CS 3013 Operating Systems — WPI, A Term 2007
Craig E. Wills — Project 4 (15 pts)
Assigned: Tuesday, October 2, 2007 — Due: Wednesday, October 10, 2007 (10:00am)

Project Description

The primary purpose of this project is to compare the performance of standard file I/O using the *read()* system call for input with memory-mapped I/O where the *mmap()* system call allows the contents of a file to be mapped to memory. Access to the file is then controlled by the virtual memory manager of the operating system. In both cases, you will need to use the *open()* and *close()* system calls for opening and closing the file for I/O.

For the project, you should write a program *proj4* that will behave similar to the *strings* command available in Linux. This command prints all strings of printable characters with length four (the default) or more. You should try executing the *strings* command on a few files to get a sense of the type of strings that are embedded in various types of files.

For your project, you do not need to print the strings of characters, but instead you will determine how many such strings of at least *four* characters exist, along with the longest such string. Your program takes a file name as command-line argument and determines strings of printable characters in the file. Printable characters occur if they match one of two routines: *isprint()*, which determines if a byte value is a printable character; and *isspace()*, which determines if a byte is a space, newline, tab, etc. Check the man pages of these routines for details and the needed include file.

Please note that the strings of characters that you find will not be NULL-terminated strings as used in C/C++, but rather these strings will terminate when followed by a non-printable byte that occurs in the file.

The following shows sample output from two executions of your program:

```
% proj4 proj4
File size: 12969 bytes.
Strings of at least length 4:  218
Maximum string length: 68 bytes

% proj4 proj.C
File size: 2894 bytes.
Strings of at least length 4:  1
Maximum string length: 2894 bytes
```

In the first execution, the program is run on its own executable file. In this case there are many non-printable characters in the file with 218 strings of at least length 4 found. The longest such string is 68 bytes. In the second execution, the program is run on a source file where all bytes in the file are printable. In this case, there is only one string, which is the length of the file.

Two interesting options to add to your program are the capability to print the strings that are found and to be able to change the minimum string length from the default of four. Both of these options are supported by the *strings* command. While these options are interesting, they are *not* required for the project and no additional credit will be given for their implementation.

Project Implementation

Now that we have described the functionality of your program, which will require all bytes of the input file to be read, the remainder of the project focuses on how your program reads the input. The default behavior of the program should be to read bytes from the file in chunks of 1024 bytes using the *read()* system call. However your program should have an optional second argument that controls the chunk size for reading or to tell the program to use memory-mapped file I/O. In the latter case your program should map the entire contents of the file to memory. The syntax of your program:

```
% proj4 srcfile [size|mmap]
```

where `srcfile` is the file on which to determine the strings of printable characters. If the optional second argument is an integer then it is the `size` of bytes to use on each loop when reading the file using the *read()* system call. Your program should enforce a chunk size limit of no more than 8192 (8K) bytes. Your program should traverse the buffer of bytes read on each iteration and keep track of printable strings as described above.

If an optional second argument is the literal string "mmap" then your program should *not* use the *read()* system call, but rather use the *mmap()* system call to map the contents of `srcfile` to memory. You should look at the man pages for *mmap()* and *munmap()* as well as the sample program `mmapexample.C` for help in using these system calls. Once your program has mapped the file to memory then it should iterate through all bytes in memory to determine strings of printable characters. You should verify that the file I/O and memory mapped options of your program show the same output for the same file as a minimal test of correctness.

Performance Analysis

Correct implementation of the project functionality using both file and memory-mapped I/O is worth 13 of the points for the project. For the remaining two points of the project you need to perform an analysis to see which type of I/O works better for different size files. For this portion of the project, you should reuse the first part of the *doit* project, which allows you to collect system usage statistics. The two usage statistics of interest for this project are major page faults and the total response (wall-clock) time. A sample invocation of your *proj4* program on itself using *doit* with the largest read size would be the following where *proj4* prints its output and then *doit* prints the resource usage statistics for the program.

```
% proj4 proj.C 8192
File size: 2894 bytes.
Strings of at least length 4:  1
Maximum string length: 2894 bytes
< resource usage statistics for proj4 process >
```

At the minimum, you must test your program running under five configurations for input files of different sizes. The five configurations are standard file I/O with read sizes of 1, 1K, 4K, and 8K bytes as well as with memory mapped I/O. You should determine performance statistics for each of these configurations on a variety of file sizes. As an aid in finding a range of file sizes, the directory `/var/lib/rpm` on the CCC machines has some large files such as `Filemd5s` or `Packages`. You should look for other files with a range of sizes.

Once you have executed your program with different configurations on a range of files, you should plot your results on two graphs where the file size is on the x-axis and the system statistic of interest (major page faults or wall-clock time) is on the y-axis. Each graph should have one line for the results of each configuration.

You should include these graphs as well as a writeup on their significance in a short (1-2 pages of text) report to be submitted along with your source code. You should indicate which configurations clearly perform better or worse than others on a given performance metric and whether there is clearly a "best practice" technique to use.

Submission of Project

Use `turnin` to turn in your project using the assignment name "proj4". You should submit the source code for your program and a soft (paper) copy of your report.