Asynchronous Group Mutual Exclusion in Ring Networks (Extended Abstract)

Kuen-Pin Wu
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
kpw@orchid.ee.ntu.edu.tw

Yuh-Jzer Joung*
Department of Information Management
National Taiwan University
Taipei, Taiwan
joung@ccms.ntu.edu.tw

Abstract

The design issues for group mutual exclusion have been modeled by Joung as the Congenial Talking Philosophers*, and solutions for shared-memory models and complete message-passing networks have been proposed [2, 3]. These solutions, however, cannot be straightforwardly and efficiently converted to ring networks where each philosopher can only communicate directly with its two neighboring philosophers. As rings are also a popular network topology, in this paper we focus the Congenial Talking Philosophers on ring networks and present an efficient and highly concurrent distributed algorithm for the problem.*

1 Introduction

The design issues for mutual exclusion between groups of processes have recently been modeled by Joung [2] as the *Congenial Talking Philosophers*, which concerns a set of N philosophers $p_1, p_2, \ldots, p_N$ which spend their time thinking alone and talking in a forum. Initially, all philosophers are thinking. From time to time, when a philosopher is tired of thinking, it wishes to attend a forum of its choice. Given that there is only one meeting room, a philosopher attempting to enter the meeting room to attend a forum can succeed only if the meeting room is empty (and in this case the philosopher starts the forum), or some philosopher interested in the same forum is already in the meeting room (and in this case the philosopher joins this ongoing forum). We assume that when a philosopher has attended a forum, it spends an unpredictable but finite amount of time in the forum. After a philosopher leaves a forum (and exits the meeting room)[1], it returns to thinking. The problem is to design an algorithm for the philosophers satisfying the following requirements:

mutual exclusion: if some philosopher is in a forum, then no other philosopher can be in a different forum simultaneously.

bounded delay: a philosopher attempting to attend a forum will eventually succeed.

concurrent entering: if some philosophers are interested in a forum and no philosopher is interested in a different forum, then the philosophers can attend the forum concurrently.

Solutions for the Congenial Talking Philosophers problem for shared-memory models and complete message-passing networks have been proposed [2, 3]. These solutions, however, cannot be straightforwardly and efficiently converted to ring networks where each philosopher can only communicate directly with its two neighboring philosophers. As rings are also a popular network topology, in this paper we focus the problem on ring networks where the N philosophers $p_0, p_1, \ldots, p_{N-1}$ are connected via a ring network so that each philosopher p_i can only send messages directly to its immediate successor p_{i+1}. (Unless stated, otherwise, additions and subtractions on indices of philosophers are to be interpreted modulo N.) We shall assume that philosophers are distinguished by their unique IDs, and that every message will eventually be delivered to its destination. However, we do not require messages to be delivered in the order sent.

As we shall see, some solutions for the Congenial Talking Philosophers problem may be easy to design. However, their "performance" in practical applications may be unsatisfactory. So Joung has also proposed four measurements to evaluate the algorithms. We shall briefly discuss these measurements in the following section. For more details, please refer to [2, 3]. Section 3 presents a straightforward solution to the Congenial Talking Philosophers problem. The solution easily satisfies the three problem requirements, and has complexity measures that "appear" to be acceptable. However, when putting the algorithm in simulation, its weakness becomes evident: the performance is only slightly better than imposing strict mutual exclusion on every entry to the meeting room. A more concurrent solution is therefore presented in Section 4. Section 5 concludes.

2 Complexity Measures

Solutions for the Congenial Talking Philosophers problem can be evaluated from the following four perspectives:

*This research was supported in part by the National Science Council, Taipei, Taiwan, under Grants NSC 85-2213-E-002-059 and NSC 86-2213-E-002-053.

[1]Throughout the paper, "in a forum" is synonymously with "in the meeting room." So, "to attend/leave a forum" is synonymously with "to enter/exit the meeting room."

messages, *time*, *context-switch*, and *degree of concurrency*.

Message complexity is concerned with how many messages the system generates per entry to the critical section—the meeting room.

For time, one is concerned with how long a philosopher p_i should wait before it enters the critical section from the time it has made a request (by sending out messages to other philosophers). More formally, define a *passage through* X by p_i to be an interval $[t_1, t_2]$, where t_1 is the time p_i enters the meeting room to attend X, and t_2 is the time it exits the meeting room. The *attribute* of the passage is denoted by $\langle p_i, \mathsf{X}\rangle$. When no confusion is possible, intervals and attributes are used interchangeably to represent passages. We often say that the passage is *initiated* at t_1, and is *completed* at t_2. The passage is *ongoing* at any time in between t_1 and t_2.

Let T be the set of passages that may be initiated after a philosopher p_i has made a request for a forum, and that must be completed before p_i can enter the meeting room. The *time* complexity is measured by $\Omega(T)$, where $\Omega(T)$ is the size of a minimal set $R \subset T$ such that for every passage $\alpha \in T$, every time instance in α is contained in some $\beta \in R$.

The *context-switch complexity* is measured by the maximum number of rounds of passages that may be initiated after a philosopher has made a request for X, but before a round of X is initiated in which p_i can make a passage through X. A *round of passages through* Y (or simply a round of Y) is a maximal set of consecutive passages through forum Y.

The *(maximum) degree of concurrency* is defined by the maximum number of entries to the meeting room that can still be made while some philosopher is in the meeting room and another philosopher is waiting for a different forum.

Note that the above complexity measures all concern worst-case scenarios. Due to the dynamic nature of the problem, an average-case analysis is extremely complicated and so simulation studies [5] are also suggested by Joung [3].

3 A Straightforward Solution

We first present a straightforward solution to the Congenial Talking Philosophers problem over a ring network. In the algorithm, which we shall refer to as CTP-Ring1 (where CTP stands for Congenial Talking Philosophers), each philosopher is in one of the following three states: $thinking$, meaning that it is not interested in any forum; $waiting$, meaning that it is waiting for a forum; and $talking$, meaning that it is in a forum. Moreover, each philosopher maintains a variable SN recording the maximum sequence number seen by the philosopher. The variable will be maintained similar to Lamport's logical clock [4]. That is, each philosopher initializes its SN to 0 and, whenever it learns of a sequence number larger than its own SN, it advances its SN to that number. Furthermore, a philosopher increments its SN by 1 when it wishes to attend a forum.

Initially, every philosopher is in state $thinking$. When a philosopher p_i wishes to attend a forum X, it enters state $waiting$, increments its SN by 1, and sends a request message $Req(\langle i, sn\rangle, \mathsf{X})$ to its successor p_{i+1}, where sn is the new value of p_i's SN. Philosopher p_i remains in state $waiting$ until its request $Req(\langle i, sn\rangle, \mathsf{X})$ is returned. Then, p_i enters the meeting room to attend X. After finishing the forum, p_i returns to state $thinking$.

When p_{i+1} receives a request message $Req(\langle j, sn_j\rangle, \mathsf{X})$ from p_i, it either forwards the request to its successor p_{i+2}, or detains the request in its message queue; the decision depends on p_{i+1}'s state. p_{i+1} forwards the request if one of the following conditions is satisfied: (1) it is in state $thinking$, (2) it is also interested in X, or (3) it is interested in a different forum and has a priority lower than p_j. A philosopher's priority is assigned as follows: when the philosopher has sent out a request $Req(\langle j, sn_j\rangle, \mathsf{X})$, it obtains a priority $\langle j, sn_j\rangle$, and it possesses the priority until it has left X. A priority $\langle j, sn_j\rangle$ is higher than $\langle k, sn_k\rangle$, denoted by $\langle j, sn_j\rangle > \langle k, sn_k\rangle$ if, and only if, $sn_j < sn_k$, or $sn_j = sn_k$ and $j < k$. For simplicity, we assume that a philosopher's priority is set to a minimal value $\langle i, \infty\rangle$ when it is not interested in any forum; that is, when it is in state $thinking$.

On the other hand, p_{i+1} detains the request if none of the above three conditions is satisfied. In other words, p_{i+1} detains the request if it is interested in a different forum Y and has a priority higher than $\langle j, sn_j\rangle$. p_{i+1} detains the request until it has finished Y. Then, p_{i+1} forwards the request (and all other requests it has held) to its successor p_{i+2}.

It can be seen that the algorithm satisfies all three requirements of the Congenial Talking Philosophers problem. For mutual exclusion, observe that by the way sequence numbers are maintained, if a philosopher p_j requests a forum after it has received a request $Req(\langle i, sn\rangle, \mathsf{X})$ issued by p_i, then p_j must obtain a priority for its request lower than $\langle i, sn\rangle$. So, when p_j's request circles to p_i, if the request is for a forum different from X, then the request must be detained by p_i until p_i has finished X for its request $Req(\langle i, sn\rangle, \mathsf{X})$. Therefore, p_j cannot attend a different forum while p_i is in X. Moreover, given that priorities are unique, when two philosophers request different forums concurrently, the request issued by the low-priority philosopher will be detained by the high-priority philosopher until the high-priority one has finished a forum. So, the two philosophers cannot be in different forums simultaneously. Because neither logically dependent requests nor concurrent requests can violate mutual exclusion, mutual exclusion is guaranteed.

For bounded delay, observe that a philosopher releases all detained messages to its successor when it finishes a forum. Given that a philosopher spends only a finite time in a forum and that every message will eventually be delivered to its target, a philosopher that has issued a request along the ring will eventually receive the request and, so, will eventually attend a forum.

For concurrent entering, observe that philosophers will not detain one another's request if they are interested in the same forum. Given that philosophers not interested in a forum will not detain any request, if some philosophers are interested in the same forum and no philosopher is interested in a different forum, then the philosophers can attend the forum concurrently.

We now consider various complexity measures of CTP-Ring1. It is easy to see that the algorithm requires

N messages for each philosopher to attend a forum. For time and context-switch complexity, we first calculate the maximum number of passages that may proceed p_i's passage through X after p_i has made a request $Req(\langle i, sn_i\rangle, \mathsf{X})$. Suppose that another philosopher p_j has also made a request $Req(\langle j, sn_j\rangle, \mathsf{Y})$. Obviously, if $sn_j > sn_i$, then p_j cannot make a passage through Y until p_i has made a passage through X (unless $\mathsf{Y} = \mathsf{X}$). So assume that $sn_j \leq sn_i$. Since messages need not be transmitted in the order sent, p_j may receive its request while p_i's request is still on the way to, say, p_{i+1}. So p_j may not have any knowledge about p_i's sequence number sn_i. Therefore, p_j's next request may still have a sequence number less than or equal to sn_i (but greater than sn_j). So p_j can still make another passage before p_i can attend X. Note that even if the sequence number of the new request is equal to sn_i, p_j may still attend a forum before p_i does if $j < i$. Hence, p_j can generate at most $sn_i - sn_j + 1$ passages before p_i can attend X. Note that there is no restriction on the type of forums through which the $sn_i - sn_j + 1$ passages are made. So, the $sn_i - sn_j + 1$ passages may constitute $sn_i - sn_j + 1$ rounds.

Moreover, by the way sequence numbers are maintained and that every request message must circulate along the ring once, if there is no pending request then all philosophers have the same sequence number maintained by their SNs. As the maximum SN in the system increases by 1 only if the philosopher holding this number has initiated a new request, in the worst case, the difference between any two SNs cannot exceed $N - 1$. To reach this case, some philosopher must have initiated a request, yielding its SN to some number c; the request then causes the next philosopher to increase its SN to $c + 1$ when initiating another request; the second request then causes the next philosopher to increase its SN to $c + 2$ when initiating a third request; and so on. So, in the worst case, when some philosopher p_i has initiated a request with sequence number sn, the other $N - 1$ philosophers may have all made a request, and have the following sequence numbers $sn-(N-1)$, $sn-(N-2)$, $\ldots$, and $sn-1$ maintained by their SNs. If p_i's ID is the smallest, then, by the previous argument, at most $N+(N-1)+\ldots+2 = \frac{N^2+N-2}{2}$ passages may be initiated before p_i can make a passage through a forum. As the $\frac{N^2+N-2}{2}$ passages need not overlap or aiming at the same forum, both the time complexity and context-switch complexity of CTP-Ring1 are $O(N^2)$.

To measure the algorithm's degree of concurrency, assume that p_j is in the meeting room, and p_i is waiting for a different forum. Let sn be the sequence number of p_i's request. Similar to above argument, the other $N - 2$ philosophers could all have made a request with sequence numbers $sn-(N-2)$, $sn-(N-3)$, $\ldots$, and $sn-1$. Then, at most $(N-1)+(N-2)+\ldots+2 = \frac{N^2-N-2}{2}$ passages may be initiated before p_i can attend its forum. So the degree of concurrency is also $O(N^2)$.

Some simulation results are summarized in Figure 1. In the simulation, we have set up a system of N philosophers and m forums. Each time a philosopher wishes to attend a forum, it randomly chooses one of the m forums to attend, and the choice follows a uniform distribution. The time a philosopher stays in states *thinking* and *talking* follows an exponential distribution with means $\mu_{thinking}$ and $\mu_{talking}$ respectively. The message transmission time also follows an exponential distribution with a mean μ_{link_delay}. For comparison, we have also measured the behavior of the algorithm when philosophers use the meeting room in a mutually exclusive style. This is done by designating one unique forum to each philosopher. In the figure, we use $m = N^*$ to denote this scenario. We have also set up the case $m = 1$ where maximum concurrency should be allowed as no two requests will ever conflict. The average round size measures the average number of passages per round; the average capacity measures, in average, the maximum number of philosophers that can be in the meeting room simultaneously per round. It is not difficult to see that the settings in the figure represent a very high contention situation to the meeting room. From the tables we can see that CTP-Ring1 provides virtually no concurrency. For example, when $m = 3$, it is likely that one third of the pending requests are targeting at the same forum very often. However, the simulation results indicate that the behavior of the system is only slightly better than the case $m = 30^*$ where the philosophers use the meeting room in a mutually exclusive style![2]

$N = 30,\ \mu_{thinking} = 50\text{ms},\ \mu_{talking} = 250\text{ms},\ \mu_{link\_delay} = 2\text{ms}$

	$m=1$	$m=3$	$m=10$	$m=30^*$
average waiting time (ms)	20.12	6108.46	7292.95	7873.7
average context switches	0	19.08	25.91	28.86
average round size	NA	1.51	1.11	1
average capacity	30	1.49	1.11	1
average messages per entry	30	30	30	30
average throughput (entry/sec)	92.13	4.66	3.95	3.67

Simulation results for Algorithm CTP-Ring1.

$N = 30,\ \mu_{thinking} = 50\text{ms},\ \mu_{talking} = 250\text{ms},\ \mu_{link\_delay} = 2\text{ms}$

	$m=1$	$m=3$	$m=10$	$m=30^*$
average waiting time (ms)	84.92	1538.9	3227.53	7899.85
average context switches	0	1.93	5.34	28.87
average round size	NA	14.64	5.37	1
average capacity	30	13.11	5.06	1
average messages per entry	60	60	60	60
average throughput (entry/sec)	76.66	16.06	8.46	3.65

Simulation results for Algorithm CTP-Ring2.

Figure 1. Some simulation results

To see why CTP-Ring1 has such a poor performance, observe that philosophers' requests are granted according to their priorities: a high-priority philosopher must enter the meeting room before a low-priority one. As a result, suppose that a number of philosophers are interested in two different forums (say, X and Y), and their priorities are ordered in such a way that two consecutive philosophers in the ordering are interested in different forums (that is, the first philosopher is interested in X, the second philosopher is interested in Y, the third philosopher is interested in X, the fourth philosopher is interested in Y, and so on.) Then, the philosophers would proceed their forums in a purely interleaving and sequential style: the first philosopher estab-

[2]Some Java applets animating the algorithms for the Congenial Talking Philosophers problem presented in the paper can be found in `http://joung.im.ntu.edu.tw/congenial/`.

lishes forum X, then the second philosopher establishes forum Y, then the third philosopher establishes forum X, then the fourth philosopher establishes forum Y, and so on. Note that, for this case no concurrency is allowed by the algorithm because only one philosopher can be in a forum at a time, albeit there are several other philosophers interested in the same forum. In the following section we present a modification of the algorithm that requires $2N$ messages per request, but offers a much better performance.

4 An Improved Algorithm

As discussed above, the low concurrency of CTP-Ring1 is due to the fact that, if two philosophers p_i and p_j are interested in the same forum but there is a third philosopher p_k interested in a different forum and has a priority in between p_i and p_j, then p_i and p_j cannot attend the same forum concurrently because the low-priority philosopher p_j must wait for p_k to finish a forum before it can attend a forum. Our solution for the problem is to let p_i issue another message along the ring to "capture" p_j (and other philosophers that are also interested in the same forum) so that the captured philosophers can attend the forum concurrently with the captor.

To do so, we introduce a new state *checking* for the philosophers. As before, a philosopher's priority is set to $\langle i, sn\rangle$ when it issues a request $Req(\langle i, sn\rangle, \mathsf{X})$. However, the priority may be "upgraded" (but at most once) before the philosopher attends forum X. The priority is reset to a minimal value $\langle i, \infty\rangle$ when the philosopher returns to state *thinking*. (Thus, a philosopher carries a priority all the time. In particular, its priority in state *thinking* is always lower than that of a philosopher interested in a forum.)

Like the previous algorithm, when a philosopher p_i wishes to attend a forum X, it enters state *waiting*, increments its SN by 1, and sends a request message $Req(\langle i, sn\rangle, \mathsf{X})$ to its successor p_{i+1}, where sn is the new value of p_i's SN. Another philosopher p_j in the ring, upon receiving the request, forwards the request if it is also interested in X or has a priority lower than $\langle i, sn\rangle$; otherwise, p_j detains the request in its message queue until it has finished a forum. When the request is returned to p_i, p_i enters state *checking* to issue a confirmation message of the form $Confirm\_C(\langle i, sn\rangle, \mathsf{X})$ to its successor p_{i+1}. The purpose of the message is twofold: to capture philosophers that are waiting for X, and to make sure that no philosopher is still in a different forum. Philosopher p_i must wait until the message is returned before it can attend X.

When a philosopher p_j receives $Confirm\_C(\langle i, sn\rangle, \mathsf{X})$, where $j \neq i$, it processes the message similarly as request messages: p_j forwards the request if it is also interested in X or has a priority lower than $\langle i, sn\rangle$; otherwise, p_j detains the request. Note that all messages (requests and confirmations) detained by a philosopher are released when the philosopher has finished a forum. In addition, if p_j is also interested in X, is in state *waiting*, and has a priority lower than $\langle i, sn\rangle$, then p_j is *captured* by p_i. In this case, p_j assumes p_i's priority and also enters state *checking* to issue a confirmation message of the form $Confirm(j, \langle i, sn\rangle, \mathsf{X})$ along the ring. (p_j's request message circulating somewhere in the ring then becomes obsolete.) Unlike p_i's confirmation message, p_j's confirmation message serves only one purpose: to make sure that no philosopher is still in a different forum. Therefore, a philosopher receiving p_j's confirmation message processes the message similarly to the other form of confirmation messages (i.e., $Confirm\_C$, where "_C" means "with capturing"), except that the new form of confirmation messages cannot capture the receiving philosophers. This prevents p_j from capturing p_i after p_i has attended X and has made a new request to re-attend X, thereby preventing p_i and p_j from capturing each other repeatedly (which could also result in a violation of mutual exclusion).

Philosopher p_i enters state *talking* to attend X when its confirmation message $Confirm\_C$ $(\langle i, sn\rangle, \mathsf{X})$ is returned. After p_i has finished the forum, it forwards all the messages it has detained to its successor p_{i+1} and then returns to state *thinking*. Note that because some low-priority philosophers may have been captured by p_i to concurrently attend X, after p_i has exited X, some philosophers may still be in X while another philosopher p_k whose priority is higher than the captured philosophers is waiting for a different forum Y. Moreover, because p_i will release all messages held in its message queue when it exits X, p_k may have received its own request $Req(\langle k, sn'\rangle, \mathsf{Y})$ after p_i has exited X but before all the philosophers captured by p_i have exited X. Therefore, p_k cannot immediately enter state *talking* to attend Y when it receives its own request. Instead, it must initiate a confirmation message $Confirm\_C(\langle k, sn'\rangle, \mathsf{Y})$ to make sure that all p_i's captured philosophers have exited X. Similarly, any philosopher captured by p_k during the time p_k's confirmation message is circulating must also initiate a confirmation message (of the form $Confirm$) along the ring before it can enter state *talking* to attend a forum.

We refer to the new algorithm as CTP-Ring2, and its complete code is given in Figure 2. The algorithm is represented by a CSP-like repetitive command of the form [1]:

$$*[g_1 \rightarrow s_1 \,\square\, g_2 \rightarrow s_2 \,\square\, \ldots \,\square\, g_k \rightarrow s_k]$$

The guarded command s_i is executed only if its guard g_i is *enabled*: The Boolean expression (if any) evaluates to true and the message specified in the guard (if any) has arrived. If more than one guarded command have an enabled guard, then one of them is chosen for execution, and the choice is nondeterministic. We assume that a guarded command whose guard is continuously enabled will eventually be executed.

It is not difficult to see that CTP-Ring2 satisfies mutual exclusion, bounded delay, and concurrent entering. Due to the space limitation, the correctness of the algorithm is proved in the full paper [6]. The algorithm has similar complexity measures as CTP-Ring1; that is, its time complexity, context-switch complexity, and degree of concurrency are all of $O(N^2)$. Again, the analysis can be found in the full paper. We remark here that, by a simple modification, the context-switch complexity of CTP-Ring2 can be further reduced to $\min(N, m+1)$. To do so, request messages now have this form:

- $Req(\langle i, sn\rangle, sn_{\max}, \mathsf{X})$

where the extra sequence number $sn_{\max}$ carried by the request is the maximum sequence number of the philosophers through which the request has passed. Initially, $sn_{\max}$

```
1  *[wish to attend a forum of X ⟶
2      SN := SN + 1;
3      target := X;
4      priority := ⟨i, SN⟩;
5      state := waiting;
6      send Req(⟨i, SN⟩, X) to p_{i+1};
7  □ receive Req(⟨j, sn⟩, Y) ⟶
8      SN := max(SN, sn);
9      [ i = j ⋀ state = waiting ⋀ priority = ⟨j, sn⟩ ⟶
10         state := checking;
11         send Confirm_C(⟨i, sn⟩, target) to p_{i+1};
12     □ i ≠ j ⋀ (target = Y ⋁ (priority < ⟨j, sn⟩) ) ⟶
13         send Req(⟨j, sn⟩, Y) to p_{i+1};
14     □ i ≠ j ⋀ target ≠ Y ⋀ priority > ⟨j, sn⟩ ⟶
15         add Req(⟨j, sn⟩, Y) to message_queue;
16     □ else ⟶ skip; /* p_i receives its own obsolete request */ ]
17 □ receive Confirm_C(⟨j, sn⟩, Y) ⟶
18     [ i = j ⟶ /* receive p_i's own confirmation message */
19         state := talking;
20         attend a forum of target;
21     □ i ≠ j ⋀ target ≠ Y ⋀ priority > ⟨j, sn⟩ ⟶
22         add Confirm_C(⟨j, sn⟩, Y) to message_queue;
23     □ else ⟶ send Confirm_C(⟨j, sn⟩, Y) to p_{i+1};
24         [ target = Y ⋀ state = waiting
                        ⋀ priority < ⟨j, sn⟩ ⟶
25           state := checking; /* captured by p_j */
26           priority := ⟨j, sn⟩; /* assume the captor's priority */
27           send Confirm(i, ⟨j, sn⟩, target) to p_{i+1}; ] ]
28 □ receive Confirm(j, ⟨k, sn⟩, Y) ⟶
29     [ i = j ⟶ /* receive p_i's own confirmation message */
30         state := talking;
31         attend a forum of target;
32     □ i ≠ j ⋀ target ≠ Y ⋀ priority > ⟨k, sn⟩ ⟶
33         add Confirm(j, ⟨k, sn⟩, Y) to message_queue;
34     □ else ⟶ send Confirm(j, ⟨k, sn⟩, Y) to p_{i+1}; ]
35 □ exit a forum of target ⟶
36     state := thinking;
37     target := ⊥;
38     priority := ⟨i, ∞⟩; /* reset the priority to a minimal value */
39     for each msg ∈ message_queue do send msg to p_{i+1};
40     message_queue := ∅;
41 ]
```

variables & messages:

- $state$: the state of p_i. Its value ranges over $\{thinking, waiting, checking, talking\}$, and is initialized to $thinking$.
- SN: the sequence number maintained by p_i. It is initialized to 0.
- $target$: the forum p_i wishes to attend, or $\perp$ otherwise. It is initialized to $\perp$.
- $priority$: p_i's priority. It is initialized to a minimal value $\langle i, \infty \rangle$.
- $message\_queue$: the queue of messages detained by p_i. It is initialized to empty.
- $Req(\langle i, sn \rangle, \mathsf{X})$: a request message by p_i to attend a forum X, where $\langle i, sn \rangle$ is p_i's priority when it makes the request.
- $Confirm\_C(\langle i, sn \rangle, \mathsf{X})$: a confirmation message issued by p_i (with priority $\langle i, sn \rangle$) to capture philosophers and to make sure that no philosopher is still in a different forum.
- $Confirm(i, \langle j, sn \rangle, \mathsf{X})$: a confirmation message issued by p_i to make sure that no philosopher is still in a different forum. $\langle j, sn \rangle$ is the priority of p_i assumed from its captor p_j.

Figure 2. Algorithm CTP-Ring2 **for** p_i**.**

equals to p_i's SN when p_i issues the request (and note that sn is also equal to p_i's SN). Upon receiving the request, a philosopher p_j adjusts its own SN to be maximum value of its current SN and $sn_{\max}$. When p_j is to forward the request to its successor (in the case that $i \neq j$), it adjusts $sn_{\max}$ to be maximum value of its current SN and $sn_{\max}$. (Note that if the request is to be held by p_j, then in between the time p_j receives the request and the time it releases the request, p_j's sequence number SN may have been adjusted a number of times.)

An extra sequence number is also added to confirmation messages so that they now have one of the following two forms:

- $Confirm\_C(\langle i, sn \rangle, sn_{\max}, \mathsf{X})$
- $Confirm(i, \langle j, sn \rangle, sn_{\max}, \mathsf{X})$.

The extra parameter $sn_{\max}$ is initialized and used as in the new request messages.

The simulation results of the new algorithm are also given in Figure 1. From the results we can see that the new algorithm significantly improves the performance of CTP-Ring1.

5 Conclusions

We have presented two distributed algorithms CTP-Ring1 and CTP-Ring2 for the Congenial Talking Philosophers problem on a ring network where each philosopher can only communicate directly with its two neighboring philosophers. CTP-Ring1 requires N messages per entry to the meeting room, and the time complexity, context-switch complexity, and degree of concurrency are all $O(N^2)$, where N is the total number of philosophers in the ring. In contrast, CTP-Ring2 requires $2N$ messages, has also $O(N^2)$ time complexity and degree of concurrency, but has a better context-switch complexity of $\min(N, m+1)$, where m is the total number of forums the philosophers may attend. Although, statically, the two algorithms do not differ very much from various complexity measures, the difference between their dynamic performance is significant. As we have shown in the simulation, CTP-Ring1 performs poorly as compared to CTP-Ring2; it out-performs CTP-Ring2 only in message complexity.

References

[1] C. A. R. Hoare. Communicating sequential processes. *CACM*, 21(8):666–677, Aug. 1978.

[2] Y.-J. Joung. Asynchronous group mutual exclusion (extended abstract). In *Proc. 17th ACM PODC*, pp. 51-60, 1998.

[3] Y.-J. Joung. The congenial talking philosophers problem in computer networks. Technical report, Dept. of Info. Management, National Taiwan University, Taipei, Taiwan, 1998.

[4] L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21(7):558–565, July 1978.

[5] A. M. Law and W. D. Kelton. *Simulation modeling & analysis*. McGraw-Hill, 1991.

[6] K.-P. Wu and Y.-J. Joung. Asynchronous group mutual exclusion in ring networks. Technical report, Dept. of Info. Management, National Taiwan University, Taipei, Taiwan, 1999.