The road to Dekker's algorithm

First attempt

```
/* PROCESS 0 /*                    /* PROCESS 1 */
.                                  .
.                                  .
while (turn != 0)                  while (turn != 1)
   /* do nothing */ ;                 /* do nothing */;
/* critical section*/;              /* critical section*/;
turn = 1;                          turn = 0;
.                                  .
```

- This solution guarantees mutual exclusion.
- Drawbacks:
  - Processes must strictly alternate in their use of their critical section
  - If one process fails, the other process is permanently blocked
  - Busy waiting: the thwarted process can do nothing productive until it gets permission to enter its critical section

Second attempt

```
/* PROCESS 0 */                    /* PROCESS 1 */

.                                  .
.                                  .
while (flag[1])                    while (flag[0])
   /* do nothing */;                  /* do nothing */;
flag[0] = true;                    flag[1] = true;
/*critical section*/;              /* critical section*/;
flag[0] = false;                   flag[1] = false;
.                                  .
```

- This solution does not guarantee mutual exclusion. Consider the following sequence:
  - P0 executes the while statement and finds flag[1] set to false.
  - P1 executes the while statement and finds flag[0] set to false
  - P0 sets flag[0] to true and enter its critical section
  - P1 sets flag[1] to true and enter its critical section

Third attempt

```
/* PROCESS 0 */
.
.
flag[0] = true;
while (flag[1])
   /* do nothing */;
/* critical section*/;
flag[0] = false;
.
```

```
/* PROCESS 1 */
.
.
flag[1] = true;
while (flag[0])
   /* do nothing */;
/* critical section*/;
flag[1] = false;
.
```

- Mutual exclusion is guaranteed again.
- Drawbacks:
  - If one process fails inside its critical section, the other process is blocked
  - If both processes set their flags to true before either has executed the while statement, then each thinks that the other has entered its critical section, causing a deadlock.

Fourth attempt

```
/* PROCESS 0 */
.
.
flag[0] = true;
while (flag[1])
{
  flag[0] = false;
  /*delay */;
  flag[0] = true;
}
/*critical section*/;
flag[0] = false;
.
```

```
/* PROCESS 1 */
.
.
flag[1] = true;
while (flag[0])
{
   flag[1] = false;
   /*delay */;
   flag[1] = true;
}
/* critical section*/;
flag[1] = false;
.
```

- Consider the following sequence of events:
  - P0 sets flag[0] to true.
  - P1 sets flag[1] to true.
  - P0 checks flag[1].
  - P1 checks flag[0].
  - P0 sets flag[0] to false.
  - P1 sets flag[1] to false.
  - P0 sets flag[0] to true.
  - P1 sets flag[1] to true.
- This sequence could be extended indefinitely, and neither process could enter its critical section.