Simulation of Different Dead lock Scenarios and Avoiding the Deadlocks

Software Requirement Specification (SRS) Document

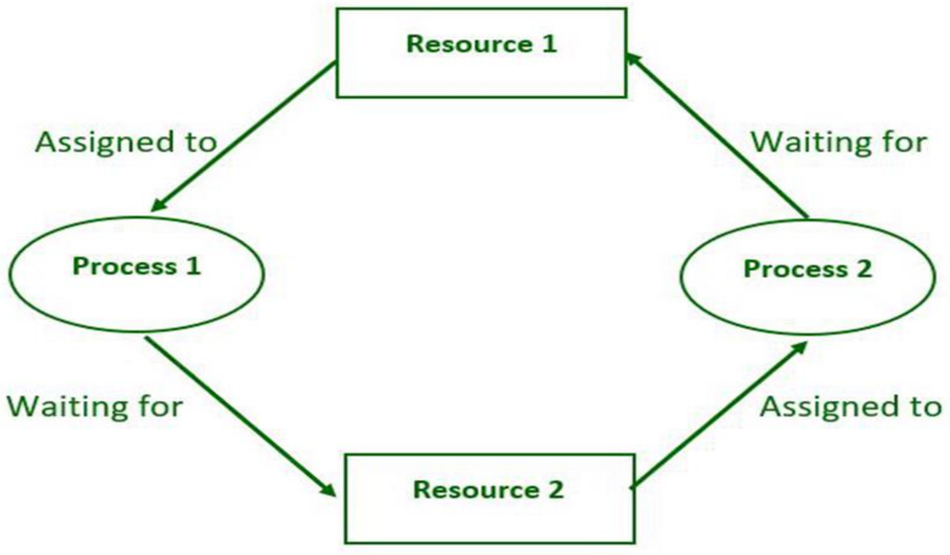
Sprint 2 Implementation

Project Timeline: 02.01.2023 to 14.01.2023

INDEX

1. Introduction: -

Deadlock in operating system is a situation which occurs when a process or thread enters a waiting state because a resource requested is being held by another waiting process, which in turn is waiting for another resource held by another waiting process. In a deadlock state a process is unable to change its state(waiting) indefinitely because the resources requested by it are being used by another waiting process.



1.1 Purpose : **-** The purpose of this document is to show how a deadlock can happens and going through the different ways to prevent the deadlock.

1.2 Intended Audience: - This document is intended to be read by the client.

1.3 Intended Use: -

- Development Team
- Maintenance Team
- Clients.

1.4 Scope: - The scope of the project is to identify the conditions for deadlocks, distinguish the different circumstances that lead to this undesirable state, and identify the methods for detection, prevention, and recovery.

2. Overall Description: -

Deadlock Conditions:

Processes do not run constantly from the time they are created until their termination; they can be identified in three different states: ready, running and blocked. At the ready state, a process is stopped, allowing some other process to run; at the running state, a process is utilizing some resource, and at the blocked state, the process is stopped and will not start running again until something triggers it to restart.

A common undesirable condition is described as deadlock, which is when two processes are at a running state, and have acquired some resources, but need to exchange resources to continue. Both processes are waiting for the other to release a "requested" resource that neither one will ever do; as a result, neither one is making any progress, therefore, reaching a deadlock state.

Deadlock Detection:

Deadlock can arise if the following four conditions hold simultaneously (Necessary Conditions)

Mutual Exclusion*:*

Two or more resources are non-shareable (Only one process can use at a time)

Hold and Wait:

A process is holding at least one resource and waiting for resources.

No Preemption:

A resource cannot be taken from a process unless the process releases the resource.

Circular Wait:

A set of processes are waiting for each other in circular form.

Deadlock Prevention :

We can prevent Deadlock by eliminating any of the above four conditions.

Deadlocks are an undesirable system state and their prevention or detection and recovery, if unavoidable, is necessary. Various methods and approaches exist Surprisingly, there are operating systems that ignore the issue altogether; in those situations, deadlock detection and recovery becomes the primary focus.

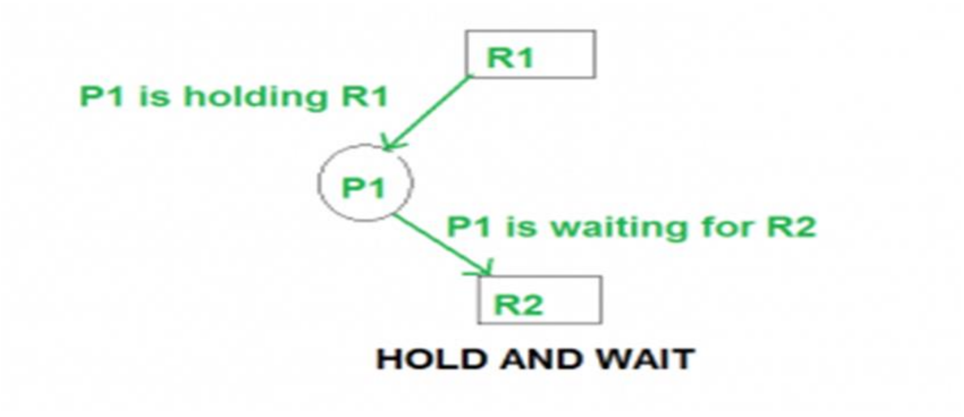
Predicting potential deadlock conditions can be complex, but there are some general conditions that if true, make deadlocks inevitable. Namely, if all the following conditions are present:
a. a mutual exclusion condition
b. no pre-emption condition
c. a hold-and-wait condition
d. a circular-wait condition

Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

Eliminate Hold and wait:

- Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization.
- The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Deadlock Avoidance:

It is very difficult to determine when a system is about to reach a state of deadlock, simply because it is nearly impossible to predict what process will request which resource before it is released by another process.
One attempt has been to identify and classify system states into: safe, unsafe and deadlocked. When the system is at a safe state it means that no deadlock has occurred and the system can still service all pending process requests. When a system is at an unsafe state, it means that some of the pending processes will not be serviced, and the danger of reaching a deadlock state is imminent. An unsafe state does not guarantee a deadlock condition in the immediate future, since the system may never have an offending process request, that could transform the unsafe state to deadlock

Deadlock avoidance can be done with Banker's Algorithm.

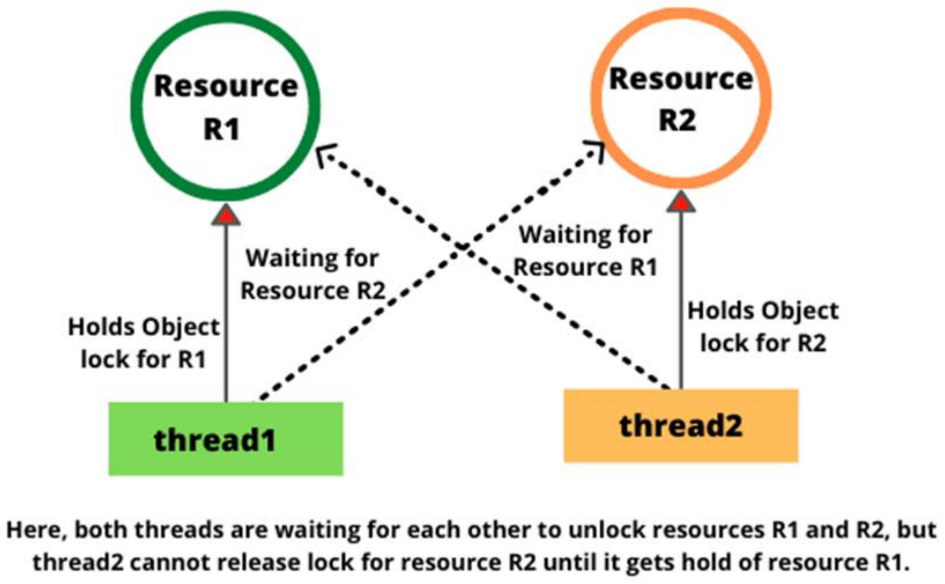
Banker's Algorithm: Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

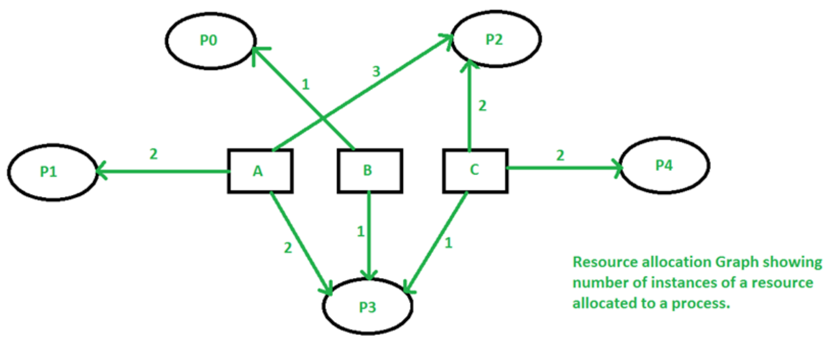
Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

1. If the request made by the process is less than equal to max need to that process.
2. If the request made by the process is less than equal to the freely available resource in the system.


Resource R1
Resource R2
Waiting for Resource R2
Waiting for Resource R1
Holds Object lock for R1
Holds Object lock for R2
thread1
thread2
Here, both threads are waiting for each other to unlock resources R1 and R2, but thread2 cannot release lock for resource R2 until it gets hold of resource R1.


P0
P1
P2
P3
P4
A
B
C
1
3
2
2
2
1
2
1
Resource allocation Graph showing number of instances of a resource allocated to a process.

2.1 Assumptions and Dependency: -

- System should have Ubuntu Linux installed.
- System should have either 4GB or more RAM.
- The service is used preferably on a desktop or laptop.

3. System Features and Requirements: -

3.1 Functions: -

3.1.1 G1_FR01-> The application should display a main menu to select options such as create a deadlock, prevent deadlock etc...

3.1.2 G1_FR02-> The application should display further sub menu options based on selected menu options.

3.1.3 G1_FR03-> The application should validate the menu options at each level. If Any incorrect option or entry by the user should display an error.

3.1.4 G1_FR04-> For simulating deadlock few processes should be created and ran to demo the deadlock scenario

3.1.5 G1_FR05-> A different set of processes should be created and run to display the demo of preventing deadlock

3.1.6 G1_FR06-> A comparison should be presented w.r.t deadlock and non-deadlock scenarios.

3.1.7 G1_FR07-> A set of processes are run to simulate deadlock detection

3.1.8 G1_FR08-> Implement banker's algorithm (or any other) for deadlock detection. This is an optional case.

3.1.9 G1_FR09-> Logging support with various levels such as info, debug, error etc.

3.2 Technical Requirements

3.2.1 G1_TR01-> Process Synchronization

3.2.2 G1_TR03-> Socket Programming in C - TCP

3.2.3 G1_TR04-> Support of statistics

3.3 Non-Functional Requirements

3.3.1 G1_NFR01-> Multi-file multi-directory solution is expected.

3.3.1 G1_NFR02-> makefile to build application.

3.3.1 G1_NFR03-> Use valgrind tool on application executable to detect memory
leak. Final valgrind report to be submitted in “reports”
directory.

3.3.1 G1_NFR04-> Level 0 DFD (context diagram), Level 1 DFD, Flow diagram
and 2 flowcharts showing core functions logic.

3.3.1 G1_NFR05-> Use CUnit to automate unit testing.

3.3.1 G1_NFR06-> HLD, LLD of the system.

3.3.1 G1_NFR07-> RTM, Plan, Presentation

3.3.1 G1_NFR08-> Unit test cases and Integration test cases in UT_IT
document. Both types of test cases i.e. sunny and
rainy should be present in this document.

3.3 System Requirements: -

System Requirements are types of functional requirements. These are features that are required to order for a system to function.

3.3.1. Tools to be used:

- System Programming
- pthread library
- Linux
- C Programming
- C File Handling

- valgrind
- Make

3.3.2. Software Interface:

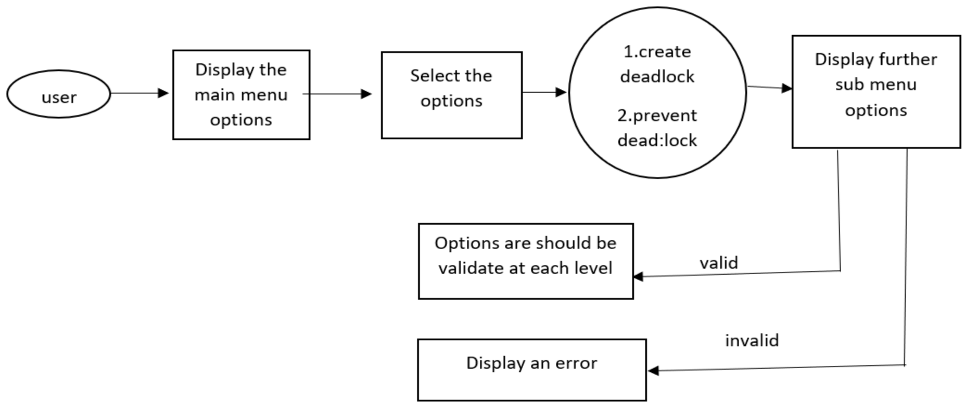
Operating System: Linux OS

3.4 System Features: -

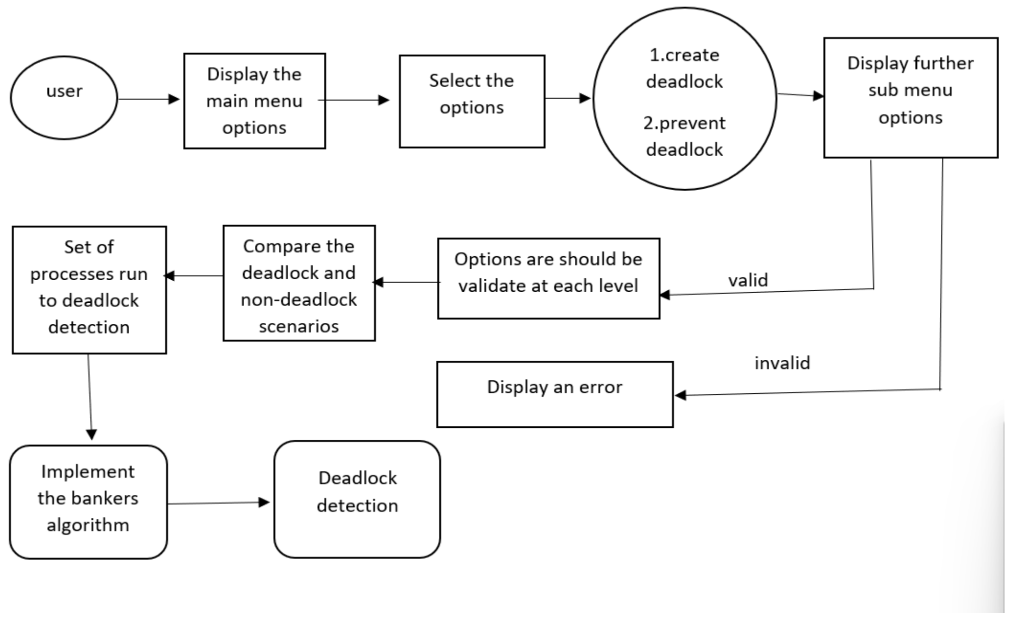
- Supportability: The system is easy to use.
- Design Constraints: The system is built using only C language.
- Reliability and availability**:** The system is available 24/7 that is whenever the user would like to use the system, they can use it up to its functionalities.
- Performance: The system will work on the user's terminal**.**

4. Dataflow Diagram:

4.1 DFD Level-0:


user
Display the main menu options
Select the options
1.create deadlock
2.prevent dead:lock
Display further sub menu options
valid
Options are should be validate at each level
invalid
Display an error

4.1 DFD Level 1 –


user
Display the main menu options
Select the options
1.create deadlock
2.prevent deadlock
Display further sub menu options
valid
invalid
Options are should be validate at each level
Display an error
Compare the deadlock and non-deadlock scenarios
Set of processes run to deadlock detection
Implement the bankers algorithm
Deadlock detection