Lab 4 - Shell Scripting

Problem Description

The objective of this lab is to use shell programming to list running processes using the */proc* interface. The script utilizes the /proc file system, extracting information from directories and files associated with each running process.

A script has been written to display a list of all the running processes. The output contains the process id (PID) and the owner of the process (user ID) as the first two columns. To simplify the flags, additional columns are indicated with their own flags. The flags that are handled are *-rss* which includes a column for the resident set size (RSS) of the process, *-comm* which prints the command that the process is executing, *-command* which prints the command line used to start the process, and *-group* which prints the group id of the process.

The lab consists of 5 different phases, each building on the next. Phase 1 parses the command line arguments, setting variables for recognized commands as "yes", and reports errors or conflicting flags.

Phase 2 uses a for loop checking if a directory *p* is in a regular expression file (*status* or *cmdline*) within */proc*. Since the current process can change while the loop is executing, there is an if statement checking to see if a directory still exists. If so, it matches directories that start with numbers and then uses the command *echo* to print a message to the terminal.

In Phase 3, it focuses on extracting details such as PID, user and group IDs, and RSS, which helps organize and prepare the data for output in the next phase. This is done using a combination of the *grep* command (global regular expression print) and *sed* (stream editor). There are two lines that converts user ID and group ID to symbolic names such as root, and netid to username.

Moving on to Phase 4, the for loop from Phase 2 expands in string order and not numerical order. To fix this issue, the output is written without the headers to a temporary file in the */tmp* directory. The *ps* command adjusts the width of column dynamically to ensure the output is neatly aligned. Also, each file is appended using the append redirection ">>", instead of overwriting the file.

Lastly in Phase 5, the program concludes by printing headers to the terminal, using if statements to ensure that the correct headers are printed. Using the sort numeric command *sort -n*, it sorts the temporary file by the first column and prints it. Finally, the temporary file is removed before the program exits.