// C program to implement Peterson’s Algorithm

// for producer-consumer problem.

#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

#include &lt;unistd.h&gt;

#include &lt;time.h&gt;

#include &lt;sys/types.h&gt;

#include &lt;sys/ipc.h&gt;

#include &lt;sys/shm.h&gt;

#include &lt;stdbool.h&gt;

#define \_BSD\_SOURCE

#include &lt;sys/time.h&gt;

#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) // Returns a random number between 1 and n

{

time\_t t;

srand((unsigned)time(&amp;t));

return (rand() % n + 1);

}

int main()

{

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); // time stamp

if (shmid1 &lt; 0 || shmid2 &lt; 0 || shmid3 &lt; 0 || shmid4 &lt; 0) {

perror(&quot;Main shmget error: &quot;);

exit(1);

}

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

int ix = 0;

while (ix &lt; BSIZE) // Initializing buffer

SHM3[ix++] = 0;

struct timeval t;

time\_t t1, t2;

gettimeofday(&amp;t, NULL);

t1 = t.tv\_sec;

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(&quot;Producer shmat error: &quot;);

exit(1);

}

bool\* flag = SHM1;

int\* turn = SHM2;

int\* buf = SHM3;

int index = 0;

while (\*state == 1) {

flag[j] = true;

printf(&quot;Producer is ready now.\n\n&quot;);

\*turn = i;

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

;

// Critical Section Begin

index = 0;

while (index &lt; BSIZE) {

if (buf[index] == 0) {

int tempo = myrand(BSIZE \* 3);

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

buf[index] = tempo;

break;

}

index++;

}

if (index == BSIZE)

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

printf(&quot;Buffer: &quot;);

index = 0;

while (index &lt; BSIZE)

printf(&quot;%d &quot;, buf[index++]);

printf(&quot;\n&quot;);

// Critical Section End

flag[j] = false;

if (\*state == 0)

break;

wait\_time = myrand(PWT);

printf(&quot;Producer will wait for %d seconds\n\n&quot;, 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(&quot;Consumer shmat error:&quot;);

exit(1);

}

bool\* flag = SHM1;

int\* turn = SHM2;

int\* buf = SHM3;

int index = 0;

flag[i] = false;

sleep(5);

while (\*state == 1) {

flag[i] = true;

printf(&quot;Consumer is ready now.\n\n&quot;);

\*turn = j;

while (flag[j] == true &amp;&amp; \*turn == j)

;

// Critical Section Begin

if (buf[0] != 0) {

printf(&quot;Job %d has been consumed\n&quot;, buf[0]);

buf[0] = 0;

index = 1;

while (index &lt; BSIZE) // Shifting remaining jobs forward

{

buf[index - 1] = buf[index];

index++;

}

buf[index - 1] = 0;

} else

printf(&quot;Buffer is empty, nothing can be consumed!!!\n&quot;);

printf(&quot;Buffer: &quot;);

index = 0;

while (index &lt; BSIZE)

printf(&quot;%d &quot;, buf[index++]);

printf(&quot;\n&quot;);

// Critical Section End

flag[i] = false;

if (\*state == 0)

break;

wait\_time = myrand(CWT);

printf(&quot;Consumer will sleep for %d seconds\n\n&quot;, wait\_time);

sleep(wait\_time);

}

exit(0);

}

// Parent process will now for RT seconds before causing child to terminate

while (1) {

gettimeofday(&amp;t, NULL);

t2 = t.tv\_sec;

if (t2 - t1 &gt; RT) // Program will exit after RT seconds

{

\*state = 0;

break;

}

}

// Waiting for both processes to exit

wait();

wait();

printf(&quot;The clock ran out.\n&quot;);

return 0;

}