Won Joon Lee
15647075
CS146
Due 6/7/19

HW 5 Report

Part 1: **rename**

```
#!/bin/bash

# rename [-f] sed-command file1 file2 ...
# Need at least 2

usage() {
	echo "Usage: rename [-f] sed-substitution-command file1 file2 ..."
	exit 1
}

# Check if enough arguments
if [[ "$#" -lt 2 ]]; then
	echo "Given $# arguments. Need at least 2."
	usage
fi

flag=false
start=1

# Check for option
if [[ $1 == -* ]]; then
	if [[ $1 == -f ]]; then
		# Set flag to true and update start
		flag=true
		start=2
	else
		echo "Invalid option given: $1"
		usage
	fi
fi

args=("$@")

n=$(expr $start - 1)
```

```
SED=${args[$n]}

while [ $start -lt "$#" ];
do
	arg=${args[$start]}

	# If arg is not found
	if [ ! -f $arg ]; then
		echo File \( $arg \) Not found.
		start=$(expr $start + 1)
		continue
	fi

	# Actual Renaming Segment
	dir=$(dirname $arg)
	base=$(basename $arg)
	new=$(echo $base | sed "s$SED")

	newpath=$(echo $(awk -v d=$dir -v newname=$new 'BEGIN {printf "%s/%s", d,
newname}'))
	echo $newpath

	# Check if flag is on
	if $flag; then
		mv -f $arg $newpath
	else
		# Check if the newpath is already an existing file
		if [ -f $newpath ]; then
			echo File \( $newpath \) already exists. Please use the \'-f\' option to
overwrite.
		else
			mv -f $arg $newpath
		fi
	fi

	# Increment the index
	start=$(expr $start + 1)
done
```

Part 2: **awkcel**

```
#!/bin/bash
```

```
usage() {
	echo "Usage: awkcel awk-cmd file (tab-separated file)"
	exit 1
}

# Check if correct num of arguments
if [[ "$#" -ne 2 ]]; then
	echo "Given $# arguments. Need exactly 2."
	usage
fi

AWK=$1
file=$2

# Check if file exists
if [ ! -f $file ]; then
	echo File \( $file \) does not exist.
	usage
fi

vars=""
index=1
header=true

while read -r line;
do
#	echo $line
	# Check if line is a comment line or an empty line
	if [[ $line == \#* ]]; then
		continue
	fi

	if $header; then
		for column in $(echo $line | awk -F '\t' '{for(i=1;i<=NF;i++) print $i}')
		do
			vars+="$column=\$$index;"
			index=$(expr $index + 1)
		done

		header=false
		continue
	fi
```

```
        echo $vars
        awk -F'\t' "{$vars} $AWK" $file

done < "$file"
```

Questions:

1.a)

1.b)

Part 3: **parallel**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/sysinfo.h>
#include <sys/types.h>
#include <unistd.h>
#include <ctype.h>

void error(char *msg)
{
        printf("%s\n", msg);
        exit(1);
}

void get_shell_cmd(char commands[][BUFSIZ], int cores)
{
        char cmd[BUFSIZ];
        for (int i = 0; i < cores; ++i)
        {
                if (fgets(cmd, BUFSIZ, stdin))
                        strcpy(commands[i], cmd);
                else
                        commands[i][0] = '\0';

//              printf("cmd = %s\n", cmd);
        }
}

pid_t exec_cmd(char *args, char *shell)
{
```

```
	pid_t pid;

	if ((pid = fork()) < 0)
		error("Fork Failed.");
	else if (pid == 0)
		if (execl(shell, shell, "-c", args, (char *) NULL) < 0)
			error("Exec Failed.");
	return pid;
}

int handle_wait(pid_t p, int cores)
{
	int count = 0;
	pid_t status, w;

	if (p)
		do
		{
			w = waitpid(p, &status, WUNTRACED);
			if (w == -1 || WIFSIGNALED(status))
				++count;
		} while (!WIFEXITED(status) && !WIFSIGNALED(status));
	return count;
}

int main(int argc, char *argv[], char *envp[])
{
	int cores, argv_i, shell_found;
	char shell[BUFSIZ];
	for (argv_i = 0; argv_i < argc; ++argv_i)
	{
		if ((argv[argv_i][0]) == '-')
			switch (argv[argv_i][1])
			{
				case 's':
					strcpy(shell, argv[argv_i+1]);
					shell_found = 1;
					break;
				default:
					error("Invalid Option.");
			}

			if (shell_found) break;
```

```
	}

	if (!shell_found)
	{
		strcpy(shell, getenv("SHELL"));
		if (argc == 1 || (cores = atoi(argv[1])) == 0)
			cores = get_nprocs();
	}
	else
		if ((argv_i + 2 >= argc) || (cores = atoi(argv[argv_i + 2])) == 0)
			cores = get_nprocs();

	if (cores > get_nprocs())
		error("# of cores specified greater than available cores.");

//	printf("cores: %d, shell: %s\n", cores, shell);

	char commands[cores][BUFSIZ];
	int end = 0, mark = 0, count = 0;
	pid_t process[cores];

	while (!end)
	{
		get_shell_cmd(commands, cores);
		for (int i = 0; i < cores; ++i)
		{
			if (commands[i][0])
				process[i] = exec_cmd(commands[i], shell);
			else
			{
				end = 1;
				break;
			}
			++mark;
		}

		for (int i = 0; i < mark; ++i)
			count += handle_wait(process[i], cores);
	}

	return count;
}
```