

OPERATING SYSTEM-CSA0401

PROGRAM 21 - 30

KRITHIYA G
192421150

21. Worst Fit Memory Allocation in C

The screenshot shows a C program in a code editor. The code defines a function `worstFit` that takes an array of block sizes, the number of blocks, an array of process sizes, and the number of processes. It initializes an allocation array with -1. Then, for each process, it iterates through all blocks to find the largest available block that can accommodate the process. If found, it assigns the block to the process. The output window shows the results of the allocation.

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

Output:

Process No.	Process Size	Block No.
1	212	5
2	417	2
3	112	5
4	426	Not Allocated

=== Code Execution Successful ===

22. Best Fit Memory Allocation in C

The screenshot shows a C program in a code editor. The code defines a function `bestFit` that takes an array of block sizes, the number of blocks, an array of process sizes, and the number of processes. It initializes an allocation array with -1. Then, for each process, it iterates through all blocks to find the smallest available block that can accommodate the process. If found, it assigns the block to the process. The output window shows the results of the allocation.

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

Output:

Process No.	Process Size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

=== Code Execution Successful ===

23. First Fit Memory Allocation

```
main.c  [Icons] [Run] [Share] [Clear]

1 #include <stdio.h>
2
3 void firstFit(int blockSize[], int m, int processSize[], int n)
4 {
5     int allocation[n];
6     for (int i = 0; i < n; i++)
7         allocation[i] = -1;
8
9     for (int i = 0; i < n; i++) {
10        for (int j = 0; j < m; j++) {
11            if (blockSize[j] >= processSize[i]) {
12                allocation[i] = j;
13                blockSize[j] -= processSize[i];
14                break;
15            }
16        }
17    }
18    printf("\nProcess No.\tProcess Size\tBlock No.\n");

```

Output

Process No.	Process Size	Block No.
1	212	2
2	417	5
3	112	2
4	426	Not Allocated

=== Code Execution Successful ===

24. Demonstrate UNIX System Calls for File Management

```
main.c  [Icons] [Run] [Share] [Clear]

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4
5 int main() {
6     int fd;
7     char buffer[100];
8
9     fd = open("sample.txt", O_RDONLY);
10    if (fd < 0) {
11        perror("Failed to open file");
12        return 1;
13    }
14
15    read(fd, buffer, sizeof(buffer));
16    printf("File Contents:\n%s\n", buffer);
17
18    close(fd);
19    return 0;

```

Output

Failed to open file: No such file or directory

=== Code Exited With Errors ===

25. I/O System Calls (fcntl, seek, stat, opendir, readdir)

```
main.c  [Icons] [Run] [Share] [Clear]

1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <unistd.h>
4 #include <dirent.h>
5 #include <sys/stat.h>
6
7 int main() {
8     struct stat fileStat;
9     int fd = open("sample.txt", O_RDWR);
10    if (fd == -1) {
11        perror("open");
12        return 1;
13    }
14
15    fcntl(fd, F_SETFL, O_APPEND); // Set append mode
16    lseek(fd, 0, SEEK_END);        // Seek to end
17    write(fd, "\nAppended line.", 15); // Write to end
18
19    fstat(fd, &fileStat);

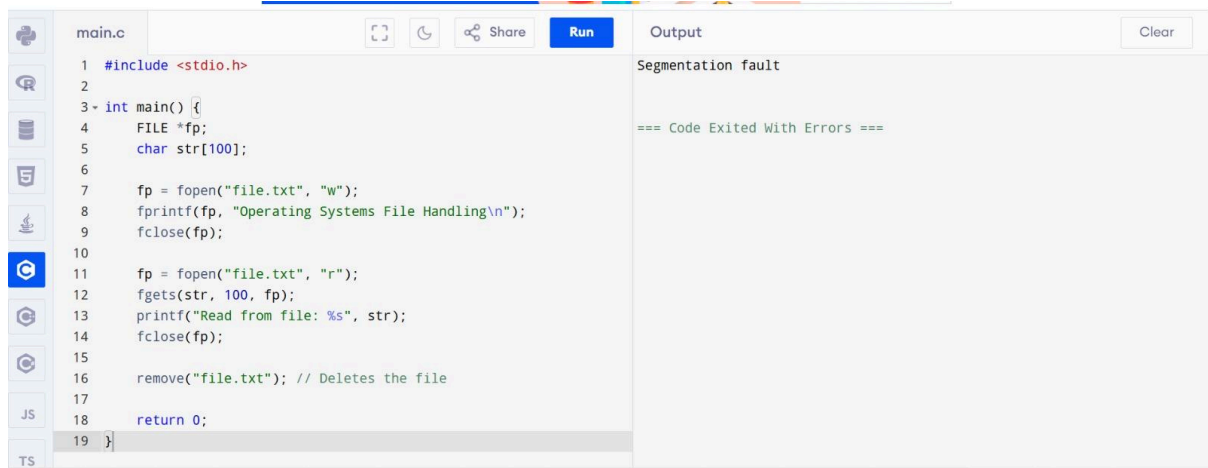
```

Output

open: No such file or directory

=== Code Exited With Errors ===

26. File Management Operations



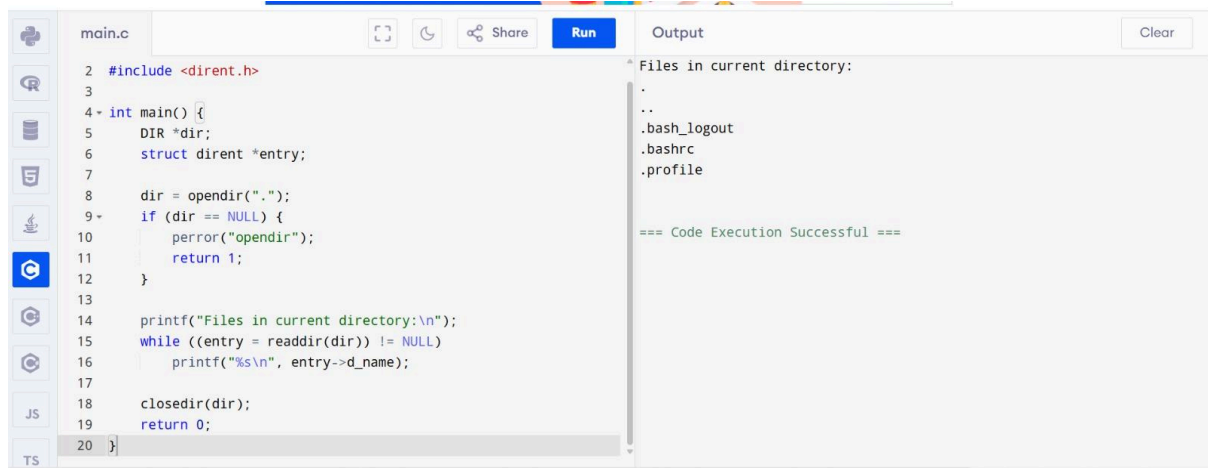
```
main.c
1 #include <stdio.h>
2
3 int main() {
4     FILE *fp;
5     char str[100];
6
7     fp = fopen("file.txt", "w");
8     fprintf(fp, "Operating Systems File Handling\n");
9     fclose(fp);
10
11    fp = fopen("file.txt", "r");
12    fgets(str, 100, fp);
13    printf("Read from file: %s", str);
14    fclose(fp);
15
16    remove("file.txt"); // Deletes the file
17
18    return 0;
19 }
```

Output

Segmentation fault

=== Code Exited With Errors ===

27. Simulate Is UNIX Command



```
main.c
2 #include <dirent.h>
3
4 int main() {
5     DIR *dir;
6     struct dirent *entry;
7
8     dir = opendir(".");
9     if (dir == NULL) {
10        perror("opendir");
11        return 1;
12    }
13
14    printf("Files in current directory:\n");
15    while ((entry = readdir(dir)) != NULL)
16        printf("%s\n", entry->d_name);
17
18    closedir(dir);
19    return 0;
20 }
```

Output

Files in current directory:

.

..

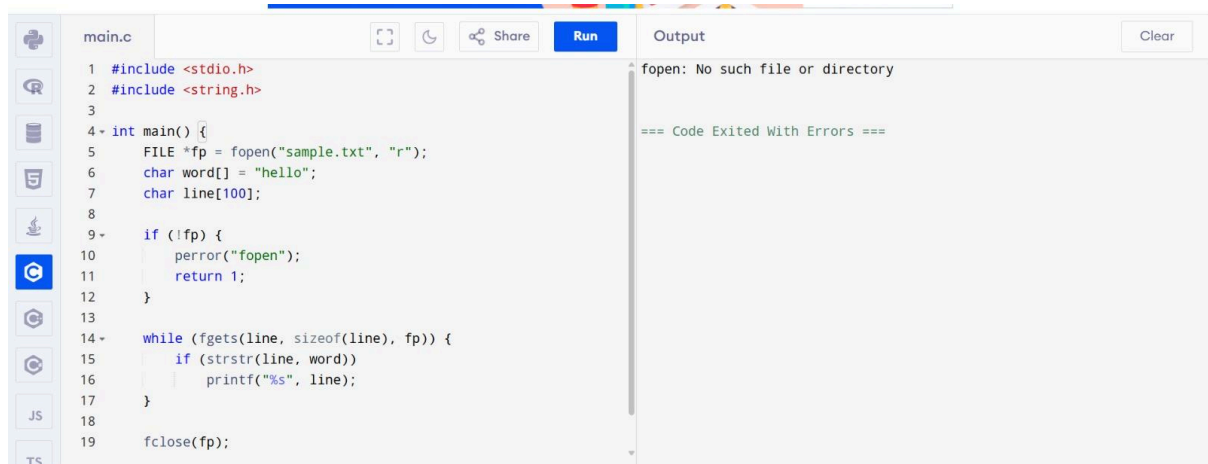
.bash_logout

.bashrc

.profile

=== Code Execution Successful ===

28. Simulate grep UNIX Command



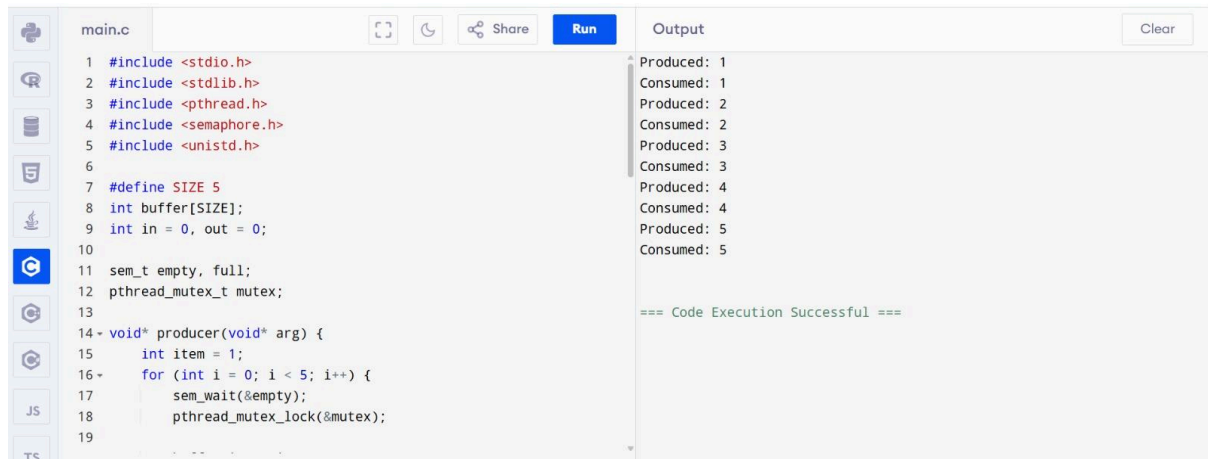
```
main.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     FILE *fp = fopen("sample.txt", "r");
6     char word[] = "hello";
7     char line[100];
8
9     if (!fp) {
10        perror("fopen");
11        return 1;
12    }
13
14    while (fgets(line, sizeof(line), fp)) {
15        if (strstr(line, word))
16            printf("%s", line);
17    }
18
19    fclose(fp);
}
```

Output

fopen: No such file or directory

=== Code Exited With Errors ===

29. Classical Process Synchronization Problem (Producer-Consumer using Semaphore)



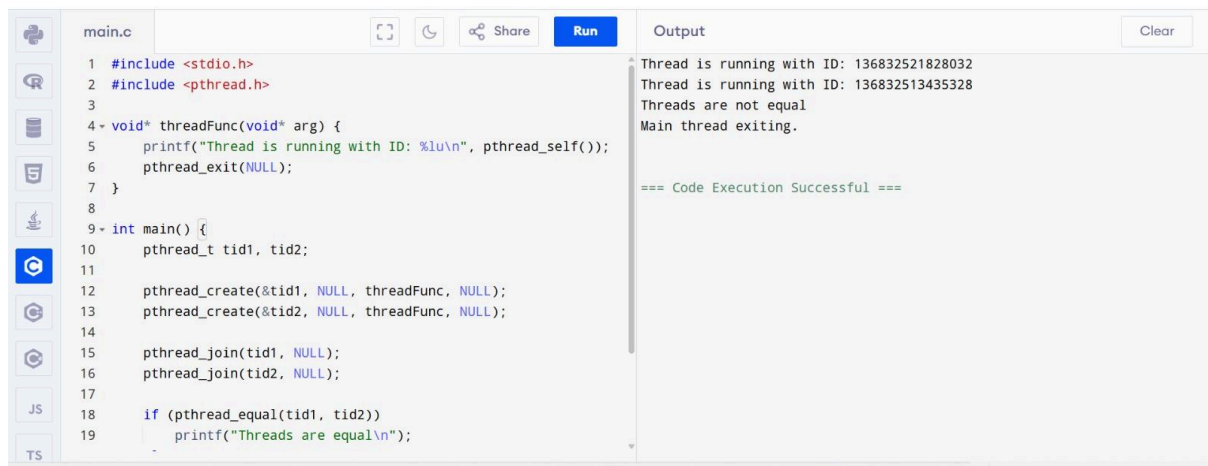
```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <semaphore.h>
5 #include <unistd.h>
6
7 #define SIZE 5
8 int buffer[SIZE];
9 int in = 0, out = 0;
10
11 sem_t empty, full;
12 pthread_mutex_t mutex;
13
14 void* producer(void* arg) {
15     int item = 1;
16     for (int i = 0; i < 5; i++) {
17         sem_wait(&empty);
18         pthread_mutex_lock(&mutex);
19
20         buffer[in] = item;
21         in = (in + 1) % SIZE;
22         item++;
23         sem_post(&full);
24         pthread_mutex_unlock(&mutex);
25         sleep(1);
26     }
27 }
28
29 void* consumer(void* arg) {
30     for (int i = 0; i < 5; i++) {
31         sem_wait(&full);
32         pthread_mutex_lock(&mutex);
33
34         item = buffer[out];
35         out = (out + 1) % SIZE;
36         pthread_mutex_unlock(&mutex);
37         printf("Consumed: %d\n", item);
38         sleep(1);
39     }
40 }
41
42 int main() {
43     pthread_t p, c;
44     pthread_create(&p, NULL, producer, NULL);
45     pthread_create(&c, NULL, consumer, NULL);
46     pthread_join(p, NULL);
47     pthread_join(c, NULL);
48     printf("=== Code Execution Successful ===\n");
49 }
```

Output

Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5

=== Code Execution Successful ===

30. Thread Concepts: Create, Join, Equal, Exit



```
main.c
1 #include <stdio.h>
2 #include <pthread.h>
3
4 void* threadFunc(void* arg) {
5     printf("Thread is running with ID: %lu\n", pthread_self());
6     pthread_exit(NULL);
7 }
8
9 int main() {
10     pthread_t tid1, tid2;
11
12     pthread_create(&tid1, NULL, threadFunc, NULL);
13     pthread_create(&tid2, NULL, threadFunc, NULL);
14
15     pthread_join(tid1, NULL);
16     pthread_join(tid2, NULL);
17
18     if (pthread_equal(tid1, tid2))
19         printf("Threads are equal\n");
20 }
```

Output

Thread is running with ID: 136832521828032
Thread is running with ID: 136832513435328
Threads are not equal
Main thread exiting.

=== Code Execution Successful ===