1)Execute the following Linux Commands
a. Admin privileges commands
b. Basic Linux commands

a. Admin privileges commands:

To switch to the root user, use the "su" command:

```
su
```

You will be prompted to enter the root user's password.

To run a command with elevated privileges, use the "sudo" command:

bash

```
sudo command
```

You will be prompted to enter your own user password, assuming your user account has sudo privileges.

b. Basic Linux commands:

To list the contents of the current directory, use the "ls" command:

bash

```
ls
```

To change to a different directory, use the "cd" command:

bash

```
cd /path/to/directory
```

To create a new directory, use the "mkdir" command:

arduino

```
mkdir new_directory_name
```

To copy a file, use the "cp" command:

bash

```
cp source_file destination_file
```

To move a file, use the "mv" command:

bash

```
mv source_file destination_file
```

To remove a file, use the "rm" command:

bash

```
rm file_name
```

To display the contents of a file, use the "cat" command:

bash

```
cat file_name
```

To search for a pattern in a file, use the "grep" command:

perl

```
grep pattern file_name
```

To view the system processes running on your machine, use the "ps" command:

```
ps
```

To shut down or reboot the system, use the "shutdown" command:

arduino

```
shutdown now
```

or

arduino

```
shutdown -r now
```

(-r option indicates reboot)

2. Write a program to print the Child process ID and Parent process ID in both Child and Parent processes

```
#include <stdio.h>
#include <unistd.h>
int main() {
   pid_t pid;
   pid = fork(); // Create a child process
   if (pid == 0) { // Child process
      printf("Child process: Child PID is %d, Parent PID is %d\n",
getpid(), getppid());
   } else if (pid > 0) { // Parent process
      printf("Parent process: Child PID is %d, Parent PID is %d\n",
pid, getpid());
   } else { // fork() failed
      printf("fork() failed.\n");
      return 1; }
   return 0;}
```

C program to implement Producer-Consumer problem using semaphore

```
#include <stdio.h>
#include <stdlib.h>#include <pthread.h>
#include <semaphore.h>#define BUFFER_SIZE 10
int buffer[BUFFER_SIZE];
int count = 0;
sem_t empty, full, mutex;
void *producer(void *arg) {
   int item = 0;
   while (1) {
      item = rand();
      sem_wait(&empty);
      sem_wait(&mutex);
      buffer[count++] = item;
      printf("Producer produced item %d, count = %d\n", item,
count);
      sem_post(&mutex);
      sem_post(&full); }}
void *consumer(void *arg) {
   int item = 0;
   while (1) {
      sem_wait(&full);
      sem_wait(&mutex);
      item = buffer[--count];
      printf("Consumer consumed item %d, count = %d\n", item,
count);
      sem_post(&mutex);
      sem_post(&empty); }}
int main() {
   pthread_t producer_thread, consumer_thread;
   sem_init(&empty, 0, BUFFER_SIZE);
   sem_init(&full, 0, 0);
   sem_init(&mutex, 0, 1);
   pthread_create(&producer_thread, NULL, producer, NULL);
   pthread_create(&consumer_thread, NULL, consumer, NULL);
   pthread_join(producer_thread, NULL);
   pthread_join(consumer_thread, NULL);
   sem_destroy(&empty);
   sem_destroy(&full);
   sem_destroy(&mutex);  return 0;}
```

3. Execute the following unix commands: cat, rmdir, mkdir, rm, cp, mv

Here are the explanations of the Unix commands cat, rmdir, mkdir, rm, cp, and mv:

cat: This command is used to display the contents of a file on the terminal. For example, cat file.txt will display the contents of the file file.txt on the terminal

1
2
3
.

rmdir: This command is used to remove an empty directory. For example, rmdir directory_name will remove the directory directory_name if it is empty

4
5
2
.

mkdir: This command is used to create a new directory. For example, mkdir directory_name will create a new directory with the name directory_name

4
1
2
.

rm: This command is used to remove files or directories. For example, rm file.txt will remove the file file.txt, and rm -r directory_name will remove the directory directory_name and all its contents recursively

4
5
2
3
.

cp: This command is used to copy files or directories. For example, cp file.txt new_file.txt will create a copy of file.txt with the name new_file.txt, and cp -r directory_name new_directory_name will create a copy of directory_name with the name new_directory_name and all its contents recursively

4
1
2
.

mv: This command is used to move or rename files or directories. For example, mv file.txt new_file.txt will rename file.txt to new_file.txt, and mv directory_name new_directory_name will rename directory_name to new_directory_name. It can also be used to move files or directories to a different location. For example, mv file.txt directory_name will move file.txt to directory_name

4
1
2

Note that these commands are case-sensitive and may have different options and arguments depending on the Unix system being used. It is important to use them with caution, especially when removing or modifying files and directories

5. Write a shell script to reverse and calculate the length of the string

```
#!/bin/bash
echo "Enter a string: "
read string
# reverse the string
reverse=""
for ((i=${#string}-1; i>=0; i--))
do
  reverse="$reverse${string:$i:1}"
done
echo "Reversed string: $reverse"
# calculate the length of the string
length=${#string}
echo "Length of the string: $length"
```

6. Write a C program to implement process management using the following

system calls fork, exec, getpid, exit, wait, close

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
int main() {
  pid_t pid;
  int status, fd;
  pid = fork();
  if (pid == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
  } else if (pid == 0) {
    printf("Child process ID: %d\n", getpid());
    fd = open("output.txt", O_WRONLY | O_CREAT | O_TRUNC,
0666);
    if (fd == -1) {
      perror("open failed");
      exit(EXIT_FAILURE);}
    dup2(fd, STDOUT_FILENO);
    close(fd);
    if (execl("/bin/ls", "ls", "-l", NULL) == -1) {
      perror("execl failed");
      exit(EXIT_FAILURE); }
  } else {
printf("Parent process ID: %d\n", getpid());
    if (wait(&status) == -1) {
      perror("wait failed");
      exit(EXIT_FAILURE);}
    if (WIFEXITED(status)) {
      printf("Child process exited with status %d\n",
WEXITSTATUS(status));
    } else if (WIFSIGNALED(status)) {
      printf("Child process terminated by signal %d\n",
WTERMSIG(status));}}
  return 0;}
```

7. Write a program to send a message (pass through command line arguments) into a message queue. Send few messages with unique message numbers

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX_MSG_LEN 100
struct msgbuf {
   long mtype;
   char mtext[MAX_MSG_LEN];};
int main(int argc, char *argv[]) {
   key_t key;
   int msgid, msg_len;
   struct msgbuf msg;
   if (argc < 2) {
      fprintf(stderr, "Usage: %s <message>\n", argv[0]);
      exit(EXIT_FAILURE);}
   key = ftok(argv[0], 'M');
   if (key == -1) {
      perror("ftok failed");
      exit(EXIT_FAILURE);}
   msgid = msgget(key, 0666 | IPC_CREAT);
   if (msgid == -1) {
      perror("msgget failed");
      exit(EXIT_FAILURE)}
unique message number
   for (int i = 1; i < argc; i++) {
      msg.mtype = i;
      strncpy(msg.mtext, argv[i], MAX_MSG_LEN);
      msg_len = strlen(msg.mtext) + 1;
      if (msgsnd(msgid, &msg, msg_len, 0) == -1) {
         perror("msgsnd failed");
         exit(EXIT_FAILURE);}
      printf("Sent message #%d: %s\n", i, msg.mtext);}
return 0;}
```

8. Write a shell program to check whether the given string is palindrome or not

```
#!/bin/sh

echo "Enter a string:"
read str

# reverse the string using the 'rev' command
rev_str=$(echo "$str" | rev)

if [ "$str" = "$rev_str" ]; then
   echo "$str is a palindrome"
else
   echo "$str is not a palindrome"
fi
```

9. Write a Shell program to check whether the given number is Armstrong or Not

```
#!/bin/sh
echo "Enter a number:"
read num
# count the number of digits in the number
num_digits=$(echo "$num" | wc -c)
num_digits=$((num_digits-1))
# initialize sum to zero
sum=0
# loop through each digit in the number
for ((i=1; i<=num_digits; i++)); do
   # extract the i-th digit
   digit=$(echo "$num" | cut -c "$i")
   # raise the digit to the power of the number of digits
   digit=$((digit**num_digits))
   # add the digit to the sum
   sum=$((sum+digit))
done
if [ "$sum" -eq "$num" ]; then
   echo "$num is an Armstrong number"
else
   echo "$num is not an Armstrong number"
fi
```

10.Write a shell script to reverse and calculate the length of the string

```
#!/bin/sh
echo "Enter a string:"
read str
# reverse the string using the 'rev' command
rev_str=$(echo "$str" | rev)
# count the length of the string using the 'wc -c' command
str_length=$(echo "$str" | wc -c)
str_length=$((str_length-1)) # subtract 1 to exclude the newline
character
echo "Original string: $str"
echo "Reversed string: $rev_str"
echo "Length of the string: $str_length"
```

11.Write a C program to implement anyone CPU scheduling algorithm

```
#include<stdio.h>
struct process {
   int pid;
   int bt;
   int rt;  };
int main() {
   int n, tq;
   float avg_wt = 0, avg_tat = 0;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter the time quantum: ");
   scanf("%d", &tq);
   struct process p[n];
   for(int i = 0; i < n; i++) {
      printf("Enter the burst time for process %d: ", i+1);
      scanf("%d", &p[i].bt);
      p[i].rt = p[i].bt;
      p[i].pid = i+1;}
   int t = 0;
   int done = 0;
   while(done < n) {
      for(int i = 0; i < n; i++) {
         if(p[i].rt > 0) {
            if(p[i].rt <= tq) {
               t += p[i].rt;
               p[i].rt = 0;
               avg_tat += t - p[i].bt;
               avg_wt += t - p[i].bt - p[i].bt;
               done++;
            } else {
               t += tq;
               p[i].rt -= tq;}}}}
   avg_tat /= n;
   avg_wt /= n;
   printf("Average turnaround time = %.2f\n", avg_tat);
   printf("Average waiting time = %.2f\n", avg_wt);
 return 0;}
```

12.Write a C program to implement anyone Disk Scheduling algorithm

```
#include<stdio.h>
#include<stdlib.h>
int main() {
   int n, head_pos, total_movement = 0;
   printf("Enter the number of requests: ");
   scanf("%d", &n);
   int requests[n];
   printf("Enter the requests: ");
   for(int i = 0; i < n; i++) {
      scanf("%d", &requests[i]);}
   printf("Enter the initial head position: ");
   scanf("%d", &head_pos);
   for(int i = 0; i < n-1; i++) {
      for(int j = 0; j < n-i-1; j++) {
         if(requests[j] > requests[j+1]) {
            int temp = requests[j];
            requests[j] = requests[j+1];
            requests[j+1] = temp;}  }}
   int start_index = 0;
   for(int i = 0; i < n; i++) {
      if(requests[i] > head_pos) {
         start_index = i;
         break;    }  }
   printf("Scanning from left to right:\n");
   for(int i = start_index; i < n; i++) {
      printf("%d -> ", requests[i]);
      total_movement += abs(requests[i] - head_pos);
      head_pos = requests[i]; }
   printf("0 -> ");
   total_movement += head_pos;
   head_pos = 0;
   for(int i = start_index-1; i >= 0; i--) {
      printf("%d -> ", requests[i]);
      total_movement += abs(requests[i] - head_pos);
      head_pos = requests[i];}
   printf("\nTotal movement: %d\n", total_movement);
   return 0;}
```

13.Write a C program to implement anyone Page replacement algorithm

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n, frames;
    printf("Enter the number of page requests: ");
    scanf("%d", &n);
    int requests[n];
    printf("Enter the page requests: ");
    for(int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);}
    printf("Enter the number of frames: ");
    scanf("%d", &frames);
    int page_faults = 0;
    int page_table[frames];
    int lru_counter[frames];
    for(int i = 0; i < frames; i++) {
        page_table[i] = -1;
        lru_counter[i] = 0;}
    for(int i = 0; i < n; i++) {
        int page = requests[i];
        int page_found = 0;
        for(int j = 0; j < frames; j++) {
            if(page_table[j] == page) {
                page_found = 1;
                lru_counter[j] = 0;
                break;}}
        if(!page_found) {
            int lru_index = 0;
            for(int j = 1; j < frames; j++) {
                if(lru_counter[j] > lru_counter[lru_index]) {
                    lru_index = j;} }
            page_table[lru_index] = page;
            lru_counter[lru_index] = 0;
            page_faults++; }
        for(int j = 0; j < frames; j++) {
            lru_counter[j]++;} }
    printf("Number of page faults: %d\n", page_faults);
    return 0;}
```