Caleb Faruki
Dr. Werschulz
September 26, 2013
OS Homework 2

2.13

3 general methods for parameter-passing to the OS include: (1) passing parameters via registers (params might exceed the # of registers, however), (2) storing the params in a memory block whose address is then passed as a consolidated param to register or (3) pushing/popping params onto/off the stack thru the OS.

2.15

The 5 major activities of the OS in file management are as follows: (1) creation/deletion of files, (2) creation/deletion of directories, (3) file/directory manipulation, (4) moving/copying files/directories, and (5) file/directory protection/permissions.

2.16

Since the OS operates on files and devices via the same abstract file interface, development efforts may be focused on writing a well-defined API. All that is needed to use a device is a hardware-specific driver file to use the system-call interface.

The downside of using the same interface is that for some devices, it might not be easy to implement certain functions. Therefore, using the same interface might come at the cost of functionality and/or performance loss.

2.18

The 2 models are the message-passing model and the shared-memory model.

For message-passing, the advantage is modularity, simplifying large tasks and isolating dangerous operations. But the cost of this is extra operations to pass these messages along.

For shared-memory, the advantage is fast, direct memory access. The disadvantage is a heightened risk of memory corruption, especially when 2+ processes are operating on the same memory addresses.

2.19

Separating mechanism and policy is important for flexibility. Designing programs and operating systems to be dynamic in accomodating policy changes means that much time and code is conserved.

Designing a "catch-all" mechanism that can accomodate any policy scenario means that source code may be reviewed, improved, and recycled. This flexibility begets other benefits since time can be allocated to other efforts, such as performance optimization, security, protection, and documentation.

2.22

The modular kernel is similar to the layered kernal insofar as each kernel section has defined, protected interfaces. But they differ in that a layered system cannot call any kernel section.

2.24

For each Java class, the compiler produces an architecture-neutral bytecode output (.class) file that will run on any implementation of the JVM.

If the JVM is implemented in software, the intepreter might interpret the bytecodes for each class instance one-at-a-time. A JIT compiler conserves time by saving the machine code for bytecode files the first time each class is invoked. Therefore, JIT compilers save a fraction of time for each subsequent invocation of those same classes.

2.26

The layered approach offers two key benefits: abstraction and ease of debugging. We may implement layers from the bottom-up, splitting the task of designing an OS into more easily workable parts. Once we've ensured that a layer is correctly functioning, we may move up a layer. When debugging the next layer, we may assume with more certainty that any bugs we encounter originate from the layer we're currently working on and not the layers below.

Contrarily, the Synthesis approach makes debugging difficult because we can't be sure that everything else in the kernel is working as intended. Moreover, no abstraction means that building this kernel is an onerous task and the finished kernel is likely monolithic and not user-friendly.

But the Synthesis approach does eliminate the overhead for a system call that comes with each layer. The net result is a system call that takes longer than does one on a non-layered system.

One more con against the layered approach is that appropriately defining the various layers is difficult unless the layers are carefully planned out.