

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

Address: U-15, Bhakti Vedant Swamy Marg, Mumbai, Maharashtra 400056

A mini – project report on

IMPLEMENTATION OF DEKKER’S ALGORITHM

prepared for work in the subject Operating Systems in the year 2014 - 2015.

Year: 2014 - 2015

Semester: V

Department: Computer Engineering

Subject: Operating Systems

Submitted to: Prof. Aruna Gawde



DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

Address: U-15, Bhakti Vedant Swamy Marg, Mumbai, Maharashtra 400056

A mini – project report on

# IMPLEMENTATION OF DEKKER’S ALGORITHM

Prepared by:

|  |  |
| --- | --- |
| SAP ID | NAME |
|  |  |
| 60004120058 | Ryan Mathews |
| 60004138001 | Drumil Bakhai |
| 60004138020 | Umang Shah |

Date: 27th October, 2014

Prof. Aruna Gawde

Professor – in – charge

ACKNOWLEDGEMENTS

We would like to take this opportunity to express our sincere gratitude and deep regards to Prof. Aruna Gawde for their impeccable guidance, monitoring and constant encouragement throughout the creation of the report. We take immense pleasure in presenting this report which is testimony to a vitally important learning opportunity that we will always cherish.

ABSTRACT

In concurrent programming, a critical section is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution. A critical section will usually terminate in fixed time, and a thread, task, or process will have to wait for a fixed time to enter it (aka bounded waiting). Some synchronization mechanism is required at the entry and exit of the critical section to ensure exclusive use, for example a semaphore.

By carefully controlling which variables are modified inside and outside the critical section, concurrent access to that state is prevented. A critical section is typically used when a multithreaded program must update multiple related variables without a separate thread making conflicting changes to that data. In a related situation, a critical section may be used to ensure a shared resource, for example a printer, can only be accessed by one process at a time.

How critical sections are implemented varies from system to system. This report deals with the Dekker’s Algorithm.

DEKKER’S ALGORITHM

Dekker's algorithm is the first known correct solution to the mutual exclusion problem in concurrent programming.It allows two threads to share a single-use resource without conflict, using only shared memory for communication.

It avoids the strict alternation of a naïve turn-taking algorithm, and was one of the first mutual exclusion algorithms to be invented.

If two processes attempt to enter a critical section at the same time, the algorithm will allow only one process in, based on whose turn it is. If one process is already in the critical section, the other process will busy wait for the first process to exit. This is done by the use of two flags, flag[0] and flag[1], which indicate an intention to enter the critical section and a turn variable that indicates who has priority between the two processes.

The algorithm may be given as:

**Init**:

flag[0] = false

flag[1] = false

turn = 0 // or 1

**P0:**

flag[0] = true;

while (flag[1] == true) {

if (turn ≠ 0) {

flag[0] = false;

while (turn ≠ 0) {

// busy wait

}

flag[0] = true;

}

}

// critical section

...

turn = 1;

flag[0] = false;

// remainder section

**P1:**

flag[1] = true;

while (flag[0] == true) {

if (turn ≠ 1) {

flag[1] = false;

while (turn ≠ 1) {

// busy wait

}

flag[1] = true;

}

}

// critical section

...

turn = 0;

flag[1] = false;

// remainder section

PROGRAM

public void choice()

{

if(choice == 4)

{

if(turn == 1){

text1.setText("P1 inside CS. P2 Waits");

choice = 1;

}

if(turn == 2)

{

text1.setText("P2 inside CS. P1 Waits");

choice = 2;

}

}

if(choice==1)

{

c1=0;

while(c2==0)

{

if(turn==2)

c1=1;

while(turn==2);

c1=0;

}

text1.setText("\nProcess P1 Enters the Critical section. S = " + (++shared) +"\n");

c1=1;

turn=2;

}

else

{

text1.setText("\nIt is the turn process P2");

}

if(choice==2)

{

c2=0;

while(c1==0)

{

if(turn==1)

c2=1;

while(turn==1);

c2=0;

}

text3.setText("\nProcess P2 enters in critical section. S = " + (--shared) +"\n");

c2=1;

turn=1;

}

else

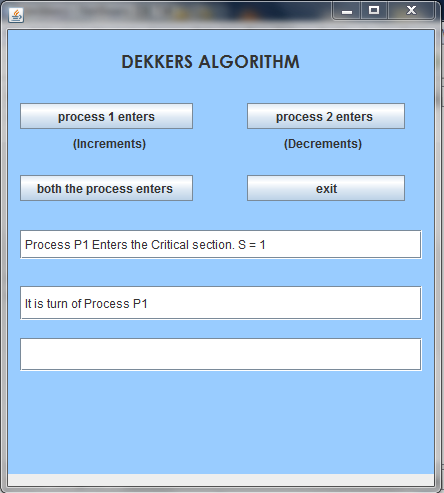
{

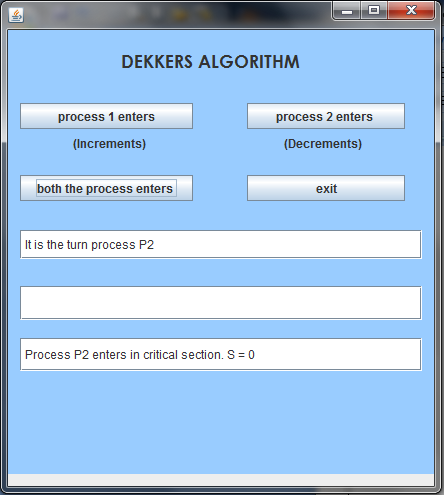
text2.setText("\nIt is turn of Process P1\n");

}

}

OUTPUT





BIBLIOGRAPHY

1. Operating Systems Concepts – Galvin, Silberschatz and Galvin
2. Operating Systems – William Stallings
3. Modern Operating Systems – Andrew S Tanenbaum

WEBLIOGRAPHY

1. <http://www.csee.wvu.edu/~jdm/classes/cs550/notes/tech/mutex/Dekker.html>
2. <http://www2.cs.uregina.ca/~hamilton/courses/330/notes/synchro/node2.html>
3. <http://hamilton.nuigalway.ie/teaching/AOS/FIVE/dekker.html>