Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
PPG for synchronization

Process Precedence Graph for synchronization
Recall (Process Precedence Graph)

**PPG (Process Precedence Graph):** Is a directed graph that is used to graphically express the order on which processes or threads are executed with respect to other processes or threads.

(A) → (B) means that A must finish before B can start.
Recall (Process Precedence Graph)

**PPG (Process Precedence Graph):** Is a directed graph that is used to graphically express the order on which processes are executed with respect to other processes or threads.

```
Begin
    A;
    CoBegin;
    B, C, D;
    CoEnd;
    E;
End
```
PPG and CoBegin/CoEnd

**Exercise:** Write CoBegin/CoEnd code for the following PPG:

![PPG Diagram]

The precedence constraints are: Process $P_1$, $P_2$, and $P_3$ should execute in parallel. However, process $P_4$ should execute after $P_2$ and $P_3$ termination, and can execute before, after, or in parallel with $P_1$. 
**Exercise:** Write CoBegin/CoEnd code for the following PPG:

```
CoBegin
  Begin
    CoBegin P_2, P_3 CoEnd
  P_4;
End
P_1;
CoEnd
```
Exercise: Write CoBegin/CoEnd code for the following PPG:

\[ P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \]
PPG and CoBegin/CoEnd

Begin

CoBegin $P_1, P_2$; CoEnd

CoBegin $P_3, P_4$; CoEnd

End

CoBegin

Begin $P_2, P_4$; End

Begin $P_1, P_3$; End

CoEnd
PPG and CoBegin/CoEnd

Begin

CoBegin $P_1$, $P_2$; CoEnd

CoBegin $P_3$, $P_4$; CoEnd

End
PPG and CoBegin/CoEnd

Begin

CoBegin \( P_1, P_2; \) CoEnd  
CoBegin \( P_3, P_4; \) CoEnd  

End

CoBeing/CoEnd cannot represent arbitrary precedence constraints. Additional constraints are added in some situations.
PPG and CoBegin/CoEnd

Another try

CoBegin

\[ P_1 \]
\[ \text{Begin } P_2, P_4; \text{ End} \]

CoEnd

\[ P_3; \]
PPG and CoBegin/CoEnd

CoBegin

\[ P_1 \]

\textbf{Begin} \( P_2, P_4; \textbf{End} \)

CoEnd

\[ P_3; \]

Fails if \( P_3 \) and \( P_4 \) need to cooperate
PPG and synchronization

Translate the following precedence graph into a code using semaphores:
Brute force. Create as many semaphores as arrows:

\[ s_{13}, s_{23}, s_{24}, s_{35}, s_{45} : \text{Semaphore} := 0; \]
### PPG and synchronization

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>code(_1);</td>
<td>code(_2);</td>
<td>P((s_13));</td>
<td>P((s_24));</td>
<td>P((s_45));</td>
</tr>
<tr>
<td>V((s_13));</td>
<td>V((s_23));</td>
<td>P((s_23));</td>
<td>code(_4);</td>
<td>P((s_35));</td>
</tr>
<tr>
<td>V((s_24));</td>
<td>code(_3);</td>
<td>V((s_45));</td>
<td>code(_5);</td>
<td></td>
</tr>
<tr>
<td>V((s_35));</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(s\_13, s\_23, s\_24, s\_35, s\_45\): Semaphore := 0; *(5 semaphores)*
**Optimization.** Let us reduce the number of semaphores by renaming $s_{xy}$ with $s_x$:

$s_1, s_2, s_3, s_4$: Semaphore := 0; (4 semaphores)
### PPG and synchronization

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>code$_1$;</td>
<td>code$_2$;</td>
<td>P(s$_1$);</td>
<td>P(s$_2$);</td>
<td>P(s$_4$);</td>
</tr>
<tr>
<td>V(s$_1$);</td>
<td>V(s$_2$);</td>
<td>P(s$_2$);</td>
<td>code$_4$;</td>
<td>P(s$_3$);</td>
</tr>
<tr>
<td>V(s$_2$);</td>
<td>code$_3$;</td>
<td>V(s$_4$);</td>
<td>code$_5$;</td>
<td></td>
</tr>
<tr>
<td>V(s$_3$);</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$s_1, s_2, s_3, s_4$: Semaphore := 0; (4 semaphores)
Classical Synchronization Problems
Readers & Writers problem
Readers & Writers problem

This problem consists of: (1) A set of reader threads $R = \{r_1, \ldots, r_n\}$ & (2) a set of writer threads $W = \{w_1, \ldots, w_m\}$. Both readers and writers share a data structure $D$ (e.g., file, database, variable, ...) such that:

- Multiple readers $R' \subseteq R$ can read at the same time.
- Only one writer $w_i \in W$ can write at a time.
- If $\exists r \in R$ that is reading, new writers have to wait (no preemption).
- If $\exists w \in W$ that is writing, new writers and readers have to wait.
- If $\exists r \in R$ that is reading, new readers can start reading.

There exist three solutions: Readers have priority, Writers have priority, and starvation free solution.
Readers & Writers problem ($| R | = 1$, $| W | = 1$)

One reader $R = \{ r \}$ and one writer $W = \{ w \}$.

Semaphore $m = 1$;

\begin{align*}
\text{r()} & \\
\{ & \\
\text{while(1)} & \\
\{ & \\
\text{acquire(m);} & \\
\text{read();} & \\
\text{release(m);} & \\
\} & \\
\} & \\
\text{w()} & \\
\{ & \\
\text{while(1)} & \\
\{ & \\
\text{acquire(m);} & \\
\text{write();} & \\
\text{release(m);} & \\
\} & \\
\} & \\
\end{align*}
Readers & Writers problem (\(| R | \geq | W | = m \geq 1\))

Multiple readers \( R = \{r_1, \ldots, r_n\} \) and multiple writers \( W = \{w_1, \ldots, w_m\} \).

1. Readers have priority. Let us discuss the solution where the Readers have priority.

i.e., when a new reader arrives and a writer is already waiting for previous readers to terminate, then the new reader can start reading with the other readers without waiting

This solution is not a starvation-free solution.

i.e., If there is a steady stream of readers, writers may not get the chance to write.
Readers & Writers problem \((\mid R \mid \geq \mid W \mid = m \geq 1)\)

Multiple readers \(R = \{r_1, \ldots, r_n\}\) and multiple writers \(W = \{w_1, \ldots, w_m\}\).

2. **Writers have priority.** This will be Lab 3.

i.e., when a new reader arrives and a writer is already waiting for previous readers to terminate, then the new reader **cannot** start reading with the other readers but has to wait.

Also

When there is waiting readers and writers during a writing, writers have the priority to be waken up over the readers at the end of the writing.

This solution is not a starvation-free solution.

i.e., If there is a steady stream of writers, readers may not get the chance to read.
Readers & Writers problem \(| R | \geq | W | = m \geq 1\)

Multiple readers \( R = \{r_1, \ldots, r_n\} \) and multiple writers \( W = \{w_1, \ldots, w_m\} \).

3. **Starvation-free.** Let us briefly discuss this solution (see course reader pp31-32).

   i.e., Readers yield for writers and writers yield for readers

   Basically

   When a reader shows up, it cannot start if a writer is waiting

   and

   When writers are waiting, one writer can start after the last current reader finishes

This solution is starvation-free.

   i.e., Busy system: ..., one writer, batch of readers, one writer, ...
End.