

Operating Systems – Lab#4 SOLUTIONS

```

1 //PRODUCER CONSUMER ALGORITHM
2 #include<stdio.h>
3 #include<stdlib.h>
4 int mutex = 1, full = 0, empty = 3, x = 0;
5 int main()
6 {
7     int n;
8     void producer();
9     void consumer();
10    int wait(int);
11    int signal(int);
12    printf("\n1.Producer\n2.Consumer\n3.Exit");
13    while (1)
14    {
15        printf("\nEnter your choice:");
16        scanf_s("%d", &n);
17        switch (n)
18        {
19            case 1: if ((mutex == 1) && (empty != 0))
20                    producer();
21                else
22                    printf("Buffer is full!!");
23                break;
24            case 2: if ((mutex == 1) && (full != 0))
25                    consumer();
26                else
27                    printf("Buffer is empty!!");
28                break;
29            case 3:
30                exit(0);
31                break;
32        }
33    }
34    return 0;
35 }

36
37 int wait(int s)
38 {
39     return (--s);
40 }
41
42 int signal(int s)
43 {
44     return(++s);
45 }
46
47 void producer()
48 {
49     mutex = wait(mutex);
50     full = signal(full);
51     empty = wait(empty);
52     x++;
53     printf("\nProducer produces the item %d", x);
54     mutex = signal(mutex);
55 }
56
57 void consumer()
58 {
59     mutex = wait(mutex);
60     full = wait(full);
61     empty = signal(empty);
62     printf("\nConsumer consumes item %d", x);
63     x--;
64     mutex = signal(mutex);
65 }

```

Operating Systems – Lab#4 SOLUTIONS

```

1 //FCFS (Arrival time and dynamic arrays)
2 #include <stdio.h>
3 #include <stdlib.h>
4 int arr_swapping(int* arr, int firstIndex, int secondIndex)
5 {
6     // Variables
7     int temp;
8
9     // Algorithm
10    temp = arr[firstIndex];
11    arr[firstIndex] = arr[secondIndex];
12    arr[secondIndex] = temp;
13
14    return 1;
15 }
16
17 int main()
18 {
19     // Variables
20     int* p, * at, * bt, * ct, * tat, * wt, i, j, n, ct_temp = 0;
21     float awt = 0, atat = 0;
22
23     // Variables initialization
24     printf_s("Enter the number of processes -- ");
25     scanf_s("%d", &n);
26
27     p = (int*)malloc(n * sizeof(int));
28     at = (int*)malloc(n * sizeof(int));
29     bt = (int*)malloc(n * sizeof(int));
30     ct = (int*)malloc(n * sizeof(int));
31     wt = (int*)malloc(n * sizeof(int));
32     tat = (int*)malloc(n * sizeof(int));
33
34     for (i = 0; i < n; i++)
35     {
36         p[i] = i + 1;
37
38         printf_s("Enter burst time for processes %d -- ", i);
39         scanf_s("%d", &bt[i]);
40
41         printf_s("Enter arrival time for processes %d -- ", i);
42         scanf_s("%d", &at[i]);
43     }
44
45     for (i = 0; i < n; i++)
46     {
47         for (j = i + 1; j < n; j++)
48         {
49             if (at[i] > at[j])
50             {
51                 arr_swapping(p, i, j);
52                 arr_swapping(at, i, j);
53                 arr_swapping(bt, i, j);
54             }
55             else if (at[i] == at[j])
56             {
57                 if (p[i] > p[j])
58                 {
59                     arr_swapping(p, i, j);
60                     arr_swapping(at, i, j);
61                     arr_swapping(bt, i, j);
62                 }
63             }
64         }
65     }
66
67     for (i = 0; i < n; i++)
68     {
69         if (i == 0)
70             ct[i] = at[i] + bt[i];
71         else
72         {
73             if (ct[i - 1] < at[i])
74                 ct[i] = ct[i - 1] + bt[i] + (at[i] - ct[i - 1]);
75             else
76                 ct[i] = ct[i - 1] + bt[i];
77         }
78
79         tat[i] = ct[i] - at[i];
80         wt[i] = tat[i] - bt[i];
81
82         awt += wt[i];
83         atat += tat[i];
84     }
85
86     printf_s("\n\n Average waiting time = %f", awt / n);
87     printf_s("\n\n Average turnaround time = %f", atat / n);
88     printf_s("\n\n \t Process \t Arrival Time \t Burst Time \t Waiting Time \t Turnaround Time \n\n");
89     for (i = 0; i < n; i++)
90     {
91         printf_s("\t P%d \t\t %d \t\t %d \t\t %d \t\t %d \t\t %d \n", p[i], at[i], bt[i], wt[i], tat[i]);
92     }
93 }

```

```

1 //SJF (Arrival time and dynamic arrays)
2 #include "stdio.h"
3 #include "stdlib.h"
4 int arr_swapping(int* arr, int firstIndex, int secondIndex)
5 {
6     // Variables
7     int temp;
8     // Algorithm
9     temp = arr[firstIndex];
10    arr[firstIndex] = arr[secondIndex];
11    arr[secondIndex] = temp;
12    return 1;
13 }
14
15 int main()
16 {
17     // Variables
18     int* p, * at, * bt, * ct, * tat, * wt, i, j, n, bt_min = 0, bt_min_pos;
19     float awt = 0, atat = 0;
20     // Variables initialization
21     printf_s("Enter the number of processes -- ");
22     scanf_s("%d", &n);
23
24     p = (int*)malloc(n * sizeof(int));
25     at = (int*)malloc(n * sizeof(int));
26     bt = (int*)malloc(n * sizeof(int));
27     ct = (int*)malloc(n * sizeof(int));
28     wt = (int*)malloc(n * sizeof(int));
29     tat = (int*)malloc(n * sizeof(int));
30
31     for (i = 0; i < n; i++)
32     {
33         p[i] = i;
34         printf_s("Enter burst time for processes %d -- ", i);
35         scanf_s("%d", &bt[i]);
36
37         printf_s("Enter arrival time for processes %d -- ", i);
38         scanf_s("%d", &at[i]);
39     }
40
41     for (i = 0; i < n; i++)
42     {
43         for (j = i + 1; j < n; j++)
44         {
45             if (at[i] > at[j])
46             {
47                 arr_swapping(p, i, j);
48                 arr_swapping(at, i, j);
49                 arr_swapping(bt, i, j);
50             }
51             else if (at[i] == at[j])
52             {
53                 if (bt[i] > bt[j])
54                 {
55                     arr_swapping(p, i, j);
56                     arr_swapping(at, i, j);
57                     arr_swapping(bt, i, j);
58                 }
59             }
60         }
61     }

```

Operating Systems – Lab#4 SOLUTIONS

```

    }
    for (i = 0; i < n; i++)
    {
        if (i == 0)
            ct[i] = at[i] + bt[i];
        else
        {
            bt_min = bt[0];
            for (j = i; j < n; j++)
            {
                if (at[j] <= ct[i - 1])
                {
                    if (bt[j] < bt_min)
                    {
                        bt_min = bt[j];
                        bt_min_pos = j;
                    }
                }
            }
            arr_swapping(p, i, bt_min_pos);
            arr_swapping(at, i, bt_min_pos);
            arr_swapping(bt, i, bt_min_pos);
            if (ct[i - 1] < at[i])
                ct[i] = ct[i - 1] + bt[i] + (at[i] - ct[i - 1]);
            else
                ct[i] = ct[i - 1] + bt[i];

        }
        tat[i] = ct[i] - at[i];
        wt[i] = tat[i] - bt[i];
        awt += wt[i];
        atat += tat[i];
    }

91     }
92     printf_s("\n\n Average waiting time = %f", awt / n);
93     printf_s("\n\n Average turnaround time = %f", atat / n);
94     printf("\n\n \t Process \t Arrival Time \t Burst Time \t Waiting Time \t Turnaround Time \n");
95     for (i = 0; i < n; i++)
96         printf("\t P%d \t\t %d \t\t %d \t\t %d \t\t %d \t\t %d \n", p[i], at[i], bt[i], wt[i], tat[i]);
97 }

```