Extending the Operating System

Abstract:

Operating System is an essential part of any computer system. This paper discusses the ways of extending the Operating system by various ways like recompiling the kernel, Libraries etc. Extensibility is the need of time. It gives the user more "POWER" over the machine. Every user can have his own "personalized" copy of operating system. The approach of extending the Operating System at user level is more advantageous than other methods. An example "mounting an FTP server" is then discussed to show the extension at user level. It is implemented by intercepting (hooks) system calls.

Introduction:

Operating System is system software that acts as an intermediary between a user of a computer and the computer hardware. It is an essential part of any computer system. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

There are so many times when a computer user thinks that there should be a "feature" offered by the operating system. At that point of time, she either installs existing software or code the solution himself. Various methods by which Operating system is extended are listed below.

Approaches of Extending the Operating System:

- Editing the Operating System.
- User level Static and Dynamic Libraries.
- By intercepting the system calls.
- Application specific modifications.
- User level plug-ins.
- Device Drivers.
- Network Server.

Each of these is discussed below in detail.

Editing the Operating System: The simplest way to modify the operating system is to recompile its kernel. It is the easiest and straightforward one. It needs that the user isn't a novice and has administrator privileges.

Static and Dynamic Libraries: This needs that user codes himself some libraries which can be linked later either statically or dynamically with applications. Most applications do not directly access the operating system, but use library functions

embedded in standard libraries. Instead of modifying the binaries or the operating system kernel, it suffices to make changes to these libraries.

By intercepting system calls: This method is the most efficient one. It intercepts the system calls and notifies the other processes whenever a process enters a system call or exit a system call.

Application Specification modifications: In this approach the features of "extended operating system" is specific to only an application. It is more like of modifying the callback functions in Windows.

Device Drivers: Instead of recompiling the whole kernel, the modification can be done to device drivers for each device. This needs the root privileges.

Network Server: The cleanest of all is this method with no intrusion to operating system; install a network server which provides new functionalities at standardized interface. This approach also needs root privileges.

User-level plug-ins: When a one time modification to the Operating System can be tolerated, a flexible strategy is to add hooks to operating system so that system calls can trigger addition functions that extend the functionality.

The comparison of all the approaches is summarized below for comparative methods.

S.No	Approach	Needs Recompilation	Needs Re-linking	Needs roots privileges	Overhead
1.	Editing the Operating System	Yes	No	Yes	Low
2.	Libraries	No	Yes	Yes	Low
3.	Intercepting System Calls	No	No	No	High
4.	Application Specific Modification	No	Yes	No	Low
5.	Device Drivers	Yes	No	Yes	Low
6.	Network Server	No	No	Yes	Medium
7.	User level Plug-ins	Yes	No	No	Low

Analysis of Approach "Intercepting the System Call And User-level Plug-ins"

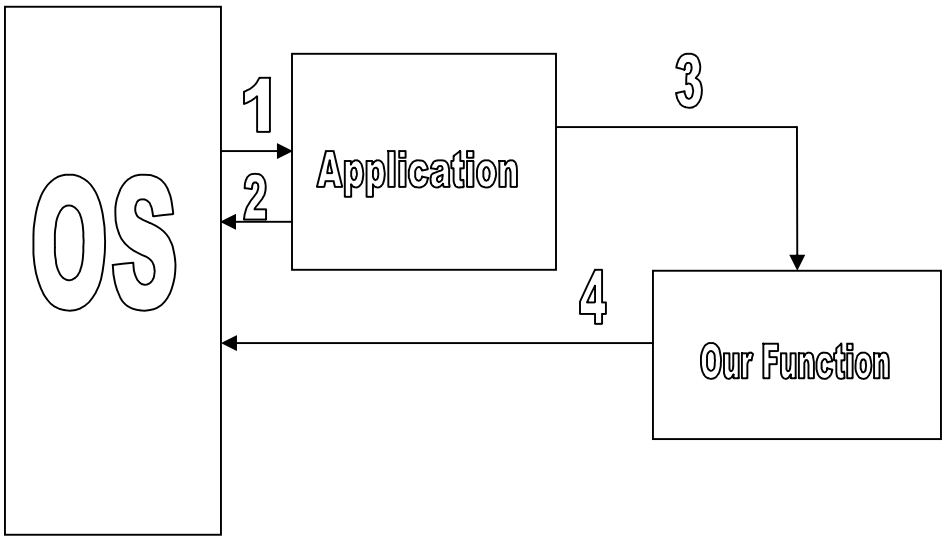
Most modern operating system provides the functionality of intercepting system calls at the user level. A process can be notified when another process enters or exits selected system call. While the original motivation for this functionality was debugging and tracing of system calls, this mechanism can also be used to alter their behavior. The latter approach (i.e. user level plug-ins) needs to intercept some system calls (which are needed to extend the OS) and then calling our own functions at that time. The system calls can be *read()* or any other (In UNIX). These two approaches are implemented at user level i.e. you do not need the root privileges to extend your operating system!!! Or you do not have to pursue the administrator to get your job done. The user has to tell its operating system which all system calls to intercept. Once intercepted the operating system can call the user defined functions (which are supposed to increase the functionality of operating system) or notified another process about the event. There are various advantages of extending the operating system at user level. They are listed below:

1. Every user has his own taste. She can therefore, tweak the operating system according to his tastes. She does not have to put his settings forcefully on other users and every user can have a personalized copy of operating system.
2. The middle man "administrator" between user and operating system is not needed here.
3. Sometimes, the feature is needed only in some special cases. Therefore it is easier at user level to implement this.

The approaches explained above can be best implemented in C or C++ as it gives the maximum power at lowest level to the user. The code can be implemented using the following algorithm:

```
Tell the operating system which all system calls to interrupt;
Start the process;
While process is in running state
      Wait for the system call;
      Determine the parameters of the call and call your function;
     Continue the system call which was interrupted;
     Return to the Operating System;
 End while;
```

The model of our operating system, application and our functions is as shown in the figure. The whole idea is divided into 4 steps. The steps are numbered from 1 to 4. The details for each step are explained below.



Step1: On request our application is started. There is continuous transfer of information between the operating system and application. The application initially provides the list of system calls to operating system which are to be intercepted. The application keeps the track of the all system calls being done. If the system call do not match with the any in the list the proceeding are done as usual.

Step 2: When the system call do not match with any in the list, the application interacts with operating system as usual. The application either returns any result or ask for something to operating system using this step.

Step 3: If the system call matches with one in the list the application calls our function with the suitable parameters. This function gets executed and after that resumes the system call. The result is returned using step 4.

Step 4: The function defined by us returns the value to operating through this step.

There can arise many problems with this approach. All the problems are listed below:

1. Some of the kernel features are hidden from the user (In Windows). Therefore we may not be able to utilize our function most efficiently.
2. The problem of overhead is there. Our application will keep looking for the system calls.

3. The user may have been given only read only permissions to the system. So we assume that the user has given a directory with write permissions.

The above two explained approaches (i.e. User level plug ins and intercepting system calls) can be used together to make a "mounted FTP" (File transfer protocol or as a Redmond Developer says Fido To Phetch) on the workstation when provided with correct username and password i.e. a user can see the transfers from or to the server as if they are from a local drive.

Our Example: "Mounted" FTP

Our example: "Mounted" FTP uses the two approaches explained above. Other approaches can be also used but it can be best implemented using these two. We make an application EFTP (Extended FTP!!!) which initially tells operating system all system calls which are to be interrupted. It then keeps track of all the calls and if the system call is in the list, the call is interrupted and function *uftp()* is called with some parameters. After the function completes the execution, the system call is resumed. The whole idea of this application is that the user should feel that she is accessing the remote file as it is local.

Whenever the user tries to access any file remotely, the application can either make a soft link to the file (if using UNIX) at any directory or the system can download the file and then show its contents to the user (The same idea is used for viewing web pages). The latter method can eat up your hard disk space so make sure that the space doesn't exceed much. Or you can download the file only when it is needed for reading or writing. Note that it is assumed that the user has given the application the right username and password to access the data on the server.

Conclusion:

There are many methods to extend your operating system and have a personalized copy of it. Each approach has its own advantages and disadvantages. The approach can be chosen according to the needs. If the change has to be permanent and is needed every time, it's advisable to modify the kernel. If it is not possible to modify the kernel as in Windows then making libraries or user level plug-ins can be useful. The method of "intercepting the system calls" to extend the operating call is a useful one though there is processor overhead. Other approaches can also be used to extend your operating system and they may give better results than this method.