

1)

Peterson Algorithm:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/wait.h> // Include this for wait()

#include <stdbool.h>


#define BSIZE 8 // Buffer size

#define PWT 2 // Producer wait time limit

#define CWT 10 // Consumer wait time limit

#define RT 10 // Program run-time in seconds


int shmid1, shmid2, shmid3, shmid4;

key_t k1 = 5491, k2 = 5812, k3 = 4327, k4 = 3213;

bool* SHM1;

int* SHM2;

int* SHM3;


int myrand(int n) {

    return (rand() % n + 1);

}


int main() {

    // Create shared memory segments

    shmid1 = shmget(k1, sizeof(bool) * 2, IPC_CREAT | 0660); // flag
```

22BCE0682 Siddhant Bhagat

```
shmidx = shmget(k2, sizeof(int) * 1, IPC_CREAT | 0660); // turn
shmidx = shmget(k3, sizeof(int) * BSZ, IPC_CREAT | 0660); // buffer
shmidx = shmget(k4, sizeof(int) * 1, IPC_CREAT | 0660); // state
```

```
if (shmidx < 0 || shmidx < 0 || shmidx < 0 || shmidx < 0) {
    perror("Main shmget error");
    exit(1);
}
```

```
SHM3 = (int*)shmat(shmidx, NULL, 0);
int ix = 0;
while (ix < BSZ) // Initializing buffer
    SHM3[ix++] = 0;
```

```
int* state = (int*)shmat(shmidx, NULL, 0);
*state = 1;
```

```
int wait_time;
int i = 0; // Consumer
int j = 1; // Producer
```

```
if (fork() == 0) { // Producer code
    SHM1 = (bool*)shmat(shmidx, NULL, 0);
    SHM2 = (int*)shmat(shmidx, NULL, 0);
    SHM3 = (int*)shmat(shmidx, NULL, 0);
    if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 == (int*)-1) {
        perror("Producer shmat error");
        exit(1);
    }
}
```

```
bool* flag = SHM1;
```

22BCE0682 Siddhant Bhagat

```
int* turn = SHM2;

int* buf = SHM3;

int index;

while (*state == 1) {

    flag[j] = true;

    *turn = i;

    while (flag[i] == true && *turn == i) ;

    // Critical Section Begin

    index = 0;

    while (index < BSIZE) {

        if (buf[index] == 0) {

            int tempo = myrand(BSIZE * 3);

            printf("Job %d has been produced\n", tempo);

            buf[index] = tempo;

            break;

        }

        index++;

    }

    if (index == BSIZE)

        printf("Buffer is full, nothing can be produced!!!\n");

    printf("Buffer: ");

    for (int k = 0; k < BSIZE; k++)

        printf("%d ", buf[k]);

    printf("\n");

    // Critical Section End

    flag[j] = false;

    if (*state == 0)

        break;
```

22BCE0682 Siddhant Bhagat

```
wait_time = myrand(PWT);

printf("Producer will wait for %d seconds\n\n", wait_time);

sleep(wait_time);

}

exit(0);

}

if (fork() == 0) { // Consumer code

    SHM1 = (bool*)shmat(shmid1, NULL, 0);
    SHM2 = (int*)shmat(shmid2, NULL, 0);
    SHM3 = (int*)shmat(shmid3, NULL, 0);
    if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 == (int*)-1) {
        perror("Consumer shmat error");
        exit(1);
    }

    bool* flag = SHM1;
    int* turn = SHM2;
    int* buf = SHM3;
    int index;
    flag[i] = false;
    sleep(5);
    while (*state == 1) {
        flag[i] = true;
        *turn = j;
        while (flag[j] == true && *turn == j) ;

        // Critical Section Begin
        if (buf[0] != 0) {
            printf("Job %d has been consumed\n", buf[0]);
            buf[0] = 0;
```

22BCE0682 Siddhant Bhagat

```
index = 1;

while (index < BSIZE) { // Shift remaining jobs forward
    buf[index - 1] = buf[index];
    index++;
}

buf[index - 1] = 0;
} else {
    printf("Buffer is empty, nothing can be consumed!!!\n");
}

printf("Buffer: ");
for (int k = 0; k < BSIZE; k++)
    printf("%d ", buf[k]);
printf("\n");
// Critical Section End

flag[i] = false;
if (*state == 0)
    break;

wait_time = myrand(CWT);
printf("Consumer will sleep for %d seconds\n\n", wait_time);
sleep(wait_time);
}

exit(0);
}

// Parent process will wait for RT seconds before causing child to terminate
sleep(RT);

*state = 0;

// Waiting for both processes to exit
wait(NULL);
```

22BCE0682 Siddhant Bhagat

```
wait(NULL);

printf("The clock ran out.\n");

return 0;

}
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <time.h>
5  #include <sys/types.h>
6  #include <sys/ipc.h>
7  #include <sys/shm.h>
8  #include <sys/wait.h> // Include this for wait()
9  #include <stdbool.h>
10
11 #define BSIZE 8 // Buffer size
12 #define PWT 2 // Producer wait time limit
13 #define CWT 10 // Consumer wait time limit
14 #define RT 10 // Program run-time in seconds
15
16 int shmid1, shmid2, shmid3, shmid4;
17 key_t k1 = 5491, k2 = 5812, k3 = 4327, k4 = 3213;
18 bool* SHM1;
19 int* SHM2;
20 int* SHM3;
21
22 int myrand(int n) {
23     return (rand() % n + 1);
24 }
25
26 int main() {
27     // Create shared memory segments
28     shmid1 = shmget(k1, sizeof(bool) * 2, IPC_CREAT | 0660); // flag
29     shmid2 = shmget(k2, sizeof(int) * 1, IPC_CREAT | 0660); // turn
30     shmid3 = shmget(k3, sizeof(int) * BSIZE, IPC_CREAT | 0660); // buffer
31     shmid4 = shmget(k4, sizeof(int) * 1, IPC_CREAT | 0660); // state
32
33     if (shmid1 < 0 || shmid2 < 0 || shmid3 < 0 || shmid4 < 0) {
34         perror("Main shmget error");
35         exit(1);
36     }
37
38     SHM3 = (int*)shmat(shmid3, NULL, 0);
39     int ix = 0;
40     while (ix < BSIZE) // Initializing buffer
```

22BCE0682 Siddhant Bhagat

```
40 while (ix < BSIZE) // Initializing buffer
41     SHM3[ix++] = 0;
42
43 int* state = (int*)shmat(shmid4, NULL, 0);
44 *state = 1;
45
46 int wait_time;
47 int i = 0; // Consumer
48 int j = 1; // Producer
49
50 if (fork() == 0) { // Producer code
51     SHM1 = (bool*)shmat(shmid1, NULL, 0);
52     SHM2 = (int*)shmat(shmid2, NULL, 0);
53     SHM3 = (int*)shmat(shmid3, NULL, 0);
54     if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 == (int*)-1) {
55         perror("Producer shmat error");
56         exit(1);
57     }
58
59     bool* flag = SHM1;
60     int* turn = SHM2;
61     int* buf = SHM3;
62     int index;
63
64     while (*state == 1) {
65         flag[j] = true;
66         *turn = i;
67         while (flag[i] == true && *turn == i) ;
68
69         // Critical Section Begin
70         index = 0;
71         while (index < BSIZE) {
72             if (buf[index] == 0) {
73                 int tempo = myrand(BSIZE * 3);
74                 printf("Job %d has been produced\n", tempo);
75                 buf[index] = tempo;
76                 break;
77             }
78             index++;
79         }
80     }
81 }
```

22BCE0682 Siddhant Bhagat

```
80         if (index == BSIZE)
81             printf("Buffer is full, nothing can be produced!!!\n");
82         printf("Buffer: ");
83         for (int k = 0; k < BSIZE; k++)
84             printf("%d ", buf[k]);
85         printf("\n");
86         // Critical Section End
87
88         flag[j] = false;
89         if (*state == 0)
90             break;
91         wait_time = myrand(PWT);
92         printf("Producer will wait for %d seconds\n\n", wait_time);
93         sleep(wait_time);
94     }
95     exit(0);
96 }
97
98 if (fork() == 0) { // Consumer code
99     SHM1 = (bool*)shmat(shmid1, NULL, 0);
100     SHM2 = (int*)shmat(shmid2, NULL, 0);
101     SHM3 = (int*)shmat(shmid3, NULL, 0);
102     if (SHM1 == (bool*)-1 || SHM2 == (int*)-1 || SHM3 == (int*)-1) {
103         perror("Consumer shmat error");
104         exit(1);
105     }
106
107     bool* flag = SHM1;
108     int* turn = SHM2;
109     int* buf = SHM3;
110     int index;
111     flag[i] = false;
112     sleep(5);
113     while (*state == 1) {
114         flag[i] = true;
115         *turn = j;
116         while (flag[j] == true && *turn == j) ;
117     }
```


22BCE0682 Siddhant Bhagat

```
118 // Critical Section Begin
119 if (buf[0] != 0) {
120     printf("Job %d has been consumed\n", buf[0]);
121     buf[0] = 0;
122     index = 1;
123     while (index < BSIZE) { // Shift remaining jobs forward
124         buf[index - 1] = buf[index];
125         index++;
126     }
127     buf[index - 1] = 0;
128 } else {
129     printf("Buffer is empty, nothing can be consumed!!!\n");
130 }
131 printf("Buffer: ");
132 for (int k = 0; k < BSIZE; k++)
133     printf("%d ", buf[k]);
134 printf("\n");
135 // Critical Section End
136
137 flag[i] = false;
138 if (*state == 0)
139     break;
140 wait_time = myrand(CWT);
141 printf("Consumer will sleep for %d seconds\n\n", wait_time);
142 sleep(wait_time);
143 }
144 exit(0);
145 }
146
147 // Parent process will wait for RT seconds before causing child to terminate
148 sleep(RT);
149 *state = 0;
150
151 // Waiting for both processes to exit
152 wait(NULL);
153 wait(NULL);
154 printf("The clock ran out.\n");
155 return 0;
156 }
```

Output:

22BCE0682 Siddhant Bhagat

```
input
Job 8 has been produced
Buffer: 8 0 0 0 0 0 0 0
Producer will wait for 1 seconds

Job 10 has been produced
Buffer: 8 10 0 0 0 0 0 0
Producer will wait for 2 seconds

Job 18 has been produced
Buffer: 8 10 18 0 0 0 0 0
Producer will wait for 2 seconds

Job 8 has been consumed
Buffer: 10 18 0 0 0 0 0 0
Consumer will sleep for 4 seconds

Job 11 has been produced
Buffer: 10 18 11 0 0 0 0 0
Producer will wait for 1 seconds

Job 10 has been produced
Buffer: 10 18 11 10 0 0 0 0
Producer will wait for 2 seconds

Job 3 has been produced
Buffer: 10 18 11 10 3 0 0 0
Producer will wait for 2 seconds

Job 10 has been consumed
Buffer: 18 11 10 3 0 0 0 0
Consumer will sleep for 7 seconds

The clock ran out.

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

2)

Critical Section Solution:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <pthread.h>
```

```
#include <unistd.h>
```

```
#define NUM_THREADS 5
```

```
// Global variable to be accessed in the critical section
```

```
int shared_resource = 0;
```

22BCE0682 Siddhant Bhagat

```
// Mutex for protecting the critical section
```

```
pthread_mutex_t mutex;
```

```
// Entry section
```

```
void enter_critical_section() {
```

```
    pthread_mutex_lock(&mutex);
```

```
}
```

```
// Critical section
```

```
void access_critical_section(int thread_id) {
```

```
    printf("Thread %d is entering the critical section.\n", thread_id);
```

```
    // Simulate some work in the critical section
```

```
    int temp = shared_resource;
```

```
    sleep(1); // Simulate time-consuming task
```

```
    shared_resource = temp + 1;
```

```
    printf("Thread %d has updated shared_resource to %d.\n", thread_id, shared_resource);
```

```
}
```

```
// Exit section
```

```
void exit_critical_section() {
```

```
    pthread_mutex_unlock(&mutex);
```

```
}
```

```
void* thread_function(void* arg) {
```

```
    int thread_id = *((int*)arg);
```

```
    // Entry section
```

```
    enter_critical_section();
```

```
    // Critical section
```

22BCE0682 Siddhant Bhagat

```
access_critical_section(thread_id);

// Exit section
exit_critical_section();

return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];

    // Initialize the mutex
    pthread_mutex_init(&mutex, NULL);

    // Create threads
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, thread_function, &thread_ids[i]);
    }

    // Join threads
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&mutex);

    printf("Final value of shared_resource: %d\n", shared_resource);
    return 0;
}
```

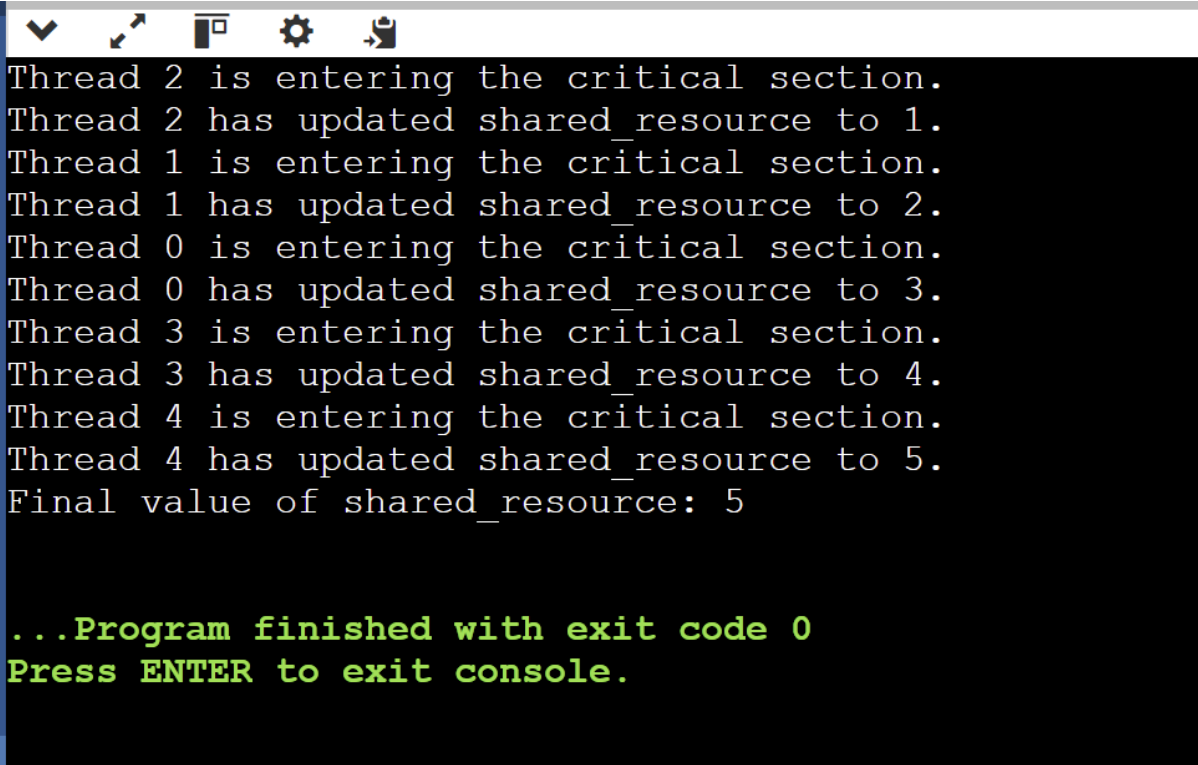
22BCE0682 Siddhant Bhagat

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <unistd.h>
5
6  #define NUM_THREADS 5
7
8  // Global variable to be accessed in the critical section
9  int shared_resource = 0;
10
11 // Mutex for protecting the critical section
12 pthread_mutex_t mutex;
13
14 // Entry section
15 void enter_critical_section() {
16     pthread_mutex_lock(&mutex);
17 }
18
19 // Critical section
20 void access_critical_section(int thread_id) {
21     printf("Thread %d is entering the critical section.\n", thread_id);
22     // Simulate some work in the critical section
23     int temp = shared_resource;
24     sleep(1); // Simulate time-consuming task
25     shared_resource = temp + 1;
26     printf("Thread %d has updated shared_resource to %d.\n", thread_id, shared_resource);
27 }
28
29 // Exit section
30 void exit_critical_section() {
31     pthread_mutex_unlock(&mutex);
32 }
33
34 void* thread_function(void* arg) {
35     int thread_id = *((int*)arg);
36
37     // Entry section
38     enter_critical_section();
39 }
```

22BCE0682 Siddhant Bhagat

```
40 // Critical section
41 access_critical_section(thread_id);
42
43 // Exit section
44 exit_critical_section();
45
46 return NULL;
47 }
48
49 int main() {
50     pthread_t threads[NUM_THREADS];
51     int thread_ids[NUM_THREADS];
52
53     // Initialize the mutex
54     pthread_mutex_init(&mutex, NULL);
55
56     // Create threads
57     for (int i = 0; i < NUM_THREADS; i++) {
58         thread_ids[i] = i;
59         pthread_create(&threads[i], NULL, thread_function, &thread_ids[i]);
60     }
61
62     // Join threads
63     for (int i = 0; i < NUM_THREADS; i++) {
64         pthread_join(threads[i], NULL);
65     }
66
67     pthread_mutex_destroy(&mutex);
68
69     printf("Final value of shared_resource: %d\n", shared_resource);
70     return 0;
71 }
72
```

Output:

A terminal window with a dark background and a light blue title bar. The title bar contains several icons: a checkmark, a cursor, a window icon, a gear, and a document. The output text is as follows:

Thread 2 is entering the critical section.
Thread 2 has updated shared_resource to 1.
Thread 1 is entering the critical section.
Thread 1 has updated shared_resource to 2.
Thread 0 is entering the critical section.
Thread 0 has updated shared_resource to 3.
Thread 3 is entering the critical section.
Thread 3 has updated shared_resource to 4.
Thread 4 is entering the critical section.
Thread 4 has updated shared_resource to 5.
Final value of shared_resource: 5

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