6G7Z1004 Advanced Computer Networks and Operating Systems
(Lab 5: Concurrency: threads & synchronization)
This lab was originally designed by Prof. Rene Doursat, MMU, UK

In this lab session, we will continue looking at the race conditions problem and perform deep analysis of one of the proposed solutions to ensure mutual exclusion from the "Critical Section" of a program.

The road to Dekker's algorithm

Dijkstra reported an algorithm for mutual exclusion for two processes, designed by the Dutch mathematician Dekker. Following Dijkstra, we develop the solution in stages here. This approach has the advantage of illustrating many of the common bugs encountered in developing concurrent programs.

a) Two processes run concurrently and share an integer, `turn`. What does this implementation achieve, but what are its drawbacks?

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

b) In fact, each process should have its own key to the CR so that if one fails, the other can still access its CR. Therefore, they now share a 2-boolean array, `flag`. But why is this second attempt wrong?

```
/* 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;
.                                  .
```

c) Perhaps we can fix this problem with a simple interchange of two statements. What is this third attempt achieving, yet why is it still flawed?

```
/* 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;
.
```

d) Let's make processes more "deferential": each process now sets its flag to indicate its desire to enter its CR but is prepared to reset the flag to defer to the other process. This is close to a correct solution but is still flawed, why?

```
/* 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;
.
```