**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

shmid2 = shmget(k2, sizeof(int) \* 1, IPC\_CREAT | 0660); // turn

shmid3 = shmget(k3, sizeof(int) \* BSIZE, IPC\_CREAT | 0660); // buffer

shmid4 = shmget(k4, sizeof(int) \* 1, IPC\_CREAT | 0660); // state

if (shmid1 < 0 || shmid2 < 0 || shmid3 < 0 || shmid4 < 0) {

perror("Main shmget error");

exit(1);

}

SHM3 = (int\*)shmat(shmid3, NULL, 0);

int ix = 0;

while (ix < BSIZE) // Initializing buffer

SHM3[ix++] = 0;

int\* state = (int\*)shmat(shmid4, NULL, 0);

\*state = 1;

int wait\_time;

int i = 0; // Consumer

int j = 1; // Producer

if (fork() == 0) { // Producer 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("Producer shmat error");

exit(1);

}

bool\* flag = SHM1;

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;

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;

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);

wait(NULL);

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

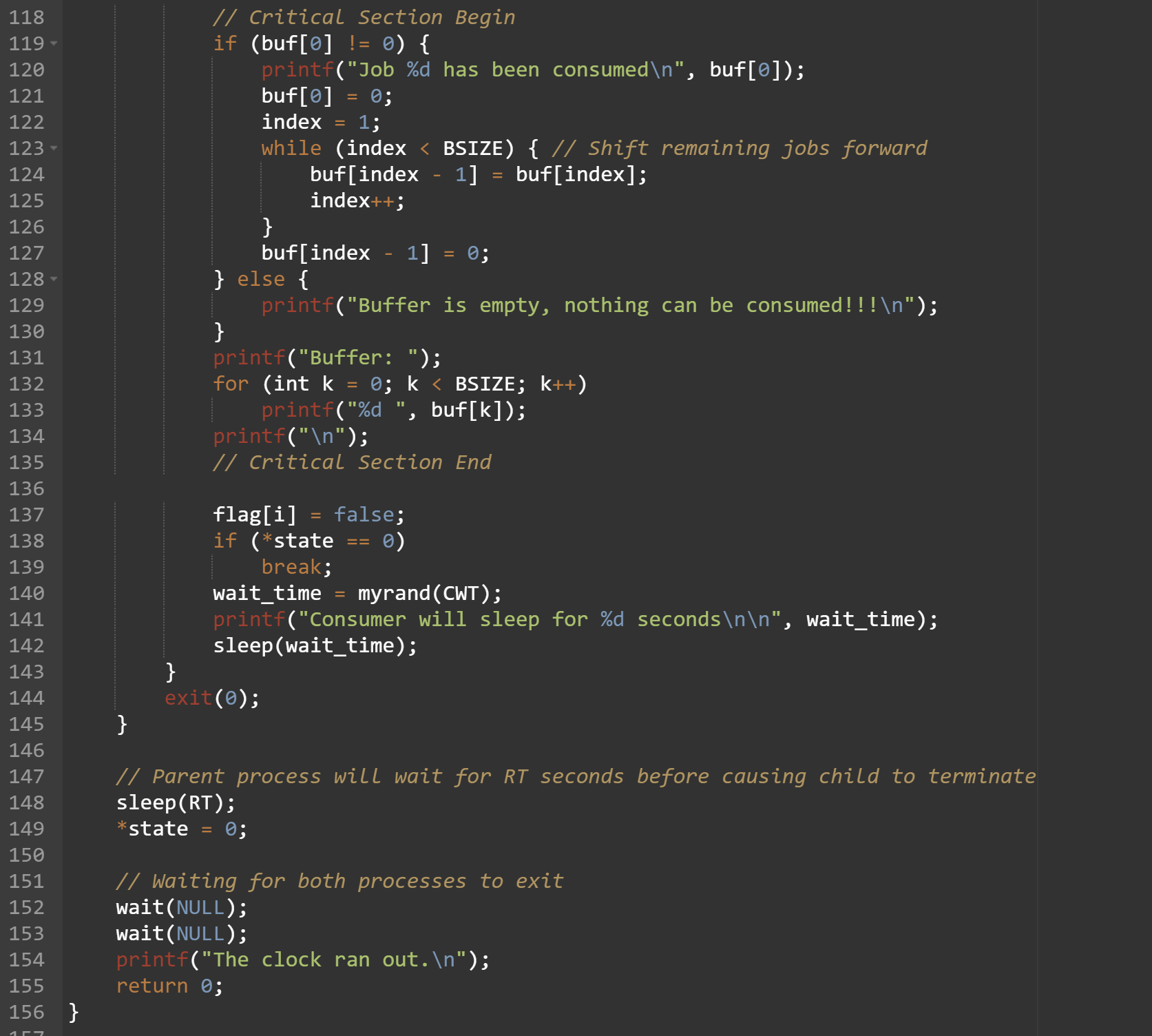
return 0;

}

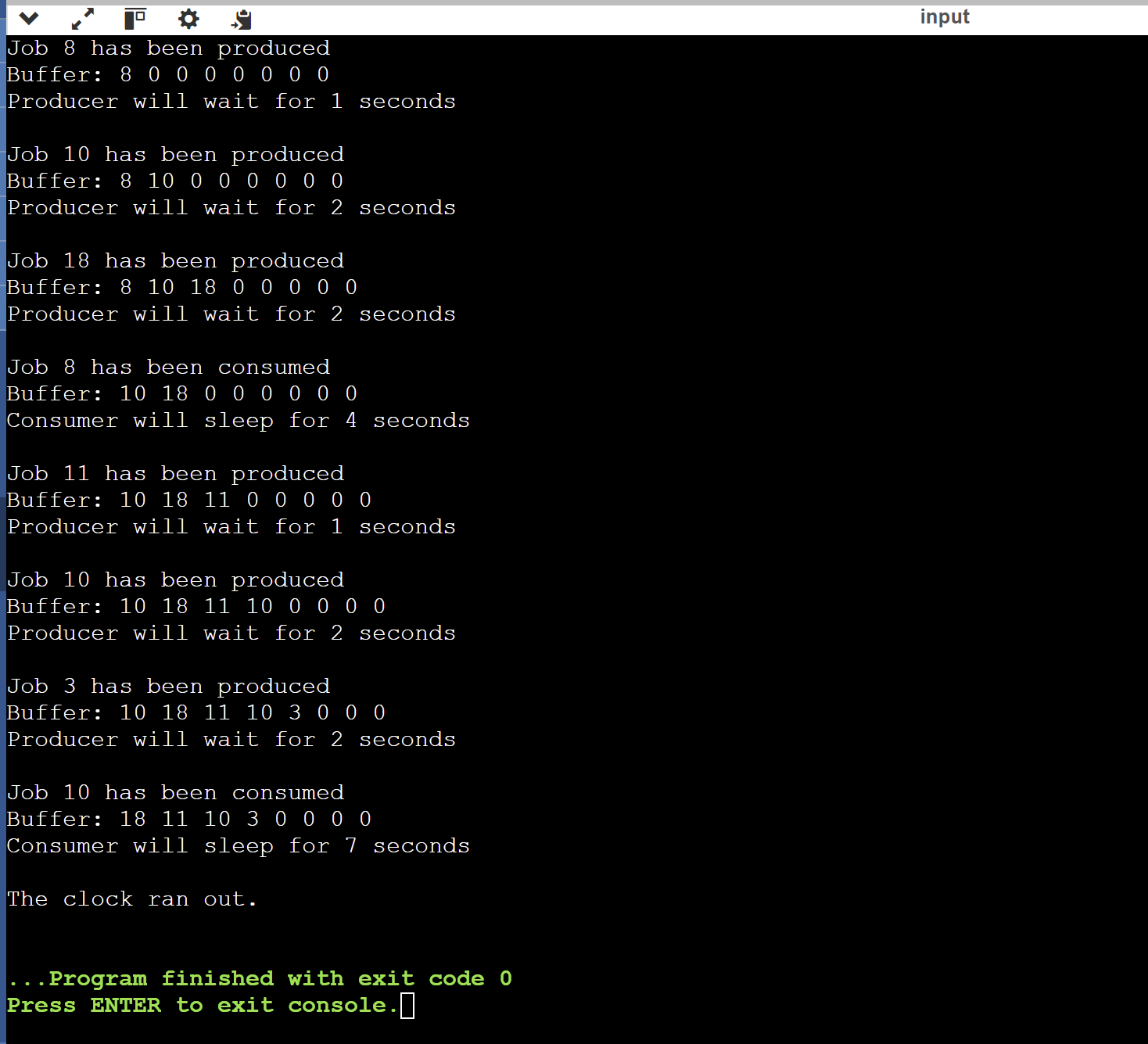








**Output:**



**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;

// 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

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);

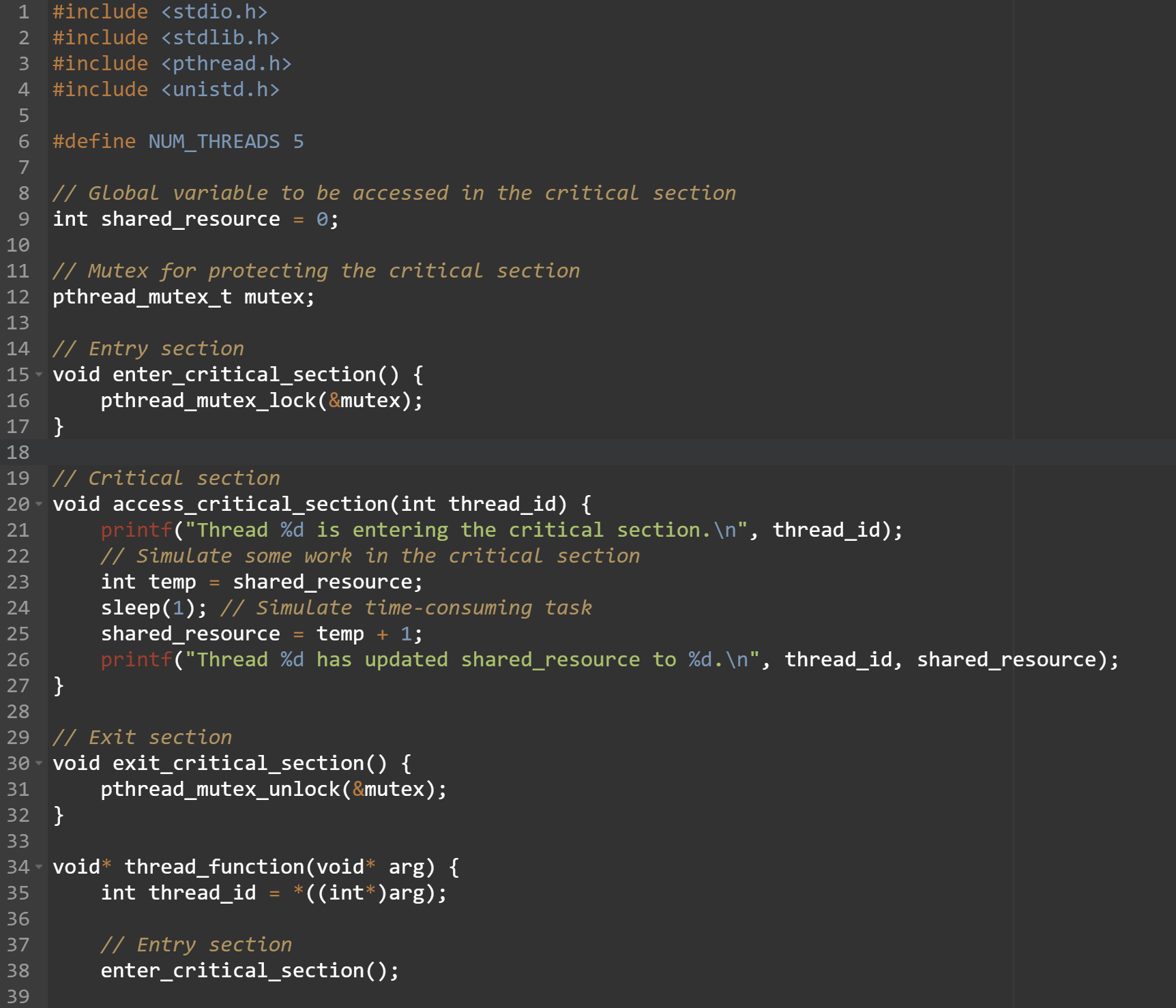
}

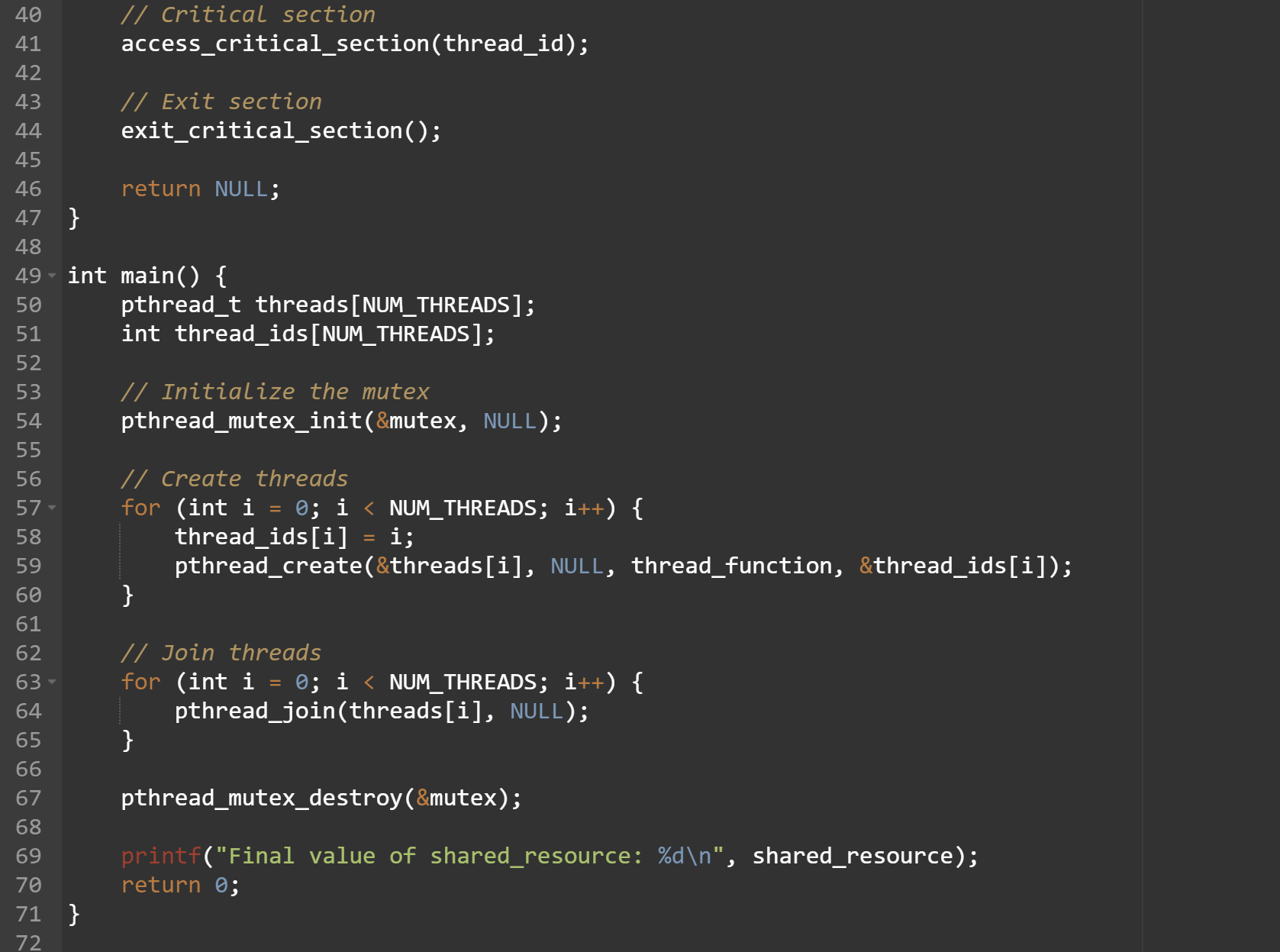
pthread\_mutex\_destroy(&mutex);

printf("Final value of shared\_resource: %d\n", shared\_resource);

return 0;

}





**Output:**

