DISTRIBUTED OPERATING SYSTEMS

Mutual Exclusion

Lecture 5

همگام سازی فرآیند ها(Process)

برای همگام سازی فرآیند ها بحث دوبدوناسازگاری یا انحصار متقابل Mutual Exclusion مطرح می شود.

بخش بحرانی نه همزمان نه همروند در اختیار 2 تا فرآیند نباشد.

باید اجازه دهیم فقط یک فرآیند از بخش بحرانی استفاده کند در صورتی فرآیند بعد می تواند از بخش بحرانی استفاده کند که فرآیند اول کارش تمام شده باشد.

الگوریتم های انحصار متقابل

الگوریتم های انحصار متقابل توزیع شده:

1- روش مبتنی بر نشانه (Token based)

2- روش مبتنی بر اجازه (Permission Based)

روش مبتنی بر نشانه

از طریق ارسال پیام خاصی بین فرآیند ها انجام می شود که نشانه نام دارد.

فقط یک نشانه وجود دارد و هر کسی که آن نشانه را دارد می تواند به منبع مشترک دستیابی داشته باشد. پس از اتمام دستیابی، نشانه به فرآیند بعدی فرستاده می شود.

این روش جلوگیری از بن بست و گرسنگی را تضمین می کند.

عیب عمده این روش این است که، وقتی نشانه مفقود می شود، باید رویه توزیع شده ی دقیقی اجرا شود تا تضمین گردد که نشانه جدیدی ایجاد شده است و این نشانه یکتا است.

روش مبتنی بر اجازه

در این روش فرآیندی که منتظر دستیابی به منبع است، ابتدا نیاز به اجازه یک فرآیند هماهنگ کننده یا سایر فرآیند ها دارد.

انواع روشهایی که برای اعطای چنین اجازه ای وجود دارد:

1- الگوریتم متمرکز

2- الگوریتم نامتمرکز

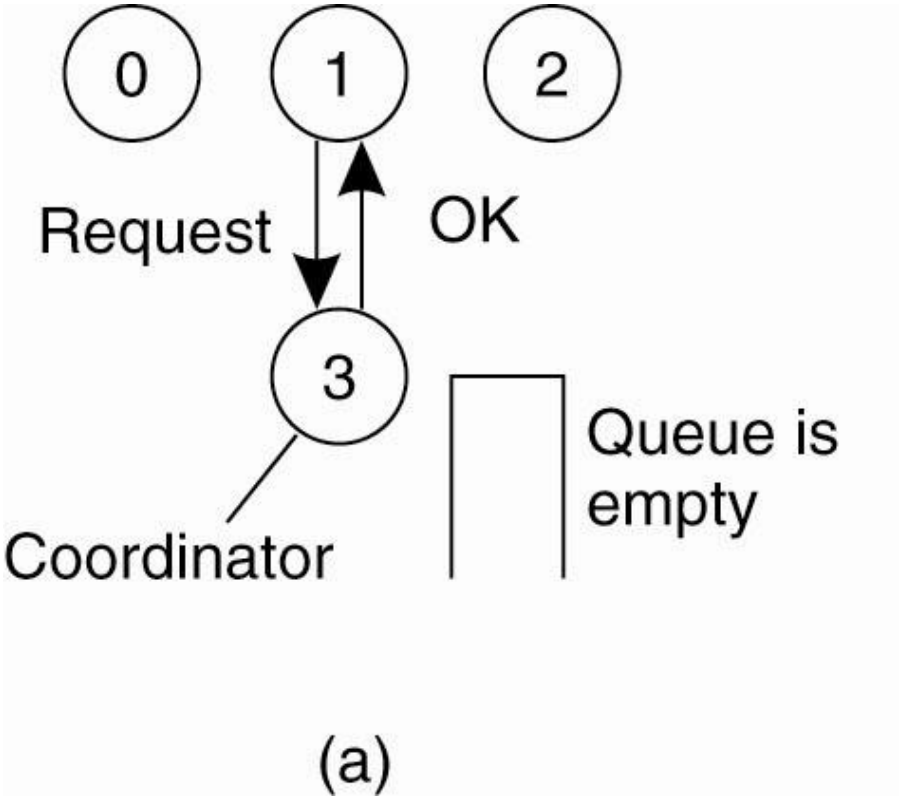
3- الگوریتم توزیع شده

الگوریتم متمرکز (1)

راحت ترین راه برای رسیدن به انحصار متقابل در سیستم توزیع شده، شبیه سازی چگونگی انجام آن در سیستم تک پردازنده ای است.

یک فرآیند به عنوان هماهنگ کننده انتخاب می شود. هر وقت فرآیندی می خواهد به منبع مشترکی دست یابد، پیام درخواست را به هماهنگ کننده می فرستد و می گوید به کدام منبع می خواهد دستیابی داشته باشد و کسب اجازه می کند.

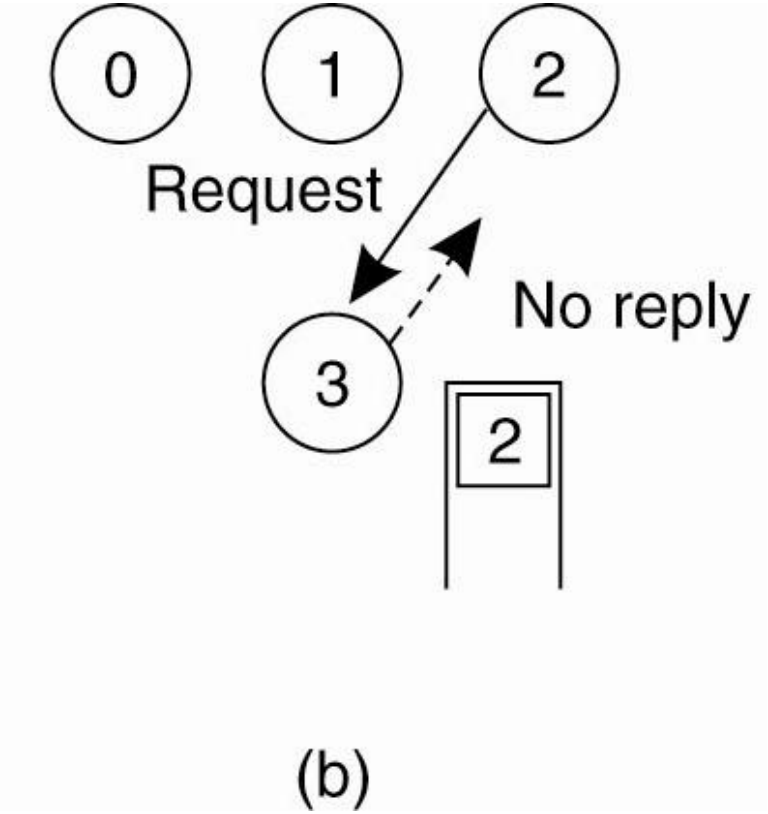
الگوریتم متمرکز (۲)



فرآیند ۱ از هماهنگ کننده اجازه می خواهد تا به منبع مشترک دست یابد.

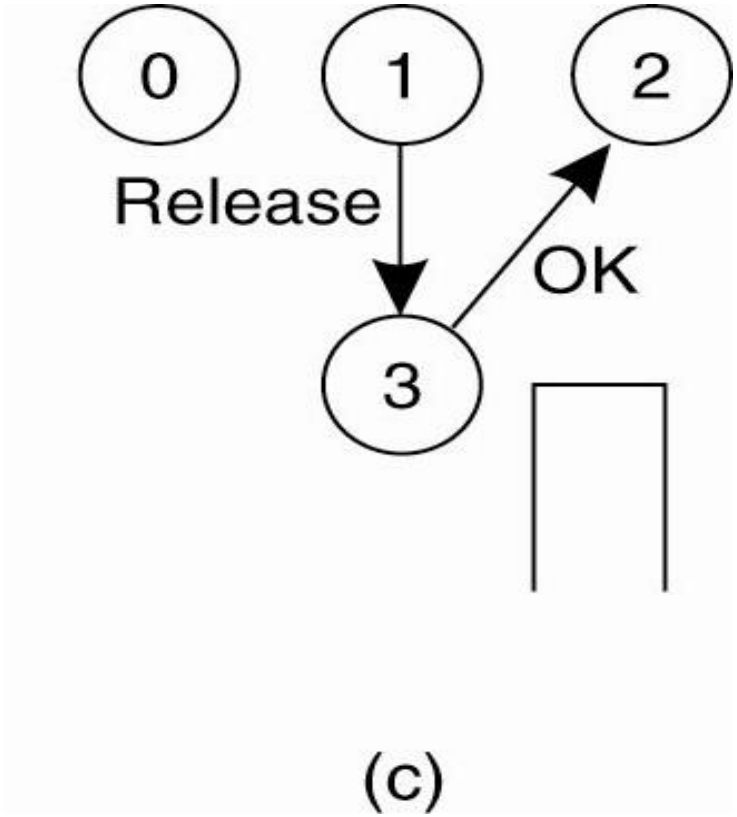
سپس هماهنگ کننده اجازه می دهد.

الگوریتم متمرکز (۳)



فرآیند ۲ اجازه می خواهد تا به همان منبع دست یابد. هماهنگ کننده پاسخ نمی دهد.

الگوریتم متمرکز (۴)



وقتی فرآیند ۱ منبع را آزاد می کند، به هماهنگ کننده اعلان می کند، که آن نیز به فرآیند ۲ پاسخ دهد.

ویژگی های الگوریتم متمرکز

پیاده سازی و مدیریت آن خیلی ساده است. انحصار متقابل را تضمین می کند.

گرسنگی وجود ندارد.

هماهنگ کننده، یک نقطه شکست دارد واگر خراب شود، کل سیستم از کار می افتد.

تنها یک هماهنگ کننده می تواند گلوگاه برای کارایی محسوب شود.

الگوریتم نامتمرکز

یک روش رأی گیری مبتنی بر سیستم DHT است و روش آن این است که هماهنگ کننده مرکزی را بسط می دهد.

فرض می شود هر منبع n بار تکثیر شده است. هر کپی دارای هماهنگ کننده خاص خودش برای کنترل دستیابی توسط فرآیندهای همزمان است.

هر وقت فرآیندی می خواهد به منبعی دستیابی داشته باشد،لازم است از هماهنگ کننده رأی جمع کند. $m>n/2$

ویژگی های الگوریتم نامتمرکز

نسبت به الگوریتم متمرکز کمتر دچار آسیب پذیری می شود. یک نقطه شکست **وجود ندارد.**

گلوگاه وجود ندارد (صف وجود ندارد).

الگوریتم توزیع شده (1)

وقتی فرآیندی می خواهد به منبع مشترکی دست یابد، پیامی می سازد که شامل نام منبع، شماره فرآیند آن و زمان (منطقی) فعلی است. سپس پیام را به تمام فرآیند ها گروه خودش ارسال می کند.

وقتی فرآیندی پیام درخواست را از فرآیند دیگری دریافت می کند، یکی از سه حالت مختلف آینده را انجام می دهد:

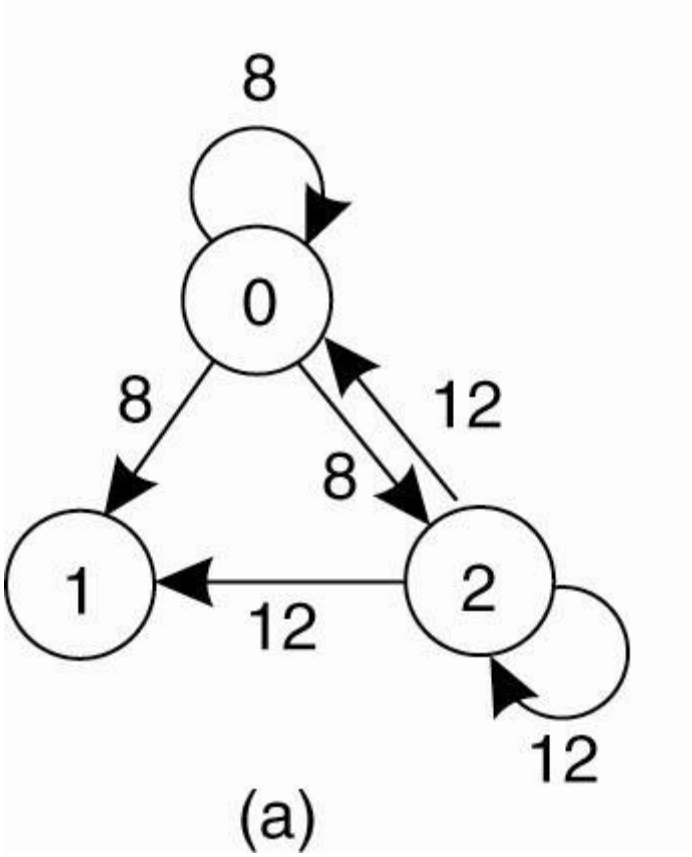
الگوریتم توزیع شده (۲)

۱- اگر گیرنده در حال دستیابی به منبع نباشد و نخواهد به آن دست یابد، پیام **OK** را به فرستنده ارسال می کند.

۲- اگر گیرنده به منبع دستیابی داشته باشد، در عوض، درخواست را در صف قرار می دهد.

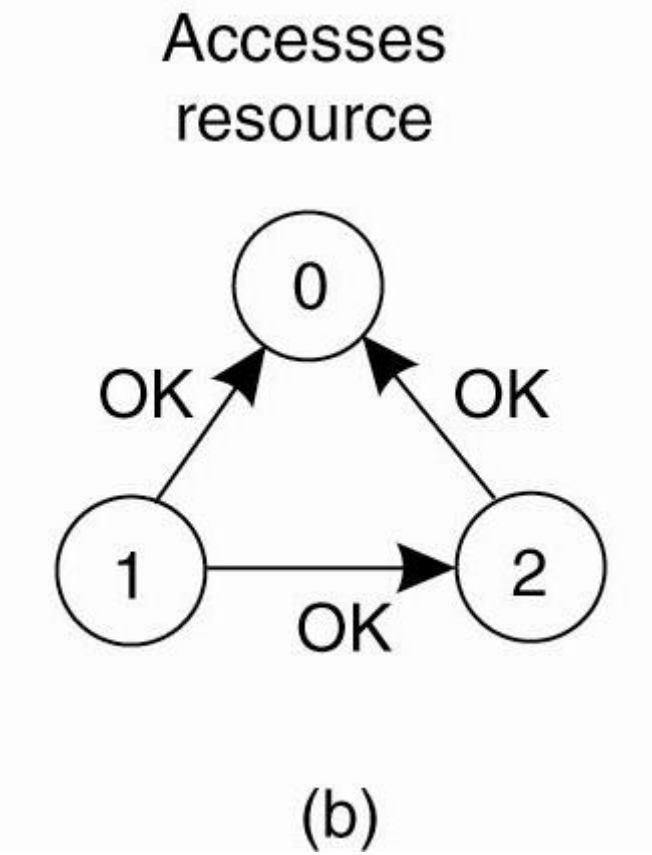
۳- اگر گیرنده بخواهد به منبع دستیابی داشته باشد ولی هنوز موفق نشده است، مهر زمان پیام ورودی را با مهر زمانی موجود در پیامی که به هر کسی فرستاده است، مقایسه می کند. کمترین مهر زمانی برنده است.

الگوریتم توزیع شده (۳)



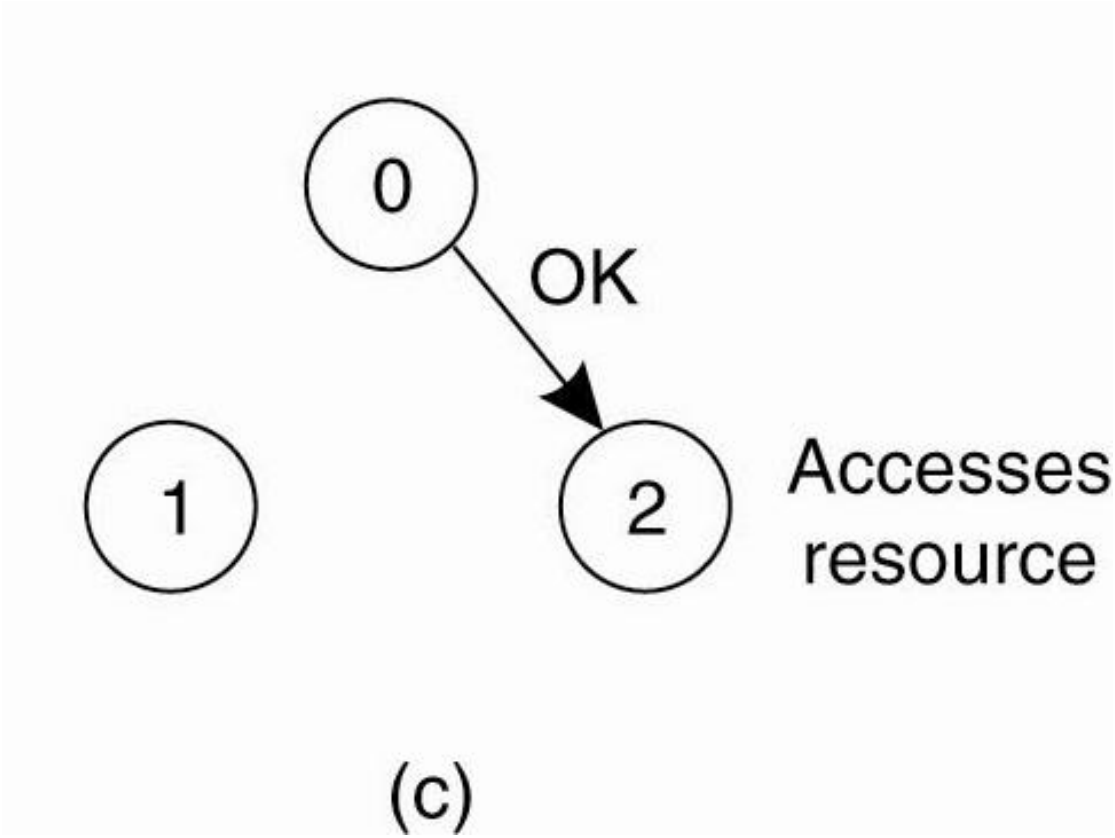
دو فرآیند می خواهند همزمان به منبع مشترک دستیابی داشته باشند.

الگوریتم توزیع شده (۴)



فرآیند صفر مهر زمان کمتری دارد، لذا برنده است.

الگوریتم توزیع شده (۵)



وقتی فرآیند صفر کارش را انجام داد، پیام OK را می فرستد و فرآیند ۲ می تواند به کارش ادامه دهد.

Maekawa's Algorithm (Quorum-based)

- **Request set**
  - each node has a request set (quorum)
  - when the node wants to enter the critical section, it sends its request to all nodes in its request set
  - the request set of each node does not include all nodes in the system
  - the intersection of any two request sets is non-empty
- **Example**
  - consider three nodes, X, Y, and Z
  - X’s request set include nodes X and Y
  - Y’s request set include nodes Y and Z
  - Z’s request set include nodes Z and X

Maekawa's Algorithm (Quorum-based)

- **Request the CS:**
  1. P_i multicasts *request* (t_i, i) to its request set, including itself
  2. P_j upon receiving the request
     a) if it is not currently locked, then locks itself and sends reply (j) to Pi
     b) otherwise, puts the request in a queue (in the order of the timestamp)
- **Enter the CS:**
  1. if P_i has received reply messages from all sites in its request set, then it enters the CS
- **Release the CS:**
  1. P_i upon exiting CS, sends *release* (t_i) to all processors in its request set
  2. P_j upon receiving the message
     a) if the waiting queue is not empty then it removes the entry in the queue and sends *reply* (j) to that node
     b) otherwise, unlocks itself

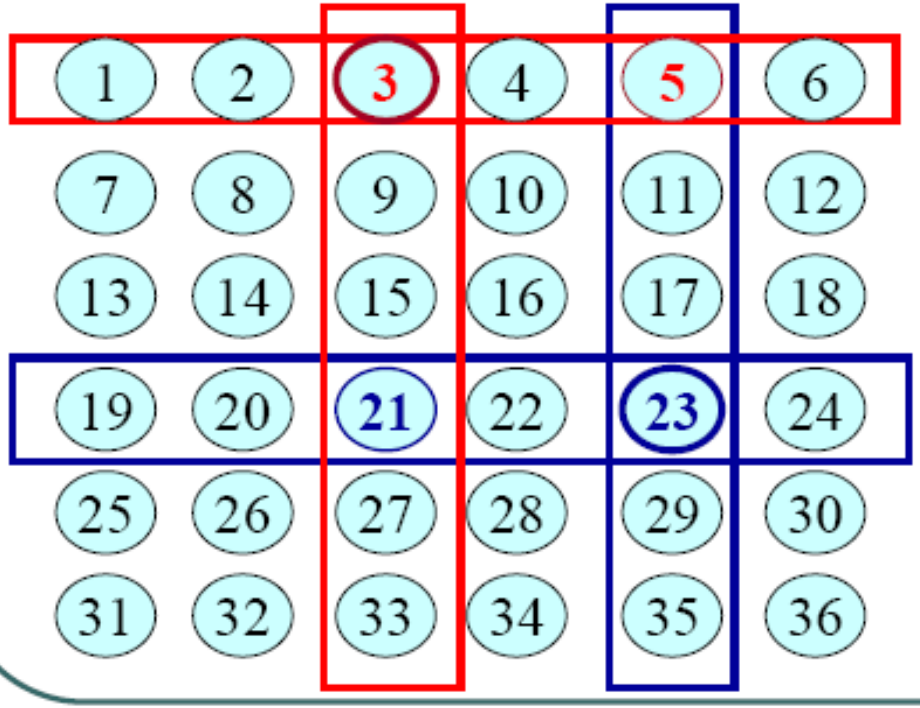
Maekawa's Algorithm Properties

- requires a total ordering of events
- requires $3\sqrt{N}$ messages per request
- response time in a very low load
  - 2T
  - send K–1 request messages sent in parallel (T)
  - send K–1 response messages sent in parallel (T)
- has the potential deadlock problem

Potential Deadlock Problem in Maekawa's Algorithm

Maekawa's algorithm

- Possibility of deadlock illustrated



- p_3 and p_{23} compete
- p_5 votes for p_3
- p_{21} votes for p_{23}

Neither p_3 nor p_{23} receives all the votes of its voting district: Deadlock

ویژگی های الگوریتم توزیع شده

سوال: آیا همواره الگوریتم توزیع شده از الگوریتم متمرکز بهتر است؟

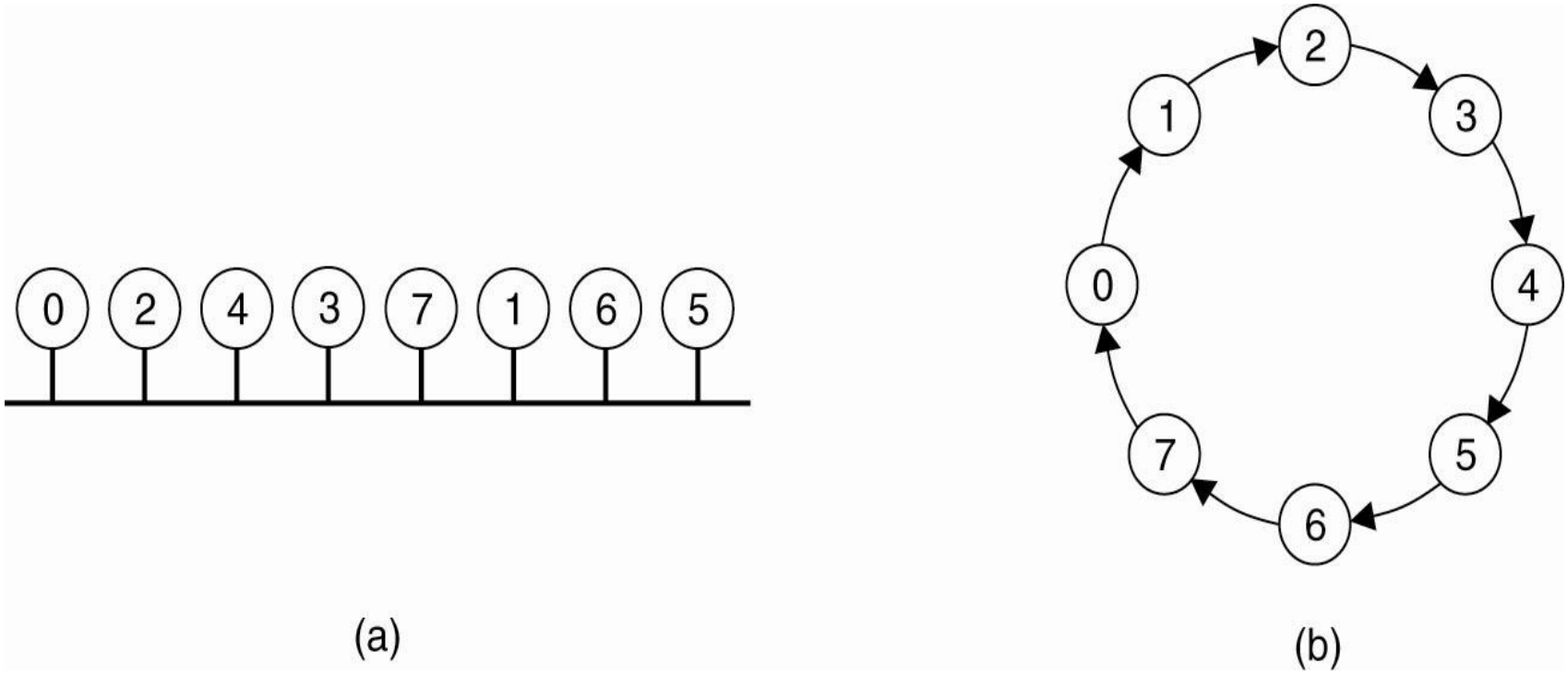
همانند الگوریتم متمرکز، انحصار متقابل بدون بن بست و گرسنگی تضمین می شود.

n نقطه شکست وجود دارد! و اگر یکی از کار بیفتد همه از کار می افتند.

حجم پیام ها خیلی بالا است.

نسبت به الگوریتم متمرکز کندتر، پیچیده تر، گران تر و با توانمندی کمتر است!

الگوریتم حلقه نشانه (1)



a) گروه نامرتبی از فرآیند ها در شبکه.

b) حلقه منطقی که در نرم افزار ساخته می شود.

الگوریتم حلقه نشانه (۲)

در شکل a یک شبکه خطی داریم (مثل اترنت) که فاقد ترتیب فرآیند ها است.

در شکل b به هر فرآیند مکانی در حلقه داده می شود. موقعیت های حلقه ممکن است به ترتیب عددی آدرس های شبکه یا ابزارهای دیگری تخصیص یابد. مهم نیست که ترتیب چه باشد. مهم این است که هر فرآیند می داند بعد از آن چه فرآیندی قرار دارد.

الگوریتم حلقه نشانه (۳)

وقتی حلقه آماده شد، به فرآیند 0 ، یک نشانه داده می شود. این نشانه در حلقه دور می زند. از فرآیند k به فرآیند k+1 می رود، و بررسی می کند که آیا این فرآیند نیاز به دستیابی به منبع مشترک دارد یا خیر. اگر نیاز داشته باشد، فرآیند اجرا می شود، پس از اتمام کارهایش، منابع را آزاد می کند. پس از پایان کار، نشانه را در ادامه حلقه آزاد می کند.

ویژگی های الگوریتم حلقه نشانه

در هر زمان فقط یک فرآیند نشانه را در اختیار دارد.

گرسنگی رخ نمی دهد.

اگر نشانه مفقود شود، باید دوباره تولید کرد. (تشخیص مفقود شدن نشانه دشوار است)

اگر فرآیندی خراب شود، الگوریتم دچار درد سر می شود، اما ترمیم آن نسبت به موارد دیگر آسان تر است.

Raymond's Tree-Based Algorithm

- the processors are structured as a tree and the token is placed at the root node
- the tree restructures when the token moves
- Request the CS (going up the tree):
  1. P_i send *request* (*i*) to its parent and puts the request in its queue if it does not hold the token
  2. P_j upon receiving the request
     a) puts the request in its queue
     b) if it has not sent a request to its parent then
        - sends *request* (*j*) to its parent

     c) otherwise (a request has already been sent to its parent for another child node)
        - does nothing

Raymond's Tree-Based Algorithm

- Request the CS (going down the tree):
  3. root site upon receiving the request
     a) puts the request in its queue
     b) executes (DTPR)
  4. P_j, upon receiving the token,
     a) if it was not requesting to enter CS or its request was not on the top of its queue then executes (DTPR)
- D. delete the top entry from its requesting queue
- T. send the token to the requesting child
- P. update parent pointer to point to the requesting child
- R. if its request queue is non-empty then send a request to the new parent

Raymond's Tree-Based Algorithm

- Enter the CS:

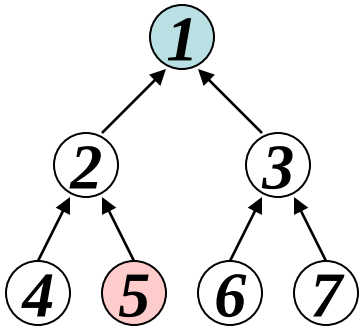
  if P_i has received the token and its request is on the top of its queue then it enters the CS

- Release the CS:

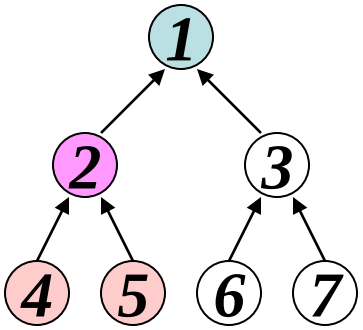
  P_i upon exiting CS

  if its queue is not empty, then executes (DTPR)

Raymond's Tree-Based Algorithm-- Example

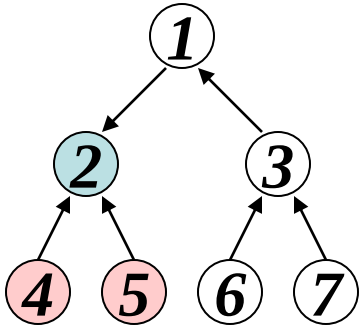


1. token is at node 1
node 5 made a request

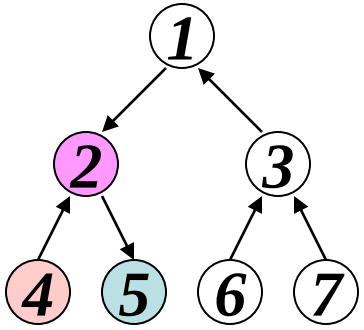


2. node 2 receives the request, it sends the request to node 1

3. node 4 also sends a request, node 2 receives it

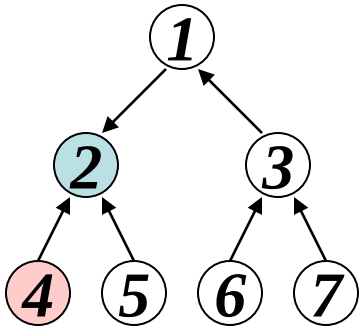


4. token is at node 2 now
node 2 becomes the root

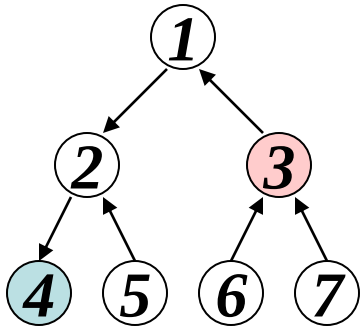


5. node 5 gets the token, it enters CS
6. node 2 sends a request to node 5

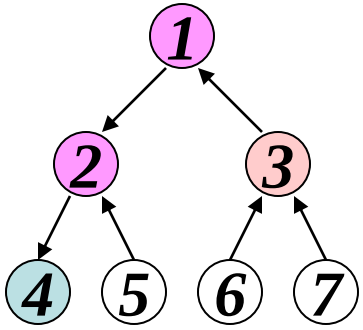
Raymond's Tree-Based Algorithm-- Example



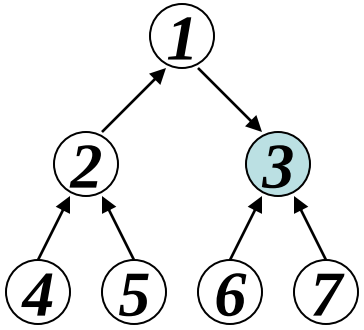
7. node 5 sends the token to node 2



8. node 4 gets the token, it enters CS
9. node 3 sends a request



10. the request from node 2 comes to node 4



11. node 3 gets the token, and becomes the root

Raymond's Tree-Based Algorithm -- Properties

- the node with the token is always the root node
- requires the nodes on the entire path, from requester to root, to be alive in order to process a request
  - still has the lost token problem
- requires 2 log*N* messages per request in average
  - longest path: 2 log*N* (when the root is at the leaf of the original tree)
  - best case: 0 messages
  - worst case: 4 log*N* messages (2 log*N* to the root, 2 log*N* back with token)
- response time
  - the message passing has to be done sequentially
  - the average response time: T log*N*
  - the best case response time: 0
  - the worst case response time: 4T log*N*

مقایسه چهار الگوریتم انحصار متقابل

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Decentralized	3mk, k = 1,2,...	2 m	Starvation, low efficiency
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

الگوریتم انتخاب (Election)

در اغلب الگوریتم های توزیع شده لازم است یک فرآیند به عنوان هماهنگ کننده یا آغاز کننده عمل کند.

اگر تمام فرآیند ها دقیقاً یکسان باشند، هیچ راهی برای انتخاب آن ها وجود ندارد.

بطور کلی، الگوریتم انتخاب سعی می کند فرآیندی با شماره بالاتر را به عنوان هماهنگ کننده انتخاب نماید.

الگوریتم های انتخاب سنتی

1- الگوریتم توانمند (Bully Algorithm)

2- الگوریتم حلقه (Ring)

الگوریتم توانمند (Bully)

همیشه توانمند باقی می ماند.

وقتی فرآیندی می بیند که هماهنگ کننده به درخواست ها پاسخ نمی دهد، انتخاب را شروع می کند.

در هر لحظه، فرآیند می تواند پیام ELECTION را از یکی از همکاران با شماره پایین بگیرد. وقتی چنین پیامی می رسد، گیرنده، پیام OK را به فرستنده ارسال می کند تا نشان دهد که زنده است و آن را تحویل خواهد گرفت.

الگوریتم توانمند (Bully) – ادامه

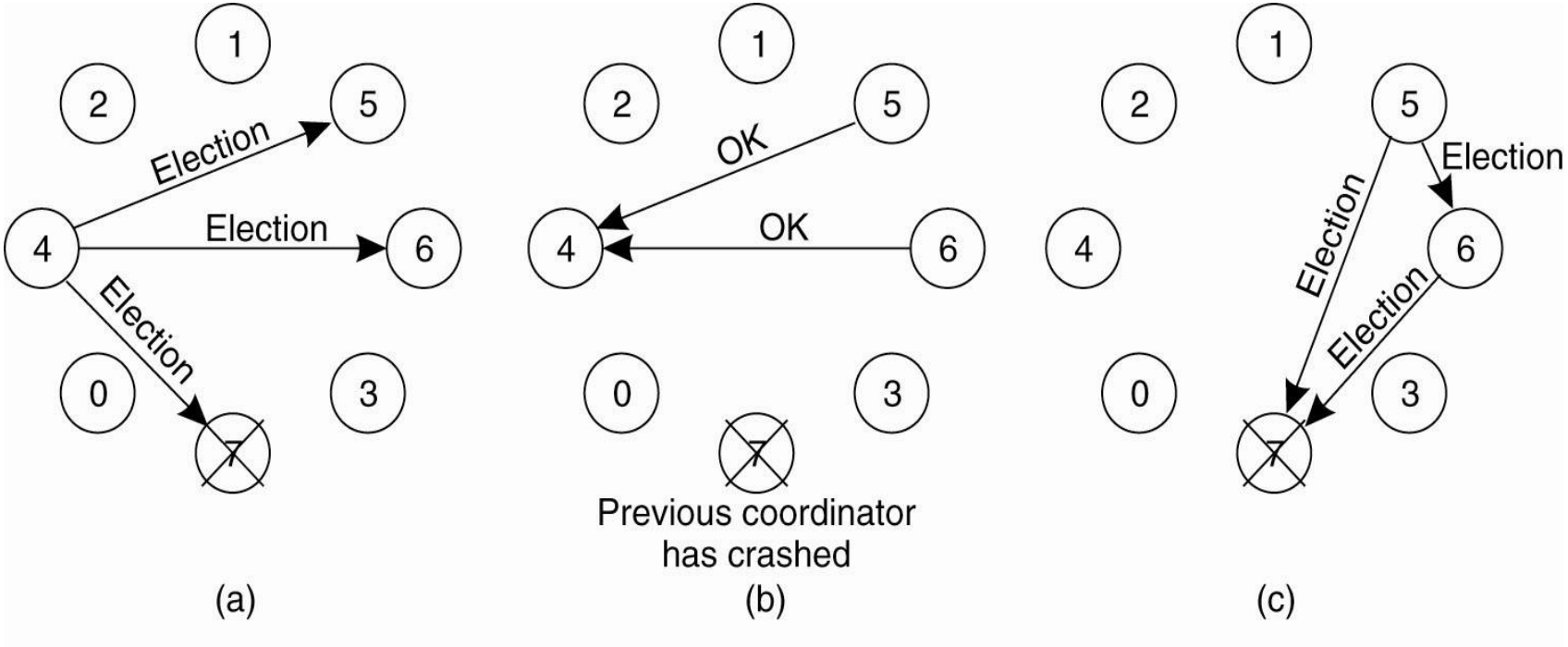
فرآیند P ، انتخاب را به صورت زیر انجام می دهد:

۱- P پیام ELECTION را به تمام فرآیندهایی با شماره بالاتر ارسال می کند.

۲- اگر هیچ کدام پاسخ ندهند، P انتخاب را برنده می شود و به عنوان هماهنگ کننده عمل می کند.

۳- اگر یکی از آن ها پاسخ دهد به عنوان هماهنگ کننده انتخاب می شود و کار P خاتمه می یابد.

الگوریتم توانمند (Bully) – ادامه

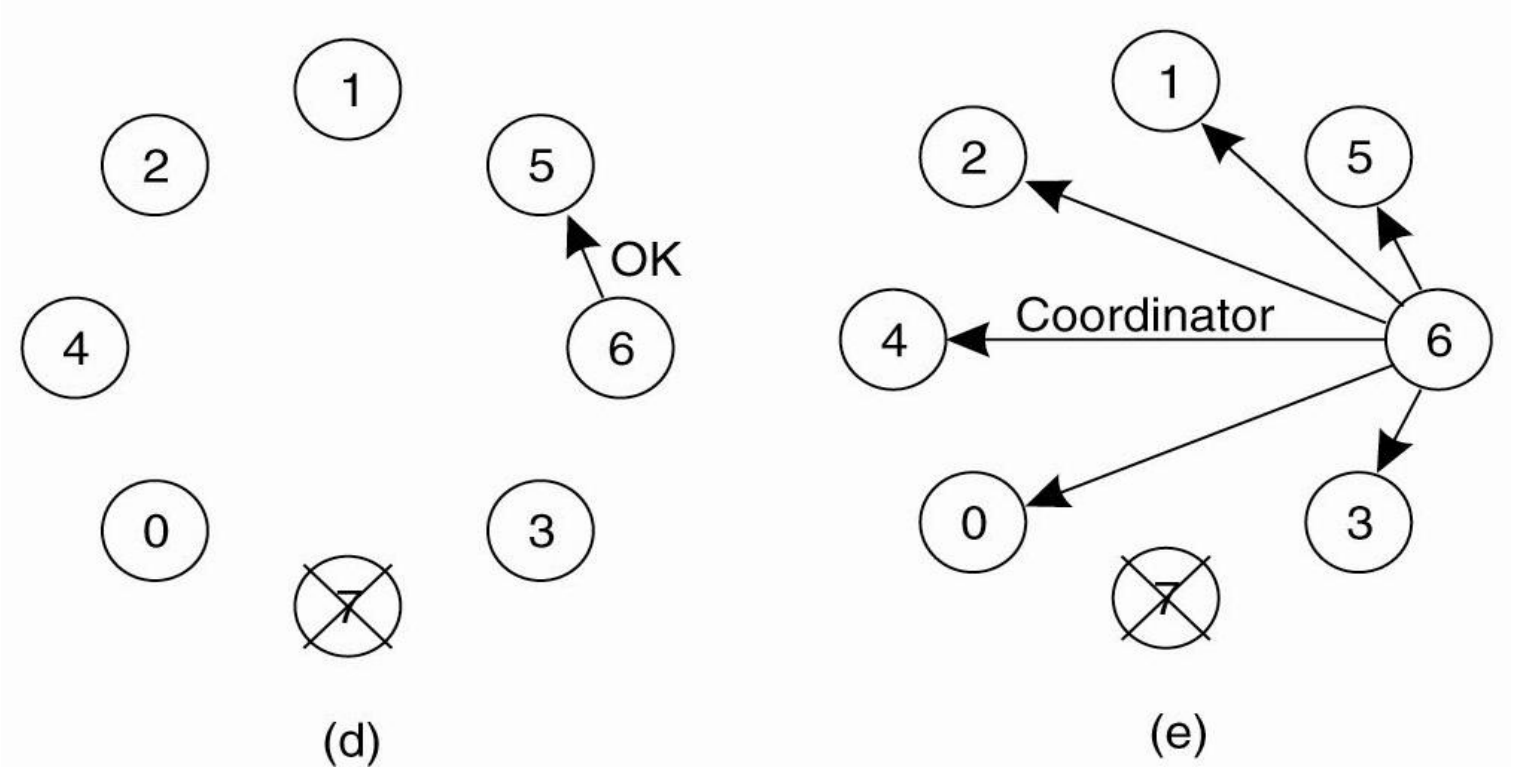


a) فرآیند 4 انتخابات را انجام می دهد.

b) فرآیند های 5 و 6 پاسخ می دهند و به 4 می گویند که متوقف شود.

c) اکنون 5 و 6 انتخابات را انجام می دهند.

الگوریتم توانمند (Bully) – ادامه



d) فرآیندهای ۶ به ۵ می گوید که متوقف شود.

e) فرآیند ۶ برنده می شود و به همه خبر می دهد.

الگوریتم حلقه (۱)

این الگوریتم از نشانه استفاده نمی کند.

فرض می کنیم فرآیندها به طور منطقی یا فیزیکی مرتب اند، به طوری که هر فرآیند می داند بعدی او کدام است.

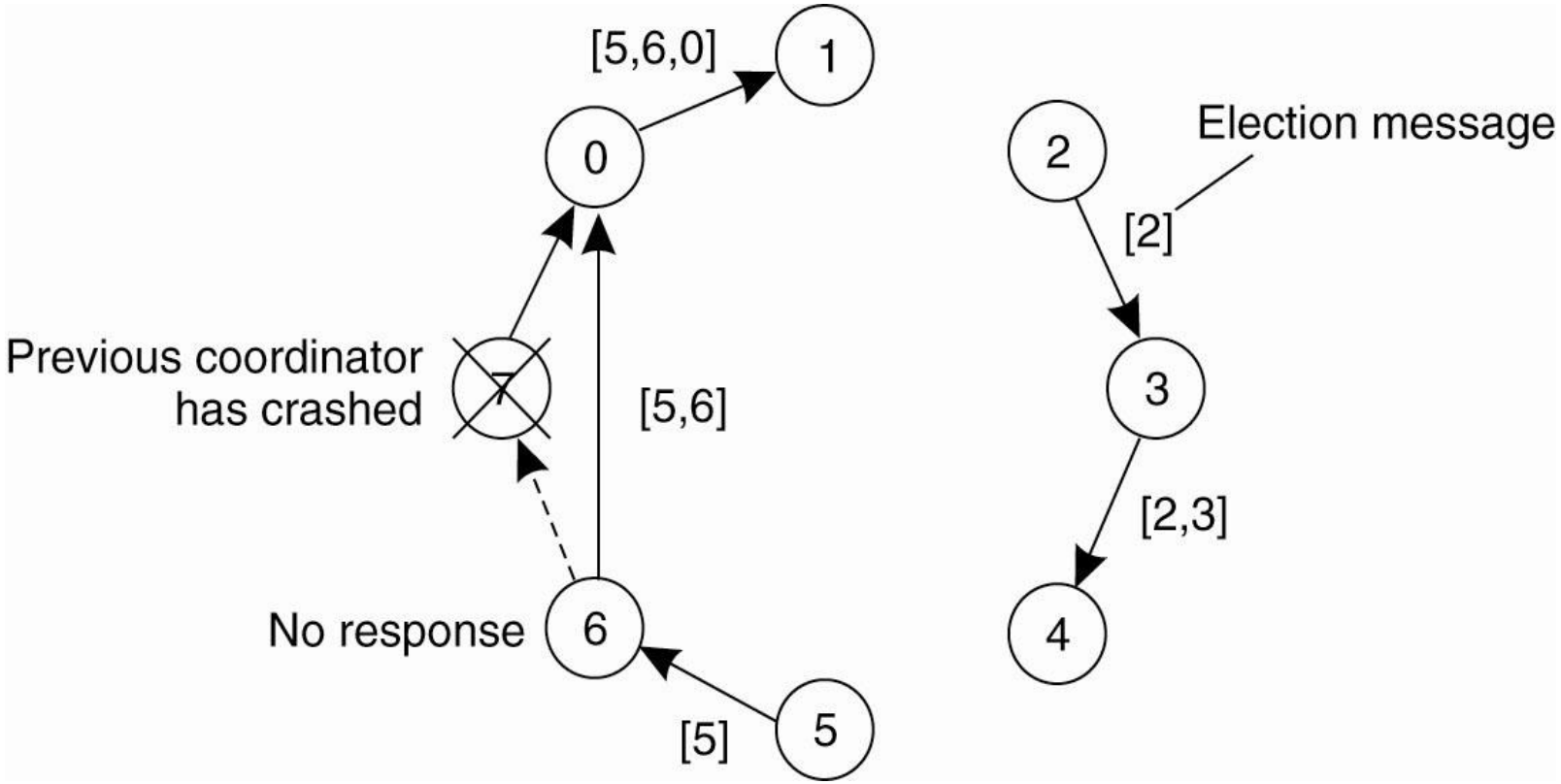
الگوریتم حلقه (۲)

وقتی فرآیندی متوجه می شود که هماهنگ کننده عمل نمی کند، خودش را کاندیدای انتخاب هماهنگ کننده معرفی می کند یک پیام ELECTION می سازد که شامل شماره فرآیند خودش است و آن را به بعدی خودش می فرستد.

اگر بعدی او از بین برود، فرستنده از آن عبور می کند و به عدد بعدی در حلقه یا عدد بعد از خودش می رود تا یک فرآیند در حال اجرا پیدا شود.

سرانجام پیام به فرآیندی بر می گردد که آن را شروع کرد.

الگوریتم حلقه (۳)



الگوریتم انتخاب با استفاده از حلقه