LAD - 1

Title: WSL Installation Process in Windows

Windows Subsystem for Linux (WSL) enables developers to run a GNU/Linux environment directly on Windows without the need for a virtual machine.

Process:

i) Enable WSL:
- Open Powershell as Administer.
- Run the command:
  wsl --install
- This will enable WSL, install the Linux kernel, & set up the default distribution (Ubuntu).

ii) Restart Your System: After enabling WSL, restart your computer.

iii) Check WSL Version:
- To confirm WSL 2 is installed:
  wsl --list --verbose

  Ensure the version column displays "2".

iv) Install Additional Distributions (Optional):
- Open the Microsoft Store, search for your desired Linux distribution, & install it.

v) Launch WSL:
- Open your installed distribution from the start menu to begin using Linux.

LAB-2

Title: Execute Linux Commands with Syntax & Examples

Command	Description	Syntax/Example
pwd	Print the current working directory.	pwd
ls	List files & directories.	ls
cd	Change the current directory	cd /home
rm -d [directory_name]	Remove an empty directory	rm -d test_dir
clear	Clear the terminal screen	clear
ls -ls	List files & directories with detailed information	ls -ls
sudo	Execute a command as a superuser.	sudo apt update
cat file.txt	Display the contents of a file.	cat file.txt
cat > file.txt	Create a new file & write content to it.	cat > newfile.txt
history	Display the list of executed commands	history
date	Display the current date.	date
time	Display system time	time

LAB - 3

Title: Running C Program & Process Creation in Linux

→ To compile & execute a C program on Linux:

i) Write the C Program: Create a file using a text editor, eg: nano program.c, & write your C code.

ii) Compile the Program:

```
gcc program.c -o program
```

iii) Run the program:

```
./program
```

Example: Process Creation:

```
#include<stdio.h>
#include <unistd.h>
int main() {
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child process\n");
    } else {
        printf("Parent process\n");
    }
    return 0;
}
```

Compile & execute the program:

```
gcc process.c -o process
./process
```

LAB-4

Title: Simulation of Producer-Consumer Problem Using Semaphores

→ Code:

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int count = 0;
sem_t empty, full;
pthread_mutex_t mutex;
void *producer(void *arg){
   for(int i = 0; i<10; i++){
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[count++] = i;
      printf("Produced: %d\n", i);
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
   return NULL;
}
void *consumer(void *arg){
   for(int i=0; i<10; i++){
      sem_wait(&full);
      pthread_mutex_lock(&mutex);
      int item = buffer[--count];
      printf("Consumed: %d\n", item);
      pthread_mutex_unlock(&mutex);
      sem_post(&empty);
   }
   return NULL;
}
```

```
int main() {
    pthread_t prod, cons;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

LAB - 5

Title: Simulation of CPU scheduling algorithms (1)
First-Come, First-Served (FCFS)

→ Code:

```
#include<stdio.h>
void findWaitingTime(int process[], int n, int bt[], int wt[]) {
    wt[0] = 0;
    for(int i = 1; i<n; i++) {
        wt[i] = bt[i-1] + wt[i-1];
    }
}
void findTurnAroundTime(int process[], int n, int bt[], int wt[], int tat[]) {
    for(int i=0; i<n; i++) {
        tat[i] = bt[i] + wt[i];
    }
}
void findAvgTime(int processes[], int n, int bt[]) {
    int wt[n], tat[n];
    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);
    printf("Processes BurstTime Waiting time Turn around time \n");
    for(int i=0; i<n; i++) {
        printf(" %d\t %d\t %d\t %d\t\n", processes[i], bt[i],
                                        wt[i], tat[i]);
    }
}
int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof processes / size of processes[0];
    int burst_time[] = {10, 5, 8};
    findAvgTime(processes, n, burst_time);
    return 0;
}
```

LAB-6

Title: Simulation of Contiguous Memory Allocation Algorithms

Best Fit Algorithm

→ Code:

```
#include <stdio.h>
#define MAX 25
void bestFit(int blockSize[], int m, int processSize[], int n){
    int allocation[n];
    for(int i = 0; i<n; i++){
        allocation[i] = -1;
    }
    for (int i=0; i<n; i++){
        int bestIdx = -1;
        for(int j=0; j<m; j++){
            if (blockSize[j] >= processSize [i]){
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]){
                    bestIdx = j;
                }
            }
        }
        if ( bestIdx != -1){
            allocation[i] = bestIdx;
            blockSize[bestIdx] -= processSize[i];
        }
    }
    printf("Process No. Process Size Block No. \n");
    for (int i=0; i<n; 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");
    }
}
int main(){
    int blockSize[] = {200, 500, 200, 300, 600};
    int processSize[]= {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n= sizeof(processSize) / sizeof(processSize[0]);
    bestFit(blockSize, m, processSize, n);
    return 0;
}
```

LAB-7

Title: Simulation of Page Replacement Algorithms
Least Recently Used (LRU)

→ Code:

```
include <stdio.h>
int findLRU(int time[], int n){
  int i, minimum = time[0], pos = 0;
  for(i = 1; i < n; ++i){
    if(time[i] < minimum){
      minimum = time[i];
      pos = i;
    }
  }
  return pos;
}
void LRU(int pages[], int n, int capacity){
  int frames[capacity], counter = 0, time[capacity];
  int flag1, flag2, pos, faults = 0;
  for(int i = 0; i < capacity; ++i){
    frames[i] = -1;
  }
  for(int i = 0; i < n; ++i){
    flag1 = flag2 = 0;
    for(int j = 0; j < capacity; ++j){
      if(frames[j] == pages[i]){
        counter++;
        time[j] = counter;
        flag1 = flag2 = 1;
        break;
      }
    }
    if(flag1 == 0){
      for(int j = 0; j < capacity; ++j){
        if(frames[j] == -1){
          counter++;
          faults++;
          frames[j] = pages[i];
          time[j] = counter;
          flag2 = 1;
          break;
        }
      }
    }
```

```
        if (flag2 == 0) {
            pos = findLRU(time, capacity);
            counter++;
            faults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        printf("\n");
        for(int j = 0; j < capacity; ++j) {
            if (frames[j] != -1) {
                printf("%d\t", frames[j]);
            } else {
                printf("-\t");
            }
        }
    }
}
int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity = 3;
    LRU(pages, n, capacity);
    return 0;
}
```

LAB-8

Title: Simulation of Disk Scheduling Algorithms

Shortest Seek Time First (SSTF)

→ Code

```
#include <stdio.h>
#include <stdlib.h>
int findClosest(int head, int request[], int n, int visited[]) {
  int min = __INT_MAX__, index = -1;
  for(int i= 0; i<n; i++){
    if(!visited[i]){
      int distance = abs(head- request[i]);
      if(distance <min) {
        min = distance;
        index = i;
      }
    }
  }
  return index;
}
void SSTF(int request[], int n, int head) {
   int visited[n];
   for(int i= 0; i<n; i++) {
      visited[i] = 0;
  }
   printf("Sequence of disk access: \n");
   for(int i= 0; i<n; i++){
     int index = findClosest(head, request, n, visited);
     visited[index] = 1;
     printf("%d → ", request[index]);
     head = request[index];
  }
  printf("\n");
}
int main() {
  int request[] = {176, 79, 34, 60, 92, 11, 41, 114};
  int n= sizeof(request) / sizeof(request[0]);
  int head = 50;
  SSTF(request, n, head);
  return 0;
}
```