1. List out the number of lines, characters and words present in a file using the necessary UNIX command.

```
wc filename.txt
```

2. Create a process and print the process id of the current process and its parent process.

```
#include <stdio.h>

#include <unistd.h>

int main() {

   printf("Current Process ID: %d\n", getpid());

   printf("Parent Process ID: %d\n", getppid());

   return 0;

}
```

3. Write a Shell program to count the number of vowels in a line of text

```
#!/bin/bash

echo "Enter a line of text:"

read line

echo "$line" | grep -o -i '[aeiou]' | wc -l
```

4. Write a shell program to find the sum and average of four integers.

```
#!/bin/bash

echo "Enter four integers:"

read a b c d

sum=$((a + b + c + d))

avg=$(echo "$sum / 4" | bc -l)

echo "Sum: $sum"

echo "Average: $avg"
```

5. Give the correct UNIX commands that display a list of users who are currently logged in to a computer and which prints the calendar of the current month and year

```
Who

Cal
```

6. Write a shell program to check whether the given number is positive or negative.

```
#!/bin/bash

echo "Enter a number:"

read num

if [ "$num" -gt 0 ]; then

    echo "Positive number"

elif [ "$num" -lt 0 ]; then

    echo "Negative number"

else

    echo "Zero"

fi
```

7. Using the appropriate UNIX command print the last 10 lines of the user specified file to standard output.

```
tail -n 10 filename.txt
```

8. Write a C program for implementing an FCFS Scheduling algorithm by displaying process id, average waiting time and average turnaround time.

```
#include <stdio.h>

int main() {

    int n, i, wt[10], tat[10], bt[10];

    float avg_wt = 0, avg_tat = 0;

    printf("Enter number of processes: ");

    scanf("%d", &n);

    for(i=0; i<n; i++) {

        printf("Enter burst time for P%d: ", i+1);

        scanf("%d", &bt[i]);
    }
    wt[0] = 0;
    for(i=1; i<n; i++) {
        wt[i] = wt[i-1] + bt[i-1];
    }
    for(i=0; i<n; i++) {
        tat[i] = wt[i] + bt[i];
        avg_wt += wt[i];
        avg_tat += tat[i];
        printf("P%d\tWT=%d\tTAT=%d\n", i+1, wt[i], tat[i]);
    }
    printf("Average Waiting Time: %.2f\n", avg_wt/n);
    printf("Average Turnaround Time: %.2f\n", avg_tat/n);
    return 0;
}
```

9. Write a C program for implementing an SJF Scheduling algorithm by displaying process id, average waiting time and average turnaround time.

```
#include <stdio.h>
struct Process {
    int pid;
    int bt;
    int wt;
    int tat;
};
void sortByBurstTime(struct Process p[], int n) {
    struct Process temp;
    for (int i = 0; i < n-1; i++) {
        for (int j = 0; j < n-i-1; j++) {
            if (p[j].bt > p[j+1].bt) {
                temp = p[j];
                p[j] = p[j+1];
                p[j+1] = temp;
            }
        }
    }
}
int main() {
    int n;
    struct Process p[20];
    float total_wt = 0, total_tat = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Burst Time for Process P%d: ", p[i].pid);
        scanf("%d", &p[i].bt);
    }
    sortByBurstTime(p, n);
    p[0].wt = 0;
    for (int i = 1; i < n; i++) {
        p[i].wt = p[i-1].wt + p[i-1].bt;
    }
    for (int i = 0; i < n; i++) {
        p[i].tat = p[i].wt + p[i].bt;
        total_wt += p[i].wt;
        total_tat += p[i].tat;
    }
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", p[i].pid, p[i].bt, p[i].wt, p[i].tat);
    }
    printf("\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);
    return 0;
}
```

10. Write a C program for implementing Inter process communication using shared memory concept.

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
int main() {
    key_t key = ftok("shmfile",65);
    int shmid = shmget(key, 1024, 0666|IPC_CREAT);
    char *str = (char*) shmat(shmid, (void*)0, 0);
    printf("Write Data : ");
    fgets(str, 1024, stdin);
    printf("Data written in memory: %s\n", str);
    shmdt(str);
    return 0;
}
```

11. Write a C program for implementing the Round Robin Scheduling algorithm.

```
#include <stdio.h>
struct Process {
    int pid;
    int bt;
    int rt;
    int wt;
    int tat;
};
int main() {
    int n, tq;
    struct Process p[20];
    int time = 0, done;
    float total_wt = 0, total_tat = 0;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Burst Time for Process P%d: ", p[i].pid);
        scanf("%d", &p[i].bt);
        p[i].rt = p[i].bt;
    }
    printf("Enter Time Quantum: ");
    scanf("%d", &tq);
    do {
        done = 1;
        for (int i = 0; i < n; i++) {
            if (p[i].rt > 0) {
                done = 0;
                if (p[i].rt > tq) {
                    time += tq;
                    p[i].rt -= tq;
                } else {
                    time += p[i].rt;
                    p[i].wt = time - p[i].bt;
                    p[i].rt = 0;
                }
            }
        }
    } while (!done);
    for (int i = 0; i < n; i++) {
        p[i].tat = p[i].wt + p[i].bt;
        total_wt += p[i].wt;
        total_tat += p[i].tat;
    }
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", p[i].pid, p[i].bt, p[i].wt, p[i].tat);
    }
    printf("\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);
    return 0;
}
```

12. Write a C program for implementing the semaphore synchronization tool.

```
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
```

```
sem_t mutex;
int count = 0;
void* thread(void* arg) {
    sem_wait(&mutex);
    count++;
    printf("Thread %ld: Count = %d\n", (long)arg, count);
    sem_post(&mutex);
    return NULL;
}
int main() {
    pthread_t t1, t2;
    sem_init(&mutex, 0, 1);
    pthread_create(&t1, NULL, thread, (void*)1);
    pthread_create(&t2, NULL, thread, (void*)2);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&mutex);
    return 0;
}
```

13. Write a C program for implementing LRU page replacement algorithm.

```
#include <stdio.h>
int findLRU(int time[], int n) {
    int min = time[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (time[i] < min) {
            min = time[i];
            pos = i;
        }
    }
    return pos;
}
int main() {
    int frames, pages[100], n, frame[10], time[10];
    int faults = 0, hit = 0, counter = 0;
    printf("Enter number of pages: ");
    scanf("%d", &n);
    printf("Enter the page reference string:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &pages[i]);
    }
    printf("Enter number of frames: ");
    scanf("%d", &frames);
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
    }
    for (int i = 0; i < n; i++) {
        int found = 0;
        for (int j = 0; j < frames; j++) {
            if (frame[j] == pages[i]) {
                hit++;
                time[j] = ++counter;
                found = 1;
                break;
            }
        }
        if (!found) {
            int pos = -1;
            for (int j = 0; j < frames; j++) {
                if (frame[j] == -1) {
                    pos = j;
                    break;
                }
            }
            if (pos == -1) {
                pos = findLRU(time, frames);
            }
            frame[pos] = pages[i];
            time[pos] = ++counter;
            faults++;
        }
        printf("Step %d: ", i + 1);
        for (int j = 0; j < frames; j++) {
            if (frame[j] != -1)
                printf("%d ", frame[j]);
            else
                printf("- ");
        }
        printf("\n");
    }
    printf("\nTotal Page Faults = %d\n", faults);
    printf("Total Page Hits = %d\n", hit);
return 0;}
```