A modified Conveyor-belt Algorithm to solve the critical section problem

Nilanjan Chanda (-nilanjanchanda22@gmail.com)  
West Bengal University of Technology

Research Article

Keywords: bounded waiting, critical section, mutual exclusion, conveyor belt algorithm

Posted Date: September 21st, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1938524/v2

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License
A modified Conveyor-belt Algorithm to solve the critical section problem

Nilanjan Chanda\textsuperscript{1*}

\textsuperscript{1*}Department of Computer Application, Asansol Engineering College, Sen Raleigh Road, Asansol, 713305, West Bengal, India.

Corresponding author(s). E-mail(s): nilanjanchanda22@gmail.com;

Abstract

In the domain of Operating System critical problem and its related concepts play an important role. Process Management is the most vital part for executing various programs required where every process competes for CPU and may use some common variables and changes made to these variables in concurrent processing can produce undesirable results. In this paper a different type of algorithm is proposed which deals effectively for solving the critical section problem. This algorithm is based on the working of a conveyor belt that a given set of processes shall circulate over the conveyor belt and any process should be able to execute their critical section when it is its turn while the other processes must continue to circulate, waiting for their turn to enter the critical section until all of them are executed.

Keywords: bounded waiting, critical section, mutual exclusion, conveyor belt algorithm

1 Introduction

Every process in the queue is gathered to be executed. While during execution two or more processes come to a point when they demand to execute their critical section. While concurrent processing may give rise to many problems such as out-of-order change of a common variable which needs to be synchronized\textsuperscript{[1]}. There are several algorithms for execution of critical section
by the processes in a mutually exclusive manner[2]. This is obvious that a system which is controlled and monitored by belt has to be checked for maintenance of flow of processes without expected deviation. Therefore, to know the actual working principal of conveyor belt[3] and its applications in an intelligent way by controlling its operations to develop a more expert method for conveyor belt algorithm[4]. While there are various types of algorithms depending on these two categories: permission based[5, 6] (where the process entering critical section needs permission from other processes), token based[7–9](where the process will enter critical section receiving the token). Some other algorithm is based on multiple k-entries using directed graph[10]. In this proposed work, the author have developed a conveyor belt algorithm[11, 12] to solve critical section problem in operating system which does not require any token or permission protocol or use any directed graph as well for finding the solution to critical section problem because of its iterative nature. The author proposed an algorithm whose mechanism is analogous to management of products in an assembly line of any manufacturing unit. Referring back to the analogy mentioned above a scheduling queue is similar to a conveyor belt and here the processes are the items which are manufactured and are being sent to the next level of execution[13]. A role of a conveyor belt is to transfer the products from one section to the next, while transferring the products are examined to pass a certain criterion to be admitted to the next section where those products which does not meet the required standards at that moment is halted for a while and the chance is given to the next process which meet the conditions until they are ready to fulfil the conditions required. The process scheduling for critical section can be handled in a similar fashion. While checking for solution of critical section for the processes those who don’t meet the conditions required is rolled down the conveyor-belt like arrangement for the next process to come up. This queue is iterative until all the processes executed their critical section.

2 Proposed Methodology

2.1 Initialization

The mechanism adopted in this algorithm is the syntactic representation of the gross working of a conveyor belt from the surface level. By principle a conveyor has two belts i.e., upper belt and lower belt which moves objects from one point of reference to the another, where both the movements are opposite with respect to each other. Firstly, the belts have been denoted using two arrays named as u[N] and l[N] respectively where u denotes upper and l denotes lower; whereas N denotes the length of both arrays. In the program all the processes are inserted in an array named thread[N] which acts as the process queue. All these variables including the required counters are declared globally for their better utilization while using them in various function calls preventing undesirable changes in their values while execution.
2.2 Arrangement

A critical section problem consists of four parts namely, i) entry section ii) critical section iii) exit section iv) remainder section. Here the entry section is executed by a function declared for this task where the conveyor-belt decides which process will enter the critical section based on the three general conditions of a critical section problem. The critical section is performed by the next function where the process uses the resource it needs to execute where it sets the lock variable after entering ensuring no other process gets permission while it is executing. The exit section is executed by another function declared which free the occupied resource and resets the lock variable.

2.3 Working Mechanism

In the main function the processes are created and are inserted into the array thread of length N. After the creation one function is called which acts the entry section where the processes are moved in the array u[N] indicating as the ‘upper belt’ of the conveyor. To check how many processes are entering and leaving the array two counter variables are employed for that task namely full initialized as f and the other is empty initialized as e. The counter variables f and e are updated respectively whenever any process has left or entered the array u[N] maintaining the flow of processes in the entry section. The function now checks whether there is any lock or not. If there is no lock, the first process enters into the critical section which is done by calling the function where it sets the lock variable to 1. In the critical section the process acquires a resource. For satisfying the condition of mutual exclusion the next processes are moved to the array l[N] which indicates the ‘lower belt’ of the conveyor until the critical section is freed by the acquired process. The process executing in the critical section uses a flag counter. When the process is executing by using a loop, the next processes are moved to the array l[N] and the flag is incremented by one each time a process is rolled down to l[N] until the flag counter reaches x(where x denotes the time quantum) while the critical section is engaged by a previous process and thus the maximum number of processes waiting in the array l[N] . If the conditions are met, the resource is freed and the control comes out of the loop. Now, the exit section is executed by evoking a function where the lock variable is reset and the control is returned back to the function which was controlling the entry section where, the next process which is standing in the array u enters the critical section upon finding the lock variable is 0 and set it to 1 indicating that the critical section is again engaged. In a similar fashion the next x number of processes are moved to the array l[N] and waits till the critical section is occupied for the quantum time frame of x units and released when the flag reaches x, i.e., flag=x. After this the control comes to the function of exit section where the lock variable is again reset to 0. This iteration is continued until all the processes in the given set of process queue(PQ) have executed their critical section after which the execution stops until the next set of processes are inserted in the array u[N] to start it again.


2.4 Schematic Diagram

The working mechanism of the proposed algorithm is demonstrated below. The general structural layout of the algorithm is shown in Fig. 1, here all the processes are queued in the process queue (PQ) which are ready to be placed on the upper belt denoted by $u[N]$ with $N$ number of segments ($i_0, i_1, \ldots, i_n - 1$). The lower belt is denoted by $l[N]$ with $N$ number of segments ($i_0, i_1, \ldots, i_n - 1$). Transfer of processes from upper belt to lower belt is shown by the downward facing arrow denoted by ‘rolling down’ while the processes transferred back to the upper belt is shown by the upward facing arrow denoted by ‘rolling up’. The lock variable is initialized to 0 indicating its empty at the start.

Here Fig. 2 shows the movement of processes in follow after they are being placed on the upper belt.
A modified Conveyor-belt Algorithm to solve the critical section problem

Fig. 3

The very first process enters the critical section and set the lock indicating its occupied Fig. 3 upon which the next process is rolled down to the lower belt.

Fig. 4

To sustain the conditions for critical section every waiting process in the lower belt waits for certain time interval $t_L$ (where $t_L =$ time required to traverse the lower belt) must get chance to enter the critical section when they are rolled up on the upper belt after a certain number of processes have passed through that time interval $t_L$ before them Fig. 4.

3 Algorithm

Below is the pseudo-code of the algorithm.
A modified Conveyor-belt Algorithm to solve the critical section problem

```
repeat
    for i:0 to N-1
        u[i]=thread[i]
        f:=f+1
    end;
repeat
    if lock=false
        f:=f-1
    CRITICAL SECTION
    lock=true;
    while lock=true
        do
            i:=i+1
            l[i]=u[i]
            j:=j+1
            flag:=flag+1;
            if flag:=x
                break;
        done
    lock=false;
until f!=0;
```

4 System Configuration

Processor: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz RAM: 4.00 GB System type: 64-bit operating system, x64-based processor.

5 Experimented Results

Through executing the algorithm, as per the conveyor belt analogy as shown throughout Fig. 1-4. In order to test the efficacy of the algorithm, the author executed the algorithm 10 times with number of processes as input to the process queue to compute the number of steps taken while increasing the input size by adding 5 processes after every successful execution shown in Table 1. After performing asymptotic analysis of the algorithm and computed the time complexity of the program which is O(N) shown in the graph Fig. 5 below. As this algorithm executes each process under certain time quantum, all the three cases of time complexity are found to be similar i.e., O(N).
**A modified Conveyor-belt Algorithm to solve the critical section problem**

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Number of processes and their running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input size-N</td>
<td>Number of steps</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

Fig. 5  Time complexity graph

6 Conclusion

This algorithm maintained the three necessary conditions required for a critical section problem during its execution.

1. **Mutual exclusion:** When one process is executing in its critical section no other process is allowed to enter the critical section using the lock variable.

2. **Progress:** No process is idle, if it can’t execute in its critical section then it is moved to the array l giving chance to the other processes standing next to enter the critical section to maintain the flow of processes.

3. **Bounded Waiting:** Every process can execute the critical section for a certain time frame. When requesting for critical section and not getting access the process is moved to array l making it to wait for certain amount of time while other processes those who get the access are making progress.

Acknowledgments. I would like to convey my gratitude and respect towards my college professor Dr Pintu Pal, Department of Computer Application, Asansol Engineering College for guiding and encouraging me throughout my research work helping to improve this paper and my research experience.
7 Future Scope

Furthermore, while this experiment provided this approach taking inspiration from a real-world application for solving critical section problem in a neat manner, observational studies are required to gain more insight about the utility and sophistication of this algorithm to derive, for instance, better methods for implementing this algorithm and improving the time complexity.

Declarations

Funding
Not Applicable.

Competing interests
Not Applicable.

Ethics approval
Not Applicable.

Consent to participate
Not Applicable.

Consent for publication
Not Applicable.

Author’s Contribution
The author confirms sole responsibility for the following: study conception and design, data collection, analysis and interpretation of results, and manuscript preparation.

References


A modified Conveyor-belt Algorithm to solve the critical section problem


