Change Your OS, Not Your Code

OS CHANGER®

Save time.
Stay on Schedule.
Reduce your work.

Re-use Your Code with OS Changer

Executive Summary

In embedded software development, we go to great lengths to save time. We spend a lot of time learning about the developer's tools and setting up the environment. We perform these tasks in order to save time.

OS Changer is a tool for saving time in embedded development.

OS Changer was created to help developers reduce the amount of re-work necessary during the transition to a new operating system. It recognizes your old OS API calls in the application and automatically handles the work to make them run with your new operating system. It manages all aspects of the porting process, from differing function names to the parameter types. All of this is done without affecting the performance of your application. OS Changer also includes profiling features for application optimization. It is a tool for developers to re-use existing application code, so they don't have to throw away months and years of work or manually perform the time-consuming task of porting their software every time the operating system platform changes.

OS Changer has been verified with many operating system vendors and used by leading companies all over the world. It supports porting your application from major operating systems including VxWorks, pSOS, Linux/POSIX, µITRON, Nucleus, Windows(Win32), µC/OS, FreeRTOS and RTLinux. It supports porting your application to multiple target operating systems, including VxWorks, Android, Linux, RT Linux, Windows, LynxOS, QNX, UNIX, Solaris, NetBSD, µITRON, MQX, Nucleus, ThreadX, T-Kernel and FreeRTOS.

Note about this paper:

In some places, purely developer topics are discussed. The topics aren't overly technical, but they sometimes make references to computer science topics. These boxes are meant to dig a little deeper into the technical details revolving around OS Changer. If you aren't technical or you would rather not know, feel free to skip over these sections.

PREFACE –
(Read this if you are in the embedded market)

Time constraints and change are the norm

Let's face it…

Most people will never understand the intricacies of an embedded system. Even among coders and programmers, working on embedded systems is rare. In today's job market, many "programmers" will never see the inside of a development board, manipulate registers on a chip, or write an Ethernet driver. They'll never understand the relationship between a compiler and a linker. They'll never know how to debug code.

And they'll never understand the embedded developer's most critical resource –

Time and ***Effort.***

That puts embedded developers in a special class.

As embedded engineers, we go to great lengths to save time. We spend a lot of time learning about the development system - the hardware, the tools, the operating system, the drivers and the application specifications. We have to spend time setting up the environment and getting the tools configured so that we can quickly write, compile, and debug our application code. Many embedded engineers create elaborate setups to ease the development process. They create special scripts and shortcuts and directory structures to ease the development time.

Why do we do that? Well, we do it to make it easier on ourselves, but really it boils down to saving time.

The focus on time is a simple way of communicating your progress on the development project. Depending on the technical ability of your manager, they may or may not be interested in all the technical details of the project. They may want to hear about the byte-swapping routing you wrote or how efficiently your code is running. But, when it comes down to it, they only care about "Time." *Is it on schedule? When will it be done? How much longer?* All these questions are about time.

Therefore, we are always on the lookout for tools that helps us save time and effort.

In embedded development, many activities affect the development time and schedule. Here are just a few:

- New software features
- Changing out hardware
- Adding a new device
- Changing specifications
- Moving to a new tool chain
- Changing or upgrading the OS
- Upgrading to a new tools version
- Changing drivers
- Adding more memory to the system
- Adding new output devices to the application

That brings us to the next point. All of the items above require something to change. Whether it's a new feature or a new board, changes will be needed in your software.

If you've been in the embedded world for any length of time, you've come to accept that some type of change is reasonable and expected. What isn't necessary is the idea that all change is the same. There are good changes and bad changes. A good change is a change to improve performance or to eliminate inefficient operations. A bad change is to perform unnecessary rework that is tedious and error-prone.

Bad changes can be project killers. After weeks of toiling away at coding, it makes no sense to start all over again. It's much more effective if you can re-use your existing code and keep your application development moving forward.

This paper is about a product from MapuSoft called OS Changer. It was developed to help embedded engineers who are changing or upgrading operating systems without the need to change your application software. OS Changer allows you to re-use your existing code without a huge amount of rework or re-writing. It helps make the change faster, more efficient, and with fewer errors.

And that means that you are saving time. Even during change.

Introduction

OS Changer is a tool for saving time. It is specifically written for embedded engineers who need to change operating systems. It supports applications written in C/C++ and is provided in full source-code format. If you are considering a change in real-time operating systems, then this tool should be part of your arsenal.

OS Changer was created to help developers eliminate the re-work necessary during the transition to a new operating system or when you go through an OS upgrade. Our customers have already spent significant time writing their application, creating software that ranges in millions lines of code. OS Changer allows them to re-use existing application code, so they don't have to throw away months and even years of work.

By eliminating the manual porting effort, OS Changer saves the project schedule by shortening the time to market. Choose the appropriate OS Changer Interface for your current OS. OS Changer then uses a Cross-OS target specific module to provide the connection to your new target OS.

OS Changer supports porting your application from the major operating systems including VxWorks, pSOS, Linux/POSIX, uITRON, Nucleus, Windows(Win32), µC/OS, FreeRTOS and RTLinux.

It supports porting your application to various target operating system including VxWorks, Android, Linux, RT Linux, Windows, LynxOS, QNX, UNIX, Solaris, NetBSD, uITRON, MQX, Nucleus, ThreadX, T-Kernel and FreeRTOS.

OS Changer reduces the extra time you spend in learning a new operating system's API, allows you reuse your existing code, and can help in profiling and optimizing your code.

OS Changer is the tool to seamlessly move to your new operating system.

Making the Change with OS Changer -

Sometimes bad things happen in embedded development.

Here's a common scenario. You've made it through part of the development project. There have been some minor hang-ups getting started, but development is moving along. You have parts of your application running and you are steadily making progress on the specs. The schedule is a little tight, but so far so good.

Then suddenly – out of nowhere you get the word that you are switching operating systems.

What do you do then? Well, you start to panic a bit. There's a good chance that all of that old code is going to need to be re-written. However, that's exactly were OS Changer should be used.

OS Changer makes that transition easier. It allows you to use your old code without going through the tedious process of changing all the function calls, verifying that all the parameters are the same type and order, the return values are the same, the initialization code is the same, and the startup sequences of the operating system are all in the same order.

The reason why this is important is simple. After you've already invested time and effort into your application, making these kinds of changes in your application code for a new operating system is going to be tedious and error-prone. It's best to try to minimize as many failure points as possible.

Your focus should be on minimizing the amount of re-work. You should also want to minimize the introduction of programming errors into the transition process. OS Changer keeps this transition effort to a minimum and reduces much of the time-consuming, labor-intensive work that distracts you from the actual product development.

You may be asking yourself how reasonable it is to expect an operating system change in the middle of a project. It happens more than you'd think. Of course, the decision to switch operating systems is a difficult one and often unexpected at the start of the project. No developer goes into a project saying, "Hey, let's switch operating systems in the middle of the project."

There are often technical reasons to switch operating systems. There could be hardware issues or there could be functionality deficiencies in the current operating system. There are also executive decisions to switch or business reasons such as acquisitions that require a change of the operating system. When Linux first came around, it wasn't uncommon to see projects switching for business reasons (no royalties). More recently, many development projects are switching to Android, a new operating system introduced by Google.

Regardless of the reason, OS Changer makes that transition simple and efficient.

What are the technical reasons to switch operating systems?

Outside of business decisions and money-matters, there are actually several technical reasons to switch operating systems. Here are a few:

- Lack of drivers/component software
- Discontinued support for your hardware platform
- Incomplete support of interrupt handling
- Dismal performance at basic tasks like start-up, memory allocation, interrupt handling, etc.
- Missing functionality (e.g. mutex, shared memory, process support, signals, etc.)
- Lack of full register save on context switch (e.g. think register windows via SPARC)
- Moving to a certified or secure operating system

In all cases, the pain of switching can be eased if you use OS Changer.

BSPs and Drivers

When making the change to a new operating system, you may also be changing hardware at the same time. When this happens, the question of a BSP (board support package) and device driver software often arise.

OS Changer eliminates the re-work for your application software. While the OS Changer I/O interface can be used for device driver development, there will still need to be BSP and device driver code supplied by the operating system vendor.

Mythbusting - It's not a wrapper.

Some embedded engineers attempt to write their own wrappers for porting software. This often turns out to be a mistake that creates more work. They don't realize the magnitude of the effort involved and often they end up with a poor implementation. Generally this is due to fundamental differences in behavior of the kernel resources between the two operating systems. OS Changer contains most of the features necessary to switch operating systems and does not follow a typical wrapper implementation method.

Using wrappers is often a way to 'short-cut' the transition process from old application code to new application code. Wrappers are useful in some cases. But, to call OS Changer a wrapper is misleading.

OS Changer is a full-featured porting tool that eases your effort across operating systems. OS Changer has been verified with many operating system vendors and used by leading companies all over the world.

It recognizes your old OS API calls in the application and automatically handles the work to make them run with your new operating system. It manages all aspects of the porting process, from differing function names to the parameter types. All of this is done without affecting the performance of your application. OS Changer also includes profiling features for application optimization.

On Wrappers…

A wrapper is a name used to describe a serious of functions that do nothing but convert input/output parameters from one function to another. An example of a wrapper is this:

```
int Old_OS_Wrapper_Function_call_1 (int id){
        unsigned int ReturnCode;
        ReturnCode = New_OS_call((TASK_ID)id);
        return ((int)ReturnCode);

};
```

This is used as a shortcut for going through the entire code base and changing all the references to the old function call. A wrapper has its place in programming, but it is not the most elegant solution. When you are changing APIs, transitioning to new feature, or deprecating old calls, wrappers are a nice way to help make the transition. They should be used as a short-term solution though. Using them for a full application can be error-prone and dangerous.

Ideally, a wrapper-less call will NOT make references to the new OS calls, but instead will actually contain the implementation code required to accomplish the old call as shown below:

```
int Old_OS_Wrapper_Function_call_1 (int id){
        unsigned int ReturnCode;

        /* your own code implementation, below */
        return ((int)ReturnCode);
};
```

Optimization and Profiling

What is optimization?

Optimization can mean many things in embedded systems. It could reference the initialization of the start-up task, interrupt handling, or a general improvement in code size. In this paper, when we say optimization, we are referencing 'application-specific performance and code-size optimization,' - using OS Changer to make improvements to your application's performance and memory footprint.

The optimization with OS Changer can be used to profile the performance of your application or it can be used to profile the interface and interaction between your application and the operating system.

OS Changer also includes profiling functionality. By using the profiler with OS Changer, you are able to optimize your application to meet your own specific performance requirements. The real benefit of the profiling comes during the transition from one operating system to the other. You'll be able to make sure that your application is running exactly the same on the new operating system as it was on the old operating system.

Why should you worry about the performance during your transition? Each operating system in the embedded market has small discrepancies and operating nuances. While they all work the same in theory, it's the small differences that can make the biggest difference in performance.

With the OS Changer profiling options, you can record multiple sessions and track of the metrics for performance. Then, as you make the change to the new operating system, you can keep a running record of the application's performance. As you are progressing, you can generate a Timing Comparison Report to compare two different sets of measurements and reports. This will help you guarantee that your application is performing efficiently and accurately.

For example, in operating systems there are often differences in the start-up time of the operating system, the creation of new tasks, or the invocation of semaphores. These small changes could cause unexpected behavior in your application. Or, even worse, these differences could cause a major slowdown of your application.

By using the profiling option with OS Changer, you can compare and contrast various performance metrics whenever you change your OS and hardware platforms. You take

the metrics before the change and compare them with your metrics after the change by using the Timing Comparison Report. It's a convenient way to maintain application performance.

The OS Changer profiling option also makes it easy to view the data. You can view the data numerically or graphically. The graphic view can be displayed as a bar graph, line graph, pie chart, or scatter chart. You can also easily switch between the views to find the display option that best suits you.

Re-Cap

OS Changer eases the transition from one operating system to another. When you are making the change, it's important to re-use as much of your existing code base as possible. Further, you want the transition to be easy and painless. It helps make the change faster, more efficient, and with fewer errors.

OS Changer supports porting your application from the major operating systems including VxWorks, pSOS, Linux/POSIX, uITRON, Nucleus, Windows(Win32), µC/OS, FreeRTOS and RTLinux.

It supports porting your application to various target operating system including VxWorks, Android, Linux, RT Linux, Windows, LynxOS, QNX, UNIX, Solaris, NetBSD, uITRON, MQX, Nucleus, , ThreadX, T-Kernel and FreeRTOS .

OS Changer is written in C and is provided in source code format. It could be used within your C/C++ and/or Ada applications.

About Mapusoft

MapuSoft Technologies (MT) is the number one provider of embedded software re-usability solutions and services that are designed to protect software investment by providing customers a greater level of flexibility and control with product development. In addition to off-the-shelf tools, MT offers porting, integration, support and training services to help developers easily migrate from legacy platforms to the next generation. We believe that our advanced software and vision will revolutionize the embedded software industry. We are working hard to provide software that is practical, familiar, financially reasonable, and easily operable. We provide full source code with no royalty fees. Our licensing strategy makes it extremely affordable for you to incorporate our products into your embedded applications. In addition, our attention to engineering detail provides you with robust software and requires minimal technical maintenance.

About OS Changer

OS Changer is a C/C++ source-level virtualization technology that allows you to easily re-use your software developed for one OS on another OS, while providing real-time performance. It eliminates the manual porting effort, saves money and shortens the time to market. The appropriate OS Changer Interface connects to your existing application that was developed on your current OS, while the Cross-OS target specific module (specific to your target OS) provides the connection to the OS you are moving to.

For more information

- To download MapuSoft's free software evaluation visit: http://mapusoft.com/downloads/
- To learn more about our licenses and request a quote visit: http://connect.mapusoft.com/contactus.html
- OS Changer Porting Kit Overview Datasheet: http://www.mapusoft.com/wp-content/uploads/documents/os-changer.pdf
- OS Changer - VxWorks Porting Kit Technical Datasheet: http://www.mapusoft.com/wp-content/uploads/documents/VxWorks-techsheet.pdf
- OS Changer - pSOS Porting Kit Technical Datasheet: http://www.mapusoft.com/wp-content/uploads/documents/pSOS-techsheet.pdf
- OS Changer - Nucleus Porting Kit Technical Datasheet: http://www.mapusoft.com/wp-content/uploads/documents/Nucleus-techsheet.pdf

- OS Changer – Windows Porting Kit Technical Datasheet: http://www.mapusoft.com/wp-content/uploads/documents/Windows-techsheet.pdf

- OS Changer Porting Kit API coverage and feature support information please click here for the current Release Notes: http://www.mapusoft.com/wp-content/uploads/documents/Release_Notes.pdf

Contact Information

For more information about OS Changer or Mapusoft, please contact us at:

US Headquarters
MapuSoft Technologies, Inc.
Unit 50197
Mobile, AL 36605

Tel: (251) 665-0280
Toll Free: 1-877-MAPUSOFT (1-877-627-8763)
Fax: (251) 665-0288

E-mail: info@mapusoft.com or sales@mapusoft.com

For a complete listing of International Offices, please click here: http://mapusoft.com/contact/

Mapusoft's Complete Product Line: