## (EXPERIMENT NO : 1) AIM: Hands on Unix Commands.

```
# 1. Create a directory named PracticeOS
mkdir PracticeOS
# 2. Move into the PracticeOS directory
cd PracticeOS
# 3. Create two files file1.txt and file2.txt
touch file1.txt file2.txt
# 4. Write content to file1.txt
echo "This is Operating System Practical." > file1.txt
# 5. Append content to file1.txt
echo "This is an additional line." >> file1.txt
# 6. Display the content of file1.txt
cat file1.txt
# 7. Copy file1.txt to file3.txt
cp file1.txt file3.txt
# 8. Rename file2.txt to renamedfile.txt
mv file2.txt renamedfile.txt
# 9. Display the list of files with details
ls -l
# 10. Delete file3.txt
rm file3.txt
# 11. Go one directory back
cd ..
 # 12. Delete the PracticeOS directory and its contents
rm -r PracticeOS
```

**FILE HANDLING COMMANDS**

```
 # 1. Create a new file
touch file1.txt
# 2. Add content to the file
echo "This is a sample file." > file1.txt
# 3. View the file contents
cat file1.txt
# 4. Copy the file
cp file1.txt file2.txt
# 5. Rename the file
mv file2.txt renamedfile.txt
# 6. Append content to the file
echo "This is additional content." >> file1.txt
# 7. Display first 5 lines of the file
head -n 5 file1.txt
# 8. Display last 5 lines of the file
tail -n 5 file1.txt
# 9. Delete a file
rm renamedfile.txt
# 10. View large files page by page
more file1.txt
# 11. View files with backward navigation
less file1.txt
```

## (EXPERIMENT NO : 2)AIM: Shell programming for file handling.

```
#!/bin/bash
echo "File Handling Menu"
echo "1. Create File"
echo "2. Read File"
echo "3. Append to File"
echo "4. Delete File"
echo "Enter your choice:"
read choice
case $choice in
 1) echo "Enter file name to create:"
      read filename
      touch $filename
      echo "File '$filename' created successfully."
      ;;
 2)  echo "Enter file name to read:"
      read filename
      if [ -f $filename ]; then
         echo "File Contents:"
         cat $filename
      else
         echo "File does not exist."
      fi
      ;;
 3)  echo "Enter file name to append:"
      read filename
      if [ -f $filename ]; then
         echo "Enter text to append:"
         read text
         echo $text >> $filename
         echo "Text appended successfully."
      else
         echo "File does not exist."
      fi
      ;;
4)   echo "Enter file name to delete:"
      read filename
      if [ -f $filename ]; then
         rm $filename
         echo "File deleted successfully."
      else
         echo "File does not exist."
      fi
      ;;
   *)
      echo "Invalid choice."
      ;;      esac
```

**(EXPERIMENT NO : 3) AIM: Shell Script programming using the commands grep, awk, and sed.**

```
#!/bin/bash
echo "File Processing using grep, awk, and sed"
# Create a sample file
echo -e "1 Alice 85\n2 Bob 78\n3 Charlie 92\n4 David 65" > students.txt
echo "Choose an operation:"
echo "1. Search for a student using grep"
echo "2. Display students with marks using awk"
echo "3. Replace student name using sed"
read choice
case $choice in
  1)
    echo "Enter student name to search:"
    read name
    echo "Search Result:"
    grep "$name" students.txt
    ;;
  2)
    echo "Displaying student names and marks:"
    awk '{print "Name: " $2 ", Marks: " $3}' students.txt
    ;;
  3)
    echo "Enter the student name to replace:"
    read oldname
    echo "Enter the new student name:"
    read newname
    sed -i "s/$oldname/$newname/g" students.txt
    echo "Name replaced successfully. Updated file content:"
    cat students.txt
    ;;
  *)
    echo "Invalid choice."
    ;;
esac
```

**(EXPERIMENT NO : 4) AIM: Implementation of various CPU scheduling algorithms (FCFS, SJF) Objective: To study and implement various CPU scheduling algorithms.**

***FCFS***

```
#include <stdio.h>
int main() {
int n, i;     int bt[20], wt[20], tat[20];
float avg_wt = 0, avg_tat = 0;
printf("Enter number of processes: ");
scanf("%d", &n);
printf("Enter burst time for each process:\n");
for(i = 0; i < n; i++) {
printf("Process %d: ", i+1);
scanf("%d", &bt[i]);    }
wt[0] = 0;
for(i = 1; i < n; i++) {
wt[i] = wt[i-1] + bt[i-1];   }
printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
for(i = 0; i < n; i++) {     tat[i] = bt[i] + wt[i];
avg_wt += wt[i];      avg_tat += tat[i];
printf("%d\t\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i], tat[i]);   }
printf("\nAverage Waiting Time = %.2f", avg_wt/n);
printf("\nAverage Turnaround Time = %.2f\n", avg_tat/n);
return 0;  }
```

***SJF***

```
#include <stdio.h>
int main() {
  int n, i, j, temp, bt[20], p[20], wt[20], tat[20];
  float avg_wt = 0, avg_tat = 0;
  printf("Enter number of processes: ");
  scanf("%d", &n);
  printf("Enter burst time for each process:\n");
  for(i = 0; i < n; i++) {
    printf("Process %d: ", i+1);
    scanf("%d", &bt[i]);
    p[i] = i + 1;  // process ID     }
  for(i = 0; i < n - 1; i++) {
    for(j = i + 1; j < n; j++) {
      if(bt[i] > bt[j]) {
        temp = bt[i];         bt[i] = bt[j];
        bt[j] = temp;         temp = p[i];
        p[i] = p[j];            p[j] = temp;   }  }  }
  wt[0] = 0;
  for(i = 1; i < n; i++) {
    wt[i] = wt[i-1] + bt[i-1];     }
  printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
  for(i = 0; i < n; i++) {
    tat[i] = bt[i] + wt[i];
    avg_wt += wt[i];
    avg_tat += tat[i];
    printf("%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);   }
  printf("\nAverage Waiting Time = %.2f", avg_wt/n);
  printf("\nAverage Turnaround Time = %.2f\n", avg_tat/n);
  return 0;      }
```

**(EXPERIMENT NO : 5) AIM: Implementation of various page replacement algorithms (FIFO) Objective: To study and implement various Page Replacement Algorithms.**

```
#include <stdio.h>
int main() {
   int i, j, n, frames, pages[30], temp[10], page_faults = 0, k = 0, flag;
   printf("Enter the number of pages: ");
   scanf("%d", &n);
   printf("Enter the page reference string: ");
   for(i = 0; i < n; i++) {
      scanf("%d", &pages[i]);
   }
   printf("Enter the number of frames: ");
   scanf("%d", &frames);
   // Initialize frames with -1 (empty)
   for(i = 0; i < frames; i++) {
      temp[i] = -1;
   }
   printf("\nPage Replacement Process (FIFO):\n");
   for(i = 0; i < n; i++) {
      flag = 0;
      // Check if page is already in frames
      for(j = 0; j < frames; j++) {
         if(temp[j] == pages[i]) {
            flag = 1;  // Page Hit
            break;
         }
      }
      // If page is not found, replace oldest
      if(flag == 0) {
         temp[k] = pages[i];
         k = (k + 1) % frames;  // Circular replacement
         page_faults++;
         // Print current frame status
         for(j = 0; j < frames; j++) {
            if(temp[j] != -1)
               printf("%d\t", temp[j]);
            else
               printf("-\t");
         }
         printf("\n");
      }
   }
   printf("\nTotal Page Faults: %d\n", page_faults);
   return 0;
}
```

**(EXPERIMENT NO : 6) AIM: Concurrent programming using system calls (fork and v-fork). Objective: Implementation of fork( ) and v-fork() system call in C.**

***Fork***

```
#include <stdio.h>

#include <unistd.h>

int main() {

   pid_t pid;
   pid = fork();  // Create a child process
   if (pid < 0) {
      printf("Fork failed.\n");
      return 1;
   } else if (pid == 0) {
      // Child process
      printf("This is the Child Process. PID: %d\n", getpid());
      printf("Child Process Completed.\n");
   } else {
      // Parent process
      printf("This is the Parent Process. PID: %d\n", getpid());
      printf("Parent Process Completed.\n");
   }
   return 0;
}
```

***Vfork.***

```
#include <stdio.h>
#include <unistd.h>
int main() {
   pid_t pid;
   pid = vfork();  // Create a child process (vfork)
   if (pid < 0) {
      printf("vfork failed.\n");
      return 1;
   } else if (pid == 0) {
      // Child process
      printf("This is the Child Process. PID: %d\n", getpid());
      printf("Child is exiting...\n");
      _exit(0);  // Must use _exit() to prevent affecting parent process
   } else {
      // Parent process
      printf("This is the Parent Process. PID: %d\n", getpid());
      printf("Parent Process Completed.\n");
   }
   return 0;
}
```

**(EXPERIMENT NO : 7) AIM: Implementation of Synchronization primitives – Semaphore for Producer Consumer Problem.**

```
#include <stdio.h>
int mutex = 1; // Semaphore for critical section
int full = 0;  // Counts filled slots
int empty = 5; // Buffer size (max slots)
void wait(int *s) {
  while (*s <= 0); // Busy wait
  (*s)--;      }
void signal(int *s) {
  (*s)++;   }
void producer() {
  wait(&mutex);  // Enter critical section
  wait(&empty);  // Check if buffer has empty slot
  full++;      // Produced an item
  printf("Producer produced an item. Full: %d, Empty: %d\n", full, empty - 1);
  signal(&mutex); // Exit critical section      }
void consumer() {
  wait(&mutex);  // Enter critical section
  wait(&full);   // Check if buffer has filled slot
  empty++;      // Consumed an item
  printf("Consumer consumed an item. Full: %d, Empty: %d\n", full - 1, empty);
  signal(&mutex); // Exit critical section      }
int main() {
  int n, i;
  printf("1. Producer\n2. Consumer\n3. Exit\n");
  for (i = 1; i <= 10; i++) {  // Run 10 operations
    printf("\nEnter your choice: ");
    scanf("%d", &n);
    switch (n) {
      case 1:
        if (empty == 0)
          printf("Buffer is full. Cannot produce.\n");
        else
          producer();
        break;
      case 2:
        if (full == 0)
          printf("Buffer is empty. Cannot consume.\n");
        else
          consumer();
        break;
      case 3:
        return 0;
      default:
        printf("Invalid choice.\n");
        break;   }   }
  return 0;
}
```

**(EXPERIMENT NO : 8) AIM: Implementation of Bankers algorithm Objective: To study and implement Bankers algorithm.**

```
#include <stdio.h>
int main() {      int n, m, i, j, k;
   int alloc[10][10], max[10][10], avail[10];
   int need[10][10], finish[10], safeSeq[10];
   int count = 0;
  printf("Enter the number of processes: ");       scanf("%d", &n);
   printf("Enter the number of resource types: ");      scanf("%d", &m);
   printf("Enter the Allocation Matrix:\n");
   for (i = 0; i < n; i++) {
   for (j = 0; j < m; j++) {
   scanf("%d", &alloc[i][j]);       }        }
printf("Enter the Maximum Matrix:\n");
 for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++) {
         scanf("%d", &max[i][j]);       }      }
printf("Enter the Available Resources:\n");
   for (j = 0; j < m; j++) {
      scanf("%d", &avail[j]);   }
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++) {
         need[i][j] = max[i][j] - alloc[i][j];       }
      finish[i] = 0; // Initialize all processes as not finished    }
printf("\nNeed Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < m; j++) {
         printf("%d ", need[i][j]);    }
      printf("\n");    }
   int y = 0; // index for safe sequence
   while (count < n) {
      int found = 0;
      for (i = 0; i < n; i++) {
         if (finish[i] == 0) {
            int canAllocate = 1;
            for (j = 0; j < m; j++) {
               if (need[i][j] > avail[j]) {
                  canAllocate = 0;        break;    }    }
            if (canAllocate) {
               for (k = 0; k < m; k++) {
                  avail[k] += alloc[i][k];     }
  safeSeq[y++] = i;       finish[i] = 1;       found = 1;
               count++;    }    }    }
      if (!found) {        printf("\nSystem is not in a safe state (Deadlock may occur).\n");
         return 0;}    }    }
   printf("\nSystem is in a safe state.\nSafe Sequence: ");
   for (i = 0; i < n; i++) {
      printf("P%d ", safeSeq[i]);      }
   printf("\n");
   return 0; }
```

**(EXPERIMENT NO : 9) AIM: Implementation of various memory allocation algorithms(First fit) Objective: To study and implement various memory allocation algorithms(First fit)**

```
#include <stdio.h>
int main() {
  int blockSize[10], processSize[10], blockNum, processNum;
  int allocation[10];
    printf("Enter the number of memory blocks: ");
  scanf("%d", &blockNum);
printf("Enter the size of each memory block:\n");
  for (int i = 0; i < blockNum; i++) {
    printf("Block %d size: ", i + 1);
    scanf("%d", &blockSize[i]);
  }
  printf("\nEnter the number of processes: ");
  scanf("%d", &processNum);
  printf("Enter the size of each process:\n");
  for (int i = 0; i < processNum; i++) {
    printf("Process %d size: ", i + 1);
    scanf("%d", &processSize[i]);
    allocation[i] = -1; // Initialize all as not allocated
  }
  // First Fit Allocation
  for (int i = 0; i < processNum; i++) {
    for (int j = 0; j < blockNum; j++) {
      if (blockSize[j] >= processSize[i]) {
        allocation[i] = j; // Allocate block j to process i
        blockSize[j] -= processSize[i]; // Reduce available block size
        break;
      }
    }
  }
  // Display Allocation Result
  printf("\nProcess No.\tProcess Size\tBlock No.\n");
  for (int i = 0; i < processNum; i++) {
    printf("%d\t\t%d\t\t", i + 1, processSize[i]);
    if (allocation[i] != -1)
      printf("%d\n", allocation[i] + 1);
    else
       printf("Not Allocated\n");
  }
 return 0;
}
```

**(EXPERIMENT NO : 10) AIM: Implementation of Disk Scheduling algorithms (FCFS,SSTF) Objective: To study and implement Disk Scheduling algorithms (FCFS,SSTF)**

***FCFS***
```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int n, i, head, seek = 0, diff;
   int queue[100];
   printf("Enter the number of disk requests: ");
   scanf("%d", &n);
   printf("Enter the disk request sequence: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &queue[i]);        }
   printf("Enter the initial head position: ");
   scanf("%d", &head);
   for (i = 0; i < n; i++) {
      diff = abs(queue[i] - head);     seek += diff;
      head = queue[i];        }
   printf("\nTotal Seek Time (FCFS) = %d\n", seek);
   printf("Seek Sequence: ");
   for (i = 0; i < n; i++) {
      printf("%d ", queue[i]);         }
   printf("\n");
   return 0;          }
```
***SSTF***
```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int n, i, j, head, seek = 0, min, diff, index;
   int queue[100], visited[100] = {0};
   printf("Enter the number of disk requests: ");
   scanf("%d", &n);
   printf("Enter the disk request sequence: ");
   for (i = 0; i < n; i++) {
      scanf("%d", &queue[i]);      }
   printf("Enter the initial head position: ");
   scanf("%d", &head);
printf("\nSeek Sequence: ");
   for (i = 0; i < n; i++) {
      min = 9999;
      for (j = 0; j < n; j++) {
         if (!visited[j]) {
            diff = abs(queue[j] - head);
            if (diff < min) {
               min = diff;
               index = j;     }     }       }
      visited[index] = 1;
      seek += abs(queue[index] - head);
      head = queue[index];
      printf("%d ", head);       }
   printf("\nTotal Seek Time (SSTF) = %d\n", seek);
   return 0;      }
```