

List of Experiments (Programming Languages: C or C++)

3. a. Implement process synchronization using semaphores.
- b. Simulation of Banker's algorithm to check whether the given system is in safe state or not. Also check whether additional resource requested can be granted immediately.

3 a)

main.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <semaphore.h>
4
5  #define NUM_THREADS 2
6
7  sem_t semaphore;
8
9  void* threadFunction(void* threadId) {
10     int tid = *((int*)threadId);
11
12     printf("Thread %d is waiting...\n", tid);
13     sem_wait(&semaphore);
14
15     printf("Thread %d is inside the critical section.\n", tid);
16     // Perform critical section operations
17
18     printf("Thread %d is exiting the critical section.\n", tid);
19     sem_post(&semaphore);
20
21     pthread_exit(NULL);
22 }
23
24 int main() {
25     pthread_t threads[NUM_THREADS];
26     int threadIds[NUM_THREADS];
27
```

```

28 // Initialize semaphore
29 sem_init(&semaphore, 0, 1);
30
31 // Create threads
32 for (int i = 0; i < NUM_THREADS; i++) {
33     threadIds[i] = i;
34     pthread_create(&threads[i], NULL, threadFunction, (void*)&threadIds[i]);
35 }
36
37 // Wait for threads to complete
38 for (int i = 0; i < NUM_THREADS; i++) {
39     pthread_join(threads[i], NULL);
40 }
41
42 // Destroy semaphore
43 sem_destroy(&semaphore);
44
45 return 0;
46 }
47

```

Output

Clear

```

/tmp/AJMd0MdATH.o
Thread 0 is waiting...
Thread 0 is inside the critical section.
Thread 0 is exiting the critical section.
Thread 1 is waiting...
Thread 1 is inside the critical section.
Thread 1 is exiting the critical section.

```

b)

PURVA SHARMA 21BCE0169

main.c

```

1 #include <stdio.h>
2
3 #include <stdbool.h>
4
5
6
7 #define MAX_PROCESSES 10
8
9 #define MAX_RESOURCES 10
10
11
12
13 int available[MAX_RESOURCES];
14 int maximum[MAX_PROCESSES][MAX_RESOURCES];
15 int allocation[MAX_PROCESSES][MAX_RESOURCES];
16
17 int need[MAX_PROCESSES][MAX_RESOURCES];
18
19 bool finished[MAX_PROCESSES];
20
21
22
23
24
25 int num_processes;
26
27 int num_resources;
28
29
30
31 bool is_safe_state() {
32
33     // ... Implementation of is_safe_state function ...
34
35 }
36

```

★ main.c

```

36
37 bool is_request_valid(int process, int* request) {
38
39     // ... Implementation of is_request_valid function ...
40
41 }
42
43 bool is_safe_state_with_request(int process, int* request) {
44
45     // ... Implementation of is_safe_state_with_request function ...
46
47 }
48
49
50
51 int main() {
52
53     // Input the number of processes and resources
54
55     printf("Enter the number of processes: ");
56
57     scanf("%d", &num_processes);
58
59
60
61     printf("Enter the number of resources: ");
62
63     scanf("%d", &num_resources);
64
65
66     // Input the available resources
67
68     printf("Enter the available resources: ");
69
70     for (int i = 0; i < num_resources; i++) {
71         scanf("%d", &available[i]);
72     }
73
74
75

```

★ main.c

```

77
78     // Input the maximum resource allocation for each process
79
80     printf("Enter the maximum resource allocation for each process:\n");
81
82     for (int i = 0; i < num_processes; i++) {
83         printf("Process %d: ", i);
84
85         for (int j = 0; j < num_resources; j++) {
86             scanf("%d", &maximum[i][j]);
87         }
88     }
89
90
91
92
93
94
95     // Input the current resource allocation for each process
96
97     printf("Enter the current resource allocation for each process:\n");
98
99     for (int i = 0; i < num_processes; i++) {
100
101         printf("Process %d: ", i);
102
103         for (int j = 0; j < num_resources; j++) {
104             scanf("%d", &allocation[i][j]);
105
106             need[i][j] = maximum[i][j] - allocation[i][j];
107
108             finished[i] = false;
109
110         }
111     }
112
113
114
115

```

★ main.c

```

116
117     // Check if the system is in a safe state
118
119     if (is_safe_state()) {
120         printf("The system is in a safe state.\n");
121     } else {
122         printf("The system is not in a safe state.\n");
123     }
124
125
126
127
128
129
130
131     // Request additional resources
132
133     int request[MAX_RESOURCES];
134
135     int process;
136
137
138
139     printf("Enter the process requesting resources: ");
140
141     scanf("%d", &process);
142
143
144
145     printf("Enter the resource request for process %d: ", process);
146
147     for (int i = 0; i < num_resources; i++) {
148         scanf("%d", &request[i]);
149     }
150
151
152
153

```

```

154
155 // Check if the request can be granted immediately
156
157 if (is_request_valid(process, request)) {
158
159     if (is_safe_state_with_request(process, request)) {
160
161         printf("The request can be granted immediately.\n");
162
163     } else {
164
165         printf("The request cannot be granted immediately. It may lead to an unsafe state.\n");
166
167     }
168
169 } else {
170
171     printf("The request is invalid. It exceeds the maximum resource need or available resources.\n");
172
173 }
174
175
176
177 return 0;
178
179 }

```

Console: connection closed (Running: 26 seg)

```

Enter the number of processes: 2
Enter the number of resources: 1
Enter the available resources: 2
Enter the maximum resource allocation for each process:
Process 0: 1
Process 1: 1
Enter the current resource allocation for each process:
Process 0: 0
Process 1: 1
The system is not in a safe state.
Enter the process requesting resources: 1
Enter the resource request for process 1: 2
The request can be granted immediately.

```

PAJAMA PADHAI