**Topic Exploration Pack**

**Systems Software**

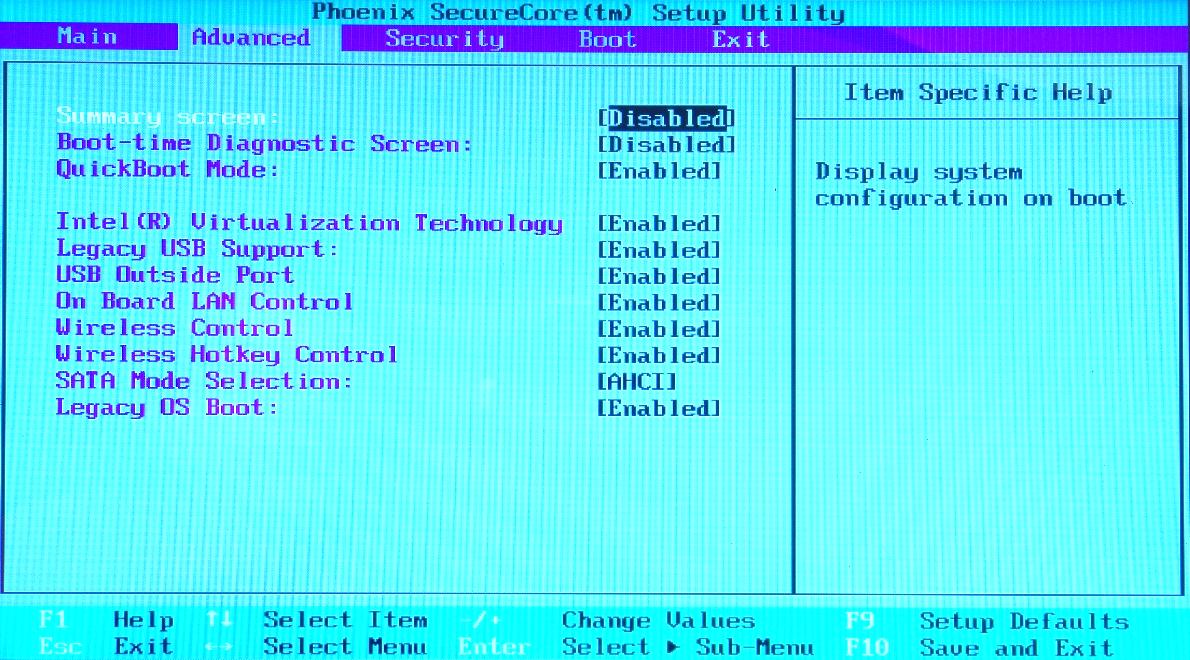
# Activity 1: studying virtual CPUs

|  |  |  |
| --- | --- | --- |
| **For each of the implementations below…** | **Briefly explain how software replaces some hardware/CPU functions** | **What was the rationale behind doing it this way?** |
| Dalvik |  |  |
| Java virtual machine |  |  |
| QEMU |  |  |
| Amazon’s cloud servers |  |  |
| Microsoft Azure |  |  |
| DOSBox |  |  |

# Activity 2: reading BIOS

On a home laptop or a desktop, enter BIOS setup (the screen that comes up right as a computer is restarted). Briefly list all the options found and who would use such a feature (regular user, tech support, administrator, gamer, corporation, programmer …)

You might see something similar to this:



|  |  |  |
| --- | --- | --- |
| **Feature** | **Used by…** | **Used for…** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Activity 3: optimising performance

Background reading: 10+ Windows 7 services you may not need (<http://www.techrepublic.com/blog/10-things/10-plus-windows-7-services-you-may-not-need/>) or equivalent services on Mac. Find out what Windows services do and why some of them can be disabled.

Task A: Comment on how services are scheduled.

Another excellent resource on the topic comes from the music software company Native Instruments:

<http://www.native-instruments.com/en/support/knowledge-base/show/752/windows-7-tuning-tips-for-audio-processing/>

Task B: Using a freeware sound editor like Audacity or a demo version of Studio One Free (<http://www.presonus.com/products/studio-one/compare-versions>), try to overburden the system by running too many tracks of audio. Then perform troubleshooting as outlined in the guides above to improve the performance.

List the processes that made the biggest difference:

|  |  |
| --- | --- |
| **The process that you  enabled/disabled** | **Explain what changed in the running  of the software** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Activity 4: timing system performance with Python

The code below runs some pointless processing and times how long it took to complete. This will be affected by other processes running in the system.

import time

st=time.time()#read system time before

for i in range(1000):

print(i\*5/3+100)#random nonsense processing

en=time.time()#read system time after

t= en-st #calculate the difference b/w end and start time

print("Process completed in {} seconds".format(t) )

The output:

1746.6666666666667
1748.3333333333333
1750.0
1751.6666666666667
1753.3333333333333
1755.0
1756.6666666666667
1758.3333333333333
1760.0
1761.6666666666667
1763.3333333333333
1765.0
Process completed in 5.173295974731445 seconds

Same program run with a USB microphone plugged in (all other things left same):

1763.3333333333333
1765.0
Process completed in 5.21029806137085 seconds

Task: Try rerunning your timing script while engaging or disengaging other activities, for example playing a YouTube video, running a file search, zipping a folder.

List your experiment conditions and the difference in time:

|  |  |
| --- | --- |
| **Experiment** | **Time difference** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |