

main.c

main.c

Share

Run

```
1 #include <stdio.h>
2
3 #define MAX 25
4
5 void worstFit(int blockSize[], int m, int processSize[], int n) {
6     int allocation[n];
7
8     // Initially no block is assigned to any process
9     for (int i = 0; i < n; i++)
10         allocation[i] = -1;
11
12     // Pick each process and find the worst fit block
13     for (int i = 0; i < n; i++) {
14         int worstIdx = -1;
15         for (int j = 0; j < m; j++) {
16             if (blockSize[j] >= processSize[i]) {
17                 if (worstIdx == -1 || blockSize[j] >
18                     blockSize[worstIdx])
19                     worstIdx = j;
20             }
21         }
22     }
23 }
```

Output

CI

Enter number of memory blocks: 5

Enter sizes of memory blocks:

Block 1: 100

Block 2: 200

Block 3: 300

Block 4: 400

Block 5: 500

Enter number of processes: 4

Enter sizes of processes:

Process 1: 412

Process 2: 215

Process 3: 342

Process 4: 235

Process No.	Process Size	Block No.
1	412	5
2	215	4
3	342	Not Allocated
4	235	3

main.c

Share

Run

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

```
// If a suitable block was found
if (worstIdx != -1) {
    allocation[i] = worstIdx;
    blockSize[worstIdx] -= processSize[i];
}
}
printf("\nProcess No.\tProcess Size\tBlock 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[MAX], processSize[MAX], m, n;
 printf("Enter number of memory blocks: ");
 scanf("%d", &m);
 printf("Enter sizes of memory blocks:\n");
 for (int i = 0; i < m; i++) {

Output

Enter number of memory blocks: 5
Enter sizes of memory blocks:
Block 1: 100
Block 2: 200
Block 3: 300
Block 4: 400
Block 5: 500
Enter number of processes: 4
Enter sizes of processes:
Process 1: 412
Process 2: 215
Process 3: 342
Process 4: 235

Process No.	Process Size	Block No.
1	412	5
2	215	4
3	342	Not Allocated
4	235	3

main.c

35

int main() {

36

int blockSize[MAX], processSize[MAX], m, n;

37

printf("Enter number of memory blocks: ");

38

scanf("%d", &m);

39

printf("Enter sizes of memory blocks:\n");

40

for (int i = 0; i < m; i++) {

41

printf("Block %d: ", i + 1);

42

scanf("%d", &blockSize[i]);

43

}

44

printf("Enter number of processes: ");

45

scanf("%d", &n);

46

47

printf("Enter sizes of processes:\n");

48

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

49

printf("Process %d: ", i + 1);

50

scanf("%d", &processSize[i]);

51

}

52

worstFit(blockSize, m, processSize, n);

53

return 0;

54

}

55

Share

Run

Output

Clear

Enter number of memory blocks: 5

Enter sizes of memory blocks:

Block 1: 100

Block 2: 200

Block 3: 300

Block 4: 400

Block 5: 500

Enter number of processes: 4

Enter sizes of processes:

Process 1: 412

Process 2: 215

Process 3: 342

Process 4: 235

Process No.	Process Size	Block No.
1	412	5
2	215	4
3	342	Not Allocated
4	235	3

main.c

main.c

2

#define MAX 25

3

void bestFit(int blockSize[], int m, int processSize[], int n) {

4

int allocation[n];

5

// Initialize all allocations to -1 (not allocated)

6

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

7

allocation[i] = -1;

8

// Pick each process and find the best fit block

9

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

10

int bestIdx = -1;

11

for (int j = 0; j < m; j++) {

12

if (blockSize[j] >= processSize[i]) {

13

if (bestIdx == -1 || blockSize[j] <

14

blockSize[bestIdx])

15

bestIdx = j;

16

}

17

// If we found a suitable block

18

if (bestIdx != -1) {

19

allocation[i] = bestIdx;

20

blockSize[bestIdx] -= processSize[i];

Output

Clear

Enter number of memory blocks: 5

Enter sizes of memory blocks:

Block 1: 100

Block 2: 300

Block 3: 250

Block 4: 125

Block 5: 237

Enter number of processes: 4

Enter sizes of processes:

Process 1: 212

Process 2: 242

Process 3: 143

Process 4: 414

Process No.	Process Size	Block No.
1	212	5
2	242	3
3	143	2
4	414	Not Allocated

main.c

Share

Run

22

}

23

// Output allocation result

24

printf("\nProcess No.\tProcess Size\tBlock No.\n");

25

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

26

printf(" %d\t\t%d\t\t", i + 1, processSize[i]);

27

if (allocation[i] != -1)

28

printf("%d\n", allocation[i] + 1);

29

else

30

printf("Not Allocated\n");

31

}

32

}

33

int main() {

34

int blockSize[MAX], processSize[MAX], m, n;

35

printf("Enter number of memory blocks: ");

36

scanf("%d", &m);

37

printf("Enter sizes of memory blocks:\n");

38

for (int i = 0; i < m; i++) {

39

printf("Block %d: ", i + 1);

40

scanf("%d", &blockSize[i]);

41

}

42

printf("Enter number of processes: ");

Output

Clear

Enter number of memory blocks: 5

Enter sizes of memory blocks:

Block 1: 100

Block 2: 300

Block 3: 250

Block 4: 125

Block 5: 237

Enter number of processes: 4

Enter sizes of processes:

Process 1: 212

Process 2: 242

Process 3: 143

Process 4: 414

Process No. Process Size Block No.

1 212 5

2 242 3

3 143 2

4 414 Not Allocated

main.c

Run

Share

```
30     printf("Not allocated\n");
31 }
32 }
33 int main() {
34     int blockSize[MAX], processSize[MAX], m, n;
35     printf("Enter number of memory blocks: ");
36     scanf("%d", &m);
37     printf("Enter sizes of memory blocks:\n");
38     for (int i = 0; i < m; i++) {
39         printf("Block %d: ", i + 1);
40         scanf("%d", &blockSize[i]);
41     }
42     printf("Enter number of processes: ");
43     scanf("%d", &n);
44     printf("Enter sizes of processes:\n");
45     for (int i = 0; i < n; i++) {
46         printf("Process %d: ", i + 1);
47         scanf("%d", &processSize[i]);
48     }
49     bestFit(blockSize, m, processSize, n);
50     return 0;
51 }
```

Output

Clear

Enter number of memory blocks: 5
Enter sizes of memory blocks:
Block 1: 100
Block 2: 300
Block 3: 250
Block 4: 125
Block 5: 237
Enter number of processes: 4
Enter sizes of processes:
Process 1: 212
Process 2: 242
Process 3: 143
Process 4: 414

Process No.	Process Size	Block No.
1	212	5
2	242	3
3	143	2
4	414	Not Allocated

main.c

Run

Share

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

```
printf("Not Allocated\n");
}

int main() {
    int blockSize[MAX], processSize[MAX], m, n;
    printf("Enter number of memory blocks: ");
    scanf("%d", &m);
    printf("Enter sizes of memory blocks:\n");
    for (int i = 0; i < m; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &blockSize[i]);
    }
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter sizes of processes:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d", &processSize[i]);
    }
    firstFit(blockSize, m, processSize, n);
    return 0;
}
```

Output

Clear

Enter number of memory blocks: 3

Enter sizes of memory blocks:

Block 1: 100

Block 2: 200

Block 3: 300

Enter number of processes: 3

Enter sizes of processes:

Process 1: 212

Process 2: 414

Process 3: 312

Process No.	Process Size	Block No.
1	212	3
2	414	Not Allocated
3	312	Not Allocated

=== Code Execution Successful ===

Execute | Source Code | Share | Help

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/stat.h>
6 #define FILENAME "example.txt"
7 int main() {
8     int fd;
9     ssize_t bytes_written, bytes_read;
10    char buffer[128];
11    // Open a file for writing (create it if it doesn't
12    // exist)
13    fd = open(FILENAME, O_CREAT | O_WRONLY | O_TRUNC,
14              S_IRUSR | S_IWUSR);
15    if (fd == -1) {
16        perror("Failed to open file for writing");
17        return 1;
18    }
19    // Write data to the file
20    bytes_written = write(fd, "Hello, UNIX system
21                          calls!\n", 25);
22    if (bytes_written == -1) {
23        perror("Failed to write to file");
24    }
25    close(fd);
```

Read from file: Hello, UNIX system calls!Read from file after
seeking: Hello, UNIX system calls!

Execute | `</>` Source Code | Share | ? Help

```
44  buffer[bytes_read] = '\0';
45  printf("Read from file: %s", buffer);
46  // Use lseek to move the file pointer
47  off_t new_offset = lseek(fd, 0, SEEK_SET);
48  if (new_offset == -1) {
49      perror("Failed to seek in file");
50      close(fd);
51      return 1;
52  }
53  // Read the file again from the beginning
54  bytes_read = read(fd, buffer, sizeof(buffer) - 1);
55  if (bytes_read == -1) {
56      perror("Failed to read from file after seeking");
57      close(fd);
58      return 1;
59  }
60  // Print the file content again
61  buffer[bytes_read] = '\0';
62  printf("Read from file after seeking: %s", buffer);
63  // Close the file
64  close(fd);
65  return 0;
66 }
```

```
Read from file: Hello, UNIX system calls!Read from file after
seeking: Hello, UNIX system calls!
```

Execute | `</>` Source Code | Share | Help

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <sys/stat.h>
5 #include <dirent.h>
6 #define FILENAME "example.txt"
7 int main() {
8     int fd;
9     struct stat file_stat;
10    DIR *dir;
11    struct dirent *entry;
12    fd = open(FILENAME, O_CREAT | O_WRONLY, 0644);
13    write(fd, "Hello, UNIX!", 13);
14    close(fd);
15    stat(FILENAME, &file_stat);
16    printf("File size: %ld bytes\n", file_stat.st_size);
17    fd = open(FILENAME, O_RDWR);
18    lseek(fd, 6, SEEK_SET);
19    write(fd, "World!", 6);
20    close(fd);
21    dir = opendir(".");
22    if (dir) {
23        while ((entry = readdir(dir)) != NULL)
24            printf("Found: %s\n", entry->d_name);
```

```
File size: 25 bytes
Found: .
Found: ..
Found: main
Found: example.txt
Found: main.c
```

Execute | `</>` Source Code | Share | Help

```
6 #define FILENAME "example.txt"
7 int main() {
8     int fd;
9     struct stat file_stat;
10    DIR *dir;
11    struct dirent *entry;
12    fd = open(FILENAME, O_CREAT | O_WRONLY, 0644);
13    write(fd, "Hello, UNIX!", 13);
14    close(fd);
15    stat(FILENAME, &file_stat);
16    printf("File size: %ld bytes\n", file_stat.st_size);
17    fd = open(FILENAME, O_RDWR);
18    lseek(fd, 6, SEEK_SET);
19    write(fd, "World!", 6);
20    close(fd);
21    dir = opendir(".");
22    if (dir) {
23        while ((entry = readdir(dir)) != NULL)
24            printf("Found: %s\n", entry->d_name);
25        closedir(dir);
26    }
27    return 0;
28 }
```

File size: 25 bytes
Found: .
Found: ..
Found: main
Found: example.txt
Found: main.c