1 UNIX System Calls

1. Consider the C function printf() on UNIX. Is printf() implemented by the OS, or by an application-level library? What system call does printf() make internally?

2. Why are `mkdir`, `ln` and `rm` implemented as separate user-level programs, while `cd` is implemented as a built-in command?

3. On a linux machine, type the following command

   ```
   $ cat | tee output.file
   ```

   cat is a UNIX utility that prints the contents of STDIN to STDOUT. tee is a UNIX utility that prints the contents of STDIN to both STDOUT and to the file named by its argument (output.file). After you type this command, you can type in some characters followed by the newline character.

   a. While this command is running, examine the processes created:

   Use pstree to see the process hierarchy. Tell us what you find about the process hierarchy. Use 'ps x' to identify the process IDs of the processes created by cat and tee commands. Linux provides a proc pseudo-filesystem which can be used to examine the state of a process using filesystem namespace.

   Type the following command for both process-ids:

   ```
   $ ls -l /proc/pid-num/fd/
   ```

   b. What do you find? What are 0,1,2,...? What do they symlink to?

   ```
   $ ls -l /proc/self/fd/
   ```

   c. What do you find? What is 3 pointing to? Why?

   d. Which system calls are executed while you run `cat`, `tee`, and `ls` as in the above commands?

4. Assume you are given two text files containing newline-separated strings. You wish to write a program that takes as input two text files and outputs another text file that contains all the strings that are common in the two input files. For example, if the two text files are `foo` and `bar`, with the following contents:

   ```
   foo:
   hello
   os
   class

   bar:
   os
   is
   an
   interesting
   class
   ```

   Then, the output would be:

   ```
   output:
   os
   class
   ```

   (`os` and `class` are common words in the two files). Assume you are given the following functions:

i. A function `cat()` that takes input from its first argument and displays its content on the standard output (file-descriptor 1). e.g.,

```
cat(foo);
```

This function call will cause the contents of the file `foo` to get streamed to the standard output of the program.

ii. A function `sort()` that takes input from its standard input (file-descriptor 0), sorts the input text (assuming it consists of newline-separated words), and prints the output to standard output (file-descriptor 1).

iii. A function `intersect()` that takes input from file descriptors 3 and 4 (non-standard file descriptors). It assumes that both inputs (from each file descriptor) are sorted, and computes the intersection of the sorted streams of words, outputting to the standard output (file descriptor 1).

iv. The `dup()`, `pipe()` and `fork()` system calls.

You are not allowed to use any other system calls (e.g., `open`, `read`, etc.) or any other functions. Using these functions, implement a utility that given two filenames, outputs the common strings in the two files on the standard output.

5. What is the total number of processes at the end of the execution of the following program? Assume there is one process in the beginning that starts running at main. Also, assume that all system calls succeed.

```
main() {
  fork();
  fork();
  fork();
}
```

Explain.

6. Consider the following program:

```
main() {
    int fd;
    fd = open(outfile, O_RDWR)
    fork();
    write(fd, hello, 5);
    exit();
}
```

Assume all system calls finish successfully on a uniprocessor system. Also, assume that a system call cannot be interrupted in the middle of its execution. What will be the contents of the outfile file, after all processes have successfully exited? Explain briefly.

7. Consider the following program:

```
main() {
    int fd;
    fd = open(outfile, O_RDWR)
    fork();
    write(fd, hello, 5);
    exit();
}
```

Assume all system calls finish successfully on a uniprocessor system. Also, assume that a system call cannot be interrupted in the middle of its execution. What will be the contents of the outfile file, after all processes have successfully exited? Explain briefly.

8. Give two interesting applications where you think the user-defined SIGUSR1 and SIGUSR2 can be useful?

9. In UNIX, a child process may terminate before a parent calls wait(). When the parent calls wait() eventually, it still expects to read the correct exitcode that the child returned. To support this functionality, UNIX does not completely remove the process till it's parent has called wait() on it.

   Such processes that have completed execution but still have an entry in the process table are called zombie processes. Usually, the presence of zombie processes in the system for a long time indicates a bug in the program (it is a common error).

   UNIX also provides the SIGCHLD signal, which is received by the parent process whenever one of its children exits.

   In class we discussed that the shell implements "&" functionality by not calling wait() immediately. Should the shell never call wait()? When should it call wait()? Answer by providing short pseudo-code. (Hint: you may want to use the SIGCHLD signal).

10. UNIX uses `fork()` and `exec()` system calls to create new processes. On the other hand, Microsoft Windows has a system call called `CreateProcess()`. Here is a sample definition of the `CreateProcess()` call (this is not identical but similar to Windows system call):

```
Boolean CreateProcess(
    lpApplicationName,    /* Name of the executable. */
    lpCommandLine,        /* Command line parameters. */
    bInheritHandles,      /* Boolean variable indicating if the new process
                             should inherit the open file descriptors of the
                             calling process. */
    lpProcessAttributes,  /* attributes of security attributes of process. */
    lpCurrentDirectory,   /* current working directory of new process. */
    lpProcessInformation  /* A pointer to Process Information structure that
                             receives identification information about the
                             new process. */
     . . .                /* ignoring some other arguments. */
);
```

   Write code to implement the equivalent of this call on UNIX. You should show how you use `lpApplicationName`, `lpCommandLine`, `bInheritHandles`, `lpProcessAttributes`, `lpCurrentDirectory`, and `lpProcessInformation` on UNIX.

11. Write the pseudo-code for a program "`cp`" that takes two command-line arguments, say input-file and output-file, copies the input-file to the output-file.

```
Syntax:
$ cp ifile ofile

Program:

int main(int argc, char **argv)
{
  //your solution goes here.
  //Use UNIX system calls to implement the logic.
}
```