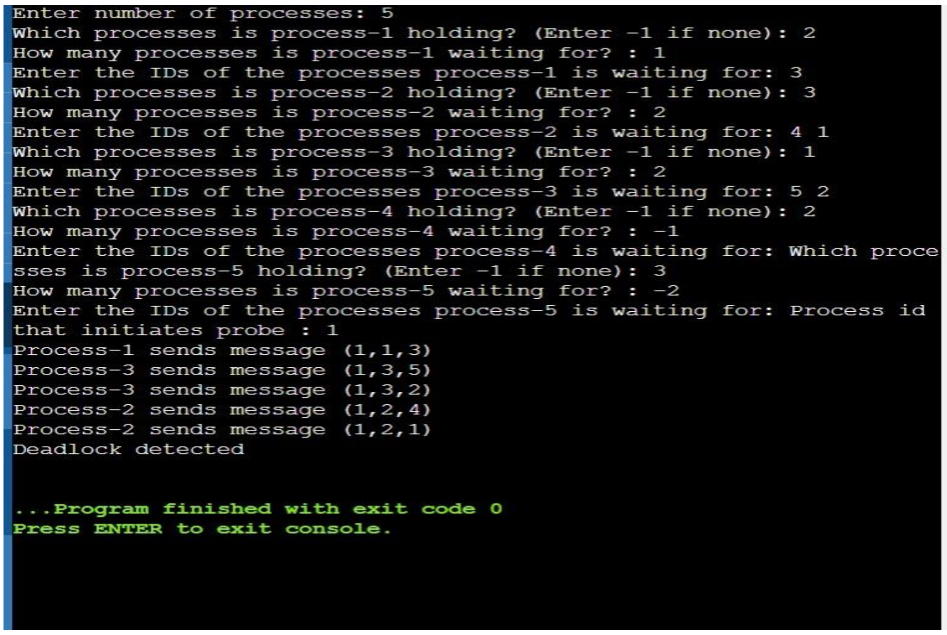
Ex.No: 03(b)	Implementation of Deadlock Detection Algorithm
Date: 05-02-2024	

Probe based Algorithm in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX_WAITING 5 // Maximum number of processes a process can wait for
struct Process
{
int id;
int holding; // Process ID of the process holding this process
int waiting[MAX_WAITING]; // Array to store Process IDs of processes this process is waiting for
int num_waiting; // Number of processes this process is waiting for
};
void probe(struct Process pList[], struct Process cur, int start, int size)
{
bool foundDeadlock = false;
// Check all waiting processes
for (int j = 0; j < cur.num_waiting; j++)
{
for (int i = 0; i < size; i++)
{
if (cur.waiting[j] == pList[i].id)
{
printf("Process-%d sends message (%d,%d,%d)\n", cur.id, start, cur.id, pList[i].id);
if (pList[i].id == start)
{
printf("Deadlock detected\n");
foundDeadlock = true;
return;
}
else
{
probe(pList, pList[i], start, size);
}
} }}}
int main()
{
int process, initiator;
printf("Enter number of processes: ");
scanf("%d", &process);
struct Process pList = (struct Process)malloc(process * sizeof(struct Process));
for (int i = 0; i < process; i++)
{
printf("Which processes is process-%d holding? (Enter -1 if none): ", i+1);
scanf("%d", &pList[i].holding);
printf("How many processes is process-%d waiting for? : ", i+1);
scanf("%d", &pList[i].num_waiting);
printf("Enter the IDs of the processes process-%d is waiting for: ", i+1);
for (int j = 0; j < pList[i].num_waiting; j++)
{
scanf("%d", &pList[i].waiting[j]);
}
pList[i].id = i+1;
}
printf("Process id that initiates probe : ");
scanf("%d", &initiator);
struct Process cur;
for (int i = 0; i < process; i++)
{
if (pList[i].id == initiator)
{
cur = pList[i];
break;}
```

```
}
probe(pList, cur, cur.id, process);
free(pList);
return 0;
}
```

Output:



```
Enter number of processes: 5
Which processes is process-1 holding? (Enter -1 if none): 2
How many processes is process-1 waiting for? : 1
Enter the IDs of the processes process-1 is waiting for: 3
Which processes is process-2 holding? (Enter -1 if none): 3
How many processes is process-2 waiting for? : 2
Enter the IDs of the processes process-2 is waiting for: 4 1
Which processes is process-3 holding? (Enter -1 if none): 1
How many processes is process-3 waiting for? : 2
Enter the IDs of the processes process-3 is waiting for: 5 2
Which processes is process-4 holding? (Enter -1 if none): 2
How many processes is process-4 waiting for? : -1
Enter the IDs of the processes process-4 is waiting for: Which proce
sses is process-5 holding? (Enter -1 if none): 3
How many processes is process-5 waiting for? : -2
Enter the IDs of the processes process-5 is waiting for: Process id
that initiates probe : 1
Process-1 sends message (1,1,3)
Process-3 sends message (1,3,5)
Process-3 sends message (1,3,2)
Process-2 sends message (1,2,4)
Process-2 sends message (1,2,1)
Deadlock detected


...Program finished with exit code 0
Press ENTER to exit console.
```

WFG based Algorithm in C:

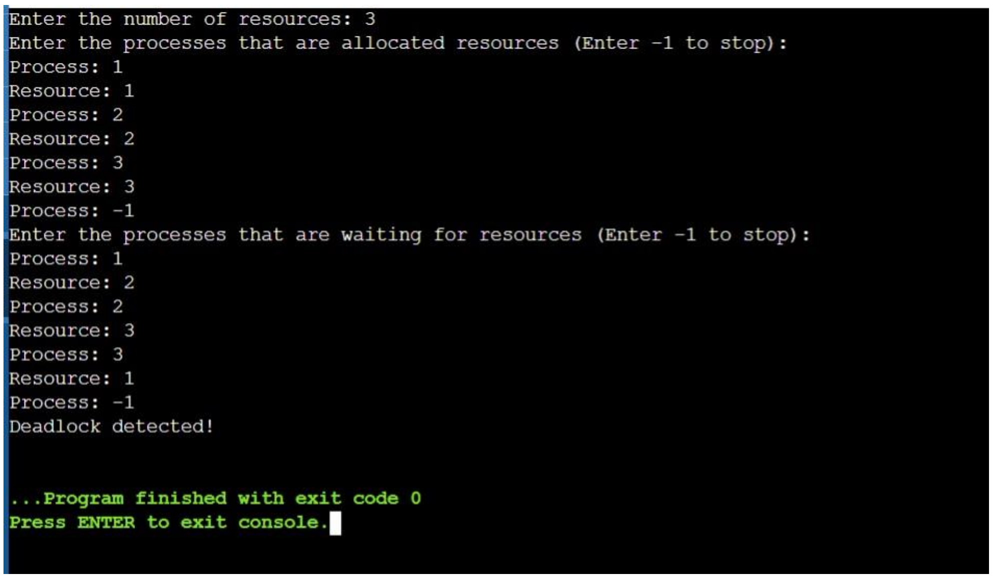
```
#include <stdio.h>
#include <stdbool.h>
#define MAX_NODES 100
// Function to detect deadlock using the WFG based algorithm
bool detectDeadlock(int graph[][MAX_NODES], int num_processes, int num_resources) {
bool visited[num_resources];
bool in_stack[num_resources];
int i,j;
// Initialize visited and in_stack arrays
for ( i = 0; i < num_resources; ++i) {
visited[i] = false;
in_stack[i] = false;
}
// Perform DFS traversal to detect cycles
for ( i = 0; i < num_resources; ++i) {
if (!visited[i]) {
int stack[MAX_NODES];
int top = -1;
visited[i] = true;
in_stack[i] = true;
stack[++top] = i;
while (top != -1) {
int node = stack[top];
bool found = false;
for ( j = 0; j < num_processes; ++j) {
if (graph[j][node]) {
if (!visited[j]) {
visited[j] = true;
in_stack[j] = true;
stack[++top] = j;
found = true;
break;
} else if (in_stack[j]) {
return true; // Cycle detected
}
}}
```

```c
if (!found) {
in_stack[node] = false;
--top;
}
}
}
}
return false;
}
int main() {
int num_resources = 0;
printf("Enter the number of resources: ");
scanf("%d", &num_resources);
int wait_for_graph[MAX_NODES][MAX_NODES] = {0};
printf("Enter the processes that are allocated resources (Enter -1 to stop):\n");
int process, resource;
while (true) {
printf("Process: ");
scanf("%d", &process);


if (process == -1) {
break;
}
printf("Resource: ");
scanf("%d", &resource);
wait_for_graph[process][resource] = 1;
}
printf("Enter the processes that are waiting for resources (Enter -1 to stop):\n");
while (true) {
printf("Process: ");
scanf("%d", &process);
if (process == -1) {
break;
}
printf("Resource: ");
scanf("%d", &resource);
wait_for_graph[process][resource] = 1;
```

```
}
int num_processes = MAX_NODES; // Assuming the maximum number of processes
if (detectDeadlock(wait_for_graph, num_processes, num_resources)) {
printf("Deadlock detected!\n");
} else {
printf("No deadlock detected.\n");
}
return 0;
}
```

Output:



```
Enter the number of resources: 3
Enter the processes that are allocated resources (Enter -1 to stop):
Process: 1
Resource: 1
Process: 2
Resource: 2
Process: 3
Resource: 3
Process: -1
Enter the processes that are waiting for resources (Enter -1 to stop):
Process: 1
Resource: 2
Process: 2
Resource: 3
Process: 3
Resource: 1
Process: -1
Deadlock detected!


...Program finished with exit code 0
Press ENTER to exit console.
```