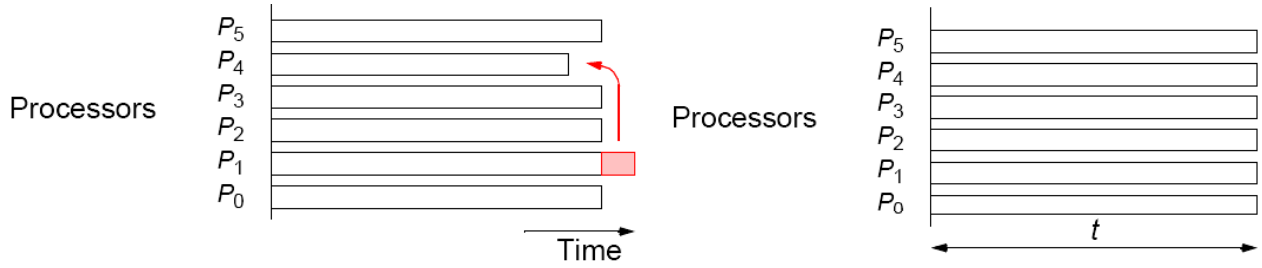
Εξισορρόπηση Φόρτου (Load Balancing) and Ανίχνευση Τερματισμού (Termination Detection)

Η εξισορρόπηση φόρτου χρησιμοποιείται για να κατανείμουμε τους υπολογισμούς δίκαια στις διάφορες διεργασίες προκειμένου να επιτευχθεί ο συντομότερος χρόνος εκτέλεσης

Το πρόβλημα της ανίχνευσης του τερματισμού είναι να ενημερωθούν όλες οι διεργασίες του παράλληλου προγράμματος ότι ο υπολογισμός έχει ολοκληρωθεί. Το πρόβλημα είναι πιο δύσκολο όταν ο υπολογισμός είναι κατανεμημένος

Εξισορρόπηση φόρτου



• Στο αριστερό σχήμα, δεν έχουμε ισοκατανομή φορτίου στις διεργασίες του παράλληλου προγράμματος. Αυτό έχει ως αποτέλεσμα, η διεργασία P_4 να τελειώνει νωρίτερα από όλες τις υπόλοιπες διεργασίες ενώ η διεργασία P_1 είναι η πιο φορτωμένη από όλες με αποτέλεσμα να τελειώνει αργότερα από όλες τις άλλες. Παρατηρείστε ότι ο συνολικός χρόνος εκτέλεσης του παράλληλου προγράμματος καθορίζεται από το χρόνο εκτέλεσης της P_1
• Στο δεξί σχήμα, έχουμε ισοκατανομή του φορτίου στις διάφορες διεργασίες και έτσι όλες οι διεργασίες τελειώνουν την ίδια χρονική στιγμή. Αυτό έχει ως αποτέλεσμα να μειώνεται και ο συνολικός χρόνος εκτέλεσης του προγράμματος

Στατική εξισορρόπηση φόρτου

• Η κατανομή των υπολογισμών στις διάφορες διεργασίες γίνεται πριν την εκτέλεση του παράλληλου προγράμματος. Η στατική εξισορρόπηση φόρτου συνήθως αναφέρεται και ως πρόβλημα απεικόνισης (mapping problem) ή πρόβλημα δρομολόγησης (scheduling problem).
• Για την εφαρμογή αυτής της μεθόδου, θα πρέπει να υπάρχει εκτίμηση του χρόνου εκτέλεσης των διαφόρων τμημάτων του προγράμματος και των αλληλεξαρτήσεων μεταξύ των τμημάτων.
• Ακολουθούν κάποιες στατικές μέθοδοι εξισορρόπησης φόρτου που έχουν προταθεί:
• Ο αλγόριθμος *Round robin* — ο αλγόριθμος αυτός μοιράζει τα έργα κυκλικά στις διεργασίες του προγράμματος επιστρέφοντας ξανά στη πρώτη διεργασία όταν δώσει έργα σε όλες τις διεργασίες.
• *Τυχαιοκρατικοί (Randomized) Algorithms* — μοιράζει με τυχαίο τρόπο τα έργα στις διαθέσιμες διεργασίες
• Αναδρομική διχοτόμηση (*Recursive bisection)* — Η μέθοδος αυτή διαιρεί αναδρομικά το πρόβλημα σε υποπροβλήματα ίσου υπολογιστικού φόρτου ελαχιστοποιώντας ταυτόχρονα την επικοινωνία μεταξύ των διεργασιών

• Κατά τη φάση του προσδιορισμού των έργων που θα αναλάβει κάθε διεργασία, είναι βασικό να λαμβάνεται υπόψη το διασυνδετικό δίκτυο που συνδέει τους επεξεργαστές στα συστήματα κατανεμημένης μνήμης. Οι διεργασίες που επικοινωνούν συχνά θα πρέπει να συνδέονται απευθείας προκειμένου να μειωθεί ο χρόνος επικοινωνίας. Το πρόβλημα αυτό στη γενική του μορφή δεν έχει λύση πολυωνυμικού χρόνου.

Η στατική μέθοδος εξισορρόπησης φορτίου έχει σημαντικά μειονεκτήματα:

• Είναι πολύ δύσκολο να υπολογιστούν με ακρίβεια οι χρόνοι εκτέλεσης των διάφορων τμημάτων του προγράμματος εκ των προτέρων χωρίς δηλ. να εκτελεστούν τα τμήματα αυτά.

• Είναι σχετικά δύσκολο να προσδιορισθεί εκ των προτέρων η καθυστέρηση που προκύπτει από την επικοινωνία μεταξύ των διεργασιών

• Σε μερικά προβλήματα, ο αριθμός των βημάτων που απαιτούνται για να φτάσουμε στη λύση δεν μπορεί να προσδιορισθεί εκ των προτέρων. Για παράδειγμα, στους αλγόριθμους διάσχισης γραφημάτων δεν είναι γνωστό εκ των προτέρων πόσα πολλά μονοπάτια θα διασχίσει ο αλγόριθμος.

Δυναμική εξισορρόπηση φόρτου

• Στη δυναμική εξισορρόπηση φόρτου, τα διαθέσιμα έργα κατανέμονται στις διεργασίες του παράλληλου προγράμματος κατά την εκτέλεση του προγράμματος.

• Αυτή η δυναμική κατανομή του φόρτου κατά την ώρα της εκτέλεσης, δημιουργεί πρόσθετη επιβάρυνση αλλά τα οφέλη είναι πιο σημαντικά σε σχέση με τα προβλήματα που μπορεί να φέρει αυτή η τεχνική.

• Υπάρχουν δύο βασικές κατηγορίες τεχνικών δυναμικής εξισορρόπησης φόρτου:

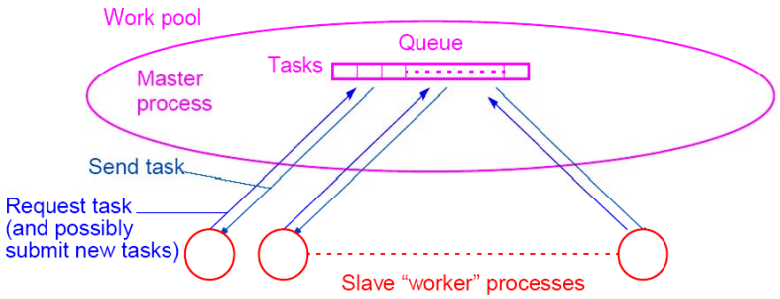
- Συγκεντρωτική (centralized)
- Αποκεντρωμένη (decentralized)

• Στην συγκεντρωτική τεχνική, τα διαθέσιμα προς εκτέλεση έργα δίνονται από μία master διεργασία στις διεργασίες slave του προγράμματος

• Στη αποκεντρωμένη τεχνική, οι διεργασίες δίνουν και παίρνουν διαθέσιμα έργα κατευθείαν η μία από την άλλη χωρίς την επίβλεψη μίας master διεργασίας.

Συγκεντρωτική δυναμική εξισορρόπηση φόρτου

- Η Master διεργασία διαθέτει τη συλλογή των έργων (work pool) που πρόκειται να εκτελεστούν.
- Τα έργα στέλνονται στις διεργασίες slave οι οποίες και τα εκτελούν. Όταν μία διεργασία ολοκληρώσει την εκτέλεση ενός έργου, ζητάει από τη master διεργασία ένα νέο προς εκτέλεση έργο.
- Είναι επίσης πιθανόν οι διεργασίες slave να παράγουν νέα έργα τα οποία υποβάλλονται στη master διεργασία.
- Τα διαθέσιμα έργα δεν είναι απαραίτητα να έχουν τον ίδιο υπολογιστικό φόρτο. Σε αυτή την περίπτωση, θα ήταν προτιμότερο να δοθούν πρώτα στις διεργασίες τα «βαρύτερα» έργα και στη συνέχεια τα πιο μικρά γιατί στην αντίθετη περίπτωση, οι διεργασίες που θα έχουν εκτελέσει μικρά έργα θα περιμένουν αδρανείς μέχρι να ολοκληρώσουν την επεξεργασία τους οι διεργασίες που έχουν αναλάβει τα βαρύτερα έργα.
- Επίσης η ουρά που κρατάει τα προς εκτέλεση έργα δεν χρειάζεται να έχει τη λογική FIFO αλλά μπορεί να είναι μία ουρά προτεραιότητας. Έργα που ενδέχεται να μας οδηγήσουν ταχύτερα στην λύση του υπό μελέτη προβλήματος μπορούν να εκτελούνται νωρίτερα από όλα τα υπόλοιπα.



Τερματισμός

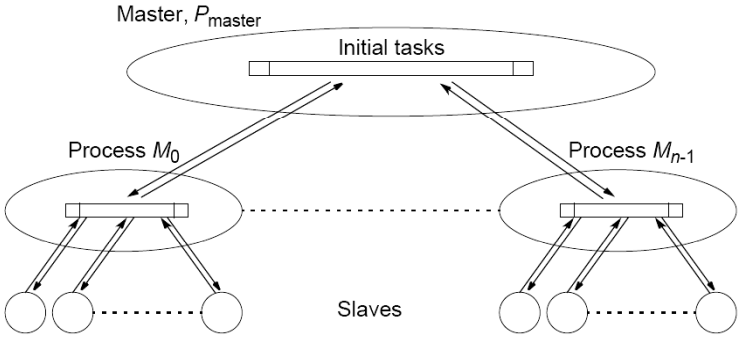
- Με τη συγκεντρωτική τεχνική, είναι σχετικά απλό για τη master διεργασία να διαπιστώσει πότε έχει ολοκληρωθεί η εκτέλεση του παράλληλου προγράμματος. Συγκεκριμένα, θα πρέπει να ικανοποιούνται δύο συνθήκες:

- Η ουρά των έργων είναι άδεια
- Κάθε slave διεργασία είναι αδρανής και επιπλέον κανένα νέο έργο δεν έχει δημιουργηθεί από κάποια διεργασία.

- Δεν αρκεί γενικά να τερματίσουμε τον υπολογισμό όταν η ουρά των έργων είναι άδεια αφού μία ή περισσότερες διεργασίες μπορεί ακόμα να εκτελούν υπολογισμούς ή ενδέχεται να παράξουν νέα έργα προς εκτέλεση.
- Υπάρχουν βέβαια προβλήματα στα οποία δεν παράγονται νέα έργα κατά την εκτέλεση του υπολογισμού. Ένα τέτοιο παράδειγμα είναι ο υπολογισμός του fractal Mandelbrot. Σε αυτή την περίπτωση, ο υπολογισμός μπορεί να τερματισθεί όταν η ουρά των έργων μείνει άδεια και όλες οι slave διεργασίες έχουν τερματίσει.

Αποκεντρωμένη Δυναμική Εξισορρόπηση Φόρτου

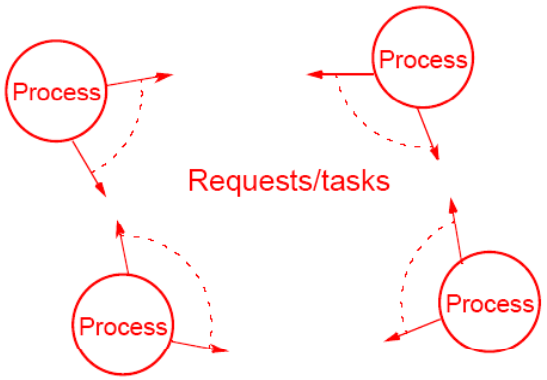
Κατανεμημένη Δεξαμενή Έργων (Distributed Work Pool)



- Το βασικό πρόβλημα στη συγκεντρωτική τεχνική είναι ότι η master διεργασία μπορεί να δίνει ένα έργο τη φορά στις διεργασίες slave
- Αυτό μπορεί να δημιουργήσει σοβαρή καθυστέρηση στην εκτέλεση του παράλληλου προγράμματος, όταν πολλές slave διεργασίες απαιτούν έργα ταυτόχρονα από τη master διεργασία
- Για αυτό το λόγο, η συγκεντρωτική τεχνική είναι κατάλληλη όταν υπάρχουν σχετικά λίγες slave διεργασίες και όταν τα έργα είναι σχετικά μεγάλα με επαρκή υπολογιστικό φόρτο.
- Όταν τα έργα είναι σχετικά μικρά και οι slave διεργασίες είναι πολλές, είναι προτιμότερη η κατανομή της δεξαμενής έργων σε περισσότερες από μία διεργασίες.
- Μία πιθανή λύση είναι να υπάρχουν ένα σύνολο «μικρότερων» masters που αναλαμβάνουν μία υποομάδα των slave διεργασιών. Η αρχική master διεργασία μοιράζει την αρχική δεξαμενή στους «μικρότερους» masters.
- Σε ένα πρόβλημα βελτιστοποίησης, οι μικρότεροι masters μπορεί να υπολογίζουν τοπικά βέλτιστα και στη συνέχεια ο master να υπολογίζει το συνολικό μέγιστο.

Πλήρως Κατανεμημένη Δεξαμενή Έργων

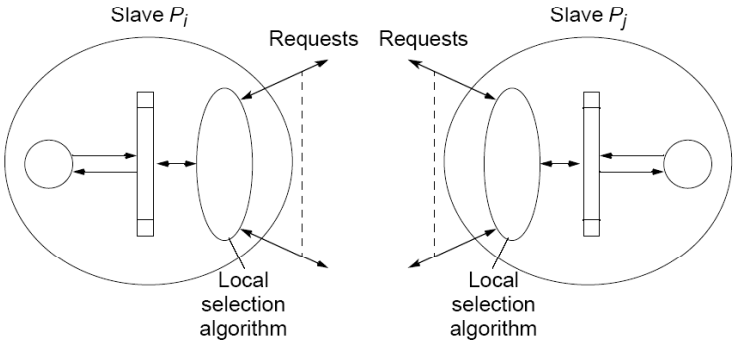
- Στη τεχνική αυτή, μετά την αρχική κατανομή των έργων στις διεργασίες (κατανεμημένη δεξαμενή) οι διεργασίες ζητούν έργα κατευθείαν από τις άλλες διεργασίες χωρίς να παρεμβάλλεται η master διεργασία



- Τα έργα μεταξύ των διεργασιών μπορούν να μεταφερθούν με δύο μεθόδους:
  - εκκίνηση από τη πλευρά του λήπτη (receiver-initiated)
  - εκκίνηση από τη πλευρά του αποστολέα (sender-initiated)

Μέθοδος εκκίνησης από τη πλευρά του λήπτη

•Σε αυτή τη μέθοδο, μία διεργασία απαιτεί έργα από άλλες διεργασίες όταν η διεργασία έχει λίγα ή δεν έχει κανένα έργο να εκτελέσει. Η μέθοδος αυτή είναι κατάλληλη σε συνθήκες μεγάλου φόρτου στο υπολογιστικό σύστημα.



•Υπάρχουν πολλές διαφορετικές μέθοδοι για την επιλογή της διεργασίας από την οποία θα ζητηθεί ένα έργο. Δύο απλοί αλγόριθμοι είναι:

- Κυκλική επιλογή (R*ound robin)* – η διεργασία *Pi* ζητά έργα από την διεργασία *Px*, όπου ο δείκτης *x* δίνεται από ένα μετρητή ο οποίος αυξάνεται μετά από κάθε αίτηση χρησιμοποιώντας modulo n αριθμητική (n διεργασίες), αποκλείοντας τη τιμή x=i
- Τυχαία επιλογή – η διεργασία *Pi* ζητά έργα από τη διεργασία *Px*, όπου ο δείκτης *x* είναι ένας αριθμός ο οποίος επιλέγεται τυχαία μεταξύ του 0 και *n* - 1 (αποκλείοντας την τιμή *i).*

• Πάντως μία επιτυχημένη επιλογή διεργασίας θα πρέπει να λαμβάνει υπόψη και το φόρτο των άλλων διεργασιών και έτσι είναι προτιμότερο να ζητούνται έργα από διεργασίες με υψηλό φόρτο.

Μέθοδος εκκίνησης από τη πλευρά του αποστολέα

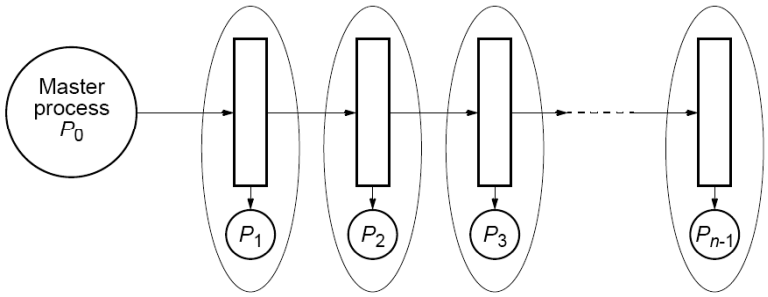
• Στη μέθοδο αυτή, μία διεργασία με υψηλό φόρτο, στέλνει έργα σε άλλες διεργασίες οι οποίες είναι πρόθυμες να δεχθούν τα νέα αυτά έργα.

• Η μέθοδος αυτή αποδίδει κυρίως σε συνθήκες χαμηλού φόρτου στο σύστημα.

• Για την επιλογή των διεργασιών που θα λάβουν τα νέα έργα, μπορούμε να χρησιμοποιήσουμε παρόμοιες μεθόδους με αυτές της τεχνικής εκκίνησης από την πλευρά του λήπτη.

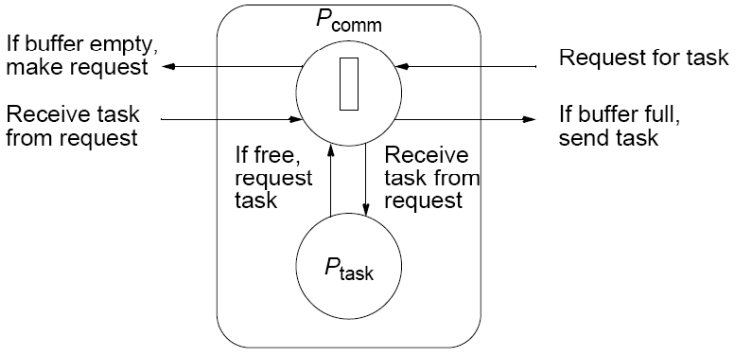
• Μπορούμε επίσης να έχουμε ένα συνδυασμό των παραπάνω μεθόδων (sender-initiated και receiver-initiated).

Εξισορρόπηση φορτίου με διεργασίες σε γραμμική διάταξη



• Στο σχήμα, κάθε διεργασία επικοινωνεί με δύο μονό διεργασίες (εκτός από τη πρώτη και τη τελευταία διεργασία που επικοινωνούν με μία μόνο). Η βασική ιδέα είναι να δημιουργήσουμε μία κατανεμημένη ουρά από έργα με κάθε διεργασία να επιτηρεί ένα διαφορετικό τμήμα της ουράς.
• Η master διεργασία (P_0) τροφοδοτεί την ουρά με έργα από το ένα άκρο και τα έργα προχωρούν προς τα δεξιά μέσα στην ουρά.
• Όταν η διεργασία, P_i (1 <= i < n), ανιχνεύσει ένα έργο στο τμήμα της ουράς που ελέγχει και εφόσον η διεργασία είναι αδρανής, παίρνει το έργο από την ουρά και αρχίζει και το εκτελεί.
• Στη συνέχεια, όλα τα έργα πιο «αριστερά» από αυτό το σημείο, μετακινούνται προς τα δεξιά στην ουρά προκειμένου να καλυφθεί το κενό που δημιουργήθηκε στην ουρά από την εκτέλεση του έργου. Ως τελικό αποτέλεσμα, ένα νέο έργο εισέρχεται στην ουρά από το αριστερό άκρο.
• Τελικά, όλες οι διεργασίες έχουν ένα έργο να εκτελέσουν και η ουρά συνεχίζει να τροφοδοτείται με νέα έργα. Επίσης, υψηλής προτεραιότητας έργα μπορούν να τοποθετούνται στην ουρά πρώτα, νωρίτερα από όλα τα υπόλοιπα έργα.

• Οι λειτουργίες ολίσθησης μπορούν να υλοποιηθούν με την ανταλλαγή μηνυμάτων μεταξύ γειτονικών διεργασιών.
• Το ακόλουθο σχήμα δείχνει τις βασικές λειτουργίες που θα πρέπει να εκτελεί κάθε διεργασία



• Ο κώδικας που υλοποιεί την παραπάνω τεχνική εξισορρόπησης φόρτου θα έχει ως εξής:
Master process (P0)

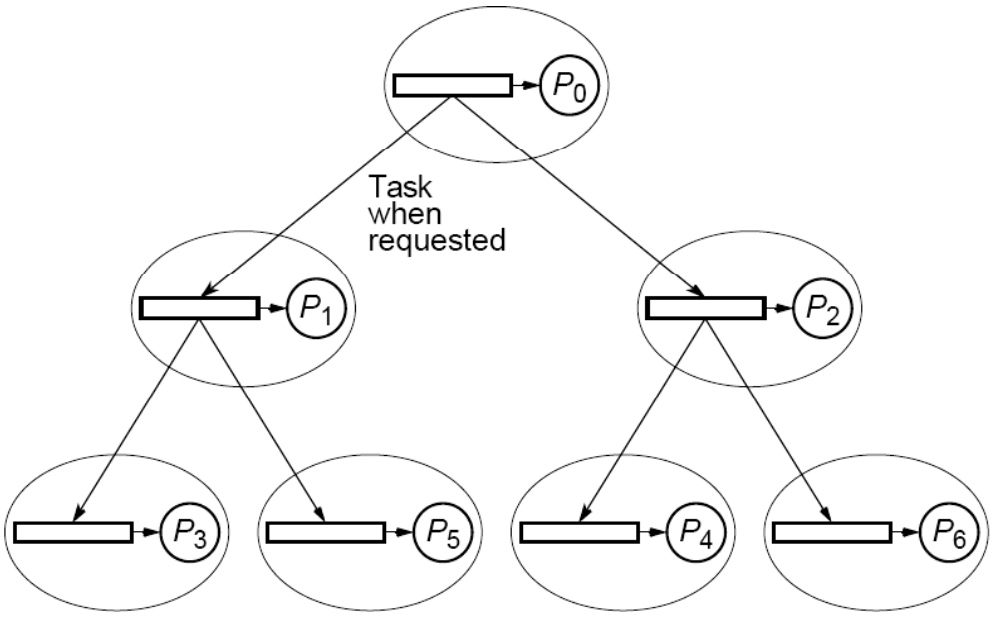
```
for (i = 0; i < no_tasks; i++) {
    recv(P1, request_tag);        /* request for task */
    send(&task, Pi, task_tag);    /* send tasks into queue */
}
recv(P1, request_tag);            /* request for task */
send(&empty, Pi, task_tag);       /* end of tasks */
```

Διεργασία *Pi* (1 < *i* < *n*)

```
if (buffer == empty) {
   send(Pi-1, request_tag);     /* request new task */
   recv(&buffer, Pi-1, task_tag);/* task from left proc */
}
if ((buffer == full) && (!busy)) {/* get next task */
   task = buffer;               /* get task*/
   buffer = empty;              /* set buffer empty */
   busy = TRUE;                 /* set process busy */
}
nrecv(Pi+1, request_tag, request);/* check msg from right */
if (request && (buffer == full)) {
   send(&buffer, Pi+1);         /* shift task forward */
   buffer = empty;
}
if (busy) {                     /* continue on current task */
   Do some work on task.
   If task finished, set busy to false.
}
```

Η μη αναστέλλουσα λειτουργία **nrecv()** είναι απαραίτητη για να ελέγξουμε αν έχει ληφθεί αίτημα από τα δεξιά. Σε μία υλοποίηση με MPI, θα χρησιμοποιούσαμε την συνάρηση **MPI_Irecv()** η οποία επιστρέφει αμέσως και «συμπληρώνει» τη δομή request με στοιχεία της λήψης που είναι σε εξέλιξη. Αυτή η δομή χρησιμοποιείται στη συνέχεια είτε από τη ρουτίνα MPI_Wait() η οποία περιμένει για την ολοκλήρωση της λήψης ή από τη ρουτίνα MPI_Test() η οποία απλά ελέγχει αν έχει ολοκληρωθεί η λήψη

• Η προηγούμενη τεχνική εξισορρόπησης φόρτου μπορεί να επεκταθεί και σε άλλες διατάξεις επικοινωνίας μεταξύ των διεργασιών πέραν της γραμμικής. Στο σχήμα που ακολουθεί οι διεργασίες είναι συνδεδεμένα σε ιεραρχική (δενδρική) διάταξη.
• Το έργο σε κάθε κόμβο περνά σε ένα από τα δύο παιδιά του, συγκεκριμένα αυτό του οποίου ο αποθηκευτικός χώρος δεν περιέχει κάποιο έργο



Κατανεμημένη Ανίχνευση Τερματισμού

• Όταν ο υπολογισμός είναι κατανεμημένος, η διαπίστωση ότι ο υπολογισμός έχει συνολικά ολοκληρωθεί μπορεί να είναι δύσκολη υπόθεση εκτός αν το πρόβλημα είναι τέτοιο ώστε μόνο μία από τις διεργασίες φθάνει στη λύση. Γενικά, ο κατανεμημένος τερματισμός τη χρονική στιγμή t θα πρέπει να ικανοποιεί τις ακόλουθες συνθήκες:

  • Θα πρέπει να ικανοποιούνται τοπικές συνθήκες τερματισμού σε κάθε διεργασία τη χρονική στιγμή *t*. Οι συνθήκες τερματισμού σε κάθε διεργασία εξαρτώνται από τη συγκεκριμένη εφαρμογή. Π.χ. στην επαναληπτική μέθοδο Jacobi, η τοπική συνθήκη τερματισμού σε μία διεργασία συμβαίνει όταν η τιμή του τοπικού αγνώστου έχει συγκλίνει στην επιθυμητή ακρίβεια.

  • Δεν υπάρχουν μηνύματα τα οποία να είναι υπό μεταφορά μεταξύ των διεργασιών τη χρονική στιγμή *t*.

• Η λεπτή διαφορά μεταξύ των παραπάνω συνθηκών τερματισμού και αυτών που δόθηκαν για τον τερματισμό σε ένα σύστημα συγκεντρωτικής εξισορρόπησης φόρτου είναι ότι τώρα λαμβάνονται υπόψη τα μηνύματα τα οποία μπορεί να είναι υπό μεταφορά μεταξύ των διεργασιών.

• Η δεύτερη συνθήκη είναι απαραίτητη επειδή ένα μήνυμα υπό μεταφορά μπορεί να επανεκκινήσει μία διεργασία που έχει ήδη τερματίσει.

• Η πρώτη συνθήκη είναι εύκολο να ανιχνευθεί αφού κάθε διεργασία μπορεί να στείλει ένα μήνυμα στη master διεργασία αφότου οι τοπικές συνθήκες τερματισμού ικανοποιούνται.

• Η δεύτερη συνθήκη είναι πιο δύσκολη να ανιχνευθεί αφού ο χρόνος που απαιτείται για να σταλούν τα μηνύματα μεταξύ των διεργασιών δεν είναι γνωστός εκ των προτέρων και εξαρτάται από τη συγκεκριμένη αρχιτεκτονική στην οποία υλοποιείται ο υπολογισμός. Η πιο απλή λύση είναι να αφήσουμε ένα επαρκές χρονικό διάστημα προκειμένου τα μηνύματα να παραληφθούν. Όμως αυτό δεν αποτελεί γενική λύση.

- Οι Bertsekas και Tsitsiklis πρότειναν μία γενική μέθοδο κατανεμημένου τερματισμού.
- Κάθε διεργασία μπορεί να είναι σε μία από τις δύο καταστάσεις

  1. Αδρανής – η διεργασία δεν έχει κάποιο έργο να εκτελέσει
  2. Ενεργή

- Η διεργασία η οποία έστειλε το αρχικό έργο σε μία διεργασία και την έκανε να αλλάξει κατάσταση και να γίνει ενεργή, γίνεται πατέρας αυτής της διεργασίας.

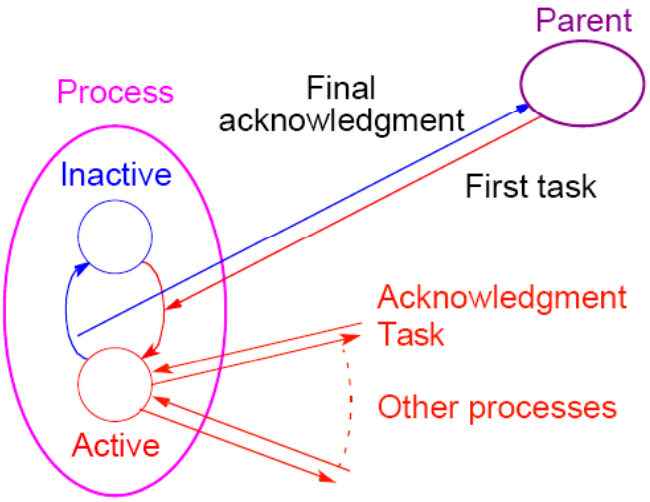
• Όταν η διεργασία λαμβάνει ένα έργο, στέλνει αμέσως ένα μήνυμα επιβεβαίωσης λήψης (acknowledgment message), εκτός αν το έργο που λαμβάνει η διεργασία προέρχεται από τη διαδικασία πατέρα. Μία διεργασία στέλνει μήνυμα ack στον πατέρα του μόνο όταν η διεργασία είναι έτοιμη να γίνει αδρανής, δηλ. όταν

  • Η τοπική συνθήκη τερματισμού ικανοποιείται (όλα τα έργα έχουν ολοκληρωθεί) και
  • έχει στείλει όλα τα μηνύματα ack για τα έργα που έχει λάβει, και
  • έχει λάβει όλα τα μηνύματα ack για τα έργα που έχει στείλει σε άλλες διεργασίες.

• Η τελευταία συνθήκη έχει ως πρακτικό αποτέλεσμα μία διεργασία να γίνεται αδρανής πάντα πριν τη διαδικασία πατέρα της.

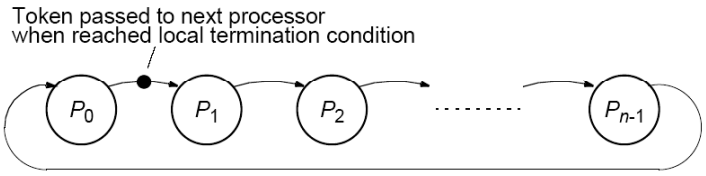
• Όταν η πρώτη διεργασία που ενεργοποιήθηκε στο πρόγραμμα γίνει αδρανής, ο υπολογισμός μπορεί να τερματίσει.

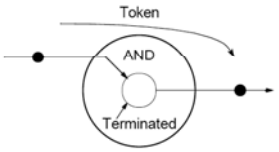
Τερματισμός χρησιμοποιώντας μηνύματα επιβεβαίωσης λήψης μηνύματος



Το παραπάνω σχήμα συνοψίζει τη διαδικασία που περιγράφηκε νωρίτερα.

Αλγόριθμος Ανίχνευσης Τερματισμού με διεργασίες σε διάταξη δακτυλίου ενός περάσματος





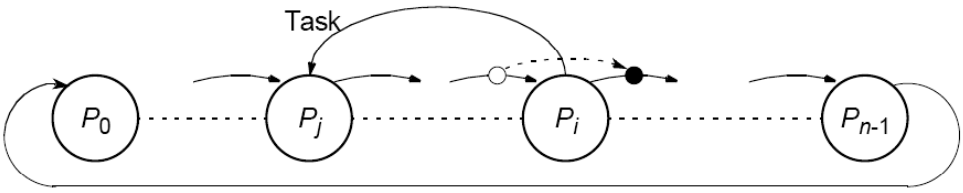
- Όταν οι διεργασίες επικοινωνούν σε διάταξη δακτυλίου μπορεί να εφαρμοσθεί ο ακόλουθος αλγόριθμος ανίχνευσης τερματισμού:
  1. Όταν η διεργασία P_0 έχει τερματίσει, παράγει ένα τεκμήριο (token) και το περνάει στη διεργασία P_1.
  2. Όταν η διεργασία P_i ($1 <= i < n$) λάβει το τεκμήριο και έχει ήδη τερματίσει, περνάει το τεκμήριο στην διεργασία P_{i+1}. Διαφορετικά, περιμένει μέχρι την ικανοποίηση της τοπικής συνθήκης τερματισμού. Από τη στιγμή που η συνθήκη επαληθευθεί, περνάει το τεκμήριο στην επόμενη διεργασία P_{i+1}. Η διεργασία P_{n-1} περνάει το τεκμήριο στη διεργασία P_0.
  3. Όταν η P_0 λάβει το τεκμήριο, γνωρίζει ότι όλες οι διεργασίες στο δακτύλιο έχουν τερματίσει. Αν χρειάζεται, η διεργασία P_0 μπορεί να στέλνει ένα μήνυμα σε όλες τις διεργασίες πληροφορώντας τες για τον συνολικό τερματισμό.
- Ο συγκεκριμένος αλγόριθμος υποθέτει ότι μία διεργασία δεν πρόκειται να ενεργοποιηθεί ξανά μετά το τοπικό τερματισμό της. Έτσι, η τεχνική αυτή δεν μπορεί να εφαρμοσθεί σε περιπτώσεις που μία διεργασία στέλνει ένα νέο έργο σε μία ήδη αδρανή διεργασία

Αλγόριθμος Τερματισμού δύο περασμάτων με τις διεργασίες σε διάταξη δακτυλίου

- Μπορούμε να τροποποιήσουμε την προηγούμενη μέθοδο έτσι ώστε να προβλέπουμε και τη περίπτωση της επανεργοποίησης μίας διεργασίας.
- Η νέα μέθοδος απαιτεί δύο περάσματα του τεκμηρίου γύρω από το δακτύλιο.
- Ο δεύτερος γύρος μπορεί να χρειάζεται γιατί η διεργασία P_i ενδέχεται να περάσει ένα έργο στην διεργασία P_j όπου j<i αφότου το τεκμήριο έχει περάσει από την P_j. Αν αυτό λοιπόν συμβεί, το τεκμήριο πρέπει να περάσει από όλες τις διεργασίες δεύτερη φορά.
- Ο τρόπος με τον οποίο οι διεργασίες μπορούν να καταλάβουν ότι ένας δεύτερος γύρος είναι απαραίτητος είναι ο «χρωματισμός» του τεκμηρίου με δύο χρώματα, μαύρο ή άσπρο, ανάλογα με το αν χρειάζεται δεύτερος γύρος ή όχι.
- Επίσης και οι διεργασίες χρωματίζονται ως μαύρες ή άσπρες.
- Η λήψη ενός μαύρου τεκμηρίου από μία διεργασία σημαίνει ότι συνολικός τερματισμός δεν έχει επιτευχθεί και ότι το τεκμήριο θα πρέπει να κυκλοφορήσει πάλι στο δακτύλιο

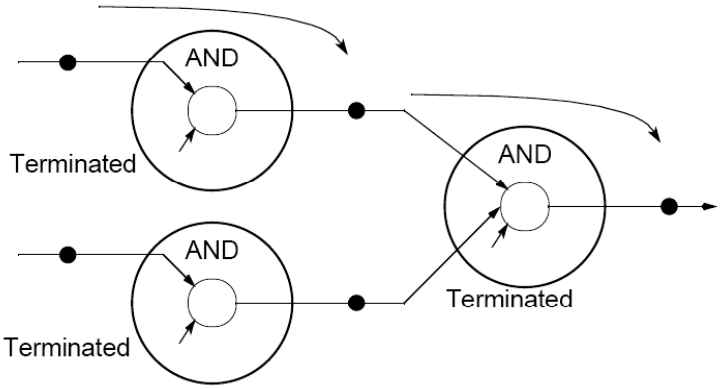
- Αναλυτικά, ο αλγόριθμος έχει ως εξής:

1. P_0 γίνεται λευκή όταν τερματίζει και παράγει ένα λευκό τεκμήριο που το στέλνει στη διεργασία P_1.
2. Κάθε διεργασία αφότου τερματίσει περνάει το τεκμήριο που έχει λάβει στην επόμενη διεργασία. Όμως το χρώμα του τεκμηρίου μπορεί να αλλάξει. Συγκεκριμένα αν η διεργασία P_i περάσει ένα έργο στην διεργασία P_j όπου $j < i$ (δηλ. σε μία διεργασία πιο αριστερά από αυτή στο δακτύλιο), γίνεται μαύρη διεργασία αλλιώς παραμένει λευκή. Μία μαύρη διεργασία θα χρωματίσει το τεκμήριο μαύρο και θα το περάσει στην επόμενη διεργασία. Μία άσπρη διεργασία θα διατηρήσει το χρώμα του τεκμηρίου που έλαβε περνώντας το στην επόμενη χωρίς αλλαγές. Αφότου η διεργασία P_i περάσει το τεκμήριο, η P_i ξαναγίνεται άσπρη.
3. Όταν η διεργασία P_0 λάβει ένα μαύρο τεκμήριο το περνάει στην επόμενη διεργασία αφού πρώτα το χρωματίσει άσπρο. Αν η διεργασία P_0 λάβει ένα άσπρο τεκμήριο, σημαίνει ότι όλες οι διεργασίες έχουν τερματίσει.



Αλγόριθμος τερματισμού με διεργασίες σε διάταξη δένδρου

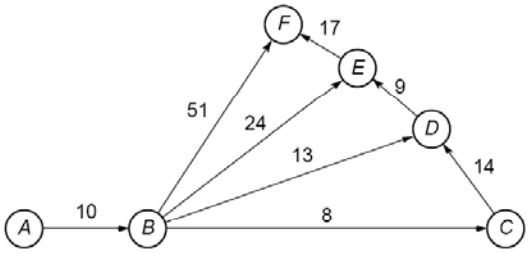
- Η παραπάνω τεχνική μπορεί να γενικευθεί με τις διεργασίες σε διάταξη δένδρου.
- Σε αυτή τη περίπτωση, μία διεργασία περνάει το τεκμήριο στον πατέρα του αν έχουν ληφθεί τεκμήρια και από τα δύο παιδιά του.
- Αυτό πρακτικά σημαίνει ότι ο υπολογισμός έχει ολοκληρωθεί σε όλο το υποδέντρο με ρίζα τη συγκεκριμένη διεργασία.



Αλγόριθμος κατανεμημένου τερματισμού με σταθερή ενέργεια

- Στον αλγόριθμο αυτό, χρησιμοποιείται μία σταθερή ποσότητα η οποία θεωρείται ως «ενέργεια» του συστήματος.
- Αυτή η ενέργεια είναι γενίκευση της έννοιας του τεκμηρίου
- Το σύστημα ξεκινά με όλη αυτή την ενέργεια συγκεντρωμένη στη διαδικασία ρίζα.
- Σε κάθε διεργασία που ζητάει έργα να εκτελέσει, η διαδικασία ρίζα δίνει μαζί με τα έργα και μέρος της ενέργειας που διαθέτει στις διεργασίες αυτές.
- Με παρόμοιο τρόπο, αν μία διεργασία λάβει αιτήσεις για έργα, η ενέργεια μοιράζεται περαιτέρω.
- Όταν μία διεργασία γίνει αδρανής, δίνει πίσω την ενέργεια που διαθέτει πριν ζητήσει νέο έργο.
- Αυτή η ενέργεια μπορεί να επιστραφεί κατευθείαν πίσω στη διεργασία-ρίζα ή στη διεργασία που έδωσε το αρχικό έργο στη συγκεκριμένη διεργασία.
- Στη δεύτερη περίπτωση, δημιουργείται μία δενδρική ιεραρχία μεταξύ των διεργασιών. Μία διεργασία δεν επιστρέφει την ενέργειά του μέχρι να συλλέξει πάλι όλη την ενέργεια που έχει μοιράσει.
- Όταν όλη η ενέργεια έχει επιστραφεί πίσω στη διεργασία ρίζα, όλες οι διεργασίες είναι αδρανείς και ο υπολογισμός μπορεί να τερματισθεί συνολικά.

Παράδειγμα εξισορρόπησης φόρτου και ανίχνευσης τερματισμού



Source \ Destination	A	B	C	D	E	F
A	•	10	•	•	•	•
B	•	•	8	13	24	51
C	•	•	•	14	•	•
D	•	•	•	•	9	•
E	•	•	•	•	•	17
F	•	•	•	•	•	•

(a) Adjacency matrix

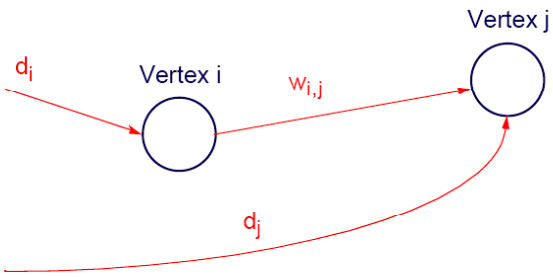
- Το πρόβλημα που θα μελετηθεί είναι η παράλληλη υλοποίηση της εύρεσης των συντομότερων διαδρομών από μία κορυφή (στο συγκεκριμένο παράδειγμα από τη κορυφή Α) προς όλες τις υπόλοιπες.
- Υπάρχουν δύο γνωστοί αλγόριθμου για την εύρεση συντομότερων διαδρομών με κοινή αφετηρία:
  - Ο αλγόριθμος Moore (1957)
  - Ο αλγόριθμος Dijkstra (1959)
- Δύο αυτοί αλγόριθμοι είναι παρόμοιοι. Στο συγκεκριμένο παράδειγμα, θα ακολουθήσουμε τον αλγόριθμο του Moore επειδή είναι πιο εύκολος να παραλληλοποιηθεί αν και κάνει περισσότερους υπολογισμούς από ότι ο αλγόριθμος του Dijkstra
- Τα βάρη στο γράφημα θα πρέπει να είναι όλα θετικά προκειμένου η μέθοδος να εφαρμοσθεί

Ο αλγόριθμος του Moore

- Με αρχή τον κόμβο αφετηρία, ο βασικός αλγόριθμος επισκέπτεται διαδοχικά κάθε κορυφή i του γραφήματος και εκτελεί τα εξής:
  - Για κάθε άλλη κορυφή j, ελέγχεται αν η διαδρομή από τη αφετηρία μέχρι την κορυφή i που διέρχεται μέσω της κορυφής i είναι συντομότερη από αυτή που έχει ευρεθεί μέχρι εκείνη τη στιγμή. Σε αυτή την περίπτωση κρατείται αυτή ως η ελάχιστη τρέχουσα απόσταση από την αφετηρία προς τη συγκεκριμένη κορυφή j. Πιο αναλυτικά αν d_i είναι η τρέχουσα ελάχιστη απόσταση από την αφετηρία προς την κορυφή i και $w_{i,j}$ είναι το βάρος της ακμής από την κορυφή i στην κορυφή j, η ενημέρωση που γίνεται σε κάθε βήμα περιγράφεται από την ακόλουθη σχέση

$$d_j = \min(d_j, d_i + w_{i,j})$$

- Ο παραπάνω αλγόριθμος επαναλαμβάνεται μέχρι να μη προκύψει καμία αλλαγή στις αποστάσεις d_j για όλες τις διεργασίες j μετά από μία επανάληψη



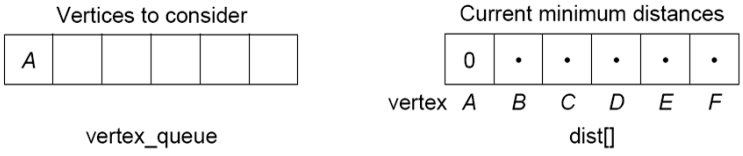
• Για την υλοποίηση του αλγορίθμου, χρησιμοποιείται μία ουρά FIFO η οποία αποθηκεύει τη λίστα των κορυφών που θα εξεταστούν. Αρχικά, μόνο η κορυφή-αφετηρία είναι στην ουρά.
• Επίσης, οι τρέχουσες μικρότερες αποστάσεις από την αφετηρία στις υπόλοιπες κορυφές αποθηκεύονται στον πίνακα d. Συγκεκριμένα, η τρέχουσα μικρότερη απόσταση προς την κορυφή i είναι αποθηκευμένη στο στοιχείο d[i]. Αρχικά, καμία από αυτές τις αποστάσεις δεν είναι γνωστές και έτσι αρχικοποιούνται στη τιμή άπειρο.
• Τα βάρη των ακμών του γραφήματος είναι αποθηκευμένα στον πίνακα w και w[i][j] είναι το βάρος της ακμής από την κορυφή *i* στην κορυφή *j* . Το βάρος αυτό είναι άπειρο αν δεν υπάρχει ακμή μεταξύ δύο κόμβων.
• Ο κώδικας που ενημερώνει τις ελάχιστες αποστάσεις σε κάθε βήμα έχει ως εξής:

```
newdist_j = dist[i] + w[i][j];
        if (newdist_j < dist[j]) dist[j] = newdist_j;
```

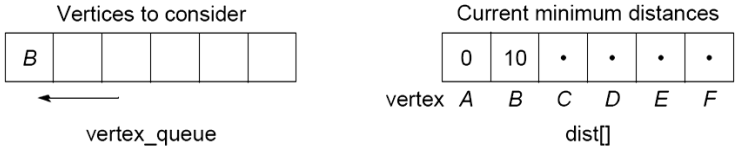
• Όταν η συντομότερη απόσταση προς την κορυφή j μειώνεται, η κορυφή j προστίθεται στην ουρά (αν δεν είναι στην ουρά). Αυτό έχει ως αποτέλεσμα, η κορυφή *j* να εξετάζεται ξανά από τον αλγόριθμο.

• Στη συνέχεια ακολουθεί η εκτέλεση του αλγορίθμου για το γράφημα του παραδείγματος

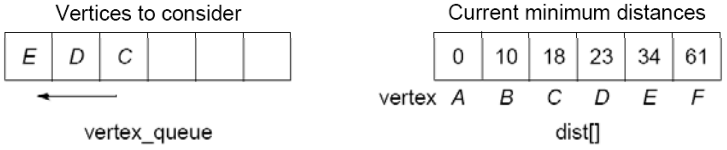
• Η αρχική κατάσταση των δύο δομών δεδομένων έχει ως εξής:



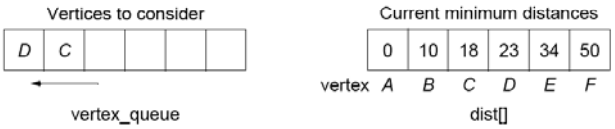
• Μετά την εξέταση της κορυφής A, θα έχουμε:



• Μετά την εξέταση των ακμών από την *B* στην *F*, *E*, *D*, και *C* έχουμε:

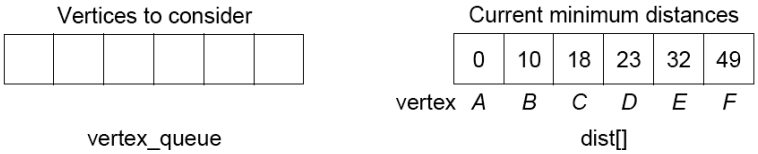


Μετά την εξέταση της ακμής από την κορυφή *E* στην *F* θα έχουμε:



Μετά την εξέταση της ακμής από την κορυφή *D* στην *E* θα έχουμε:

Μετά την εξέταση της ακμής από την κορυφή C στη *D*, δεν προκύπτουν αλλαγές. Μετά την εξέταση της ακμής *E* προς την *F* θα έχουμε:



Η ουρά έχει αδειάσει και επομένως δεν υπάρχουν άλλες κορυφές να εξετάσουμε. Ο πίνακας d περιέχει πλέον τις ελάχιστες αποστάσεις από την κορυφή A προς τις υπόλοιπες κορυφές του γραφήματος.

Ακολουθιακός κώδικας για τον αλγόριθμο του Moore

Αν η συνάρτηση **next_vertex()** επιστρέφει την επόμενη κορυφή από την ουρά των κορυφών ή την τιμή **no_vertex** εάν η ουρά είναι κενή, ο ακολουθιακός κώδικας για τον αλγόριθμο του Moore θα έχει ως εξής:

```
while ((i = next_vertex()) != no_vertex)  /* while a vertex */
  for (j = 1; j < n; j++)                 /* get next edge */
    if (w[i][j] != infinity) {            /* if an edge */
      newdist_j = dist[i] + w[i][j];
      if (newdist_j < dist[j]) {
        dist[j] = newdist_j;
        append_queue(j);          /* add to queue if not there */
      }
    }                             /*no more to consider*/
```

Παράλληλη υλοποίηση του αλγόριθμου του Moore

Θα παρουσιάσουμε πρώτα μία υλοποίηση με συγκεντρωτική δεξαμενή έργων.

Η συγκεντρωτική δεξαμενή έργων είναι ουσιαστικά η ουρά των κορυφών, **vertex_queue.**

Κάθε slave διεργασία παίρνει κορυφές από την ουρά και επιστρέφει νέες κορυφές των οποίων οι συντομότερες αποστάσεις έχουν αλλάξει.

Αφού, ο πίνακας των βαρών των ακμών του γραφήματος δεν αλλάζει κατά την εκτέλεση του παράλληλου προγράμματος, αυτή η δομή αντιγράφεται από την αρχή της εκτέλεσης σε κάθε slave διεργασία.

Ο κώδικας για τη Master διεργασία ακολουθεί

```
while (vertex_queue() != empty) {
  recv(PANY, source = Pi);    /* request task from slave */
  v = get_vertex_queue();
  send(&v, Pi);               /* send next vertex and */
  send(&dist, &n, Pi);        /* current dist array */
     .
  recv(&j, &dist[j], PANY, source = Pi);/* new distance */
  append_queue(j, dist[j]);   /* append vertex to queue */
                              /* and update distance array */
};
recv(PANY, source = Pi);      /* request task from slave */
send(Pi, termination_tag);    /* termination message*/
```

Ο κώδικας για τη slave διεργασία i

```
send(Pmaster);                  /* send request for task */
recv(&v, Pmaster, tag);         /* get vertex number */
if (tag != termination_tag) {
  recv(&dist, &n, Pmaster);     /* and dist array */
  for (j = 1; j < n; j++)       /* get next edge */
     if (w[v][j] != infinity) {/* if an edge */
        newdist_j = dist[v] + w[v][j];
        if (newdist_j < dist[j]) {
          dist[j] = newdist_j;
          send(&j, &dist[j], Pmaster);/* add vertex to queue */
        }                           /* send updated distance */
     }
}
```

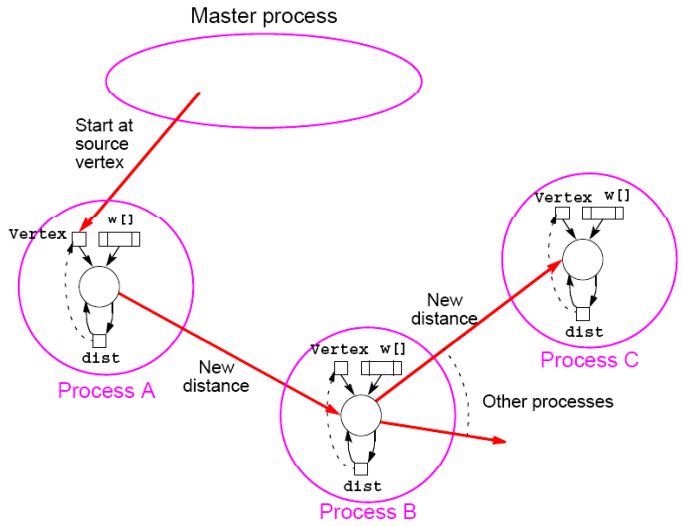
Στο παραπάνω κώδικα, η διεργασία Master στέλνει στη slave διεργασία μαζί με την κορυφή που θα επεξεργαστεί και το πίνακα d με τις τρέχουσες ελάχιστες αποστάσεις.

Αποκεντρωμένη Δεξαμενή Έργων

• Σε αυτή την τεχνική, η ουρά vertex_queue κατανέμεται στις διεργασίες. Συγκεκριμένα, η slave διεργασία i αναλαμβάνει την επεξεργασία της κορυφής *i* και επομένως κατέχει την εγγραφή της ουράς για την κορυφή *i.* Με άλλα λόγια, για κάθε κορυφή i δεσμεύεται μία θέση στην ουρά και αυτή η θέση είναι υπό τον έλεγχο της διεργασίας i.
• Ο πίνακας **dist[ ]** κατανέμεται μεταξύ των διεργασιών κατά τέτοιο τρόπο ώστε η διεργασία *i* να διατηρεί πάντα την τρέχουσα ελάχιστη απόσταση προς την κορυφή *i*.
• Επίσης, κάθε διεργασία i αποθηκεύει την i-οστη γραμμή από το πίνακα γειτνίασης του γραφήματος. Η γραμμή αυτή περιέχει τα βάρη των ακμών της κορυφής i προς όλες τις υπόλοιπες κορυφές του γραφήματος.

• Ο αλγόριθμος αναζήτησης έχει ως εξής:

  • Η επεξεργασία αρχίζει από την κορυφή *A*. Η διαδικασία που έχει αναλάβει στην κορυφή A ενεργοποιείται.
  • Αυτή η διεργασία θα βρει τις αποστάσεις προς τους γείτονες της κορυφής A.
  • Η απόσταση προς την κορυφή *j* θα σταλεί στη διεργασία *j* η οποία θα συγκρίνει τη νέα τιμή με αυτή που έχει ήδη αποθηκευμένη και θα την ενημερώσει αν η νέα απόσταση είναι μικρότερη από την αποθηκευμένη.
  • Έτσι με αυτό τον τρόπο, όλες οι ελάχιστες αποστάσεις θα ενημερωθούν κατά τη διάρκεια της αναζήτησης.
  • Αν το περιεχόμενο του **d[i]** αλλάξει, η διεργασία *i* θα ενεργοποιηθεί ξανά και αρχίζει να υπολογίζει τις αποστάσεις προς τους γείτονες της κορυφής i σε ένα νέο γύρο αναζήτησης.



Ο κώδικας για τη slave διεργασία έχει ως εξής:

```
recv(newdist, PANY);
if (newdist < dist) {
    dist = newdist;
    vertex_queue = TRUE;
} else vertex_queue =
FALSE;
If (vertex_queue == TRUE)
  for (j=0; j <n ; j++)
     if (w[j] != infinity) {
        d = dist + w[j];
        send(&d, Pj);
     }
```

Ο κώδικας για την slave διεργασία i μπορεί να απλοποιηθεί ως εξής:

```
recv(newdist, PANY);
if (newdist < dist)
  dist = newdist;              /* start searching around vertex */
  for (j = 1; j < n; j++) /* get next edge */
     if (w[j] != infinity) {
        d = dist + w[j];
        send(&d, Pj);          /* send distance to proc j */
     }
```

• Ένας μηχανισμός χρειάζεται για να διαπιστωθεί ο τερματισμός ολόκληρου του υπολογισμού. Συγκεκριμένα, θα πρέπει ο υπολογισμός να τερματίζεται όταν όλες οι διεργασίες είναι αδρανείς και όταν δεν υπάρχουν μηνύματα υπό μεταφορά.

• Η απλούστερη λύση είναι να χρησιμοποιηθούν σύγχρονες ρουτίνες επικοινωνίας όπου μία διεργασία-αποστολέας δεν μπορεί να προχωρήσει μέχρι ο παραλήπτης να λάβει το μήνυμα που εστάλη.

• Σημειώνεται επίσης ότι μία διεργασία είναι ενεργή μόνο αφότου η κορυφή της έχει τοποθετηθεί στην ουρά.

• Επίσης με τη προτεινόμενη παράλληλη υλοποίηση, είναι πιθανό πολλές διεργασίες να είναι αδρανείς και επομένως η λύση να μην είναι ιδιαίτερα αποδοτική.

• Επίσης, η λύση είναι μη πρακτική για μεγάλου μεγέθους γραφήματα αν κάθε διεργασία αναλαμβάνει μία μόνο κορυφή. Αυτό μπορεί να λυθεί αν κάθε διεργασία αναλαμβάνει περισσότερες από μία κορυφές.