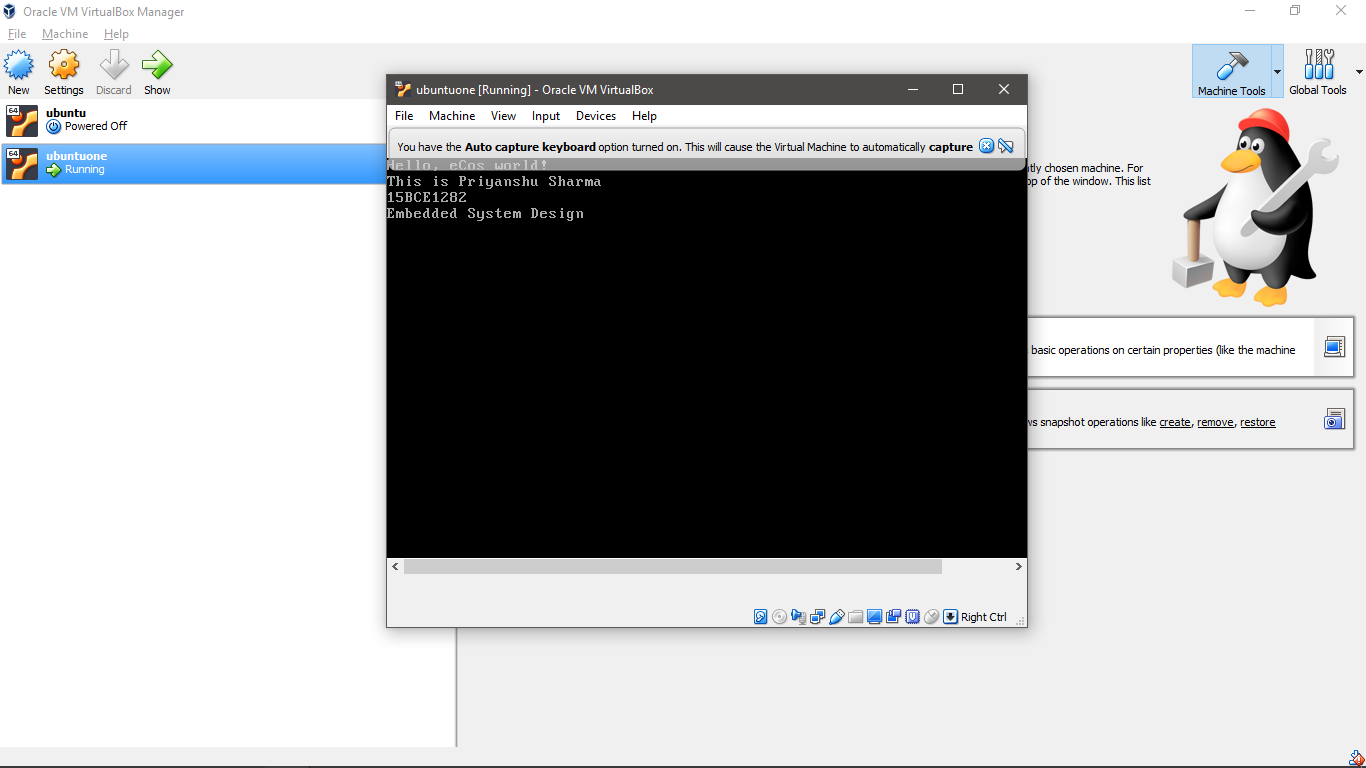
printf("15BCE1282\n");

printf("Embedded Syatem and Design\n");

return 0;

}

**OUTPUT**

****

**Two Threads Program**

#cp /opt/ecos/ecos-3.0/examples/twothreads /mnt/ecoshdd/boot/bootimg

#grub-install --root-directory=/mnt/ecoshdd /dev/sdb

**CODE**

#include <cyg/kernel/kapi.h>

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

/\* now declare (and allocate space for) some kernel objects,

like the two threads we will use \*/

cyg\_thread thread\_s[2]; /\* space for two thread objects \*/

char stack[2][4096]; /\* space for two 4K stacks \*/

/\* now the handles for the threads \*/

cyg\_handle\_t simple\_threadA, simple\_threadB;

/\* and now variables for the procedure which is the thread \*/

cyg\_thread\_entry\_t simple\_program;

/\* and now a mutex to protect calls to the C library \*/

cyg\_mutex\_t cliblock;

/\* we install our own startup routine which sets up threads \*/

void cyg\_user\_start(void)

{

printf("Entering twothreads' cyg\_user\_start() function\n");

cyg\_mutex\_init(&cliblock);

cyg\_thread\_create(4, simple\_program, (cyg\_addrword\_t) 0,

"Thread A", (void \*) stack[0], 4096,

&simple\_threadA, &thread\_s[0]);

cyg\_thread\_create(4, simple\_program, (cyg\_addrword\_t) 1,

"Thread B", (void \*) stack[1], 4096,

&simple\_threadB, &thread\_s[1]);

cyg\_thread\_resume(simple\_threadA);

cyg\_thread\_resume(simple\_threadB);

}

/\* this is a simple program which runs in a thread \*/

void simple\_program(cyg\_addrword\_t data)

{

int message = (int) data;

int delay;

printf("Beginning execution; thread data is %d\n", message);

cyg\_thread\_delay(200);

for (;;) {

delay = 200 + (rand() % 50);

/\* note: printf() must be protected by a

call to cyg\_mutex\_lock() \*/

cyg\_mutex\_lock(&cliblock); {

printf("Thread %d: and now a delay of %d clock ticks\n",

message, delay);

}

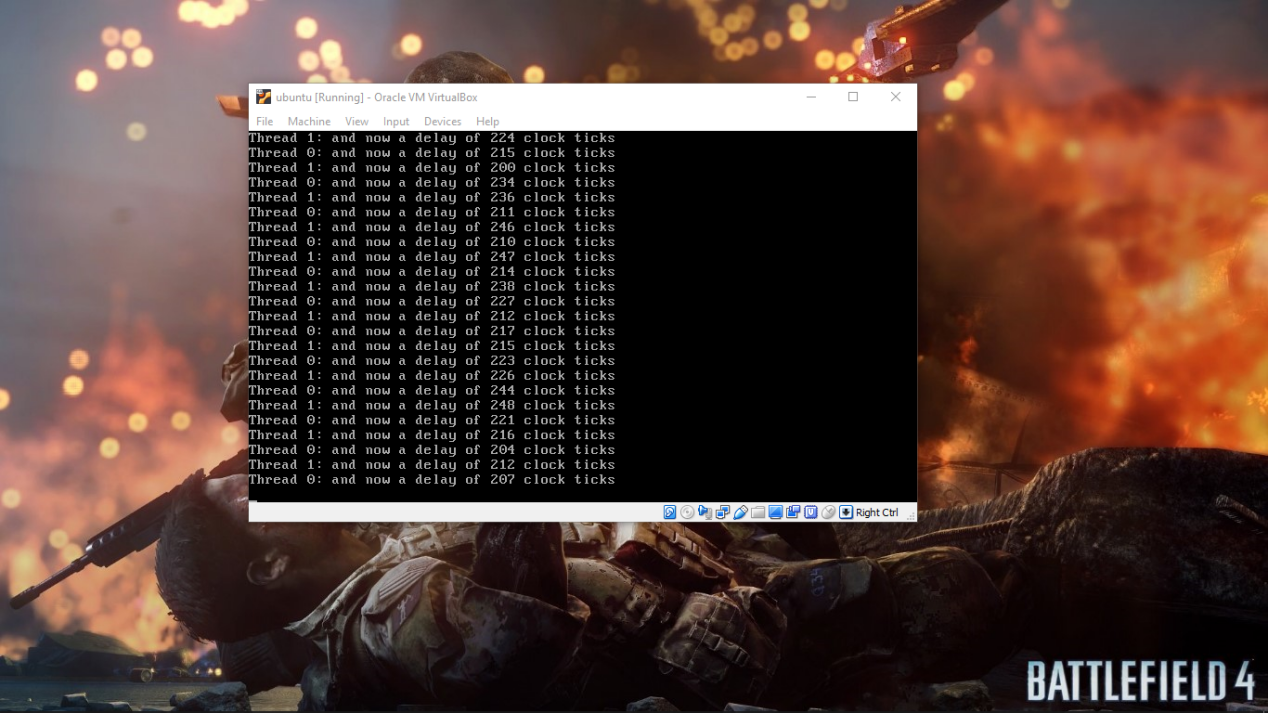
cyg\_mutex\_unlock(&cliblock);

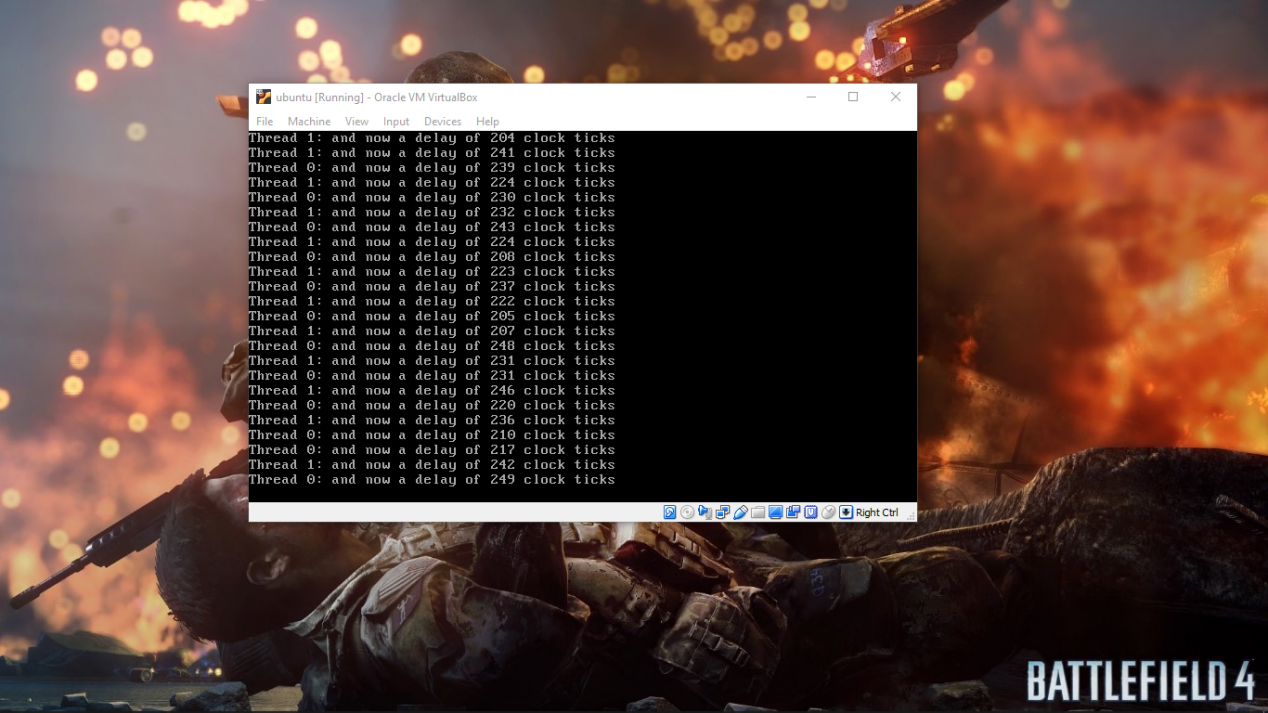
cyg\_thread\_delay(delay);

}

}

**OUTPUT**





**Simple Alarm Program**

#cp /opt/ecos/ecos-3.0/examples/simple-alarm /mnt/ecoshdd/boot/bootimg

#grub-install --root-directory=/mnt/ecoshdd /dev/sdb

**CODE**

/\* this is a very simple program meant to demonstrate

a basic use of time, alarms and alarm-handling functions

in eCos \*/

#include <cyg/kernel/kapi.h>

#include <stdio.h>

#define NTHREADS 1

#define STACKSIZE 4096

static cyg\_handle\_t thread[NTHREADS];

static cyg\_thread thread\_obj[NTHREADS];

static char stack[NTHREADS][STACKSIZE];

static void alarm\_prog( cyg\_addrword\_t data );

/\* we install our own startup routine which sets up

threads and starts the scheduler \*/

void cyg\_user\_start(void)

{

cyg\_thread\_create(4, alarm\_prog, (cyg\_addrword\_t) 0,

"alarm\_thread", (void \*) stack[0],

STACKSIZE, &thread[0], &thread\_obj[0]);

cyg\_thread\_resume(thread[0]);

}

/\* we need to declare the alarm handling function (which is

defined below), so that we can pass it to

cyg\_alarm\_initialize() \*/

cyg\_alarm\_t test\_alarm\_func;

/\* alarm\_prog() is a thread which sets up an alarm which is then

handled by test\_alarm\_func() \*/

static void alarm\_prog(cyg\_addrword\_t data)

{

cyg\_handle\_t test\_counterH, system\_clockH, test\_alarmH;

cyg\_tick\_count\_t ticks;

cyg\_alarm test\_alarm;

unsigned how\_many\_alarms = 0, prev\_alarms = 0, tmp\_how\_many;

system\_clockH = cyg\_real\_time\_clock();

cyg\_clock\_to\_counter(system\_clockH, &test\_counterH);

cyg\_alarm\_create(test\_counterH, test\_alarm\_func,

(cyg\_addrword\_t) &how\_many\_alarms,

&test\_alarmH, &test\_alarm);

cyg\_alarm\_initialize(test\_alarmH, cyg\_current\_time()+200, 200);

/\* get in a loop in which we read the current time and

print it out, just to have something scrolling by \*/

for (;;) {

ticks = cyg\_current\_time();

printf("Time is %llu\n", ticks);

/\* note that we must lock access to how\_many\_alarms, since the

alarm handler might change it. this involves using the

annoying temporary variable tmp\_how\_many so that I can keep the

critical region short \*/

cyg\_scheduler\_lock();

tmp\_how\_many = how\_many\_alarms;

cyg\_scheduler\_unlock();

if (prev\_alarms != tmp\_how\_many) {

printf(" ---> alarm calls so far: %u\n", tmp\_how\_many);

prev\_alarms = tmp\_how\_many;

}

cyg\_thread\_delay(30);

}

}

/\* test\_alarm\_func() is invoked as an alarm handler, so

it should be quick and simple. in this case it increments

the data that is passed to it. \*/

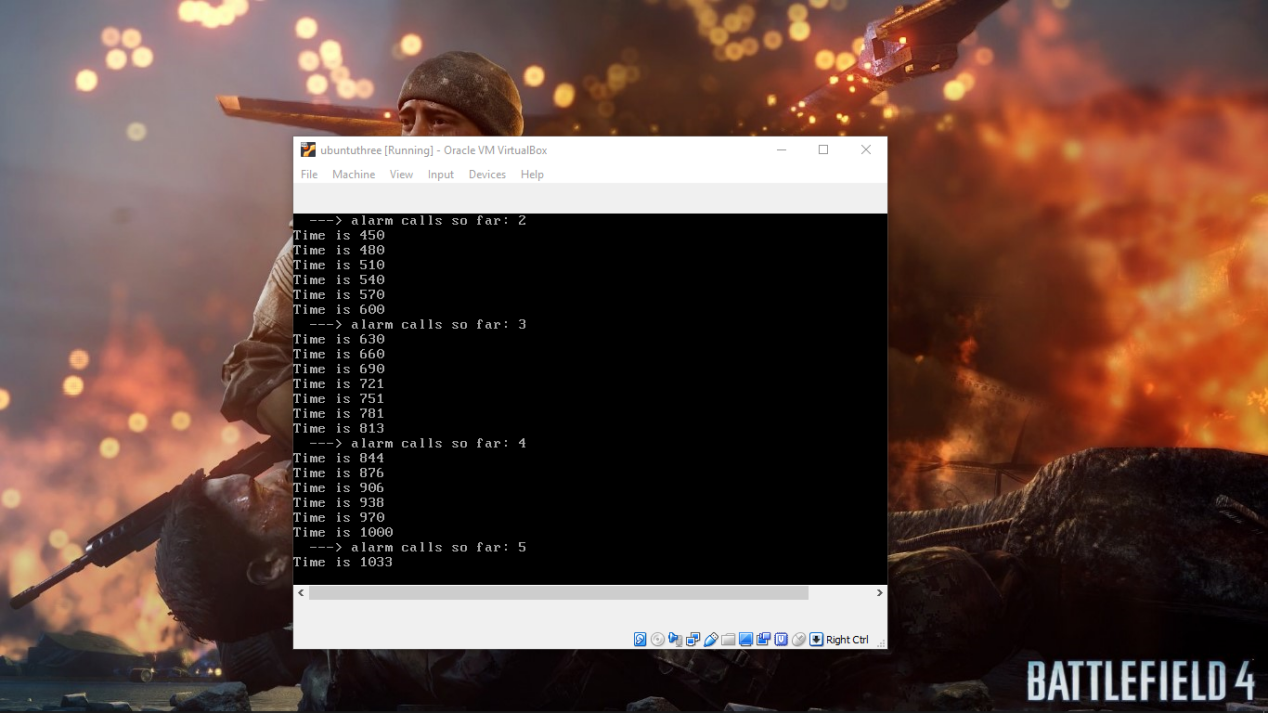
void test\_alarm\_func(cyg\_handle\_t alarmH, cyg\_addrword\_t data)

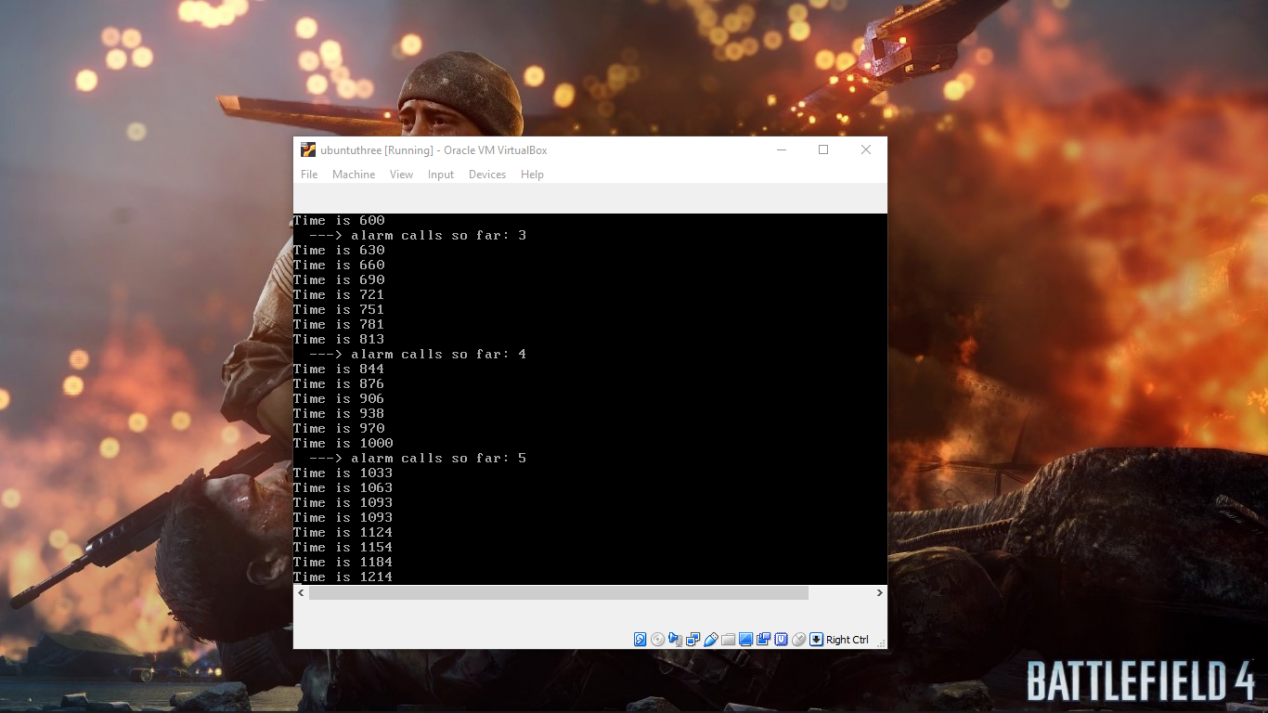
{

++\*((unsigned \*) data);

}

**OUTPUT**





**Serial Program**

#cp /opt/ecos/ecos-3.0/examples/serial /mnt/ecoshdd/boot/bootimg

#grub-install --root-directory=/mnt/ecoshdd /dev/sdb

**CODE**

/\* CONFIGURATION CHECKS \*/

#include <pkgconf/system.h> /\* which packages are enabled/disabled \*/

#ifdef CYGPKG\_KERNEL

# include <pkgconf/kernel.h>

#endif

#ifdef CYGPKG\_LIBC

# include <pkgconf/libc.h>

#endif

#ifdef CYGPKG\_IO\_SERIAL

# include <pkgconf/io\_serial.h>

#endif

#ifndef CYGFUN\_KERNEL\_API\_C

# error Kernel API must be enabled to build this example

#endif

#ifndef CYGPKG\_LIBC\_STDIO

# error C library standard I/O must be enabled to build this example

#endif

#ifndef CYGPKG\_IO\_SERIAL\_HALDIAG

# error I/O HALDIAG pseudo-device driver must be enabled to build this example

#endif

/\* INCLUDES \*/

#include <stdio.h> /\* printf \*/

#include <string.h> /\* strlen \*/

#include <cyg/kernel/kapi.h> /\* All the kernel specific stuff \*/

#include <cyg/io/io.h> /\* I/O functions \*/

#include <cyg/hal/hal\_arch.h> /\* CYGNUM\_HAL\_STACK\_SIZE\_TYPICAL \*/

/\* DEFINES \*/

#define NTHREADS 1

#define STACKSIZE ( CYGNUM\_HAL\_STACK\_SIZE\_TYPICAL + 4096 )

/\* STATICS \*/

static cyg\_handle\_t thread[NTHREADS];

static cyg\_thread thread\_obj[NTHREADS];

static char stack[NTHREADS][STACKSIZE];

/\* FUNCTIONS \*/

static void simple\_prog(CYG\_ADDRESS data)

{

cyg\_io\_handle\_t handle;

Cyg\_ErrNo err;

const char test\_string[] = "serial example is working correctly!\n";

cyg\_uint32 len = strlen(test\_string);

printf("Starting serial example\n");

err = cyg\_io\_lookup( "/dev/haldiag", &handle );

if (ENOERR == err) {

printf("Found /dev/haldiag. Writing string....\n");

err = cyg\_io\_write( handle, test\_string, &len );

}

if (ENOERR == err) {

printf("I think I wrote the string. Did you see it?\n");

}

printf("Serial example finished\n");

}

void cyg\_user\_start(void)

{

cyg\_thread\_create(4, simple\_prog, (cyg\_addrword\_t) 0, "serial",

(void \*)stack[0], STACKSIZE, &thread[0], &thread\_obj[0]);

cyg\_thread\_resume(thread[0]);

}

**OUTPUT**

