WEEK 4 5 & 6

Create a shell script that counts word occurrences in files. Save this script to a file, make it executable (`chmod +x word_count.sh`), and use it to count word occurrences in files by providing the file names as arguments.

by sahitya

A) Word Count Script

Usage

Usage: word_count.sh <file1> [<file2> ...]

Algorithm

The shell script uses awk to iterate through each word in the file(s) and counts the occurrences of each word. The results are then displayed.

Example

Here's an example of the script in action:

```
Word occurrences in sample.txt:
apple 3
orange 2
banana 1
```

Write a shell script that accept a list of file names as arguments count and report the occurrence of each word.

You can create a shell script in bash that accepts a list of file names as arguments, counts the occurrence of each word, and reports the results. Here's a simple example of such a script:

```
#!/bin/bash

# Check if at least one file name is provided as an argument
if [ $# -lt 1 ]; then
  echo "Usage: $0 <file1> [<file2> ...]"
  exit 1
fi

# Iterate through each provided file
for file in "$@"; do
  # Check if the file exists
  if [ ! -e "$file" ]; then
    echo "File not found: $file"
    continue
  fi

  # Process the file and count word occurrences
  echo "Word occurrences in $file:"
  awk '
  {
    for (i = 1; i <= NF; i++) {
      word = $i
      gsub(/[^a-zA-Z0-9]/, "", word)  # Remove non-alphanumeric characters
      if (word != "") {
        words[word]++
      }
    }
  }
  END {
    for (word in words) {
      print word, words[word]
    }
  }' "$file"
done
```

B) Factorial Script

Usage

Usage: factorial.sh <integer>

Algorithm

The shell script takes an integer as an argument and calculates its factorial using a loop. The result is then displayed.

Example

Here's an example of the script in action:

```
Factorial of 5 is 120
```

write a shell script to find the factorial of given integer.

You can write a shell script to find the factorial of a given integer using a loop or recursion. Here's an example of a shell script that calculates the factorial of an integer using a loop:

```
#!/bin/bash

# Check if the user provided an integer as an argument
if [ $# -ne 1 ] || ! [[ $1 =~ ^[0-9]+$ ]]; then
  echo "Usage: $0 <integer>"
  exit 1
fi

# Function to calculate the factorial using a loop
calculate_factorial() {
  local num=$1
  local result=1

  for ((i = 1; i <= num; i++)); do
    result=$((result * i))
  done

  echo $result
}

# Call the function with the provided integer argument
factorial=$(calculate_factorial "$1")
echo "Factorial of $1 is $factorial"
```

C) List Files Script

Usage

Usage: list_files.sh <directory_path>

Algorithm

The shell script takes a directory path as an argument and lists all the files in the specified directory.

Example

Here's an example of the script in action:

```
Files in /home/user:
file1.txt
file2.txt
file3.txt
```

write a shell script that list the all files in a directory.

You can create a simple shell script to list all the files in a directory using the `ls` command. Here's a basic example of such a script:

```
#!/bin/bash

# Check if a directory path is provided as an argument
if [ $# -ne 1 ]; then
  echo "Usage: $0 <directory_path>"
  exit 1
fi

directory_path="$1"

# Check if the provided directory exists
if [ ! -d "$directory_path" ]; then
  echo "Directory not found: $directory_path"
  exit 1
fi

# List all files in the provided directory
echo "Files in $directory_path:"
ls -p "$directory_path" | grep -v /  # List files, not directories
```

Save this script to a file, make it executable (e.g., `chmod +x list_files.sh`), and then you can use it to list all the files in a specified directory by providing the directory path as an argument. For example:

```
./list_files.sh /path/to/your/directory
```

Using System Calls: `cat` Command

Implementation

The `cat` command is implemented in C using system calls. It takes one or more file names as arguments and prints their content to the standard output.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    // Code omitted for brevity
}
```

Usage

Usage: cat <file1> [<file2> ...]

You can use awk to find the number of characters, words, and lines in a file. Here's an awk script to do that:

```
#!/bin/bash

# Check if a file name is provided as an argument
if [ $# -ne 1 ]; then
  echo "Usage: $0 <file_name>"
  exit 1
fi

file_name="$1"

# Check if the provided file exists
if [ ! -f "$file_name" ]; then
  echo "File not found: $file_name"
  exit 1
fi

# Use awk to count characters, words, and lines in the file
awk 'BEGIN {
  characters = 0
  words = 0
  lines = 0
}
{
  characters += length
  words += NF
  lines++
}
END {
  print "Characters:", characters
  print "Words:", words
  print "Lines:", lines
}' "$file_name"
```

Using System Calls: `mv` Command

Implementation

The `mv` command is implemented in C using system calls. It takes two arguments: the source file to move and the destination file where the source file should be moved to.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    // Code omitted for brevity
}
```

Usage

Usage: mv <source_file> <destination_file>

Write a C Program that makes a copy of a file using standard I/O and system calls.

To create a C program that makes a copy of a file using standard I/O and system calls, you can use functions like `fopen`, `fread`, `fwrite`, and `fclose` for standard I/O, and `open`, `read`, `write`, and `close` for system calls. Here's an example program that does just that:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
    fprintf(stderr, "Usage: %s <source_file> <destination_file>\n", argv[0]);
    return 1;
}

const char *source_file = argv[1];
const char *destination_file = argv[2;

// Using standard I/O
FILE *source_fp = fopen(source_file, "rb");
if (source_fp == NULL) {
    perror("Failed to open source file");
    return 1;
}

FILE *destination_fp = fopen(destination_file, "wb");
if (destination_fp == NULL) {
    perror("Failed to open destination file");
    fclose(source_fp);
    return 1;
}

char buffer[1024];
size_t bytes_read;

while ((bytes_read = fread(buffer, 1, sizeof(buffer), source_fp) > 0) {
    fwrite(buffer, 1, bytes_read, destination_fp);
}

fclose(source_fp);
fclose(destination_fp);

printf("File copied using standard I/O\n");

// Using system calls
int source_fd = open(source_file, O_RDONLY);
if (source_fd == -1) {
    perror("Failed to open source file");
    return 1;
}

int destination_fd = open(destination_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
if (destination_fd == -1) {
    perror("Failed to open destination file");
    close(source_fd);
    return 1;
}

while ((bytes_read = read(source_fd, buffer, sizeof(buffer)) > 0) {
    write(destination_fd, buffer, bytes_read);
    }

    close(source_fd);
    close(destination_fd);

    printf("File copied using system calls\n");

    return 0;
}
```

This program takes two command-line arguments: the source file to copy and the destination file. It uses both standard I/O and system calls to copy the contents of the source file to the destination file and then prints a message indicating which method was used.

Emulating the Unix `ls -l` Command

Emulating `ls -l`

We can emulate the `ls -l` command using system calls in C. This allows us to list the contents of a directory and retrieve file information.

Implementation

The implementation involves using system calls to list the contents of a directory and retrieve file information.

Usage

Usage: ls-l <directory_path>

Write a C program to emulate the Unix ls-l command.

To emulate the Unix `ls -l` command, you can use the `opendir`, `readdir`, and `stat` system calls to list the contents of a directory and retrieve file information. Here's a simple C program to emulate `ls -l`:

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>

void print_permissions(mode_t st_mode) {
    char permissions[11];

    permissions[0] = S_ISDIR(st_mode) ? 'd' : '-';
    permissions[1] = (st_mode & S_IRUSR) ? 'r' : '-';
    permissions[2] = (st_mode & S_IWUSR) ? 'w' : '-';
    permissions[3] = (st_mode & S_IXUSR) ? 'x' : '-';
    permissions[4] = (st_mode & S_IRGRP) ? 'r' : '-';
    permissions[5] = (st_mode & S_IWGRP) ? 'w' : '-';
    permissions[6] = (st_mode & S_IXGRP) ? 'x' : '-';
    permissions[7] = (st_mode & S_IROTH) ? 'r' : '-';
    permissions[8] = (st_mode & S_IWOTH) ? 'w' : '-';
    permissions[9] = (st_mode & S_IXOTH) ? 'x' : '-';
    permissions[10] = '\0';

    printf("%s ", permissions);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <directory_path>\n", argv[0]);
        return 1;
    }

    const char *directory_path = argv[1];
    DIR *dir;
    struct dirent *entry;
    struct stat file_info;

    dir = opendir(directory_path);
    if (dir == NULL) {
        perror("Failed to open directory");
        return 1;
    }

    while ((entry = readdir(dir)) != NULL) {
        char file_path[1024];
        snprintf(file_path, sizeof(file_path), "%s/%s", directory_path, entry->d_name);

        if (stat(file_path, &file_info) == -1) {
            perror("Failed to get file information");
            continue;
        }

        print_permissions(file_info.st_mode);
        printf("%ld ", (long)file_info.st_nlink);

        struct passwd *pw = getpwuid(file_info.st_uid);
        struct group *gr = getgrgid(file_info.st_gid);

        if (pw != NULL) {
            printf("%s ", pw->pw_name);
        }

        if (gr != NULL) {
            printf("%s ", gr->gr_name);
        }

        printf("%lld ", (long long)file_info.st_size);

        char time_buf[32];
        strftime(time_buf, sizeof(time_buf), "%b %d %H:%M", localtime(&file_info.st_mtime));
        printf("%s ", time_buf);

        printf("%s\n", entry->d_name);
    }

    closedir(dir);
    return 0;
}
```

This program takes a directory path as a command-line argument and lists the contents of the directory in a format similar to `ls -l`. It prints file permissions, number of hard links, owner, group, file size, modification time, and file name for each file in the directory.