San José State University
Department of Computer Engineering

# CMPE 142
# Operating Systems
Section 1

Spring 2021
Instructor: Ron Mak

## Assignment #8

**Assigned:** Friday, April 16
**Due:** Friday, April 23 at 11:59 PM
**Team assignment**, 100 points

### POSIX file system functions
In this assignment, your team will call POSIX file and directory functions and run the **`fswatch`** utility program as a child process to monitor file system operations.

### POSIX functions
Consider the following sequence of file system commands that one could enter on the command line of a Bash terminal window:

```
mkdir subdir1
mkdir subdir2
cd subdir1
echo "Hello, world!" >> file1.txt (do this 5 times to write 5 lines)
cp file1.txt file2.txt
mv file2.txt ../subdir2/file2.txt
cd ../subdir2
ln file2.txt file3.txt
ln -s file2.txt file2s.txt
cd ../subdir1
rm file1.txt
cd ..
rmdir subdir1
cd subdir2
chmod a+rw file2.txt
cat file2s.txt
```

Each of these file system operations can be performed in a C or C++ program using POSIX file and directory functions. See https://www.mkompf.com/cplus/posixlist.html.

Some examples:

| Bash command | POSIX function |
|---|---|
| `mkdir` | `mkdir()` |
| `cd` | `chdir()` |
| `ln` | `link()` |
| `rm` | `unlink()` |
| `echo "Hello, world" > file.txt` | `open()` *the file*<br>*write to the file*<br>`close()` *the file* |

## The `fswatch` utility program

The **`fswatch`** utility program monitors the file system and writes messages to its standard output that indicate what operations the file system performed. To run on the command line and monitor a given directory, e.g., **`mydirectory`**:

```
fswatch -xtr mydirectory
```

- **`-x`** to print file manager events
- **`-t`** to include timestamps
- **`-r`** to recurse to monitor subdirectories

The specific messages that **`fswatch`** writes for each command depend on the operating system. Some commands (such as **`cd`**) do not cause any file system operations, and so there won't be any output from **`fswatch`**.

## Install `fswatch`

- Ubuntu: https://zoomadmin.com/HowToInstall/UbuntuPackage/fswatch
- Mac OS: http://support.moonpoint.com/os/os-x/homebrew/fswatch.php

## Your program operation

As you've done in past assignments, your program should create a pipe, fork a child process, and read from the pipe what the child process writes to the pipe.

The child process should run **`fswatch`** with the above options to monitor the current directory; i.e., the child process should run

```
fswatch -xtr .
```

To run **`fswatch`**, the child process must call one of the POSIX **`exec`** functions **`execl()`**, **`execle()`**, **`execlp()`**, **`execv()`**, or **`execvp()`**.

**fswatch** writes to its standard output. To programmatically redirect its standard output to the write end of the pipe, the child process needs to call either the **dup()** or **dup2()** function before starting **fswatch**.

After forking the child process, the parent process should then call POSIX functions to execute the series of file operations equivalent to the Bash commands shown above. After each operation, the parent should read the pipe for output from **fswatch** that resulted from the file operation.

**fswatch** can lag a file operation by a second, and not every file operation causes it to produce output. Therefore, the parent process should do a nonblocking read of the pipe. If the pipe is empty, it should wait a second and try another read. If after a few tries with nothing read, it should assume **fswatch** didn't write anything and go on to the next file operation.

To organize the output and make it more intelligible, the parent process should print the equivalent Bash command, call the POSIX function(s), and then print the **fswatch** output that it read from the pipe.

Example output from the parent process (different on other operating systems):

```
----- mkdir subdir1
Fri Apr 16 01:02:19 2021 /Users/rmak/Asgn08-FS-Monitor/subdir1 Created IsDir
----- mkdir subdir2
Fri Apr 16 01:02:20 2021 /Users/rmak/Asgn08-FS-Monitor/subdir2 Created IsDir
----- cd subdir1
----- open file1.txt, append, close
Fri Apr 16 01:02:25 2021 /Users/rmak/Asgn08-FS-Monitor/subdir1/file1.txt Created Updated IsFile
----- cp file1.txt file2.txt
Fri Apr 16 01:02:26 2021 /Users/rmak/Asgn08-FS-Monitor/subdir1/file2.txt Created Updated IsFile
```
*etc.*

When the parent process is done making the sequence of file system calls, it should terminate the child process, which will also terminate **fswatch**, and print "**Done!**",

## Tips

After the parent process forks the child process, the parent should sleep for a second to give the child process a chance to start **fswatch**.

The parent process should use a large (8K?) buffer to read the pipe. The **fswatch** command can generate many lines of output in response to certain file operations.

Do not use the **system()** function. Call the POSIX file functions.

## What to submit

Submit the following to Canvas, **Assignment #8: POSIX File System Functions**.

- Source files (either C or C++) of your program.
- A text file of your program's output.

## Rubric

Your submission will be graded according to these criteria:

| Criteria | Max points |
|---|---|
| **Program structure** | **50** |
| • Parent and child processes with pipe. | • 10 |
| • Parent process does nonblocking reads on the pipe. | • 10 |
| • Child process redirects standard output to the pipe. | • 10 |
| • Child process runs **`fswatch`** with the proper options. | • 20 |
| **File operations** | **50** |
| • POSIX file and directory functions called properly. | • 35 |
| • **`fswatch`** output printed by the parent process. | • 15 |