

main.c

Share

Run

Clear

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 #define MAX 100
5
6 int main() {
7     int n, tq;
8     int bt[MAX], rem_bt[MAX];
9     int wt[MAX], tat[MAX];
10    int t = 0;
11    float avg_wt = 0, avg_tat = 0;
12
13    printf("Enter number of processes: ");
14    scanf("%d", &n);
15
16    printf("Enter burst times of each process:\n");
17    for (int i = 0; i < n; i++) {
18        printf("P%d: ", i + 1);
19        scanf("%d", &bt[i]);
20        rem_bt[i] = bt[i];
21        wt[i] = 0;
22    }
23
24    printf("Enter Time Quantum: ");
25    scanf("%d", &tq);
26
27    printf("\nGantt Chart:\n");
28    while (1) {
29        int done = 1;
30
31        for (int i = 0; i < n; i++) {
32            if (rem_bt[i] > 0) {
33                done = 0;
34
35                if (rem_bt[i] > tq) {
36                    t += tq;
37                    rem_bt[i] -= tq;
38                    printf(" | P%d | %d", i + 1, t);
39                } else {
40                    t += rem_bt[i];
41                    wt[i] = t - bt[i];
42                    rem_bt[i] = 0;
43                    printf(" | P%d | %d", i + 1, t);
44                }
45            }
46        }
47    }
```

Output

Enter number of processes: 5
Enter burst times of each process:
P1: 4
P2: 5
P3: 3
P4: 6
P5: 3
Enter Time Quantum: 2

Gantt Chart:
0 | P1 | 2 | P2 | 4 | P3 | 6 | P4 | 8 | P5 | 10 | P1 | 12 | P2 | 14 | P3 | 15 | P4 | 17 | P5 | 18 | P2 | 19 | P4 | 21

Process Burst Time Waiting Time Turnaround Time
P1 4 8 12
P2 5 14 19
P3 3 12 15
P4 6 15 21
P5 3 15 18

Average Waiting Time = 12.80
Average Turnaround Time = 17.00

=== Code Execution Successful ===

```

main.c
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 #define MAX 100
5
6 int main() {
7     int n, bt[MAX], wt[MAX], tat[MAX], p[MAX];
8     float avg_wt = 0, avg_tat = 0;
9
10    printf("Enter number of processes: ");
11    scanf("%d", &n);
12
13    printf("Enter Burst Time for each process:\n");
14    for (int i = 0; i < n; i++) {
15        printf("P%d: ", i + 1);
16        scanf("%d", &bt[i]);
17        p[i] = i + 1; // store process ID
18    }
19
20
21    for (int i = 0; i < n - 1; i++) {
22        for (int j = i + 1; j < n; j++) {
23            if (bt[j] < bt[i]) {
24                // swap burst time
25                int temp = bt[i];
26                bt[i] = bt[j];
27                bt[j] = temp;
28
29                // swap process ID
30                temp = p[i];
31                p[i] = p[j];
32                p[j] = temp;
33            }
34        }
35    }
36
37    wt[0] = 0;
38
39    for (int i = 1; i < n; i++) {
40        wt[i] = 0;
41        for (int j = 0; j < i; j++)
42            wt[i] += bt[j];
43    }

```

Output

```

Enter number of processes: 5
Enter Burst Time for each process:
P1: 4
P2: 5
P3: 3
P4: 6
P5: 3

Process Burst Time Waiting Time Turnaround Time
P3 3 0 3
P5 3 3 6
P1 4 6 10
P2 5 10 15
P4 6 15 21

Average Waiting Time = 6.80
Average Turnaround Time = 11.00

```

=== Code Execution Successful ===

main.c

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3
4 #define MAX 100
5
6 int main() {
7     int n, bt[MAX], wt[MAX], tat[MAX], rem_bt[MAX];
8     int tq, t = 0; // tq = time quantum, t = current time
9     float avg_wt = 0, avg_tat = 0;
10
11     printf("Enter number of processes: ");
12     scanf("%d", &n);
13
14     printf("Enter burst times of each process:\n");
15     for (int i = 0; i < n; i++) {
16         printf("P%d: ", i + 1);
17         scanf("%d", &bt[i]);
18         rem_bt[i] = bt[i];
19         wt[i] = 0;
20     }
21
22     printf("Enter Time Quantum: ");
23     scanf("%d", &tq);
24
25     printf("\nGantt Chart:\n");
26     while (1) {
27         int done = 1;
28         for (int i = 0; i < n; i++) {
29             if (rem_bt[i] > 0) {
30                 done = 0;
31
32                 printf(" | P%d ", i + 1);
33
34                 if (rem_bt[i] > tq) {
35                     t += tq;
36                     rem_bt[i] -= tq;
37                 } else {
38                     t += rem_bt[i];
39                     wt[i] = t - bt[i];
40                     rem_bt[i] = 0;
41                 }
42             }
43         }
44     }
```



Share

Run

Output

```
Enter number of processes: 5
Enter burst times of each process:
P1: 4
P2: 5
P3: 2
P4: 6
P5: 3
Enter Time Quantum: 2

Gantt Chart:
| P1 | P2 | P3 | P4 | P5 | P1 | P2 | P4 | P5 | P2 | P4 |

Process Burst Time Waiting Time Turnaround Time
P1 4 8 12
P2 5 13 18
P3 2 4 6
P4 6 14 20
P5 3 14 17

Average Waiting Time = 10.60
Average Turnaround Time = 14.60

--- Code Execution Successful ---
```

main.c

Share

Run

1 // Online C compiler to run C program online

2 #include <stdio.h>

3 #include <stdlib.h>

4 #include <string.h>

5 #include <sys/ipc.h>

6 #include <sys/msg.h>

7 #include <sys/types.h>

8 #include <unistd.h>

9 #include <sys/wait.h>

10

11 #define MAX 100

12

13 // Message structure

14 struct msg_buffer {

15 long msg_type; // message type

16 char msg_text[MAX]; // message content

17 };

18

19 int main() {

20 key_t key;

21 int msgid;

22 struct msg_buffer message;

23

24 key = ftok("progfile", 65);

25 msgid = msgget(key, 0666 | IPC_CREAT);

26 if (msgid == -1) {

27 perror("msgget failed");

28 exit(EXIT_FAILURE);

29 }

30

31 pid_t pid = fork();

32

33 if (pid < 0) {

34 perror("fork failed");

35 exit(EXIT_FAILURE);

36 }

37

38 if (pid == 0) {

39 sleep(1);

40 if (msgrcv(msgid, &message, sizeof(message.msg_text), 1, 0) == -1) {

41 perror("msgrcv failed");

42 exit(EXIT_FAILURE);

43 }

Output

Writer: Message sent: Hello from parent via Message Queue!

Reader: Received message: Hello from parent via Message Queue!

--- Code Execution Successful ---

main.c

Share

Run

Output

Clear

```
1 // Online C compiler to run C program online
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <sys/ipc.h>
6 #include <sys/shm.h>
7 #include <sys/types.h>
8 #include <unistd.h>
9 #include <sys/wait.h>
10 #define SHM_SIZE 1024
11
12 int main() {
13     int shmid;
14     char *shared_memory;
15     shmid = shmget((key_t)1234, SHM_SIZE, 0666 | IPC_CREAT);
16     if (shmid == -1) {
17         perror("shmget failed");
18         exit(EXIT_FAILURE);
19     }
20
21     pid_t pid = fork();
22
23     if (pid < 0) {
24         perror("fork failed");
25         exit(EXIT_FAILURE);
26     }
27
28     if (pid == 0) {
29         sleep(1); // wait for parent to write
30         shared_memory = (char*) shmat(shmid, NULL, 0);
31         if (shared_memory == (char*) -1) {
32             perror("shmat failed (child)");
33             exit(EXIT_FAILURE);
34         }
35         printf("Reader: Data read from shared memory: %s\n", shared_memory);
36         shmdt(shared_memory);
37         exit(EXIT_SUCCESS);
38     } else {
39         shared_memory = (char*) shmat(shmid, NULL, 0);
40         if (shared_memory == (char*) -1) {
41             perror("shmat failed (parent)");
42             exit(EXIT_FAILURE);
43         }
44         printf("Writer: Data written into shared memory: Hello from parent using shared memory!\n");
45         strcpy(shared_memory, "Hello from parent using shared memory!");
46         printf("Reader: Data read from shared memory: Hello from parent using shared memory!\n");
47     }
48 }
```

Writer: Data written into shared memory: Hello from parent using shared memory!
Reader: Data read from shared memory: Hello from parent using shared memory!

--- Code Execution Successful ---