OS Lab Assignment 5

## Name: Keshav Garg

## ID: 2018UCP1674

GitHub Repo link: [**link**](https://github.com/gargk747/OS-Lab/tree/master/Assignment%205)

**Dekker’s Algorithm (Two Processes):**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <sys/shm.h>

#include <sys/wait.h>

int shmID, shrID, shrID2, fav;

key\_t key1 = 54991, key2 = 784519, key3 = 882363, key4 = 23456;

int \*p1\_wants\_to\_enter,\*p2\_wants\_to\_enter, \*shared,\*favored;

int main(){

shmID = shmget(key1, sizeof(int), IPC\_CREAT | 0660);

if(shmID < 0){

printf("Shared memory 1 could not be allocated\n");

exit(1);

}

shrID = shmget(key2, sizeof(int), IPC\_CREAT | 0660);

if(shrID < 0){

printf("Shared memory 2 could not be allocated\n");

exit(1);

}

shrID2 = shmget(key3, sizeof(int), IPC\_CREAT | 0660);

if(shrID < 0){

printf("Shared memory 3 could not be allocated\n");

exit(1);

}

fav = shmget(key4, sizeof(int), IPC\_CREAT | 0660);

if(fav < 0){

printf("Shared memory 4 could not be allocated\n");

exit(1);

}

p1\_wants\_to\_enter = (int \*)shmat(shmID, NULL, 0);

favored= (int \*)shmat(fav, NULL, 0);

if (p1\_wants\_to\_enter == (int \*)-1 ) {

printf("Failed to attach p1\_wants\_to\_enter to first process\n");

exit(-1);

}

p2\_wants\_to\_enter = (int \*)shmat(shrID2, NULL, 0);

if (p2\_wants\_to\_enter== (int \*)-1 ) {

printf("Failed to attach p2\_wants\_to\_enter to first process\n");

exit(-1);

}

shared = (int \*)shmat(shrID, NULL, 0);

if (shared == (int \*)-1 ) {

printf("Failed to attach shared to first process\n");

exit(-1);

}

if (favored == (int \*)-1 ) {

printf("Failed to attach favoured to first process\n");

exit(-1);

}

\*shared = 1;

\*p1\_wants\_to\_enter = 0;

\*p2\_wants\_to\_enter = 0;

\*favored=1;

if (fork() == 0) {

srand(time(0));

p1\_wants\_to\_enter = (int \*)shmat(shmID, NULL, 0);

p2\_wants\_to\_enter = (int \*)shmat(shrID2, NULL, 0);

favored=(int \*)shmat(fav, NULL, 0);

shared = (int \*)shmat(shrID, NULL, 0);

if (p1\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p1\_wants\_to\_enter to first process\n");

exit(-1);

}

if (p2\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p2\_wants\_to\_enter to first process\n");

exit(-1);

}

if (favored == (int \*)-1 ){

printf("Failed to attach favored to first process\n");

exit(-1);

}

if (shared == (int \*)-1 ){

printf("Failed to attach shared variable to first process\n");

exit(-1);

}

int done=0;

for(done=0;done<=15;done++){

\*p1\_wants\_to\_enter=1;

while(\*p2\_wants\_to\_enter == 1){

if(\*favored == 2){

\*p1\_wants\_to\_enter = 0;

while(\*favored == 2);

\*p1\_wants\_to\_enter = 1;

}

}

\*shared+=10;

printf("Process 1: %d\n",\*shared);

\*favored = 2;

\*p1\_wants\_to\_enter = 0;

}

}

else {

if (fork() == 0){

srand(time(0));

p1\_wants\_to\_enter = (int \*)shmat(shmID, NULL, 0);

p2\_wants\_to\_enter = (int \*)shmat(shrID2, NULL, 0);

favored=(int \*)shmat(fav, NULL, 0);

shared = (int \*)shmat(shrID, NULL, 0);

if (p1\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p1\_wants\_to\_enter to first process\n");

exit(-1);

}

if (p2\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p2\_wants\_to\_enter to first process\n");

exit(-1);

}

if (favored == (int \*)-1 ){

printf("Failed to attach favored to first process\n");

exit(-1);

}

if (shared == (int \*)-1 ){

printf("Failed to attach shared variable to first process\n");

exit(-1);

}

int done=0;

for(done=0;done<=15;done++){

\*p2\_wants\_to\_enter=1;

while(\*p1\_wants\_to\_enter == 1){

if(\*favored == 1){

\*p2\_wants\_to\_enter = 0;

while(\*favored == 1);

\*p2\_wants\_to\_enter = 1;

}

}

\*shared+=15;

printf("Process 2: %d\n",\*shared);

\*favored = 1;

\*p2\_wants\_to\_enter = 0;

}

}

else{

sleep(1);

wait(NULL);

wait(NULL);

}

}

}

**Output:**

**Peterson’s Algorithm (Two Processes):**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <sys/shm.h>

#include <sys/wait.h>

int shmID, shrID, shrID2, fav;

key\_t key1 = 54991, key2 = 784519, key3 = 882363, key4 = 23456;

int \*p1\_wants\_to\_enter,\*p2\_wants\_to\_enter, \*shared,\*favored;

int main(){

shmID = shmget(key1, sizeof(int), IPC\_CREAT | 0660);

if(shmID < 0){

printf("Shared memory 1 could not be allocated\n");

exit(1);

}

shrID = shmget(key2, sizeof(int), IPC\_CREAT | 0660);

if(shrID < 0){

printf("Shared memory 2 could not be allocated\n");

exit(1);

}

shrID2 = shmget(key3, sizeof(int), IPC\_CREAT | 0660);

if(shrID < 0){

printf("Shared memory 3 could not be allocated\n");

exit(1);

}

fav = shmget(key4, sizeof(int), IPC\_CREAT | 0660);

if(fav < 0){

printf("Shared memory 4 could not be allocated\n");

exit(1);

}

p1\_wants\_to\_enter = (int \*)shmat(shmID, NULL, 0);

favored= (int \*)shmat(fav, NULL, 0);

if (p1\_wants\_to\_enter == (int \*)-1 ) {

printf("Failed to attach p1\_wants\_to\_enter to first process\n");

exit(-1);

}

p2\_wants\_to\_enter = (int \*)shmat(shrID2, NULL, 0);

if (p2\_wants\_to\_enter== (int \*)-1 ) {

printf("Failed to attach p2\_wants\_to\_enter to first process\n");

exit(-1);

}

shared = (int \*)shmat(shrID, NULL, 0);

if (shared == (int \*)-1 ) {

printf("Failed to attach shared to first process\n");

exit(-1);

}

if (favored == (int \*)-1 ) {

printf("Failed to attach favoured to first process\n");

exit(-1);

}

\*shared = 1;

\*p1\_wants\_to\_enter = 0;

\*p2\_wants\_to\_enter = 0;

\*favored=1;

if (fork() == 0) {

srand(time(0));

p1\_wants\_to\_enter = (int \*)shmat(shmID, NULL, 0);

p2\_wants\_to\_enter = (int \*)shmat(shrID2, NULL, 0);

favored=(int \*)shmat(fav, NULL, 0);

shared = (int \*)shmat(shrID, NULL, 0);

if (p1\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p1\_wants\_to\_enter to first process\n");

exit(-1);

}

if (p2\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p2\_wants\_to\_enter to first process\n");

exit(-1);

}

if (favored == (int \*)-1 ){

printf("Failed to attach favored to first process\n");

exit(-1);

}

if (shared == (int \*)-1 ){

printf("Failed to attach shared variable to first process\n");

exit(-1);

}

int done=0;

for(done=0;done<=15;done++){

\*p1\_wants\_to\_enter=1;

\*favored=1;

while(\*p2\_wants\_to\_enter && (\*favored==2));

\*shared+=10;

printf("Process 1: %d\n",\*shared);

\*p1\_wants\_to\_enter=0;

}

}

else {

if (fork() == 0){

srand(time(0));

p1\_wants\_to\_enter = (int \*)shmat(shmID, NULL, 0);

p2\_wants\_to\_enter = (int \*)shmat(shrID2, NULL, 0);

favored=(int \*)shmat(fav, NULL, 0);

shared = (int \*)shmat(shrID, NULL, 0);

if (p1\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p1\_wants\_to\_enter to first process\n");

exit(-1);

}

if (p2\_wants\_to\_enter == (int \*)-1 ){

printf("Failed to attach p2\_wants\_to\_enter to first process\n");

exit(-1);

}

if (favored == (int \*)-1 ){

printf("Failed to attach favored to first process\n");

exit(-1);

}

if (shared == (int \*)-1 ){

printf("Failed to attach shared variable to first process\n");

exit(-1);

}

int done=0;

for(done=0;done<=15;done++){

\*p2\_wants\_to\_enter=1;

\*favored=2;

while(\*p2\_wants\_to\_enter && (\*favored==1));

\*shared+=15;

printf("Process 2: %d\n",\*shared);

\*p2\_wants\_to\_enter=0;

}

}

else{

sleep(1);

wait(NULL);

wait(NULL);

}

}

}

**Output:**

**Dekker’s Algorithm (Two Threads):**

// gcc -pthread <name.c>

// This program creates two threads, each thread

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

int x = 0; // Shared variable

#define THREAD1 0

#define THREAD2 1

#define TRUE 1

#define FALSE 0

int p1\_wantstoenter;

int p2\_wantstoenter;

int favouredProcess;

void Initialization(void) {

p1\_wantstoenter=0;

p2\_wantstoenter=0;

favouredProcess=1;

srand(time(NULL));

}

void \*func1(void \*s) {

int i, k;

for (i=0; i < 5; i++) {

p1\_wantstoenter=1;

while(p2\_wantstoenter)

{

if(favouredProcess==2)

{

p1\_wantstoenter=0;

while(favouredProcess==2)

;

p1\_wantstoenter=1;

}

}

// Critical Section - shared variable is incremented

x = x+1;

printf("[%2d] : Thread 1 in Critical Section (%d).\n", i+1,x);

favouredProcess=2;

p1\_wantstoenter=0;

// Random delay added

k = (int) ((3.0\*rand())/RAND\_MAX);

sleep(k);

}

}

void \*func2(void \*s) {

int i, k;

for (i=0; i < 5; i++) {

p2\_wantstoenter=1;

while(p1\_wantstoenter)

{

if(favouredProcess==1)

{

p2\_wantstoenter=0;

while(favouredProcess==1)

;

p2\_wantstoenter=1;

}

}

// Critical Section - shared variable is incremented

x = x+1;

printf("[%2d] : Thread 2 in Critical Section (%d).\n", i+1,x);

favouredProcess=1;

p2\_wantstoenter=0;

// Random delay added

k = (int) ((5.0\*rand())/RAND\_MAX);

sleep(k);

}

}

int main() {

pthread\_t Thread1, Thread2;

// Initialized the lock then fork 2 threads

Initialization();

// Create two threads (both run func)

pthread\_create(&Thread1, NULL, func1, (void \*) THREAD1);

pthread\_create(&Thread2, NULL, func2, (void \*) THREAD2);

// Wait for the threads to end.

pthread\_join(Thread1, NULL);

pthread\_join(Thread2, NULL);

return 0;

}

**Output:**

**Peterson’s Algorithm (Two Threads):**

// gcc -pthread <name.c>

// This program creates two threads, each thread

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

int x = 0; // Shared variable

#define THREAD1 0

#define THREAD2 1

#define TRUE 1

#define FALSE 0

int p1\_wantstoenter;

int p2\_wantstoenter;

int favouredProcess;

void Initialization(void) {

p1\_wantstoenter=0;

p2\_wantstoenter=0;

favouredProcess=1;

srand(time(NULL));

}

void \*func1(void \*s) {

int i, k;

for (i=0; i < 5; i++) {

p1\_wantstoenter=1;

favouredProcess=2;

while(p2\_wantstoenter && favouredProcess==2)

;

// Critical Section - shared variable is incremented

x = x+1;

printf("[%2d] : Thread 1 in Critical Section (%d).\n", i+1,x);

p1\_wantstoenter=0;

// Random delay added

k = (int) ((3.0\*rand())/RAND\_MAX);

sleep(k);

}

}

void \*func2(void \*s) {

int i, k;

for (i=0; i < 5; i++) {

p2\_wantstoenter=1;

while(p1\_wantstoenter && favouredProcess==2)

;

// Critical Section - shared variable is incremented

x = x+1;

printf("[%2d] : Thread 2 in Critical Section (%d).\n", i+1,x);

favouredProcess=1;

p2\_wantstoenter=0;

// Random delay added

//k = (int) ((5.0\*rand())/RAND\_MAX);

sleep(k);

}

}

int main() {

pthread\_t Thread1, Thread2;

// Initialized the lock then fork 2 threads

Initialization();

// Create two threads (both run func)

pthread\_create(&Thread1, NULL, func1, (void \*) THREAD1);

pthread\_create(&Thread2, NULL, func2, (void \*) THREAD2);

// Wait for the threads to end.

pthread\_join(Thread1, NULL);

pthread\_join(Thread2, NULL);

return 0;

}

**Output:**