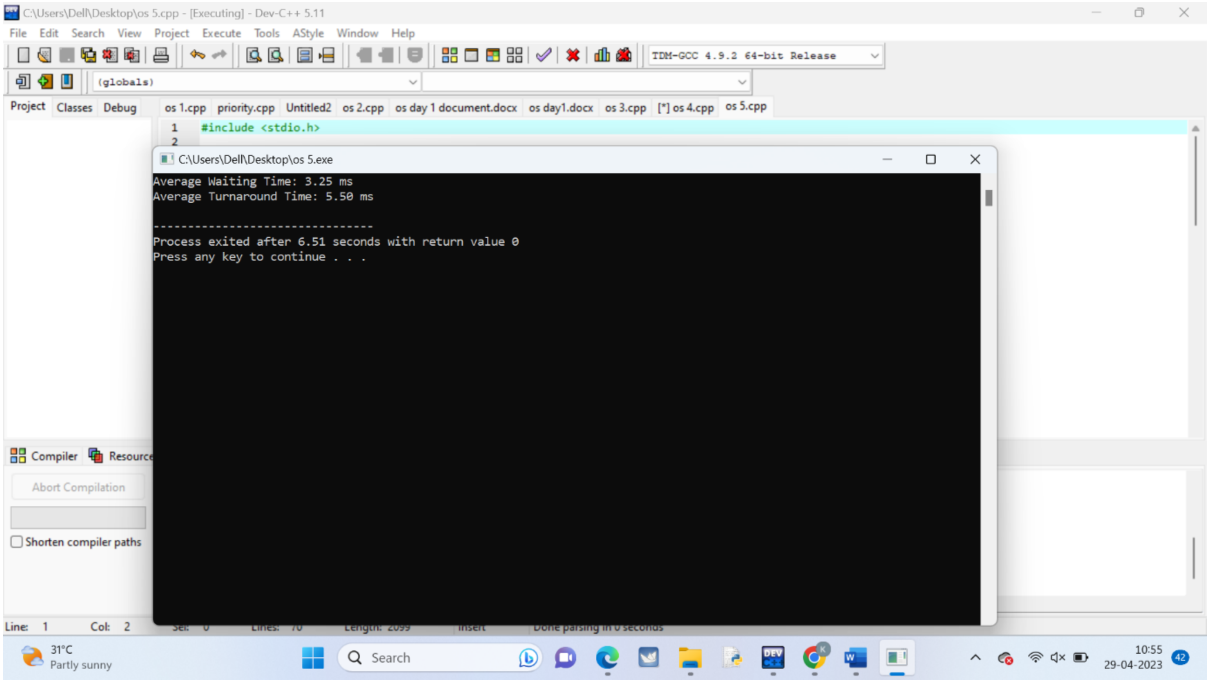
5. Write a program to compute the average waiting time and turnaround time based on Preemptive shortest remaining processing time first (SRPT) algorithm for the following set of processes, with the arrival times and the CPU-burst times given in milliseconds

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	3
P4	4	1

Program:

```
#include <stdio.h>
struct process {
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int completed;
};
int main() {
    int n = 4, t = 0, min_burst_time, min_index;
    struct process processes[] = {
        {0, 5, 5, 0, 0, 0},
        {1, 3, 3, 0, 0, 0},
        {2, 3, 3, 0, 0, 0},
        {4, 1, 1, 0, 0, 0}
    };
    while (1) {
        min_burst_time = 9999;
        min_index = -1;
                            for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= t && processes[i].completed == 0) {
                if (processes[i].remaining_time < min_burst_time) {
                    min_burst_time = processes[i].remaining_time;
                    min_index = i;
                }
            }
        }
        if (min_index == -1) {
            break;
        }
        processes[min_index].remaining_time--;
        t++;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= t && processes[i].completed == 0) {
                if (i != min_index) {
                    processes[i].waiting_time++;
                }
                if (processes[i].remaining_time == 0) {
                    processes[i].completed = 1;
                    processes[i].turnaround_time = t - processes[i].arrival_time;
                }
            }
        }
    }
    float avg_waiting_time = 0, avg_turnaround_time = 0;
    for (int i = 0; i < n; i++) {
        avg_waiting_time += processes[i].waiting_time;
        avg_turnaround_time += processes[i].turnaround_time;
    }
    avg_waiting_time /= n;
    avg_turnaround_time /= n;
    printf("Average Waiting Time: %.2f ms\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f ms\n", avg_turnaround_time);

    return 0;
}
```

Output:



C:\Users\Dell\Desktop\os 5.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
os 1.cpp priority.cpp Untitled2 os 2.cpp os day 1 document.docx os day1.docx os 3.cpp [*] os 4.cpp os 5.cpp
C:\Users\Dell\Desktop\os 5.exe
Compiler Resources
Abort Compilation
Shorten compiler paths
31°C Partly sunny
Search
10:55 29-04-2023

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

```
Average Waiting Time: 3.25 ms
Average Turnaround Time: 5.50 ms

--------------------------------
Process exited after 6.51 seconds with return value 0
Press any key to continue . . .
```

6. Write a C program to implement the deadlock detection algorithm for a system with 3 processes and 3 resource instances and the resource matrices are given below.

Max Matrix   Allocation Matrix

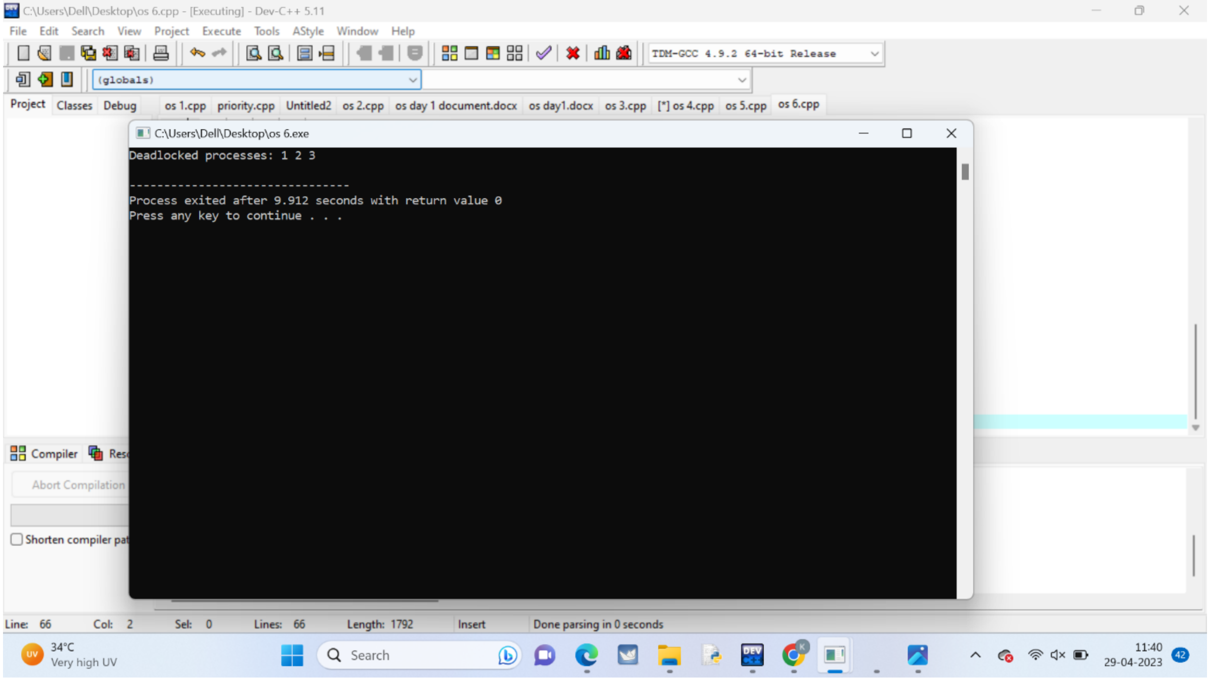
3 6 8	3 3 3
4 3 3	2 0 3
3 4 4	1 2 4

The number of available resources is [1,2,0]. Determine if the system is in a deadlock state and identify the deadlocked processes.

Program:

```
#include <stdio.h>

int main() {
    // Define the Max and Allocation matrices
    int max[3][3] = {{3, 6, 8}, {4, 3, 3}, {3, 4, 4}};
    int allocation[3][3] = {{3, 3, 3}, {2, 0, 3}, {1, 2, 4}};

    // Define the Available vector
    int available[3] = {1, 2, 0};

    // Define the Work and Finish vectors
    int work[3], finish[3] = {0, 0, 0};

    // Initialize the Work vector to the Available vector
    for (int i = 0; i < 3; i++) {
        work[i] = available[i];
    }

    // Initialize the Need matrix to the Max matrix minus the Allocation matrix
    int need[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }

    // Detect deadlock by checking for a safe sequence
    int safe = 0;
    while (safe == 0) {
        safe = 1;
        for (int i = 0; i < 3; i++) {
            if (finish[i] == 0) {
                int j;
                for (j = 0; j < 3; j++) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }
                if (j == 3) {
                    // Process i can complete
                    safe = 0;
                    finish[i] = 1;
                    for (int k = 0; k < 3; k++) {
                        work[k] += allocation[i][k];
                    }
                }
            }
        }
    }

    // Print the results
    int deadlock = 1;
    printf("Deadlocked processes: ");
    for (int i = 0; i < 3; i++) {
        if (finish[i] == 0) {
            printf("%d ", i + 1);
            deadlock = 0;
        }
    }
    if (deadlock == 1) {
        printf("None");
    }
    printf("\n");

    return 0;
}
```

Output:



```
Deadlocked processes: 1 2 3

--------------------------------
Process exited after 9.912 seconds with return value 0
Press any key to continue . . .
```

7. Write a C program to illustrate the page replacement method where the current least recently used element is replaced and determine the number of page faults for the following test case:

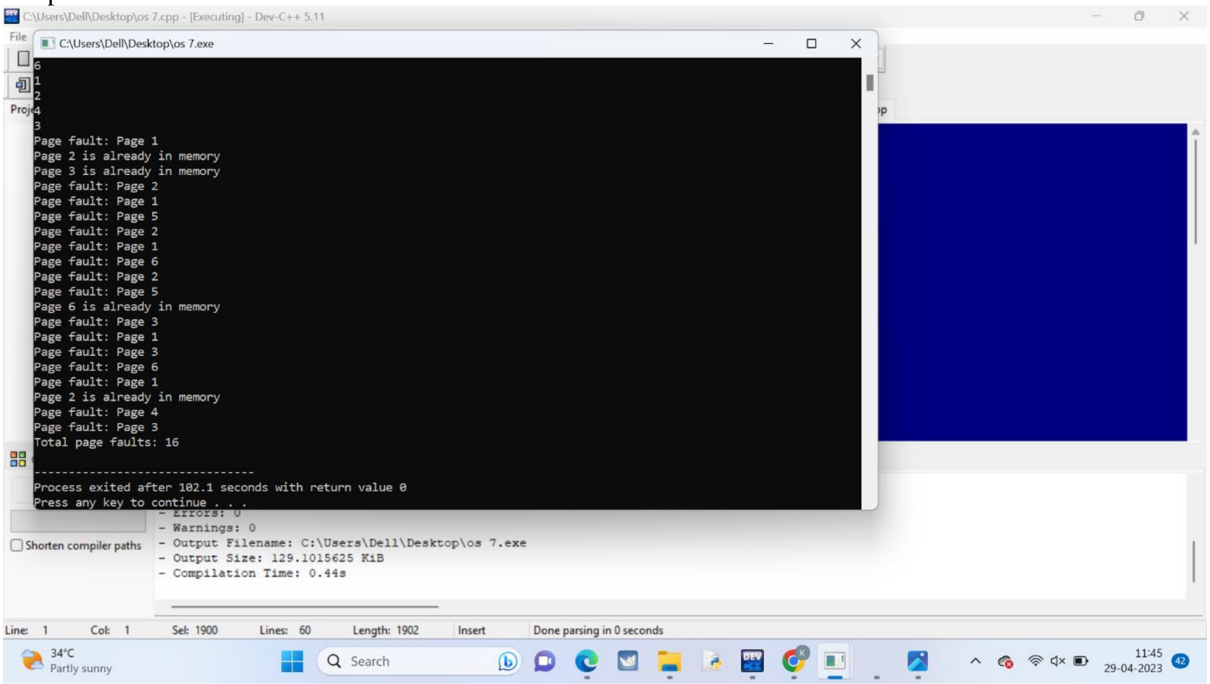
No. of page frames: 3; Page reference sequence 1,2,3,2,1,5,2,1,6,2,5,6,3,1,3,6,1,2,4 and 3.

Program:

```
#include <stdio.h>
#define MAX_PAGES 20
int main() {
    int pageFrames, pageFaults = 0, time = 0;
    int pageReferences[MAX_PAGES], pageTable[MAX_PAGES];
    int i, j, oldestPage, oldestTime;
    printf("Enter the number of page frames: ");
    scanf("%d", &pageFrames);
             printf("Enter the page reference sequence (separated by spaces): ");
    for (i = 0; i < MAX_PAGES; i++) {
        if (scanf("%d", &pageReferences[i]) != 1) {
            break;
        }
    }
    int numPages = i;
    for (i = 0; i < pageFrames; i++) {
        pageTable[i] = -1;
    }
    for (i = 0; i < numPages; i++) {
        int page = pageReferences[i];
        int inPageTable = 0;
        for (j = 0; j < pageFrames; j++) {
            if (pageTable[j] == page) {
                inPageTable = 1;
                break;
            }
        }
                          if (inPageTable) {
            printf("Page %d is already in memory\n", page);
        } else {
            pageFaults++;
            printf("Page fault: Page %d\n", page);
            oldestPage = pageTable[0];
            oldestTime = time;
            for (j = 0; j < pageFrames; j++) {
                if (pageTable[j] == -1) {
                    oldestPage = pageTable[j];
                    break;
                } else if (oldestTime > pageTable[j]) {
                    oldestPage = pageTable[j];
                    oldestTime = pageTable[j];
                }
            }
            for (j = 0; j < pageFrames; j++) {
                if (pageTable[j] == oldestPage) {
                    pageTable[j] = page;
                    break;
                }
            }
        }
        for (j = 0; j < pageFrames; j++) {
            if (pageTable[j] != -1) {
                pageTable[j]++;
            }
        }
```

```
                    time++;
    }
            printf("Total page faults: %d\n", pageFaults);
             return 0;
}
```

Output:


C:\Users\Dell\Desktop\os 7.cpp - [Executing] - Dev-C++ 5.11
File
C:\Users\Dell\Desktop\os 7.exe
Shorten compiler paths
Line: 1
Col: 1
Sel: 1900
Lines: 60
Length: 1902
Insert
Done parsing in 0 seconds
34°C
Partly sunny
Search
11:45
29-04-2023

```
6
1
2
4
3
Page fault: Page 1
Page 2 is already in memory
Page 3 is already in memory
Page fault: Page 2
Page fault: Page 1
Page fault: Page 5
Page fault: Page 2
Page fault: Page 1
Page fault: Page 6
Page fault: Page 2
Page fault: Page 5
Page 6 is already in memory
Page fault: Page 3
Page fault: Page 1
Page fault: Page 3
Page fault: Page 6
Page fault: Page 1
Page 2 is already in memory
Page fault: Page 4
Page fault: Page 3
Total page faults: 16

--------------------------------
Process exited after 102.1 seconds with return value 0
Press any key to continue . . .
```

```
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Dell\Desktop\os 7.exe
- Output Size: 129.1015625 KiB
- Compilation Time: 0.44s
```

8. Write a C program to simulate FCFS disk scheduling algorithm and execute your program and find the average head movement with the following test case:

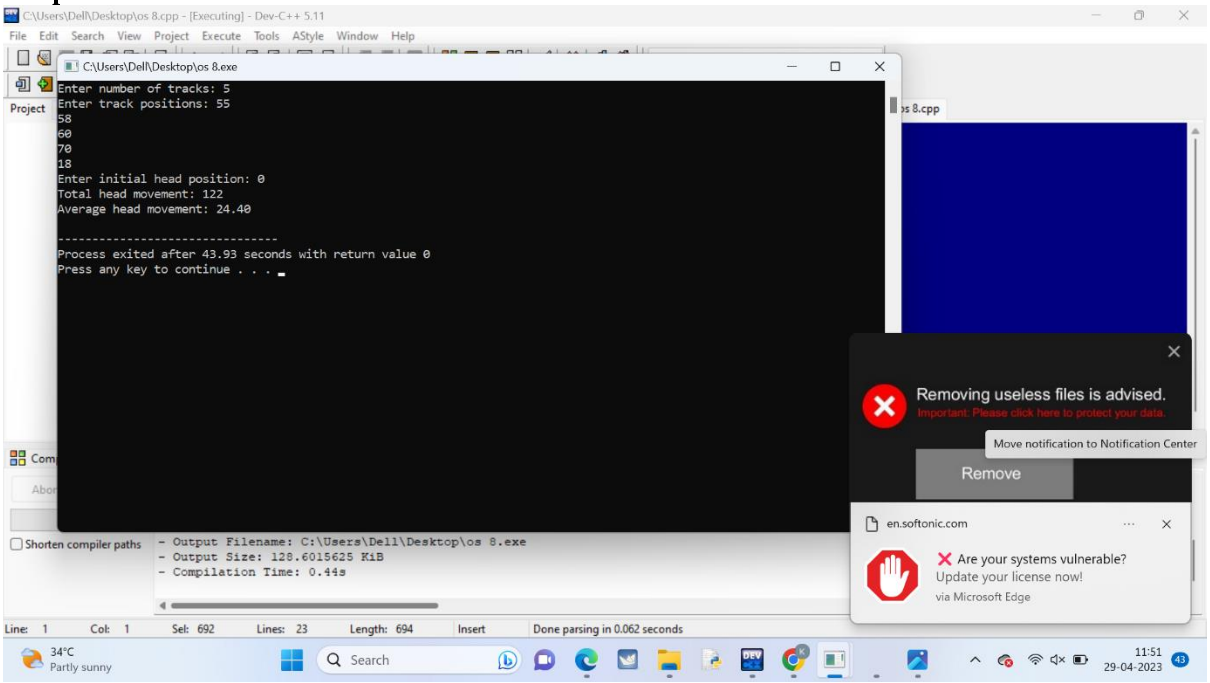
No of tracks 5; Track position:55   58   60   70   18

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_TRACKS 1000
int main() {
    int tracks[MAX_TRACKS];
    int n, head_pos, total_distance;
    printf("Enter number of tracks: ");
    scanf("%d", &n);
            printf("Enter track positions: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &tracks[i]);
    }
            printf("Enter initial head position: ");
    scanf("%d", &head_pos);
    total_distance = 0;
    for (int i = 0; i < n; i++) {
        total_distance += abs(tracks[i] - head_pos);
        head_pos = tracks[i];
    }
    printf("Total head movement: %d\n", total_distance);
    printf("Average head movement: %.2f\n", (float) total_distance / n);
```

```
        return 0;
}
```

Output:



C:\Users\Dell\Desktop\os 8.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
C:\Users\Dell\Desktop\os 8.exe
Project
os 8.cpp
Shorten compiler paths
Line: 1 Col: 1 Sel: 692 Lines: 23 Length: 694 Insert Done parsing in 0.062 seconds
34°C Partly sunny
Search
11:51 29-04-2023

```
Enter number of tracks: 5
Enter track positions: 55
58
60
70
18
Enter initial head position: 0
Total head movement: 122
Average head movement: 24.40

--------------------------------
Process exited after 43.93 seconds with return value 0
Press any key to continue . . .
```

```
- Output Filename: C:\Users\Dell\Desktop\os 8.exe
- Output Size: 128.6015625 KiB
- Compilation Time: 0.44s
```

Removing useless files is advised.
Important: Please click here to protect your data.
Move notification to Notification Center
Remove

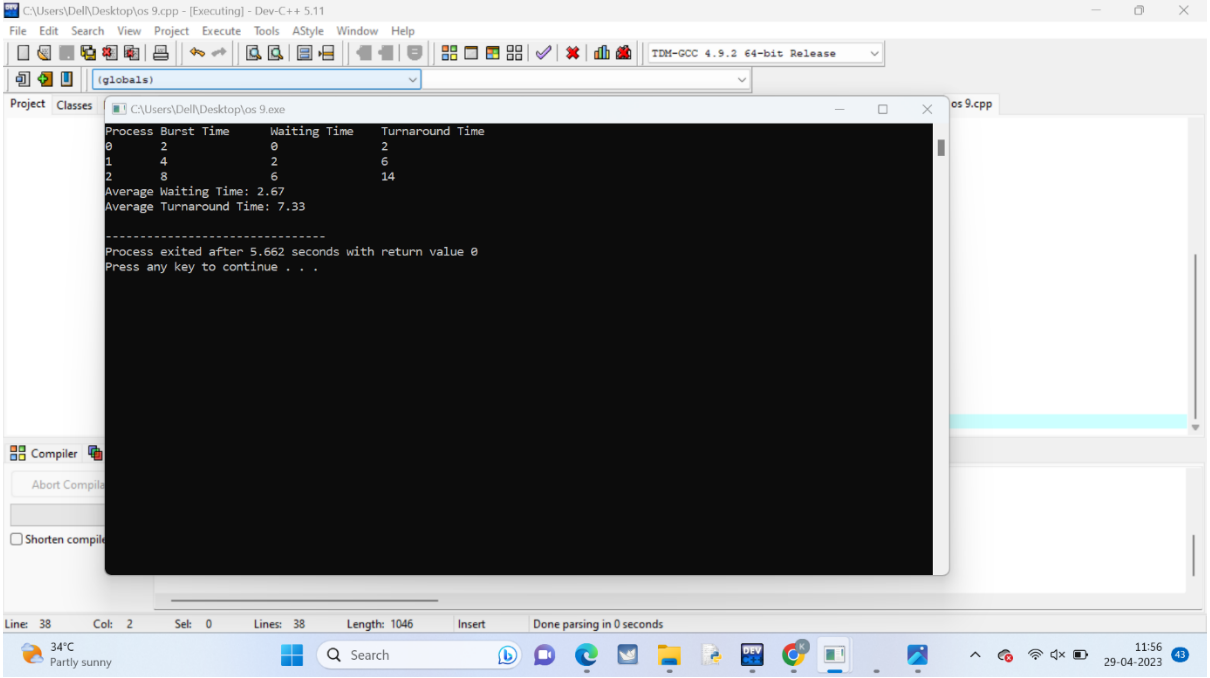
en.softonic.com
Are you systems vulnerable?
Update your license now!
via Microsoft Edge

9. Consider three processes (process id 0, 1, 2 respectively) with compute time bursts 2, 4 and 8-time units. All processes arrive at time zero. Write a program to compute the average waiting time and average turnaround time based on First Come First Serve scheduling

Program:

```
#include<stdio.h>
int main()
{
    int n = 3;
    int burst_time[] = {2, 4, 8};
    int waiting_time[n], turnaround_time[n];
    int i, j;
    waiting_time[0] = 0;
    for(i=1; i<n; i++)
    {
        waiting_time[i] = 0;
        for(j=0; j<i; j++)
        {
            waiting_time[i] += burst_time[j];
        }
    }
    for(i=0; i<n; i++)
    {
        turnaround_time[i] = waiting_time[i] + burst_time[i];
    }
    float avg_waiting_time = 0, avg_turnaround_time = 0;
    for(i=0; i<n; i++)
    {
        avg_waiting_time += waiting_time[i];
        avg_turnaround_time += turnaround_time[i];
    }
    avg_waiting_time /= n;
    avg_turnaround_time /= n;
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i=0; i<n; i++)
    {
        printf("%d\t%d\t\t%d\t\t%d\n", i, burst_time[i], waiting_time[i], turnaround_time[i]);
    }
    printf("Average Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);

    return 0;
}
```

Output:


C:\Users\Dell\Desktop\os 9.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes
C:\Users\Dell\Desktop\os 9.exe
os 9.cpp
Compiler
Abort Compila
Shorten compile
Line: 38 Col: 2 Sel: 0 Lines: 38 Length: 1046 Insert Done parsing in 0 seconds
34°C Partly sunny
Search
11:56 29-04-2023

```
Process Burst Time      Waiting Time    Turnaround Time
0       2               0               2
1       4               2               6
2       8               6               14
Average Waiting Time: 2.67
Average Turnaround Time: 7.33

--------------------------------
Process exited after 5.662 seconds with return value 0
Press any key to continue . . .
```

10. Consider the following process table with number of processes that contains allocation field (for showing the number of resources of type: A, B and C allocated to each process in the table), max field (for showing the maximum number of resources of type: A, B, and C that can be allocated to each process). Write a program to calculate the entries of need matrix using the formula: (Need)i = (Max)i - (Allocation)i

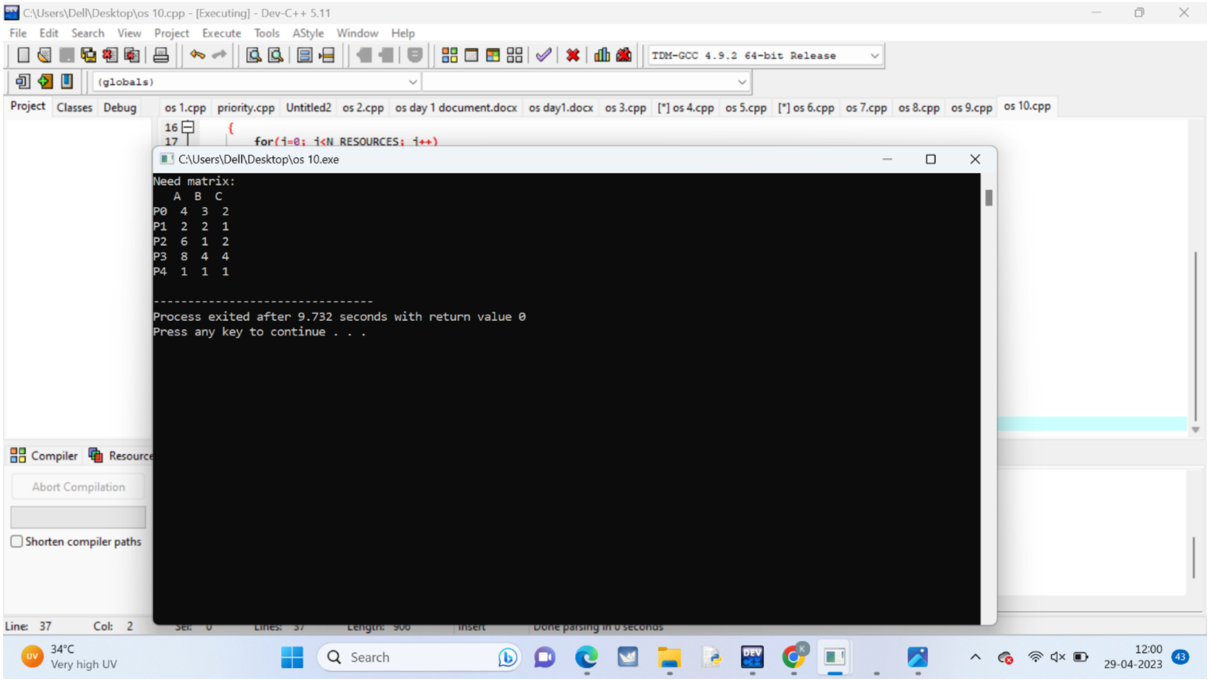
Process	Allocation	Max	Availble
	A B C	A B C	A B C
P0	1 1 2	5 4 4	3 2 1

P1	2 1 2	4 3 3	
P2	3 0 1	9 1 3	
P3	0 2 0	8 6 4	
P4	1 1 2	2 2 3	

Program:

```
#include <stdio.h>

#define N_PROCESSES 5
#define N_RESOURCES 3

int main()
{
   int allocation[N_PROCESSES][N_RESOURCES] = {{1, 1, 2}, {2, 1, 2}, {3, 0, 1}, {0, 2, 0}, {1, 1, 2}};
   int max[N_PROCESSES][N_RESOURCES] = {{5, 4, 4}, {4, 3, 3}, {9, 1, 3}, {8, 6, 4}, {2, 2, 3}};
   int available[N_RESOURCES] = {3, 3, 2};
   int need[N_PROCESSES][N_RESOURCES];
   int i, j;
   for(i=0; i<N_PROCESSES; i++)
   {
      for(j=0; j<N_RESOURCES; j++)
      {
         need[i][j] = max[i][j] - allocation[i][j];
      }
   }
   printf("Need matrix:\n");
   printf("   A  B  C\n");
   for(i=0; i<N_PROCESSES; i++)
   {
      printf("P%d ", i);
      for(j=0; j<N_RESOURCES; j++)
      {
         printf("%2d ", need[i][j]);
      }
      printf("\n");
   }

   return 0;
}
```

Output:



```
Need matrix:
   A  B  C
P0  4  3  2
P1  2  2  1
P2  6  1  2
P3  8  4  4
P4  1  1  1

--------------------------------
Process exited after 9.732 seconds with return value 0
Press any key to continue . . .
```

11. Write a C program to create 4 child processes. In the first child process, print the odd numbers. In the second child process print the even numbers. In the third child process print the multiple of 3. In the fourth child process print the multiples of 5. Print the process id for each of the processes.

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main() {
    int i, pid;
    for(i=1; i<=4; i++) {
        pid = fork();
        if(pid == 0) {
            switch(i) {
                case 1:
                    printf("Child %d (PID=%d): ", i, getpid());
                    for(int j=1; j<=10; j++) {
                        if(j%2 == 1) printf("%d ", j);
                    }
                    printf("\n");
                    break;
                case 2:
                    printf("Child %d (PID=%d): ", i, getpid());
                    for(int j=1; j<=10; j++) {
                        if(j%2 == 0) printf("%d ", j);
                    }
                    printf("\n");
                    break;
                case 3:
                    printf("Child %d (PID=%d): ", i, getpid());
                    for(int j=1; j<=10; j++) {
                        if(j%3 == 0) printf("%d ", j);
                    }
                    printf("\n");
                    break;
                case 4:
                    printf("Child %d (PID=%d): ", i, getpid());
                    for(int j=1; j<=10; j++) {
                        if(j%5 == 0) printf("%d ", j);
                    }
                    printf("\n");
                    break;
            }
            exit(0);
        }
    }
    return 0;
}
```

12. Write a C program to implement the best-fit algorithm and allocate the memory block to each process.

Test Case:

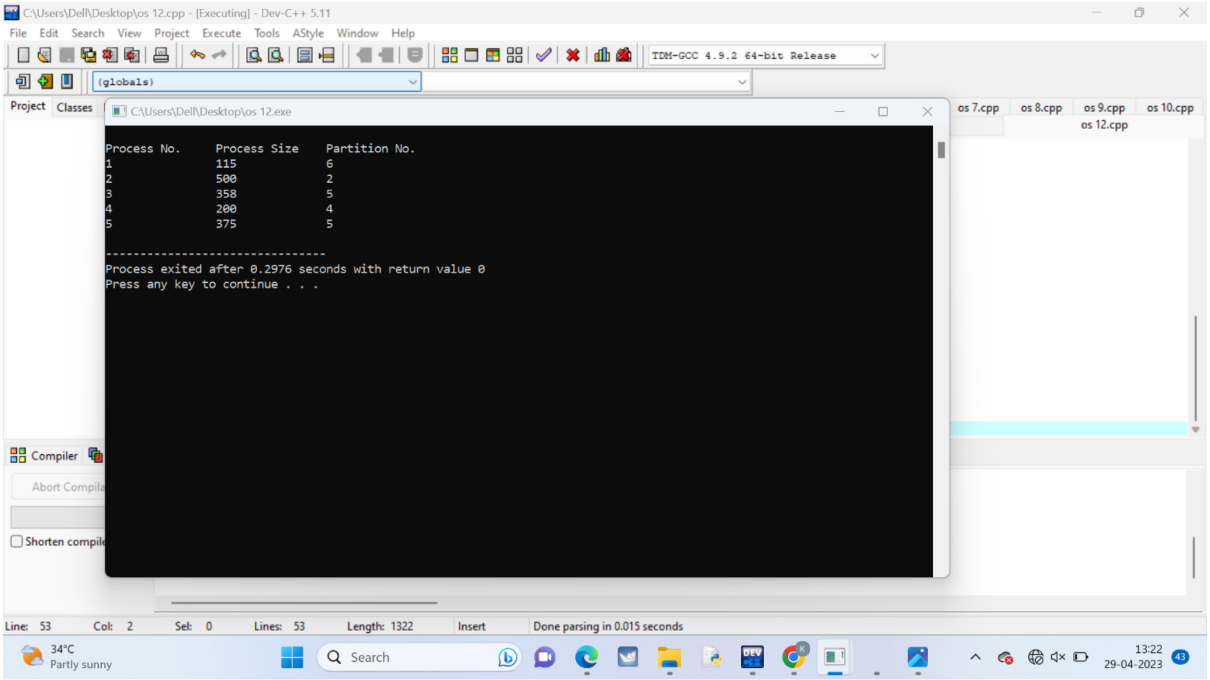
Memory partitions: 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order),

Show the outcome for the test case with the best-fit algorithms to place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)

Program:

```
#include <stdio.h>

#define MAX_PARTITIONS 6
#define MAX_PROCESSES 5

int partitions[MAX_PARTITIONS] = {300, 600, 350, 200, 750, 125};
int processes[MAX_PROCESSES] = {115, 500, 358, 200, 375};
int allocation[MAX_PROCESSES];

void best_fit()
{
    int i, j;
    int best_index;
    for (i = 0; i < MAX_PROCESSES; i++) {
        best_index = -1;
        for (j = 0; j < MAX_PARTITIONS; j++) {
            if (partitions[j] >= processes[i]) {
                if (best_index == -1) {
                    best_index = j;
                } else if (partitions[j] < partitions[best_index]) {
                    best_index = j;
                }
            }
        }
        if (best_index != -1) {
            allocation[i] = best_index;
            partitions[best_index] -= processes[i];
        } else {
            allocation[i] = -1;
        }
    }
}

void print_allocation()
{
    int i;
    printf("\nProcess No.\tProcess Size\tPartition No.\n");
    for (i = 0; i < MAX_PROCESSES; i++) {
        printf("%d\t\t%d\t\t", i+1, processes[i]);
        if (allocation[i] != -1) {
            printf("%d\n", allocation[i]+1);
        } else {
            printf("Not Allocated\n");
        }
    }
}

int main()
{
    best_fit();
    print_allocation();
    return 0;
}
```

Output:


C:\Users\Dell\Desktop\os 12.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes
os 7.cpp os 8.cpp os 9.cpp os 10.cpp
os 12.cpp
Compiler
Abort Compila
Shorten compile
Line: 53 Col: 2 Sel: 0 Lines: 53 Length: 1322 Insert Done parsing in 0.015 seconds
34°C
Partly sunny
Search
13:22
29-04-2023

C:\Users\Dell\Desktop\os 12.exe

```
Process No.     Process Size    Partition No.
1               115             6
2               500             2
3               358             5
4               200             4
5               375             5

--------------------------------
Process exited after 0.2976 seconds with return value 0
Press any key to continue . . .
```