Διάλεξη 10: Αλγόριθμοι Αμοιβαίου Αποκλεισμού σε περιβάλλον ανταλλαγής μηνυμάτων

ΕΠΛ 432: Κατανεμημένοι Αλγόριθμοι

Τι θα δούμε σήμερα

- Αλγόριθμος Χρήση Συντονιστή
- Αλγόριθμος του Lamport
- Αλγόριθμος LeLann: τοπολογία δακτυλίου

ΑΑ με Ανταλλαγή Μηνυμάτων

- Συνθήκη Ασφαλείας
  - Αμοιβαίος Αποκλεισμός: Μόνο μια διεργασία κάθε στιγμή στο κρίσιμο τμήμα
- Συνθήκες Ζωτικότητας
  - Αποφυγή Αδιεξόδου
  - Αποφυγή Παρατεταμένης Στέρησης
- Πολυπλοκότητα
  - Αριθμός μηνυμάτων για χρήση (είσοδο και έξοδο) του ΚΤ

Αλγόριθμος με Συντονιστή

- Ιδέα: Προσομοίωση του πως επιτυγχάνουμε αμοιβαίο αποκλεισμό σε συγκεντρωτικά συστήματα

- Εκλέγουμε ένα επεξεργαστή συντονιστή (πρόεδρο)
  - Ελέγχει την πρόσβαση τον άλλων επεξεργαστών στο κρίσιμο τμήμα

- Επεξεργαστές μη συντονιστές
  - Απευθύνονται στον συντονιστή για να μπορέσουν να μπουν στο κρίσιμο τμήμα
  - Μπορούν να χρησιμοποιήσουν το κρίσιμο τμήμα αφότου πάρουν την άδεια του συντονιστή

Αλγόριθμος Συντονιστή

Assume that p_c is the coordinator

Code for entry section at non-coordinator p_i :

```
Send <request> to p_c
wait to receive <reply> from p_c
enter the critical source
```

Code for exit section at non-coordinator p_i :

```
send <release> to p_c
```

Αλγόριθμος Συντονιστή

Assume that p_c is the coordinator

Upon receipt of <request> at p_c from p_i:

```
add p_i in the queue
if (!waiting)
    send <reply> to p_i
    waiting = TRUE
```

Upon receipt of <release> at p_c from p_i:

```
if (!empty(queue))
    send <reply> to next in queue
else
    waiting = FALSE
```

Ανάλυση Αλγορίθμου Συντονιστή

- **Συνθήκη Ασφάλειας (Αμοιβαίος Αποκλεισμός)**: **Ισχύει**
  - Ο συντονιστής επιτρέπει μόνο σε μια διεργασία να βρίσκεται στο κρίσιμο τμήμα ανα πάσα στιγμή

- Συνθήκες Ζωτικότητας
  - **Αποφυγή Αδιεξόδου**: διασφαλίζεται από τον συντονιστή
  - **Αποφυγή Παρατεταμένης Στέρησης**: διασφαλίζεται από την πολιτική FIFO του συντονιστή

Ανάλυση Αλγορίθμου Συντονιστή

- Πολυπλοκότητα: 3 μηνύματα
  - 2 μηνύματα για είσοδο στο ΚΤ
    - <request>, <reply>
  - 1 μήνυμα για έξοδο από το ΚΤ
    - <release>

- **Μειονεκτήματα:**
  - Σφάλμα συντονιστή καταρρέει το σύστημα
  - Θεωρεί αξιόπιστη ανταλλαγή μυνημάτων

Αλγόριθμος του Lamport

- Ιδέα: Επικοινώνησε με τους άλλους επεξεργαστές για να τους ενημερώσεις για την επιθυμία σου να χρησιμοποιήσεις το κρίσιμο τμήμα
  - Αν η αίτηση σου στάλθηκε «πριν» από άλλες αιτήσεις τότε μπορείς να χρησιμοποιήσεις το κρίσιμο τμήμα
  - Αλλιώς περιμένεις την σειρά σου

- Πως ορίζουμε την σειρά των γεγονότων;
  - Χρονοσφραγίδες Lamport

- Υποθέσεις
  - Κάθε επεξεργαστής διαθέτει μια ουρά για να κρατά τις αιτήσεις
  - Τα κανάλια προσφέρουν FIFO εγγυήσεις

Χρονοσφραγίδες Lamport

- Λογικός συγχρονισμός ρολογιών
  - Μερική ταξινόμηση γεγονότων
  - Προσπαθεί να εξηγήσει την σχέση «πριν από»

- Χρονοσφραγίδα: ένας ακέραιος αριθμός για κάθε επεξεργαστή στο σύστημα.

- Αλγόριθμος Χρονοσφραγίδας στον p_i:
  - Πριν από κάθε υπολογιστικό βήμα αύξησε την χρονοσφραγίδα
  - Συμπεριέλαβε την χρονοσφραγίδα σου σε κάθε μήνυμα που στέλνεις
  - Αν λάβεις μήνυμα με μεγαλύτερη χρονοσφραγίδα ανανέωσε θέσε την χρονοσφραγίδα σου σε μεγαλύτερη τιμή από αυτή του μηνύματος πριν επεξεργαστείς το μήνυμα

Αλγόριθμος Lamport

Entry code at process p_i:

```
send <request, (t_i, i)> to all
add <p_i, (t_i, i)> in your local request_queue_i
                    according to (t_i, i)
wait for <reply, (t_j, j)> from all
          s.t. (t_j, j) > (t_i, i)
wait for <p_i, (t_i, i)> to be head of request_queue_i
enter CS
```

Exit code at p_i:

```
remove <p_i, (t_i, i)> from request_queue_i
send <release, (t_i, i)> to all
```

Αλγόριθμος Lamport

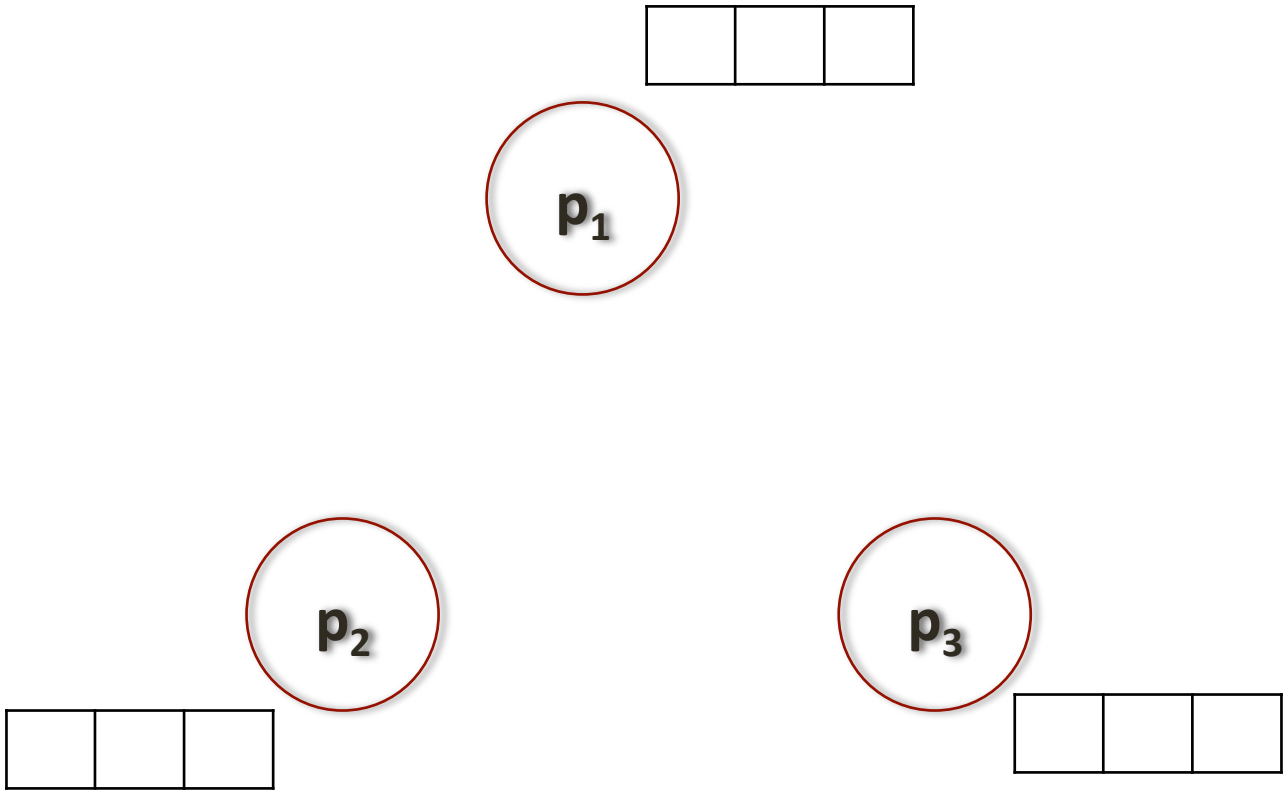
Upon receiving <request, (t_i, i)> at p_j:

```
add <p_i, (t_i, i)> in your local request_queue_j
                    according to (t_i, i)
send <reply, (t_j, j)> to p_i
```

Upon receiving <release, (t_i, i)> at p_j:

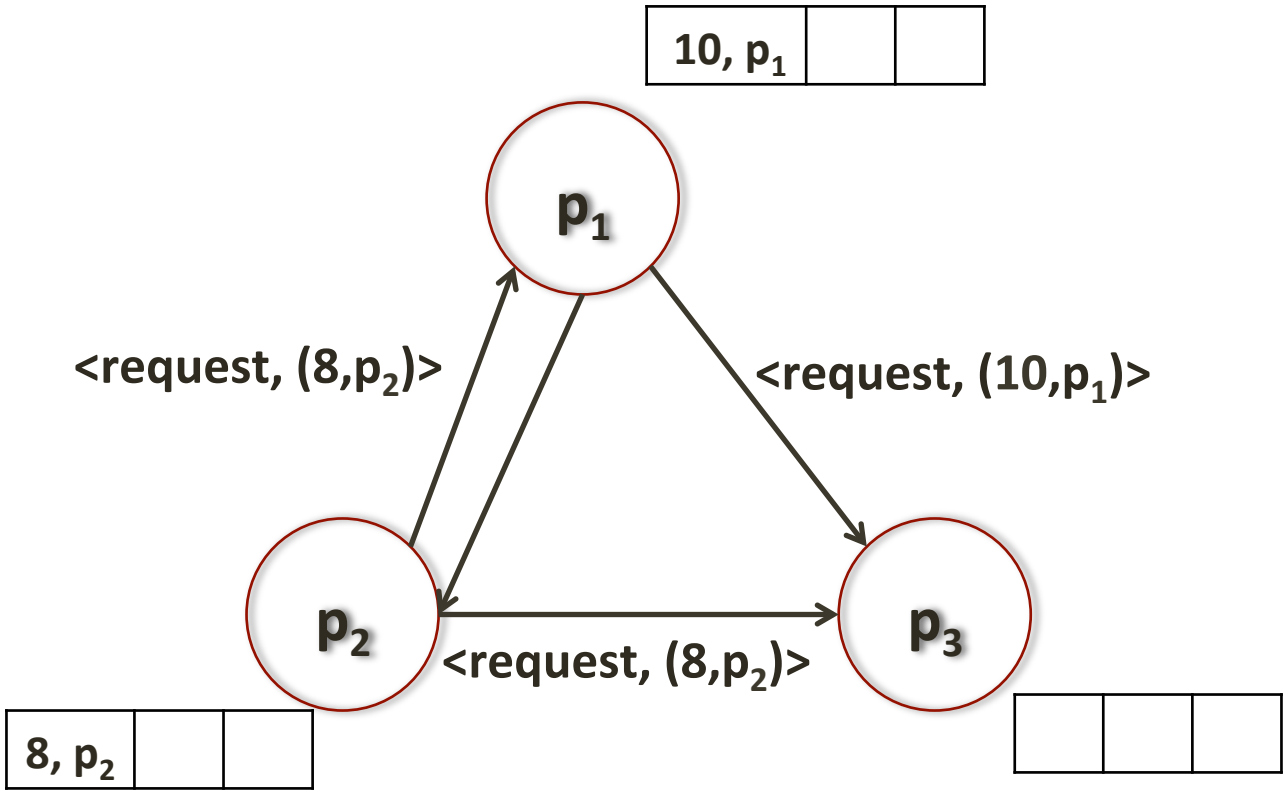
```
remove <p_i, (t_i, i)> from request_queue_j
```

Παράδειγμα Εκτέλεσης



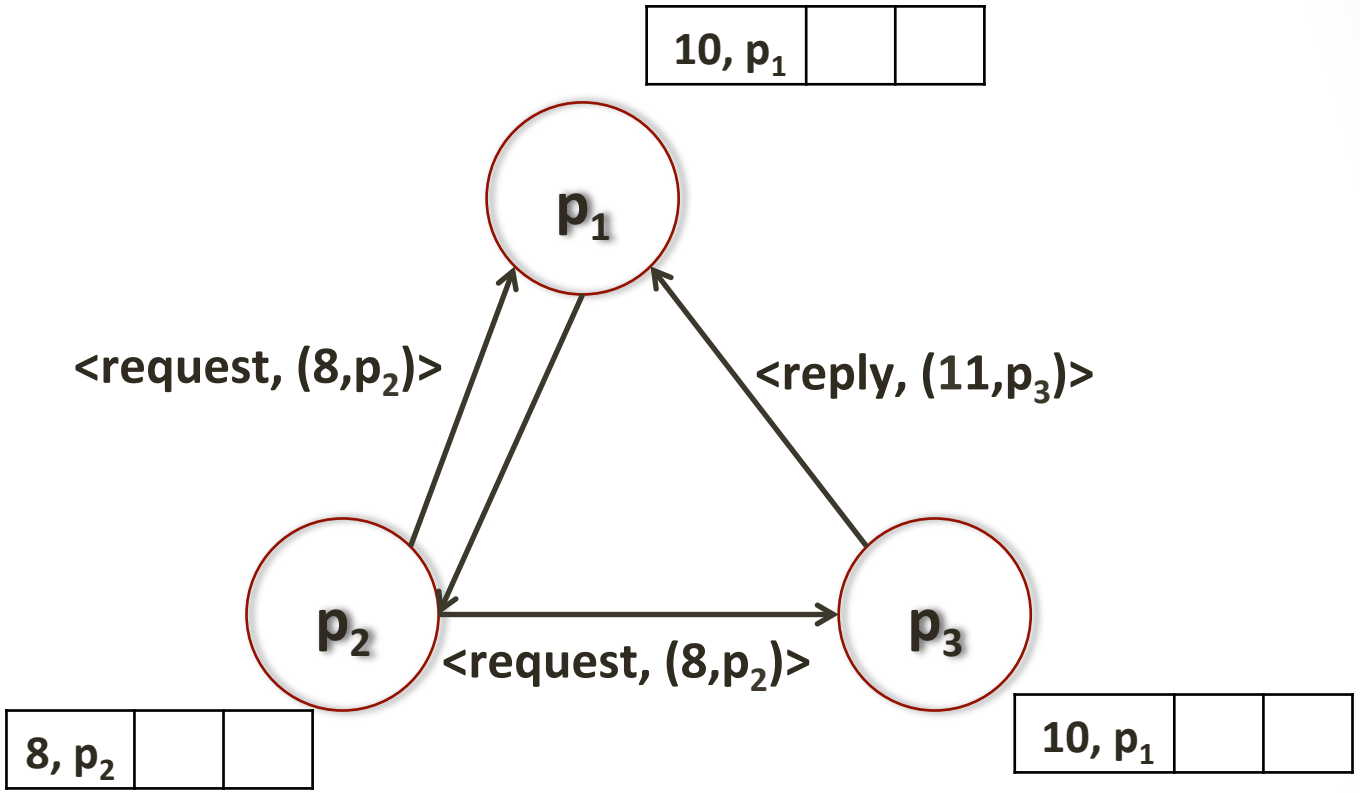
- **Έστω οι επεξεργαστές p_1 και p_2 θέλουν να εισέλθουν στο ΚΤ....**

Παράδειγμα Εκτέλεσης



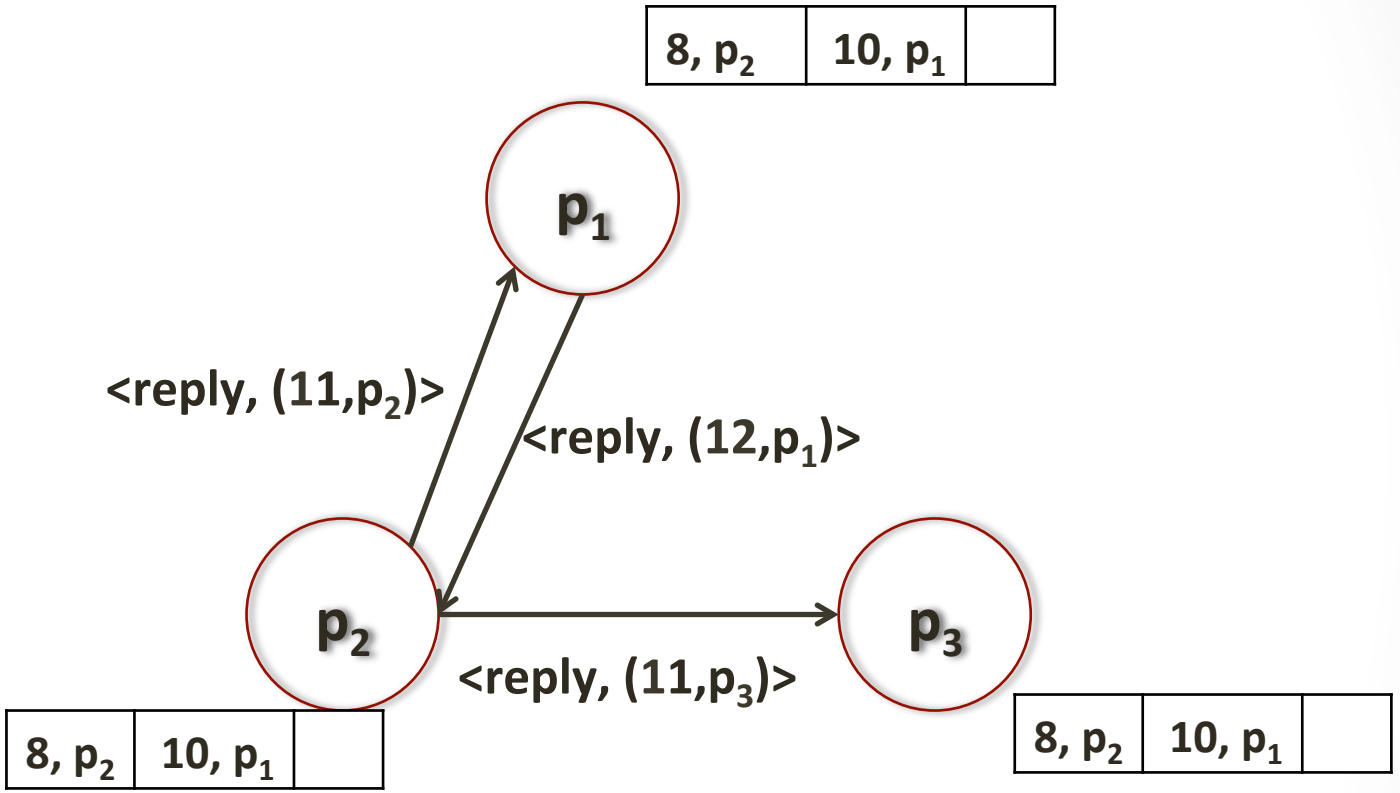
- **Βάζουν την αίτηση τους στην ουρά και στέλνουν μήνυμα σε όλους με την χρονοσφραγίδα τους....**

Παράδειγμα Εκτέλεσης



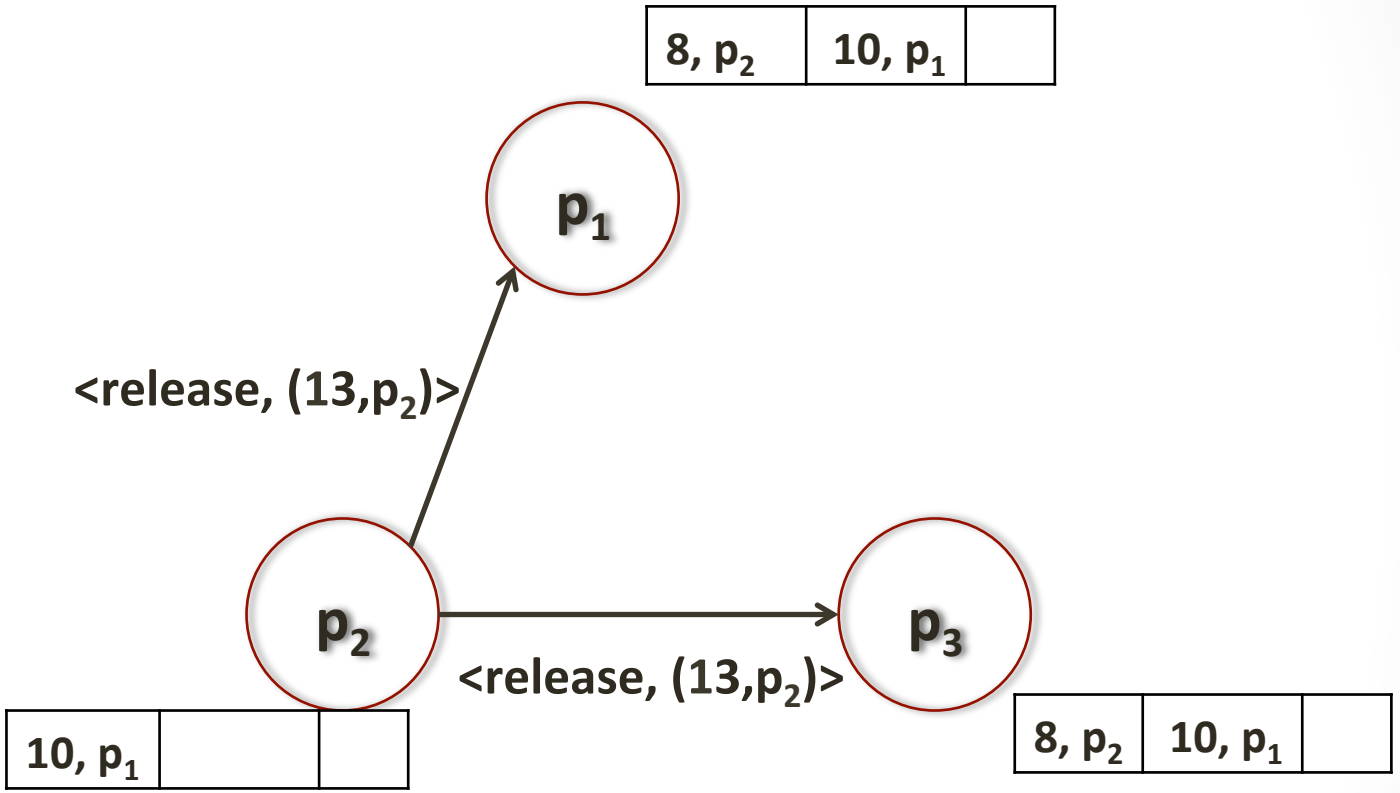
- Ο p_3 λαμβάνει πρώτα την αίτηση από τον p_1 και απαντά.
- Ο p_1 δεν μπορεί ακόμα να μπεί στο ΚΤ αφού δεν έλαβε απάντηση από τον p_2

Παράδειγμα Εκτέλεσης



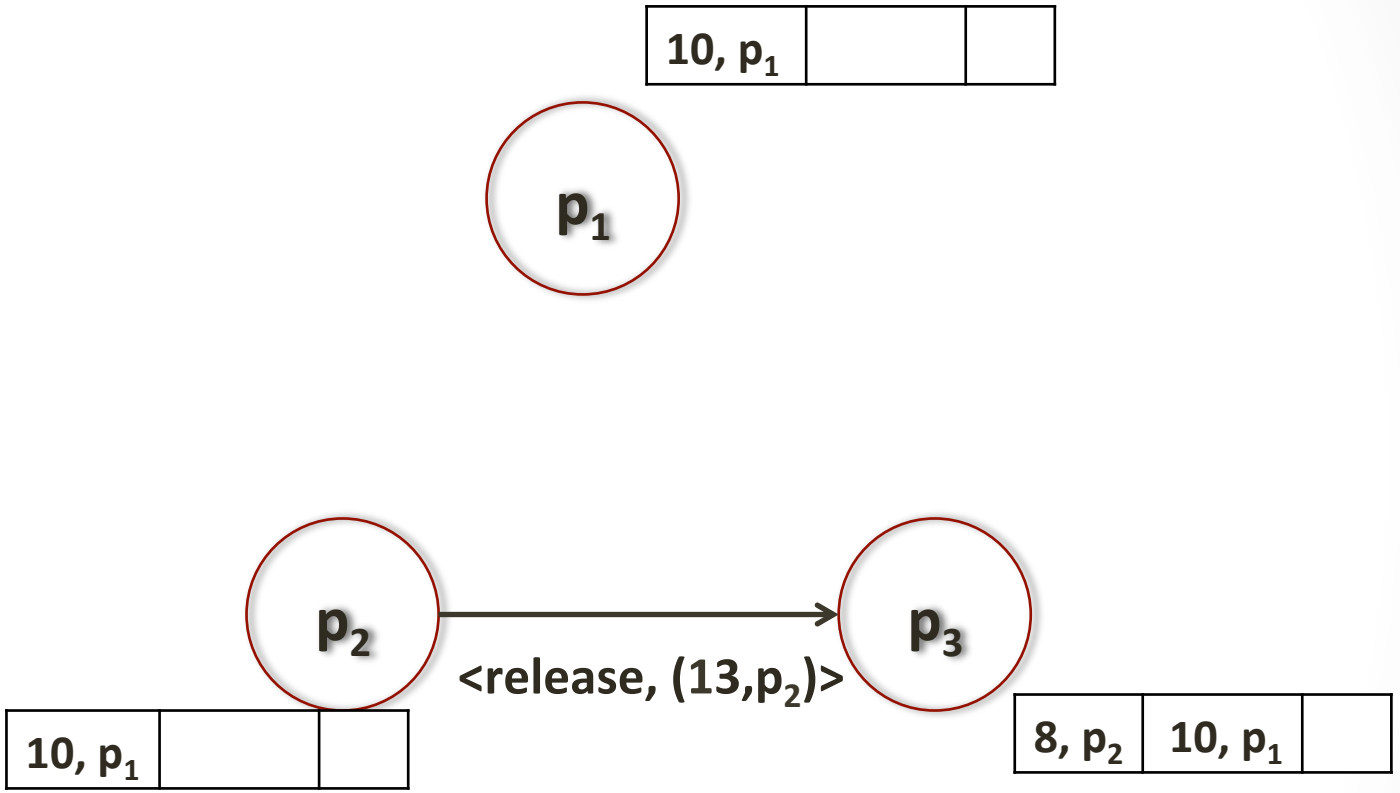
- Τα μηνύματα του p_2 λαμβάνονται από τους p_1 και p_3...
- Οι p_1 και p_3 τοποθετούν την αίτηση του p_2 πρώτη αφού έχει μικρότερη χρονοσφραγίδα

Παράδειγμα Εκτέλεσης



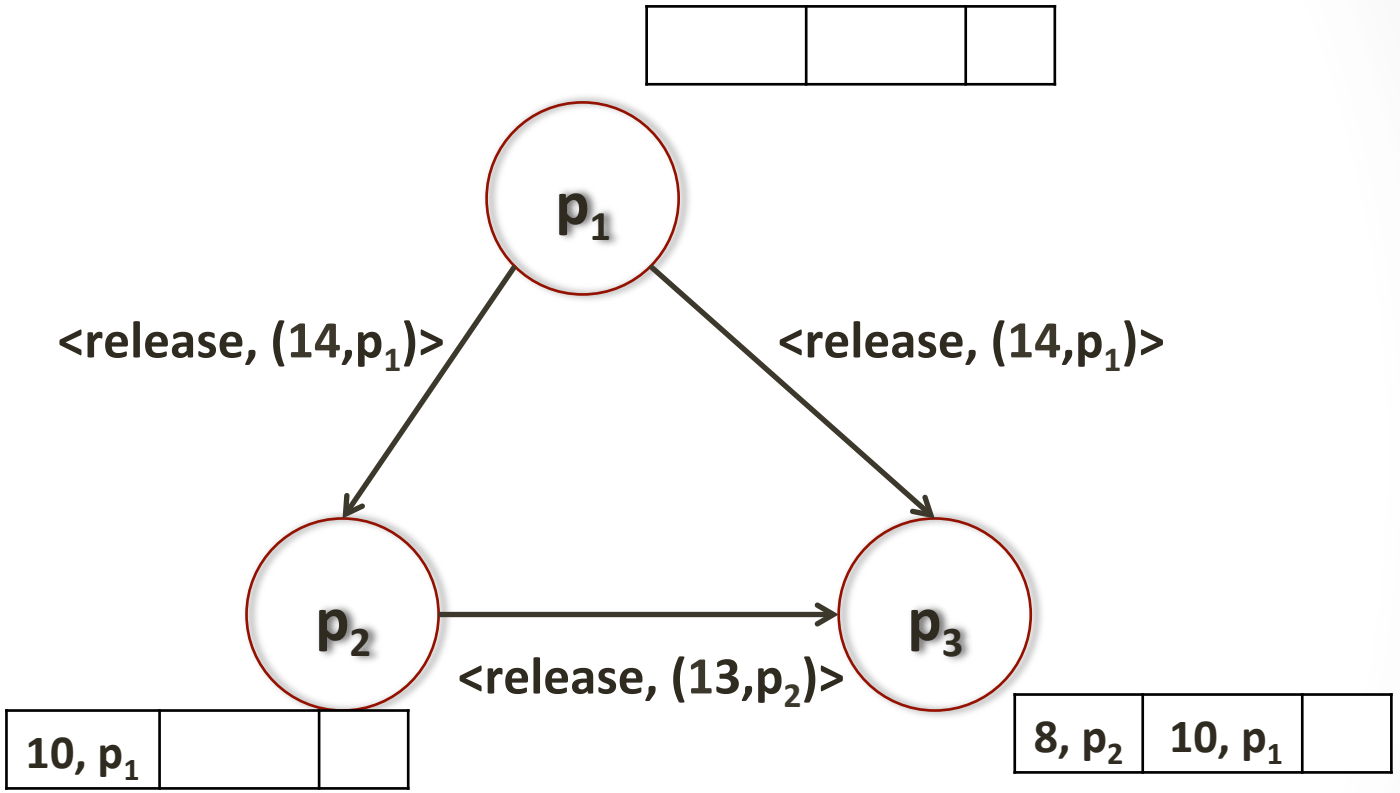
- Όταν ο p_2 τελειώσει από το ΚΤ ειδοποιεί τους υπόλοιπους...

Παράδειγμα Εκτέλεσης



- Όταν ο p_1 λάβει το μήνυμα του p2, τον αφαιρεί από την ουρά του και έρχεται η αίτηση του πρώτη...
- Τότε σταματά να περιμένει και μπαίνει στο ΚΤ...

Παράδειγμα Εκτέλεσης



- Όταν ο p_1 τελειώσει από το ΚΤ ειδοποιεί τους υπόλοιπους...

Ορθότητα Αλγορίθμου

- **Θεώρημα**: Ο αλγόριθμος του Lamport επιτυγχάνει Αμοιβαίο Αποκλεισμό
- Απόδειξη: Με αντίφαση
  - Έστω ότι μπορούν δύο επεξεργαστές p_i και p_j να μπουν στο ΚΤ ταυτόχρονα
  - Συνεπάγεται ότι για τους δυο επεξεργαστές ισχύουν τα ακόλουθα
    - Έλαβαν απάντηση από όλους με χρονοσφραγίδα μεγαλύτερη από την δική τους, και
    - Η αίτησή τους είναι στην κορυφή της ουράς
  - Έστω χ.α.τ.γ ότι η χρονοσφραγίδα που βάζει ο p_i στην αίτησή του είναι μικρότερη από αυτή του p_j
    - Επομένως ο p_i έστειλε την αίτησή του πριν παραλάβει την αίτηση του p_j
    - Και επίσης σημαίνει ότι ο p_i δεν απάντησε ακόμα στον p_j

Ορθότητα Αλγορίθμου

- Απόδειξη Συνέχεια....
  - Έστω ότι ο p_i παρέλαβε την αίτηση του p_j και απάντησε σε αυτή
  - Από την υπόθεση ότι έχουμε FIFO κανάλια έπεται ότι ο p_j θα παραλάβει την απάντηση του p_i **αφότου** παραλάβει την αίτηση του p_i. Άρα τα γεγονότα συμβαίνουν με την εξής σειρά:
    - Παραλαβή αίτησης p_i
    - Παραλαβή απάντησης p_i
  - Αφού σύμφωνα με την υπόθεσή μας ο p_j μπαίνει στο ΚΤ σημαίνει ότι η αίτησή του είναι στην αρχή της ουράς
    - Αυτό συμβαίνει μόνο αν τοποθετήσει την αίτηση του p_i μετά από την δική του
  - Αυτό όμως δεν είναι νόμιμο σύμφωνα με τον αλγόριθμο αφού η αίτηση του p_i έχει μικρότερη χρονοσφραγίδα από την αίτηση του p_j

Αντίφαση

Αποφυγή Παρατεταμένης Στέρησης

- **Θεώρημα**: Ο αλγόριθμος του Lamport αποφεύγει την παρατεταμένη στέρηση
- Απόδειξη: Με αντίφαση
  - Έστω ότι ο p_j μπαίνει στο ΚΤ πριν από τον p_i και η αίτηση του p_j είχε μεγαλύτερη χρονοσφραγίδα από την αίτηση του p_i
    - Αυτό καθιστά μια αίτηση να μην ικανοποιείται και να περιμένει επ' άπειρον
  - Έπεται ότι ο p_j
    - Έλαβε απάντηση από όλους με χρονοσφραγίδα μεγαλύτερη από την δική του, και
    - Η αίτησή του είναι στην κορυφή της ουράς
  - Αφού έλαβε απάντηση από όλους σημαίνει ότι έλαβε απάντηση και από τον p_i

Αποφυγή Παρατεταμένης Στέρησης

- Απόδειξη συνέχεια...
  - Αφού τα κανάλια είναι FIFO έπεται ότι πριν παραλάβει την απάντηση του p_i, ο p_j παρέλαβε την αίτηση του p_i
  - Σύμφωνα όμως με τον αλγόριθμο οι αιτήσεις ταξινομούντε στην ουρά σύμφωνα με την χρονοσφραγίδα τους
  - Αφού η χρονοσφραγίδα της αίτησης του p_i ήταν μικρότερη από την χρονοσφραγίδα της αίτησης του p_j τότε η αίτηση του p_i θα έπρεπε να μπει μπροστά από την αίτηση του p_j στην ουρά
  - Επομένως ο p_i θα έπρεπε να μπει στο ΚΤ πριν από το p_j

Αντίφαση

Πολυπλοκότητα Αλγορίθμου

- Για κάθε αίτηση και χρήση του ΚΤ χρειαζόμαστε
  - N-1 μηνύματα <request>
  - N-1 μηνύματα <reply>
  - N-1 μηνύματα <release>

- Άρα ο αλγόριθμος χρειάζεται συνολικά 3(N-1) μηνύματα

Αλγόριθμος LeLann: Δακτύλιος

- Ιδέα: Πέρνα μια σκυτάλη από τους επεξεργαστές. Όποιος κρατά την σκυτάλη μπορεί να μπει στο ΚΤ.

- Αν θες να μπεισ στο ΚΤ
  - Κράτα την σκυτάλη μέχρι να τελειώσεις και μετά προώθησε την

- Αν δεν θες να μπεις στο ΚΤ
  - Απλά προώθησε την σκυτάλη

Αλγόριθμος LeLann: Δακτύλιος

Entry code at process p_i:

```
wait until receive <token>

enter CS
```

Exit/No-Entry code at p_i:

```
send <token> to right neighbor
```

Ανάλυση Αλγορίθμου

- Συνθήκη Ασφάλειας: Ικανοποιεί τον Αμοιβαίο Αποκλεισμό αφού μόνο μια διεργασία έχει την σκυτάλη και μπαίνει στο ΚΤ

- Συνθήκη Ζωτικότητας: Ικανοποιεί την αποφυγή παρατεταμένης στέρησης όσο η σκυτάλη μεταφέρετε από ένα κόμβο σε άλλο

Μειονεκτήματα

- Αποτυχία Διεργασίας
  - Πρέπει να εντοπιστεί η εσφαλμένη διεργασία
  - Πρέπει να ξαναδημιουργηθεί ο δακτύλιος

- Απώλεια Σκυτάλης
  - Απαιτείται να υπάρχει πάντα μια σκυτάλη
  - Υπεύθυνη διεργασία (πρόεδρος) για να ελέγχει που βρίσκετε η σκυτάλη.

Ερωτήσεις;