**CS414 Fall 2004 Homework 8. Due in class on November 23, 2004**

2. In class, we learned about the working set algorithm for managing paged memory. In the context of this algorithm:
   a) Many programs have "phases". For example, a compiler might have an input and parsing phase, a symbol-table analysis phase, a code generation phase, etc. Why would each phase be likely to have its own working set?

      *A working set is a set of pages visited repeatedly because of spatial or temporal locality in a program: pages containing the code being executed or the data objects being accessed.. When the behavior of a program changes drastically, the set of pages would often change too.*

   b) Still in the context of a program that has phases, suppose that you discovered that all the pages for some segment of the program turn out to be in the working set from start to finish (e.g. despite the phase behavior, they stay in the working set). Give some possible explanations for what such pages might contain.

      *Perhaps these pages contain commonly used procedures that are called in both phases, such as the formatted I/O routines, or data both phases access.*

   c) Suppose that we were considering implementing a new system call to "pin" certain pages into memory. Once pinned, a page will no longer be a candidate for being paged out by the paging algorithm, and won't count against the working set of the program. Describe some situations where you might consider pinning a page using this new system call. (Note: no need to focus on programs that run in phases – part (c) is about programs in general).

      *Perhaps we know that from time to time, some data structure will be accessed intensively – but then not touched for long enough so that the algorithm might page those pages out. By pinning them we can completely shut down this undesired excess paging.*

3. We learned about dynamically linked libraries in class, too.
   a) List several advantages of using dynamic libraries

      *When many programs need to link to the same library, we save runtime memory by using shared (dynamically linked and loaded) libraries. We end up with just one copy of the code, which can be a big win if the library is huge.*

   b) We learned that each program must make its own copy of any data segments associated with a DLL, but that all programs using a DLL share the code. Explain the difference between a data segment and a code segment in just a few words (not an essay!) and then explain briefly why we need to create private copies for one, but not the other.

*Code segments have code compiled from source of the library. Data segments have the variables. Since we're pretending that each instance of the library has a private copy of the variables (and can change them) we need a copy of the segment. The code never changes and so can be a single shared segment.*

c) While developing a new game program, suppose that you discover that a commonly used DLL has a bug. You track down the source file, fix the bug, and recompile the library. Now your game program begins to work perfectly. Yet other users are suddenly complaining that many of there programs have begun to crash! Why might fixing a bug in a DLL cause problems? Would the same issue have arisen if we were using traditional statically linked libraries? Explain.

*Perhaps some program actually depended on the buggy behavior, or something else is impacted by the change and breaks, or timing changes in a way that exposes a bug in the application. With static linking only a program "relinked" to use the modified library would be affected by the change (unless you consider remote procedure calls or other networked behaviors)..*

d) A motivation for the DLL concept was to reduce the total virtual memory size. Remind us of how DLL's can give this benefit. Now describe a situation in which moving to DLLs would actually cause the virtual memory in use to rise, presumably hurting performance relative to a solution that didn't have DLLs. You can assume that the library we are thinking of is the formatted I/O library for a programming language like C or C++ if it helps you "focus" on a specific scenario in answering this question.

*Even though multiple programs have the DLL code segment in their virtual memory, in fact there is only one shared copy in physical memory. But normally we need to map the ENTIRE DLL in when we use a DLL, whereas a program would only get the PARTS of a library that it uses in a statically linked setting. Thus by moving to a DLL we can see increased memory use if we don't have lots and lots of applications using the DLL – enough so that all of its routines are needed.*

4. Consider the disk buffer pool of a typical file system.
   a) Caching file system pages has pros and cons. List a few of each.

*Pro: Better performance: no need to do a disk I/O for each access.*

*Con: File on the disk can lag the file in memory, so in a crash data can be lost. When sharing files between multiple machines, programs might be confused by these delays. Uses memory we might have put to other use, perhaps a better one.*

   b) Some studies show that as much as 50% of all files are deleted within a few seconds after being created! In fact one study of Microsoft Windows NT showed that a great many files are deleted within milliseconds after being created. How

might you exploit this knowledge in designing a disk buffer pool (cache) management strategy?

*Delay the file writes long enough so that, with luck, the file might be deleted without ever doing the I/O at all!*

5. Some designers of database systems complain that file systems and buffering only mess their performance up and that they would prefer to have complete control over (1) what gets prefetched, (2) what gets held in the kernel buffer pool, and (3) how files are mapped onto disk blocks.

   A typical scenario where this arises is when a database has some form of "index" mapping keys to blocks and records within those on the file system. For example, "Ken" might map to "14/7", meaning that the data associated with key "Ken" is in block 14. The "7" would mean that block 14 has multiple fixed-size records in it and Ken's is the $7^{th}$ record. For example, a 1024-byte block size could hold roughly 100 records each of size 100 bytes, with 24 bytes left over for other purposes.

   The index file itself would typically be a balanced tree, like a binary tree. Focusing on this case, look at each of the complaints mentioned above. Would control over the specified behavior have a potentially big performance impact, compared with the sorts of automatic policies we discussed in class?

   *(1) A database index might not be searched sequentially. If the file system thinks we are searching sequentially time could be wasted doing undesired I/O.*

   *(2) Once we scan an index block we probably won't revisit it; keeping it in memory is pointless and wastes that piece of memory.*

   *(3) A database index might perform best if the application (the database program) can control how files are positioned on the disk (for example to put blocks close together because they are often accessed sequentially). If the file system does this the database cleverness is defeated.*

6. Suppose that on a Linux file system, the GNU C compiler is in a file called "/usr/bin/gcc". Now Doug comes along and creates a (true) link to that file, calling it "/users/doug/bin/c".
   a) Doug's reason for creating the file system link is that he really hates 3-letter command names. Doug prefers to just type "c" instead of "gcc". But does he really get the same program that someone else would get when they run the program by the name "gcc"? Explain.

      *Yes. The file is really identified internally by inode number; it has two pathnames but only one inode and hence the two names are references to the same bytes.*

b) Sally the system administrator needs to install a new version of GCC. So she compiles the new version, then deletes the old one, copies the new one into place, and gives it the identical name to the old one. When Doug runs "c" will he still get the old program, or will he get the new one now?

*Doug will get the OLD version. When she deleted one name, the file never went away, it just had the link count drop by one. The new file has a new inode number and Doug's shortcut doesn't point to it!*

c) Same as part (a), but now assume that Doug created a symbolic link, not a true link.

*Here there are two files. One is the gcc program. The other, the symbolic link, is a file containing the pathname of the gcc program – not the executable image.*

d) Same as part (b), but under the same assumption – Doug's version is a symbolic link.

*Now he gets the NEW one. Here there is one pathname for the one inode and Doug's "name" doesn't identify the file by inode number but by its true pathname.*