In this lab, we will implement the deadlock scenario as modelled by the dining philosophers’ problem with three processes. The implementation requires the procedures for the individual processes and also a scheme for detecting deadlocks and recovering from them.

1. Create three concurrent processes $P_0, P_1, P_2$ and three resources $r_{0,1}, r_{1,2}, r_{2,0}$. Resource $r_{i,j}$ is used by both $P_i$ and $P_j$, but not simultaneously.

2. There is a critical section of the code when process $P_i$ uses resources $r_{i,i+1}$ and $r_{i-1,i}$ simultaneously. The arithmetic is modulo 3.

3. It must acquire these resources one after the other by making requests.

4. If it gets one of the resources, it holds on to it and waits for the other.

5. When it gets both resources, it executes for a certain amount of time and then releases both resources one after the other.

6. It continues the rest of its execution.

7. The processes are infinite loops with this sequence.
The outline of what a process must do is illustrated below for process \( P_0 \).

Process \( P_0 \):

\[
\text{repeat}
\begin{align*}
\text{repeat} & \quad \text{request resource } r_{0,1}; \\
\text{until} & \quad \text{granted.}
\end{align*}
\begin{align*}
\text{repeat} & \quad \text{request resource } r_{2,0}; \\
\text{until} & \quad \text{granted.}
\end{align*}
\text{THE ABOVE MUST BE DONE USING semaphores.}
\begin{align*}
\text{loop for} & \quad \text{a random delay time. (Critical Section)} \\
& \quad \text{Release the first resource} \\
& \quad \text{Release the second resource} \\
\text{loop for} & \quad \text{a random delay time. (Remainder Section)}
\end{align*}
\text{until false;}
\]

The programs you write should take into consideration the following:

- The acquiring and release of the common resources must be programmed using a semaphore for each common resource, shared by the two processes using that resource.

- Program a fourth process, which detects if there is a deadlock among the processes and resources, and if so, it breaks the deadlock, by forcibly releasing the resource held by a process it selects at random.

- Since all four programs run in parallel, the deadlock routine checks for deadlocks every time it executes.

- Output via print statements (redirected to a file), the execution sequences, and deadlock occurrence and recovery.