20/20

Advanced Operating Systems (621)
Homework Assignment – 2
Sandipan Dey

13. Process P_0 wants to access a resource A $\Rightarrow$ it sends the 3-tuple $< A,\ 0,\ T_{0A} >$ to all processes (conceptually including itself) asking permission and P_1 wants to access another resource B $\Rightarrow$ it sends $< B,\ 1,\ T_{1B} >$ to all processes (P_0, P_1 if we **restrict ourselves** to the **two-process / two resource system** (P_0, P_1, A, B) and also assuming no message loss).

1. When P_0 receives the request $< B,\ 1,\ T_{1B} >$ it can be in any of the 3 following states:

a) P_0 is not holding *B* and does not want to access *B*. It says OK to P_1 that can acquire *B*.

b) P_0 is holding *B*, it queues the request. Replies OK to P_1 once it's done.

c) P_0 wants to access *B* but is not holding *B*, it queues the request. Compares its own timestamp T_{0B} with T_{1B}, the process with number $\arg\min(T_{0B}, T_{1B})$ wins the access.

2. Similarly when P_1 receives the request $< A,\ 0,\ T_{0A} >$ it can be in any of the 3 following states:

a) P_1 is not holding *A* and does not want to access *A*. It says OK to P_0 that can acquire *A*.

b) P_1 is holding *A*, it queues the request. Replies OK to P_0 once it's done.

c) P_1 wants to access *A* but is not holding *A*, it queues the request. Compares its own timestamp T_{1A} with T_{0A}, the process with number $\arg\min(T_{0A}, T_{1A})$ wins the access.

The following table shows system's action upon receiving the requests $< A,\ 0,\ T_{0A} >$, $< B,\ 1,\ T_{1B} >$. For the simplicity of analysis, we assume that **no further request messages are sent** from any process to any other process.

There can be *3x3 = 9* different (joint) states the system can be in (depending upon how the processes act upon getting the request messages $< A,\ 0,\ T_{0A} >$, $< B,\ 1,\ T_{1B} >$), the different states are shown below :

	P_1 does not want A	P_1 holding A	P_1 is not holding A, wants A
P_0 doesn't want B	P_0 acquires A P_1 acquires B (No Deadlock)	P_1 releases A P_0 acquires A P_1 acquires B (No Deadlock)	$P_{\arg\min(T_{0,A},T_{1,A})}$ acquires A P_1 acquires B (No Deadlock)
P_0 holding B	P_0 releases B P_1 acquires B P_0 acquires A (No Deadlock)	P_1 releases A P_0 acquires A P_0 releases B P_1 acquires B	P_0 releases B P_1 acquires B $P_{\arg\min(T_{0,A},T_{1,A})}$ acquires A (No Deadlock)
P_0 not holding B, wants B	$P_{\arg\min(T_{0,B},T_{1,B})}$ acquires B P_0 acquires A (No Deadlock)	P_1 releases A P_0 acquires A $P_{\arg\min(T_{0,B},T_{1,B})}$ acquires B (No Deadlock)	$P_{\arg\min(T_{0,A},T_{1,A})}$ acquires A $P_{\arg\min(T_{0,B},T_{1,B})}$ acquires B (No Deadlock)

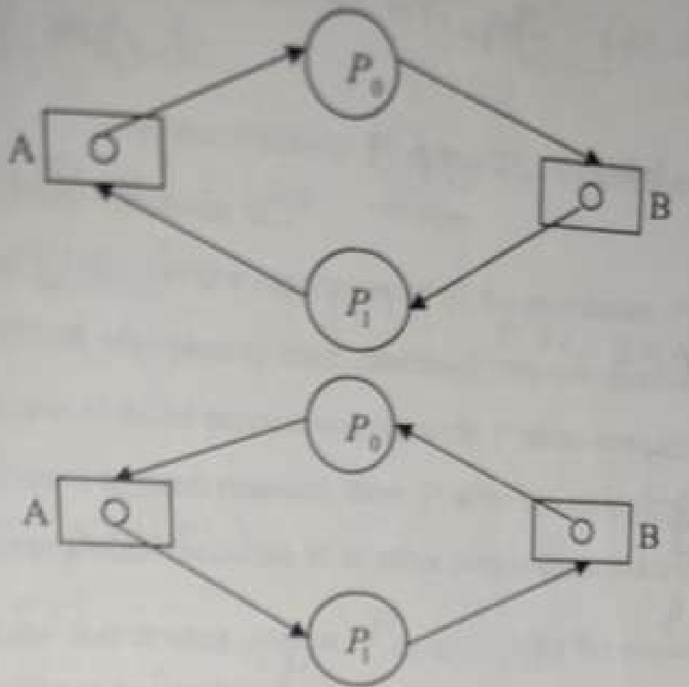
By no further messages assumption we have $P_{\arg\min(T_{0,A},T_{1,A})} = P_0$ and $P_{\arg\min(T_{0,B},T_{1,B})} = P_1$. Hence, all the cases except the one corresponding to the **center of the cell** can never have circular waiting (irrespective of the order in which the processes release / acquire critical sections (resources). But the state corresponding to the cell in the center (where both the processes hold one resource each and request for another resource) **can lead to deadlock**, depending on **whether or not** the processes are **willing** to **leave the resources** they are **currently holding**, before they get access to the other **requested resource**. This leads to two different cases:

Case-1:

If the resources A and B are **independent** $\Rightarrow$ Access of A does not necessitate access of B and vice-versa $\Rightarrow$ a process **never** requires to **hold** them **simultaneously**, i.e., processes **hold resources** (enter critical regions) **strictly sequentially** $\Rightarrow$ a process **already inside a critical region** (already holding a resource) **can't attempt to enter another one** $\Rightarrow$ it is impossible that a process can block while holding a resource that another process wants $\Rightarrow$ deadlock can never occur.

Proof (by contradiction):
Let's assume to the contrary, i.e., assume deadlock can occur even in case of sequential resource access. In this case, by assumption, both of P_0 and P_1 **can hold exactly one of resources** A and B at a given point of time.

But, since there is a deadlock, by **sufficient condition** for the **deadlock**, a **knot** must be there in the resource allocation graph **(RAG)** containing these two processes and two resources. But, the **only two knots possible** containing these 2-process / 2-resource system are shown below:



10/10

RAGs representing all possible knots for the system (P_0 , P_1 , A, B)

But in both of the above RAGs can't happen since both the processes are holding a resource and requesting another one simultaneously, a **contradiction** to **sequential access assumption**.

Thought from another perspective, by imposing the constraint of **sequential resource access**, the necessary conditions **Hold & Wait** and **Circular Wait** are violated, hence deadlock can't occur.

<u>Case-2:</u>

On the contrary, if the processes try to access multiple resources simultaneously and allowed to request for another resource while holding one, then deadlock can happen, e.g., if P_0 is holding resource A and then requests access for resource B, a **deadlock can occur** if other process P_1 tries to acquire them in the reverse order. The Ricart and Agrawala algorithm itself does not contribute to deadlock since each critical region is handled independently of all the others. The above figure shows the 2 cases where **deadlock can occur** when there is a circular waiting.

14. Let's start with a set of processes $P = \{P_1, P_2, P_3, \cdots, P_n\}$.
With the assumption of **uniqueness** of process numbers, let any two distinct processes have different numbers, i.e., $\forall P_i, \forall P_j \in P,\ P_i \neq P_j \Leftrightarrow i \neq j,\ i, j \in \{1,2,\ldots,n\}$.

Now, the set of integers being **totally ordered** under $\leq$, $\forall i, j \in Z^+,\ (i \leq j) \vee (j \leq i)$.

Also, $i \neq j$ and with $(\leq, Z)$ forming a partially ordered set (**POSET**), **without loss of generality** we can assume $i < j$ (by **anti-symmetry**).

Now, both P_i and P_j detect the demise of the current coordinator and hold an election using Bully algorithm simultaneously:

- P_i sends **ELECTION** message to $\forall P_k \in P,\ (i<k)$. Also, P_j sends **ELECTION** message to $\forall P_l \in P,\ (j<l)$.
- Since $i<j$ (by assumption), P_i sends **ELECTION** message to P_j, but P_j **never** sends **ELECTION** message to P_i.
- First let's consider the responses of all the processes P_l with $i<j<l$, both P_i and P_j send **ELECTION** message to them, so (if up) they will respond to both P_i and P_j simultaneously.
  - If one of these processes responds, it takes over and both P_i and P_j's job is done.
  - If none of them respond, then P_j wins the election and becomes the new coordinator. It sends back response to all other processes including P_i, so that they come to know the new coordinator. Hence, $P_{\max(i,j)}$ wins the election and becomes the new coordinator without any ambiguity.

The above is equivalent to running one election by $P_{\max(i,j)}$, since P_k with $\min(i,j) \le k < \max(i,j)$ will never be able to win the election, the node that will be elected will be = $ELECTED(\{P_1, P_2, \cdots, P_n\}) = \max_k \{P_k \mid \max(i,j) \le k \le n \ \wedge P_k \ is\ up\}$.

10/10

$\frac{1}{2}+\frac{2}{2}+\frac{3}{2}+\frac{4}{2}+\cdots+\frac{i}{2}$

$\varphi_i = \lceil \frac{i}{2} \rceil\ 1+t_i$

$\varphi_i - \varphi_{i-1}$

$b_i \le b_{i-1} - t_i + 1$

$\hat{c}_i = c_i + \varphi_i - \varphi_{i-1}$

$n_i \le 2^{n_i - 1}$

$c_i \le \lfloor \frac{i}{2} \rfloor + n_i - 1$

$\varphi_0 = 0$

$\varphi_i \ge \varphi_0$

$\varphi_i \le 2\varphi_{i-1}$

$\sim \varphi_{i-2}$

$\le 2^{i-1}\varphi_1 = 2^{i-1}$

$\varphi_i - \varphi_{i-1}$

$= n_i - n_{i-1}$

$\le n_i - 1$