Lab-1 →

Basic Linux Commands

Objective: To use different commands in Linux Operating system.

Theory:
Linux is an operating system like macOS or windows. It is also the most popular open source & free.

Commands:

1. ls (list):
→ The 'ls' command is used to lists files & directories in the specified directory.

2. pwd:
→ The 'pwd' command prints the current working directory.

3. cd:
→ The 'cd' command is used to changes the current working directory.

4. 'mkdir':
→ The 'mkdir' command is used to ~~changes~~ create a new directory.

5. touch:
→ The 'touch' command is commonly used to update the access & modification times of a file, but it can also be used to create an empty file if the file doesn't already exist.

6. echo:
→ The 'echo' command is used to write something into the files.

7. cat:
→ The 'cat' command is used for concatenating & displaying the content of files.

8. cp:
→ The 'cp' command is used to copy the contents of source file to the destination file. If destination file already exists, it will be overwritten.

9. mv:
→ The 'mv' command is used to rename or move file & directories.

10. rm:
→ The 'rm' command is used to remove files & directories

11. rmdir:
→ The 'rmdir' command is specifically designed to remove empty directories.

12. date:
→ The 'date' command is used to display the current date & time.

13 ps:
→ The 'ps' command is used to display information about currently running processes.

14. sudo:
→ sudo is one of the basic commands in Linux. It runs with administrative or root permissions.

15. clear:
→ The 'clear' command is used to clear the screen.

16. 'chmod':
→ The 'chmod' is used to change the file permissions.

17. useradd:
→ The 'useradd' command is used to create a new user account.

18. passwd:
→ The 'passwd' command is used to change a user password.

19. history:
→ The 'history' command is used to display a list of recently executed commands.

20. man:
→ The 'man' command in Linux is used to display the manual pages for various commands, utilities & system calls.

Conclusion:
Hence, the use of basic different Linux commands were learnt with the help of different commands.

Lab-5 →

1. Modify the program of Ex 3.2 for creating four child processes using system system call.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main( ){
        system ("ls");
        system("id");
        system ("whoami");
        system("date");
}
```

3. Write the program that creates the four process using fork() system call.

Theory

'fork()' is a system call in Linux that is used to create a new process. When 'fork()' is called, it created a new process, which is called the child process.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(){
    for(int i=0; i<4; ++i){
        if(fork()==0){
            printf("child %d created\n", i+1);
            exit(EXIT_SUCCESS);
        }
    }
    printf("Parent Process\n");
    while(wait(NULL)>0);
    return 0;
}
```

2. List the uses of exec family and write at least two programs that uses exec system call.

Ans:

Theory:

The 'exec' family of functions in C are used to replace the current process image with a new process image. They are typically used after a 'fork' system call in order to execute a different program in the child process.

The uses of 'exec' family are as follows:

* To load a different program
→ For the shell commands execution
→ For executing shell scripts
→ Dynamic loading of libraries.

Program 1:

```
#include <unistd.h>
int main() {
    execlp("ls", "ls", "-l", "/", NULL);
    return 0;
}
```

Program 2 →

```
#include <unistd.h>
int main() {
    char *args[] = {"ls", "-l", "/", NULL};
    execvp("ls", args);
    return 0;
}
```

Lab-4 →

1. Write a shell script program to display list of users currently logged in.

Theory

The shell script program is written in the Bash scripting language, denoted by the shebang '#!/bin/bash' at the beginning of the script. This indicates that the script should be interpreted by the Bash shell.

Code

```
#!/bin/bash
echo "List of users currently logged in:"
who
```

2. Write a shell script program to display the long list of directories.

Theory

The shell script program is written in the Bash scripting language, denoted by the shebang '#!/bin/bash' at the beginning of the script.

Program

```
#!/bin/bash
echo "Long list of directories:"
ls -ld */
```

3. Write a shell script program to check whether the given number is even or odd.

Theory

The 'echo -n' command is used to prompt message asking the user to enter a number. Also read command is used to read the input entered by the user.

Program

```
#!/bin/bash
echo -n "Enter a number: "
read number
if((number %2 == 0)); then
    echo "$number is even"
else
    echo "$number is odd"
fi
```

4. Write a shell script to take filename as input & delete the file.

Theory

'echo' command is used to prompt message asking the user to enter the filename.

'rm' command is used to delete the file specified by the filename variable.

Program

```
#!/bin/bash
echo -n "Enter the filename to delete"
read filename
if [-f "$filename"]; then
    rm "$filename"
    echo "File '$filename' deleted succesfully"
else
    echo "File '$filename' doesnot exist"
fi
```

5. Write a shell script to add numbers from 1 to N.

Theory

The shell script program is written in the Bash scripting language, denoted by the shebang '#!/bin/bash' at the beginning of the script.

Program

```
#!/bin/bash
echo -n "Enter the number: "
read N
sum = 0
for ((i=1; i<=N; i++)); do
    sum = $((sum+i))
done
echo "The sum of numbers from 1 to $N is: $sum"
```

8. Write a shell script to find the factorial of N.

Theory

The shell script program is written in the Bash scripting language, denoted by the shebang '#!/bin/bash' at the beginning of the script.

Program

```
#!/bin/bash
echo -n "Enter a number: "
read N
factorial = 1
for((i=1; i<=N; i++)); do
    factorial = $((factorial*i))
done
echo "The factorial of $N is: $factorial"
```

Lab-5

1. Write a program using threads that prints sum of numbers up to given positive number n.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define MAX_THREADS 4
int n;
int sum = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void * calculate_sum(void *arg) {
    int start = *(int) arg;
    int local_sum = 0;
    for (int i = start; i <= n; i = MAX_THREADS) {
        local_sum = local_sum + i;
    }
    pthread_mutex_lock(&mutex);
    sum += local_sum;
    pthread_mutex_unlock(&mutex);
    return NULL;
}
int main() {
    pthread_t threads[MAX_THREADS];
    int thread_args[MAX_THREADS];
    printf("Enter a positive number: ");
    scanf("%d", &n);
    for (int i = 0; i < MAX_THREADS; i++) {
        thread_args[i] = i + 1;
        pthread_create(&threads[i], NULL, calculate_sum, &
            thread_args[i]);
    }
    for (int i = 0; i <= MAX_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Sum of numbers upto %d is %d\n", n, sum);
    return 0;
}
```

2. Write a program that have two threads, one reads a word from keyboard & other check for valid word.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#define MAX_WORD_LENGTH 20
#define WORD_LIST_SIZE 10
char word_list[WORD_LIST_SIZE][MAX_WORD_LENGTH] = {
"apple", "sangam", "subedi", "aaditya", "khatri", "bishnu" "gautam" };
int valid_word_found = 0;
pthread_mutex_t = pthread_mutex_initilizer;
int is_valid_word (char *word) {
    for (int i=0; i<= WORD_LIST_SIZE; i++) {
        if (strcmp(word, word_list[i] == 0) {
            return 2;
        }
    }
}
void * read_words (void *args) {
    char word[MAX_WORD_LENGTH];
    printf("Enter a word:");
    scanf("%s", word);
    pthread_mutex_lock(&mutex);
    valid_word_found = is_valid_word(word);
    pthread_mutex_unlock(&mutex);
    return NULL;
}
void * check_validity (void * args) {
    while (1) {
        pthread_mutex_lock(&mutex);
        if (valid_word_found) {
            printf("valid word\n");
            valid_word_found = 0;
        }
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

```
int main(){
  pthread_t thread_read, thread_check;
  pthread_create(&thread_check, NULL, check_validity, NULL);
  while(1){
      pthread_create(&thread_read, NULL, read_words, NULL);
      pthread_join(thread_read, NULL);
  }
  return 0;
}
```

Lab-6 →

1. Write a program to demonstrate best fit, worst fit & first fit partition selection algorithm.

Program for Best fit →

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main(){
   int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
   static int bf[max], ff[max];
   printf("Enter the no. of blocks: ");
   scanf("%d", &nb);
   printf("Enter the no. of files: ");
   scanf("%d", &nf);
   printf("Enter the size of blocks\n");
   for(i=1; i<=nb; i++){
      printf("Block %d", i);
      scanf("%d", &b[i]);
   }
   printf("Enter the size of files\n");
   for(i=1; i<=nf; i++){
      printf("File %d", i);
      scanf("%d", &f[i]);
   }
   for(i=1; i<=nf; i++){
      for(j=1; j<=nb; j++){
         if(bf[j] != 1){
            temp = b[j] - f[i];
            if(temp > 0){
               if(lowest > temp){
                  ff[i] = j;
                  lowest = temp;
               }
            }
         }
      frag[i] = lowest;
      bf[ff[i]] = 1;
      lowest = 10000;
   }
   for(i=1; i<=nf && ff[i] != 0; i++){
      printf("%d\t%d\t%d\t%d\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
   }
}
```

Program for worst fit

```
#include <stdio.h>
#include <conio.h>
#define max 25
void main(){
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max], ff[max];
    printf("Enter the no. of blocks: ");
    scanf("%d", &nb);
    printf("Enter the no. of files: ");
    scanf("%d", &nf);
    printf("Enter the size of blocks");
    for(i=1; i<=nb; i++){
        printf("Block %d", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of files");
    for(i=1; i<=nf; i++){
        printf("File %d", &i);
        scanf("%d", &f[i]);
    }
    for(i=1; i<=nf; i++){
        for(j=1; j<=nb; j++){
            if(bf[j]!=1){
                temp = b[j]-f[i];
                if(temp>0){
                    ff[i]=j;
                    break;
                }
            }
        }
        frag[i]=temp;
        bf[ff[j]]=1;
        for(i=1; i<=nf; i++)
            printf("%d\t%d\t%d\t%d\t%d\n", i, f[i], ff[i], b[ff[i]],
                frag[i]);
    }
}
```

Program for first fit

```
#include <stdio.h>
#define max 25
void main() {
    int frag[max], b[max], f[max], i, j, nb, nf, temp, highest, t=0;
    static int bf[max], ff[max];
    printf("Enter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);
    printf("Enter the size of blocks: ");
    for(i=1; i<=nb; i++) {
        printf("Block %d", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of files: ");
    for(i=1; i<=nf; i++) {
        printf("File %d", i);
        scanf("%d", &f[i]);
    }
    for(i=1; i<=nf; i++) {
        for(j=1; j<=nb; j++) {
            if(bf[j] != 1) {
                temp = b[j] - f[i];
                if(temp>0) {
                    if(highest<temp) {
                        frag[i] = highest;
                        bf[ff[j]] = 1;
                        highest = 0;
                    }
                }
                ff[i] = j;
                highest = temp;
            }
        }
        printf for(i=1; i<=nf; i++) {
            printf("%d\t%d\t%d\t%d\t%d\n", i, f[i], ff[i], b[ff[i]],
            frag[i]);
        }
    }
}
```