|  |
| --- |
| bool flag[2] = {false, false}; *// οι μεταβλητές flag[0], flag[1] αρχικοποιούνται σε false*int turn; |
| P0: flag[0] = true;P0\_gate: turn = 1; **while** (flag[1]== true && turn == 1) { *// busy wait* } *// critical section* ... *// end of critical section* flag[0] = false; | P1: flag[1] = true;P1\_gate: turn = 0; **while** (flag[0] == true && turn == 0) { *// busy wait* } *// critical section* ... *// end of critical section* flag[1] = false; |

**Αλγόριθμος Peterson**

για την επίτευξη αμοιβαίου αποκλεισμού

Όπου

Οι flag[0] και flag[1] εκφράζουν την πρόθεση των P0, P1 αντίστοιχα να εισέλθουν στην κρίσιμη περιοχή. Η κάθε διεργασία, αφού εκφράσει την πρόθεσή της για είσοδο στην κρίσιμη περιοχή, παραχωρεί προτεραιότητα στην άλλη διεργασία μέσω της turn.

Η είσοδος στην κρίσιμη περιοχή παραχωρείται στην P0 αν η P1 δεν επιθυμεί να εισέλθει στην κρίσιμη περιοχή της ή η P1 έχει δώσει προτεραιότητα στην P0 θέτοντας την turn σε 0.

Η λύση Peterson πετυχαίνει:

1. **Αμοιβαίο αποκλεισμό**
2. **Πρόοδο** (όχι αδιέξοδο / deadlock), δηλαδή σε εύλογο χρονικό διάστημα κάποια άλλη ενδιαφερόμενη διεργασία (και όχι αυτή που μόλις άφησε την κρίσιμη περιοχή) θα εισέλθει στην κρίσιμη περιοχή. Μόνο οι ενδιαφερόμενες διεργασίες ανταγωνίζονται για την είσοδό τους στην κρίσιμη περιοχή – όχι οι διεργασίες που εκτελούν «ξένες» εντολές μη έχοντας δηλώσει πρόθεση εισαγωγής στην κρίσιμη περιοχή τους.
3. **Περιορισμένη αναμονή** (δίκαιη λύση) δηλαδή μια διεργασία με δηλωμένη πρόθεση θα χάσει από τις ανταγωνίστριές της λιγότερες φορές από ένα πάνω όριο. Το πάνω όριο είναι συνάρτηση του πλήθους των παράλληλων διεργασιών. Στη λύση Peterson η αναμονή είναι το πολύ ένας γύρος.