Busy Waiting Algorithms

Solution to Critical-Section Problem

1. Mutual Exclusion - If process P_i is executing in its critical section, then no other processes can be executing in their critical sections
2. Progress - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. Bounded Waiting - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted
   - Assume that each process executes at a nonzero speed
   - No assumption concerning relative speed of the N processes

Lock Variables: Software Solution

- Before entering a critical section a process should know if any other is already in the critical section or not
- Consider having a FLAG (also called lock)
- FLAG = TRUE
  - No process is in the critical section
- FLAG = FALSE
  - A process is in the critical section

```
// wait while someone else is in the
// critical region
1. while (FLAG == FALSE);
// stop others from entering critical region
2. FLAG = FALSE;
3. critical_section();
// after critical section let others enter
//the critical region
4. FLAG = TRUE;
5. noncritical_section();
```

FLAG = FALSE

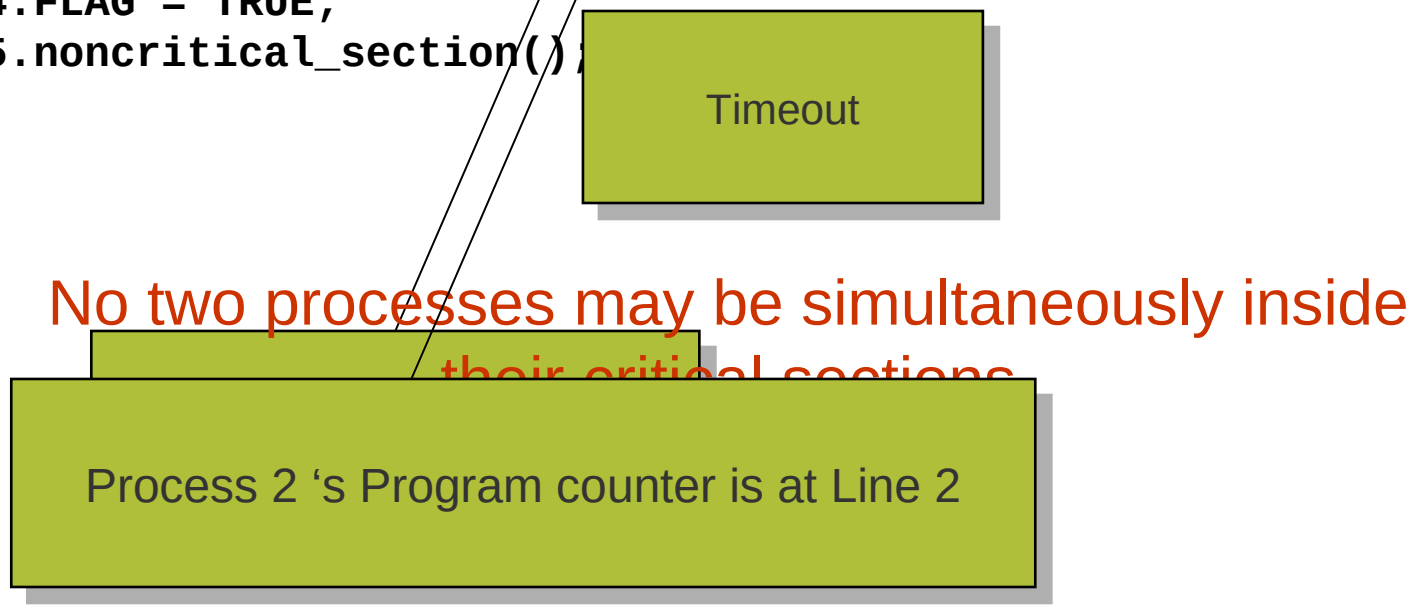
Lock Variables

Process 1

```
1.while (FLAG == FALSE);
2.FLAG = FALSE;
3.critical_section();
4.FLAG = TRUE;
5.noncritical_section();
```

Process 2

```
1.while (FLAG == FALSE);
2.FLAG = FALSE;
3.critical_section();
```



Process 1 forgot that it was Process 2’s turn

Solution: Strict Alternation

- We need to remember "Who's turn it is?"
- If its Process 1's turn then Process 2 should wait
- If its Process 2's turn then Process 1 should wait

Process 1

```
while(TRUE)
{
  // wait for turn
  while (turn != 1);
  critical_section();
  turn = 2;
  noncritical_section();
}
```

Process 2

```
while(TRUE)
{
  // wait for turn
  while (turn != 2);
  critical_section();
  turn = 1;
  noncritical_section();
}
```

Turn = 1

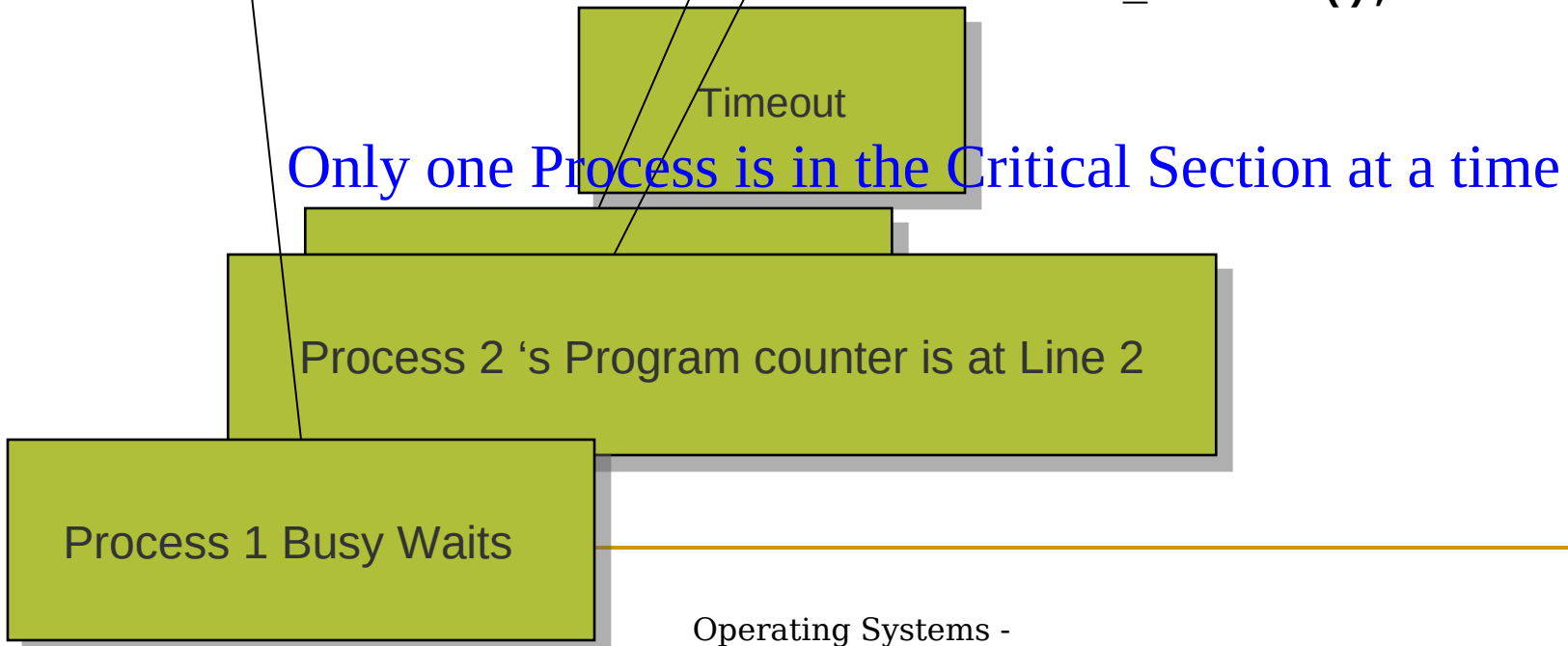
Strict Alternation

Process 1

```
While(1)
1.while (Turn != 1);

2.critical_section();

3.Turn = 2;
4.noncritical_section();
```

Process 2

```
While(1)
1.while (Turn != 2);

2.critical_section();

3.Turn = 1;
4.noncritical_section();
```



Only one Process is in the Critical Section at a time

Strict Alternation

Process 1

```
while(TRUE)
{
  // wait
  while (turn != 1);
  critical_section();
  turn = 2;
  noncritical_section();
}
```

Process 1

```
while(TRUE)
{
  // wait
  while (turn != 2);
  critical_section();
  turn = 1;
  noncritical_section();
}
```

- **Can you see a problem with this?**
- **Hint : What if one process is a lot faster than the other**

turn = 1

Strict Alternation

Process 1

```
while(TRUE)
{
  // wait
  while (turn != 1);
  critical_section();
  turn = 2;
  noncritical_section();
}
```

Process 2

```
while(TRUE)
{
  // wait
  while (turn != 2);
  critical_section();
  turn = 1;
  noncritical_section();
}
```

- Process 1
  - Runs
  - Enters its critical section
  - Exits; setting turn to 2.
- Process 1 is now in its non-critical section.
- Assume this non-critical procedure takes a long time.
- Process 2, which is a much faster process, now runs
- Once it has left its critical section, sets turn to 1.
- Process 2 executes its non-critical section very quickly and returns to the top of the procedure.

turn = 1

Strict Alternation

Process 1

```
while(TRUE)
{
  // wait
  while (turn != 1);
  critical_section();
  turn = 2;
  noncritical_section();
}
```

Process 2

```
while(TRUE)
{
  // wait
  while (turn != 2);
  critical_section();
  turn = 1;
  noncritical_section();
}
```

- Process 1 is in its non-critical section
- Process 2 is waiting for turn to be set to zero
- In fact, there is no reason why process 2 cannot enter its critical region as process 1 is not in its critical region.

Strict Alternation

- What we have is a violation of one of the conditions that we listed above

No process running outside its critical section may block other processes

Progress - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely

- This algorithm requires that the processes ***strictly alternate*** in entering the critical section
- Taking turns is not a good idea if one of the processes is ***slower***.

Reason

- Although it was Process 1's **turn**
- But Process 1 was not **interested**.
- Solution:
  - We also need to remember
    - **"Whether it is interested or not?"**

Algorithm 2

- Replace
  - **`int turn;`**
- With
  - **`bool Interested[2];`**
- **`Interested[0] = FALSE`**
  - Process 0 is not interested
- **`Interested[0] = TRUE`**
  - Process 0 is interested
- **`Interested[1] = FALSE`**
  - Process 1 is not interested
- **`Interested[1] = TRUE`**
  - Process 1 is interested

Algorithm 2

Process 0

```
while(TRUE)
{
  interested[0] = TRUE;
  // wait for turn
  while(interested[1]!=FALSE);
  critical_section();
  interested[0] = FALSE;
  noncritical_section();
}
```

Process 1

```
while(TRUE)
{
  interested[1] = TRUE;
  // wait for turn
  while(interested[0]!=FALSE);
  critical_section();
  interested[1] = FALSE;
  noncritical_section();
}
```

Algorithm 2

Process 0

```
while(TRUE)
{
  interested[0] = TRUE;
  while(interested[1]!=
```

Process 1

```
while(TRUE)
{
  erested[1] = TRUE;
  le(interested[0]!=FALSE);
```

DEADLOCK

Peterson's Solution

Combine the previous two algorithms:

```
int turn;
bool interested[2];
```

- **Interested[0] = FALSE**
  - Process 0 is not interested
- **Interested[0] = TRUE**
  - Process 0 is interested
- **Interested[1] = FALSE**
  - Process 1 is not interested
- **Interested[1] = TRUE**
  - Process 1 is interested

Peterson's Solution

Process 0

```
while(TRUE)
{ interested[0] = TRUE;
  turn = 0;
  // wait
  while(interested[1]==TRUE && turn == 0 );
  critical_section();
  interested[0] = FALSE;
  noncritical_section();
```

F && T

Timeout

Process 1

```
while(TRUE)
{
  interested[1] = TRUE;
  turn = 1;
  // wait
  while(interested[0]==TRUE && turn == 1 );
  critical_section();
  interested[1] = FALSE;
  noncritical_section();
  }
```

F && T

Peterson's Solution

Process 0

```
while(TRUE)
{ interested[0] = TRUE;
  turn = 0;
    // wait
  while(interested[1]==TRUE && turn == 0 );
  critical_section();
  interested[0] = FALSE;
  noncritical_section();
```

T && F

Timeout

Process 1

```
while(TRUE)
{
  interested[1] = TRUE;
  turn = 1;
  // wait
  while(interested[0]==TRUE && turn == 1 );
  critical_section();
  interested[1] = FALSE;
  noncritical_section();
  }
```

F && T

Can not be TRUE at the same time.
Thus used to break tie