**EXERCISE – 5**

**Name – Bhavy Kharbanda**

**SAP ID – 500082531**

**Roll Number – R214220342**

**Batch – B11**

**Date of Performance – 26/04/2021**

**Date of Submission – 03/05/2021**

OPERATING SYSTEM LAB

EXERCISE – 5.1

BANKER’S ALGORITHM

**AIM: To write a C program to simulate the Banker’s Algorithm.**

**Code:-**

**#include <stdio.h>**

**int main()**

**{**

**// P0, P1, P2, P3, P4 are the Process names here**

**int r, p, n, m, i, j, k, alloc[10][10], max[10][10], avail[10], need[10][10];**

**printf("Enter the no of processes : ");**

**scanf("%d", &p);**

**printf("\n\nEnter the no of resources : ");**

**scanf("%d", &r);**

**printf("\n\nEnter the Max Matrix for each process : ");**

**for (i = 0; i < p; i++)**

**{**

**printf("\nFor process %d : ", i + 1);**

**for (j = 0; j < r; j++)**

**scanf("%d", &max[i][j]);**

**}**

**printf("\n\nEnter the allocation for each process : ");**

**for (i = 0; i < p; i++)**

**{**

**printf("\nFor process %d : ", i + 1);**

**for (j = 0; j < r; j++)**

**scanf("%d", &alloc[i][j]);**

**}**

**printf("\n\nEnter the Current available Resources: ");**

**for (i = 0; i < r; i++)**

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

**for (i = 0; i < p; i++)**

**for (j = 0; j < r; j++)**

**need[i][j] = max[i][j] - alloc[i][j];**

**printf("\nMax matrix:\tallocation matrix:\n");**

**for (i = 0; i < p; i++)**

**{**

**for (j = 0; j < r; j++)**

**printf("%d ", max[i][j]);**

**printf("\t\t");**

**for (j = 0; j < r; j++)**

**printf("%d ", alloc[i][j]);**

**printf("\n");**

**}**

**int f[n], ans[n], ind = 0;**

**for (k = 0; k < n; k++) {**

**f[k] = 0;**

**}**

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

**for (j = 0; j < m; j++)**

**need[i][j] = max[i][j] - alloc[i][j];**

**}**

**int y = 0;**

**for (k = 0; k < 5; k++) {**

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

**if (f[i] == 0) {**

**int flag = 0;**

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

**if (need[i][j] > avail[j]){**

**flag = 1;**

**break;**

**}**

**}**

**if (flag == 0) {**

**ans[ind++] = i;**

**for (y = 0; y < m; y++)**

**avail[y] += alloc[i][y];**

**f[i] = 1;**

**}**

**}**

**}**

**}**

**printf("Following is the SAFE Sequence\n");**

**for (i = 0; i < n - 1; i++)**

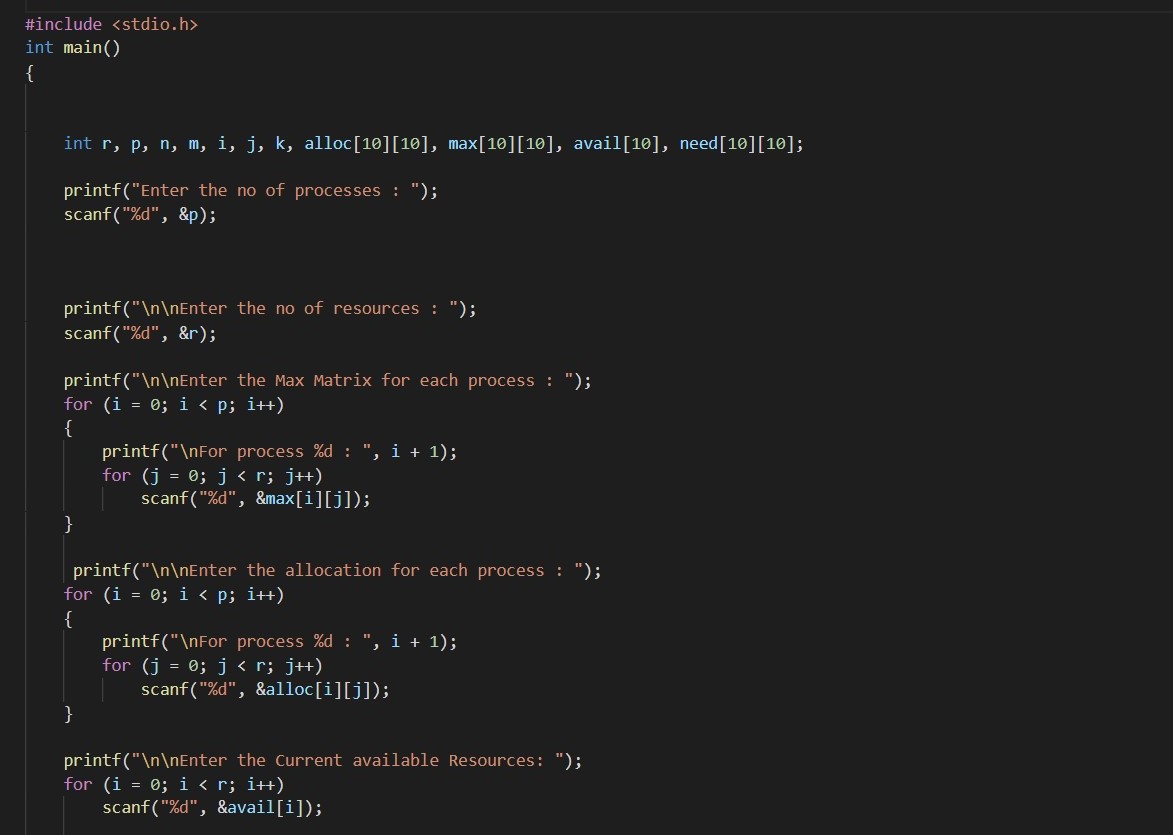
**printf(" P%d ->", ans[i]);**

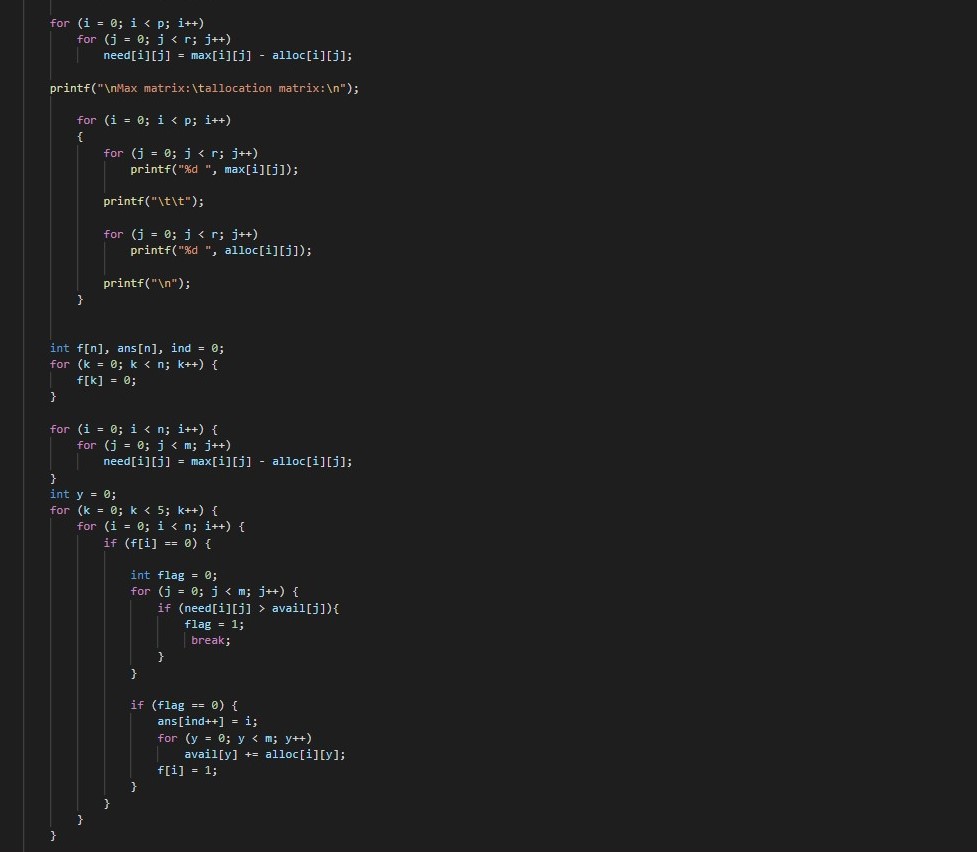
**printf(" P%d", ans[n - 1]);**

**return (0);**

**}**

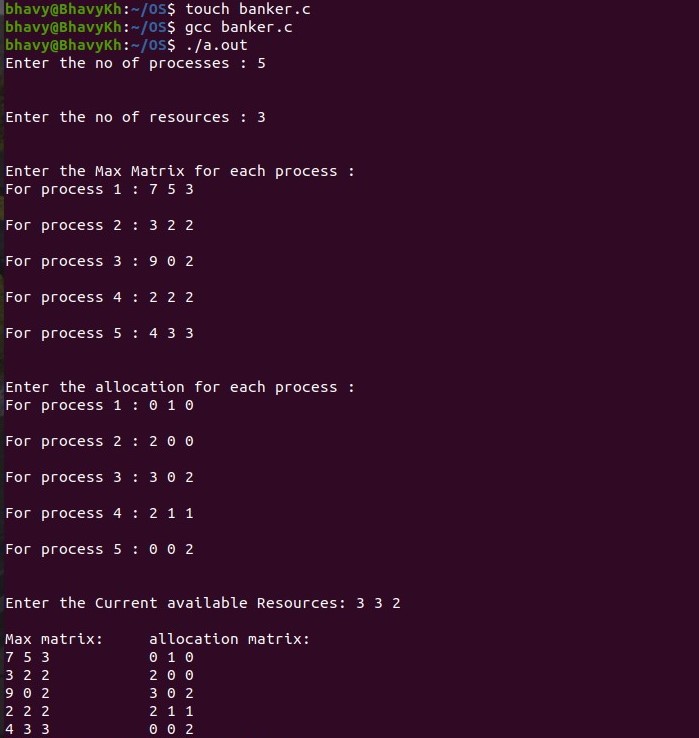
**Source Code:-**

****

****

****

**Output:-**

****

****

OPERATING SYSTEM LAB

EXERCISE – 5.2

SEMAPHORES

**AIM: To write a C program to implement the Producer & consumer Problem (Semaphore)**

**THEORY:**

**The producer-consumer problem (Also called the bounded-buffer problem.) illustrates the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process (producer) produces information and puts it in the buffer, while the other process (consumer) consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently. Herein lies the problem. What happens if the producer tries to put an item into a full buffer? What happens if the consumer tries to take an item from an empty buffer?**

**In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty.**

**Algorithm:-**

**Step 1- The Semaphore mutex, full &empty are initialized.**

**Step 2- In the Case of Producer process:**

1. **Produce an item into a temporary variable.**
2. **If there is empty space in the buffer check the mutex value for entering into the critical section.**
3. **If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.**

**Step 3- In the case of consumer process:**

1. **It should wait if the buffer is empty.**
2. **If there is any item in the buffer check for the mutex value, if the mutex == 0, remove item from buffer.**
3. **Signal the mutex value and reduce the empty value by 1.**
4. **Consume the item.**

**Step 4- Print the result.**

**Code:-**

**#include<stdio.h>**

**#include<stdlib.h>**

**int mutex=1,full=0,empty=3,x=0;**

**int main()**

**{**

**int n;**

**void producer();**

**void consumer();**

**int wait(int);**

**int signal(int);**

**printf("\n Menu");**

**printf("\n1.Producer\n2.Consumer\n3.Exit");**

**while(1)**

**{**

**printf("\nEnter your choice:");**

**scanf("%d",&n);**

**switch(n)**

**{**

**case 1: if((mutex==1)&&(empty!=0))**

**producer();**

**else**

**printf("Buffer is full!!");**

**break;**

**case 2: if((mutex==1)&&(full!=0))**

**consumer();**

**else**

**printf("Buffer is empty!!");**

**break;**

**case 3:**

**exit(0);**

**break;**

**}**

**}**

**return 0;**

**}**

**int wait(int s)**

**{**

**return (--s);**

**}**

**int signal(int s)**

**{**

**return(++s);**

**}**

**void producer()**

**{**

**mutex=wait(mutex);**

**full=signal(full);**

**empty=wait(empty);**

**x++;**

**printf("\nProducer produces the item: %d",x);**

**mutex=signal(mutex);**

**}**

**void consumer()**

**{**

**mutex=wait(mutex);**

**full=wait(full);**

**empty=signal(empty);**

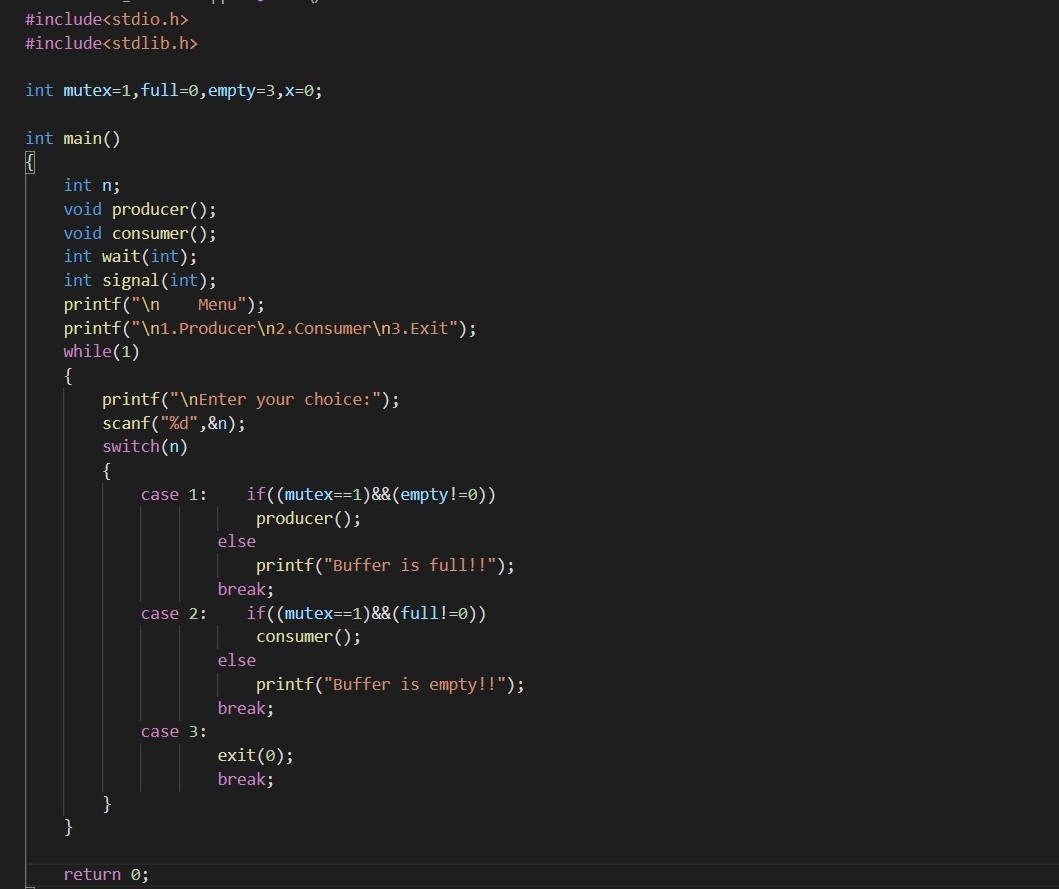
**printf("\nConsumer consumes item: %d",x);**

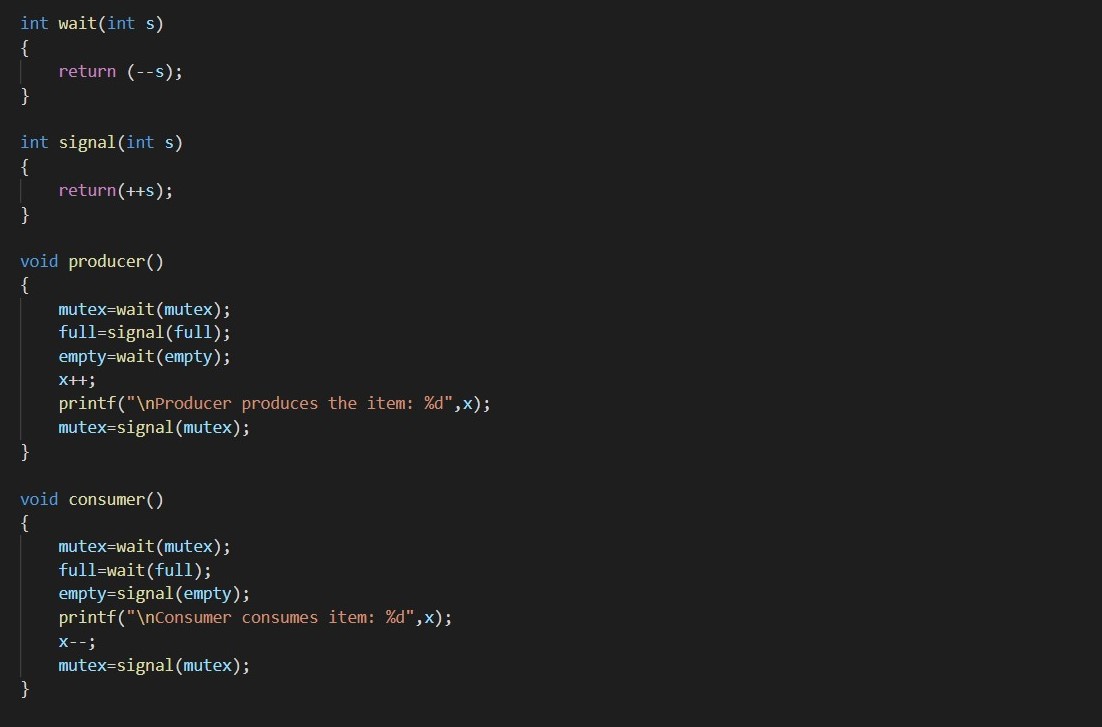
**x--;**

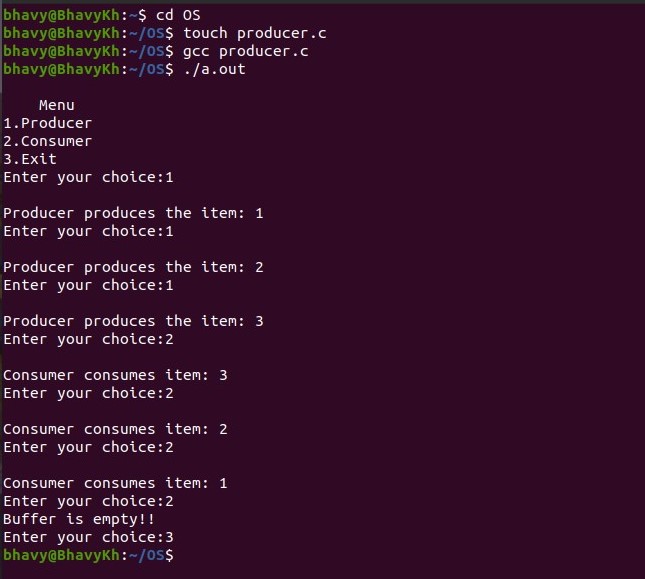
**mutex=signal(mutex);**

**}**

**Screenshot/Output:-**

****

****

****

**\*\*\*\*\*\*\*End of Exercise – 5.2\*\*\*\*\*\*\***

**THANK YOU**