A Formal Model of Quorum Based *k*-Mutex Algorithm using Input/Output Automata

Ismail M. Nur
Department of Mathematics
Postgraduate Program of Mathematics and Natural Sciences
Hasanuddin University, Makassar, Indonesia
e-mail: ismailmnur49@gmail.com

Armin Lawi
Department of Computer Science
Faculty of Mathematics and Natural Sciences
Hasanuddin University, Makassar, Indonesia
e-mail: armin@unhas.ac.id

***Abstract*—An input/output automata model is a distributed computation model in distributed systems to create a formal model or to prove a distributed algorithm. This research aims to create a formal model of quorum based *k*-mutual exclusion (*k*-mutex) algorithm. The formal model of quorum based *k*-mutex algorithm using input/output automata consists of five components: signature, state, initial state, transition, and task. The constructed model proved by using time line and execution of input/output automata. The result of research shows that the input/output automata model of quorum based *k*-mutex algorithm solved the problem of *k*-mutex given.**

Keywords—distributed systems, input/output automata; k-mutual exclusion; distributed system; quorum based algorithm.

I. INTRODUCTION

Mutual exclusion (mutex) is a condition in distributed system where system synchronizes many processes that compete each other to access the single resource so that only one process succeed in one period of time [3,4,5]. Then the problem of mutex generalized into *k*-mutual exclusion (*k*-mutex) where k (>1) processes can access the single resource together at the same time in one period of time [9].

One type of algorithm to solve mutex and *k*-mutex is a quorum-based algorithm [4,5]. It is an algorithm that constructed by using quorum system set such as coterie, *k*-coterie, etc. As far, there is no formal model for quorum based *k*-mutex algorithm. One of the model that we can use to formalize algorithm is by using input/output automata (I/O automata)[1,2,6,7,8].

I/O automata model has been used in several studies. Lynch & Tuttle introduced the I/O automata in distributed systems for the first time in 1988 [7]. Then, Goldman used I/O automata model to simulate the distributed algorithm [2]. Lynch & Jensen used I/O automata model to get a proof burn of *N*-process in mutual exclusion [8]. Therefore, this research aims to create a formal model of quorum based *k*-mutex algorithm by using I/O automata.

The paper is organized as follows. Section 2 explains the basic concepts of distributed systems, I/O Automata model, the *k*-mutual exclusion problem and its quorum-based algorithm. Section 3 provides our results and discussion, and section 4 concludes the paper.

II. BASIC CONCEPTS

A. Distributed Algorithm

Distributed system is a system consisting of separate components that are connected via a network. In this case, the components are processes and the network is a canal. In aim to make the system work, the processes communicate each other, exchange message and data asynchronously. A protocol is required to coordinate processes that work asynchronously, that protocol is a distributed algorithm [6].

B. I/O Automata Model

An I/O automata model is a distributed computation model to make a formal model of a distributed algorithm or prove the distributed algorithm that has been made [6].

Definition 1. An I/O automaton is a system consists of 5-components ($sig(A)$, $states(A)$, $start(A)$, $trans(A)$, $tasks(A)$), where:

- $sig(A)$, set of signatures
- $states(A)$, set of states (limited or not)
- $start(A)$, non-empty subset of $states(A)$ that known as first state or initial state
- $trans(A)$, relation of transition-state, where $trans(A) \subseteq states(A) \times acts(sig(A)) \times states(A)$
- $task(A)$, is a partition of task.

The first step in making the I/O automata model of an automaton A is to determine the signature, *sig(A)*. Then, determine the set of state, *states(A)* and initial state *start(A)*. After determine *sig(A)*, *states(A)* and *start(A)*, the next step is to construct a transitions, transitions in I/O automata is the algorithm itself. The last step is to determine the goal of the system or the function, *task* (A).

There are two I/O automata model in distributed systems, I/O automaton process and I/O automaton canal. But only I/O automaton process will be constructed in this research. The model of algorithm will be simulate in time line, and from the time line we can make an execution of I/O automata. Execution of I/O automata is a transition of s,π,s' where s is the state of automata before input/output action, π is input/output action, and s' is the state of automata after input/output action. Simulation should fulfill the safety, liveness, and fairness condition of *k*-mutex.

978-1-5090-5548-7/16/$31.00 ©2016 IEEE

C. k-Mutual exclusion

Definition 2. *k*-Mutual exclusion (*k*-mutex) is a condition in distributed system where at most *k* processes can access the single resource in one period of time.

Algorithm of *k*-mutex created to coordinate the processes that compete to access the single resource so that at most *k* (< n) processes successfully access it in one period of time [9]. The Algorithm is correct if it fulfill the safety, liveness, and fairness condition.

- Safety: at most *k* process can access single resource in one period of time.
- Liveness: every request to access the resource will be succeed. (Starvation-free and Deadlock-free)
- Fairness: every process has a same opportunity.

D. k-Coterie

Definition 3. Let $k > 1$. A nonempty set $C \subseteq 2^P$ is a *k*-coterie on *P* if *C* satisfies the following properties:

- **Non-intersection:** for every $d < k$ elements of $Q_1, \ldots, Q_d \in C$ that free each other, such as $Q_i \cap Q_j = \varnothing$, $1 \le i < j \le d$, there must be $Q \in C$ so that $Q \cap Q_j = \varnothing$, $1 \le i \le d$.
- **Intersection:** for every p ($> k$) elements of $Q_1, \ldots, Q_p \in C$, there must be $\{Q_i, Q_j\}$, so that $Q_i \cap Q_j \neq \varnothing$, $1 \le i \ne j \le p$.
- **Minimalist:** $Q_i \not\subset Q_j$, for all $Q_i, Q_j \in C$, $i \ne j$.

Example 1. C = {{1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}} is 2-coterie of set P = {1, 2, 3, 4}.

III. Result and Discussion

A. I/O Automata Process of k-Mutex

In order to solve *k*-mutex problem, algorithms constructed in purpose to make *k* (>1) processes can access a single resource in one period of time. In quorum based algorithm, the process that wants to access the resource will send the request to the quorum set, quorum set determined by *k*-coterie. If quorum set agreed, then process can access the resource. This process we modeled as an automaton. Fig. 1 shows the

Signature (sig (P_i)):	
Input :	*Output :*
• $Trying(Ts)_i$,	• $Critical(Ts)_i$
• $Request(Ts_j, P_j)_{j,i}$	• $Request(Ts_i, P_i)_{i,j,}$
• **Receive(***j***)**$_{j,i,}$	• **Send(***i***)** $_{i,j,}$
• $Reclaim(j)_{j,i}$	• $Reclaim(i)$ $_{i,j,}$
• $Relinquish(i)_{j,i}$	• $Relinquish(j)$ $_{i,j,}$
• **Receive(***wait***)**$_{j,i,}$	• **Send(***wait***)** $_{i,j,}$
• **Receive(***exit* P_j**)**$_{j,i}$	• **Send(***exit* P_i**)** $_{i,j,}$
• $Exit_i$	• $Reminder_i$

Fig.1. Signature of process P_i in *k*-mutex

signature of an I/O automaton process P_i in *k*-mutex.

Signature is the entire action of an automaton. Action in I/O automaton P_i are input and output itself. There are two inputs from the user to the process, $Trying(Ts)_i$ and $Exit_i$. There are two outputs from the process to the user, $Critical(Ts)_i$ and $Reminder_i$. Another message is input and output from and to another process, let P_j.

There are six output messages from process P_i to process P_j, namely $Request(Ts_i, P_i)_{i,j}$, **Send(***i***)**$_{i,j}$, $Reclaim(j)_{i,j}$, $Relinquish(j)_{i,j}$, **Send(***wait***)**$_{i,j}$, and **Send(***exit* P_i**)**$_{i,j}$. $Request(Ts_i, P_i)_{i,j,}$ is a message to request the permission. **Send(***i***)**$_{i,j,}$ is a message to give the permission. $Reclaim(j)_{i,j,}$ is a message to reclaim the permission has been given. $Relinquish(j)_{i,j,}$ is a message to restore permission. **Send(***wait***)**$_{i,j,}$ is a message to ask to wait. **Send(***exit* P_i**)**$_{i,j,}$ is an announcement message that process P_i releasing the resource. The same as the output action, also there are six input messages from process P_j to process P_i, namely $Request(Ts_j, P_j)_{j,i}$, **Receive(***j***)**$_{j,i}$, $Reclaim(j)_{j,i}$, $Relinquish(i)_{j,i}$, **Receive(***wait***)**$_{j,i}$, and **Receive(***exit* P_j**)**$_{j,i}$. $Request(Ts_j, P_j)_{j,i}$ is a message to request the permission. **Receive(***j***)**$_{j,i}$ is a permission message. $Reclaim(j)_{j,i}$ is a message to reclaim the permission. $Relinquish(i)_{j,i}$ is a restore permission maessage. **Receive(***wait***)**$_{j,i}$ is a wait message. And **Receive(***exit* P_j**)**$_{j,i}$ is an announcement message that process P_j releasing the resource.

States (*states*(P_i) and *start*(P_i)):

- $Status_i$ = {*Reminder* (***R***), *Waiting Section* (***WS***), *Critical Section* (***CS***), *Exit Section* (***ES***)}.
  Initial state is *Reminder*, $Status_i$ = ***R***.
- $\boldsymbol{AGREE_i}$, is a set of accepting permission of process P_i.
  Initial state of $\boldsymbol{AGREE_i}$ is an empty set, $\boldsymbol{AGREE_i} = \emptyset$
- $\boldsymbol{QUEUE_i}$, is a set of queue of process P_i.
  Initial state of $\boldsymbol{QUEUE_i}$ is an empty set, $\boldsymbol{QUEUE_i} = \emptyset$
- $\boldsymbol{PERM_i}$, is a set of giving permission of process P_i.
  Initial state of $\boldsymbol{PERM_i}$ is an empty set, $\boldsymbol{PERM_i} = \emptyset$

Fig. 2. States and initial state of process P_i in *k*-mutex

Fig. 2 shows the states and the initial state of an I/O automaton process P_i in *k*-mutex. There are four status in *k*-mutex, namely *Reminder* (*R*), *Waiting Section* (*WS*), *Critical Section* (*CS*), and *Exit Section* (*ES*). *R* is a status when the process not in access mode or trying mode. *WS* is a status when the process in trying mode. *CS* is a status when the process in access mode. *ES* is a status when the process releases the resource.

There are three sets of message in an I/O automaton process P_i in *k*-mutex, namely $QUEUE_i$, $AGREE_i$, and $PERM_i$. When the process P_i sends or receives request from other process then process P_i will put the request into $QUEUE_i$. When the process P_i receives permission from other process then process P_i will put the permission into $AGREE_i$. When the process P_i sends permission to other process then P_i will put IP of that process into $PERM_i$. Initial states of all sets are empty set, and for $PERM_i$ members should not be more than one.

Input Transition (*trans*(P_i)):

- ***Trying(Ts)$_i$***

Pre: State = R
Eff: State = WS
Request(Ts_i, P_i)$_{i,j}$, $i=j$, $\forall P_j \in Q$, $Q \subset C$

- ***Request(Ts_j, P_j)$_{j,i}$***

Eff: $(Ts_j, P_j) \in QUEUE_i$
case,
$(Ts_j, P_j) < max\ QUEUE_i \rightarrow$ **Send**$(i)_{i,j}$,
$(Ts_j, P_j) < max\ QUEUE_i$ and $\exists (k) \in PERM_i \rightarrow$
Reclaim$(i)_{i,k}$, $(k) \in PERM_i$
$(Ts_j, P_j) < max\ QUEUE_i$ and $\exists (i) \in PERM_i \rightarrow$
remove (i) in $PERM_i$ and **Send**$(i)_{i,j}$,
(Ts_j, P_j) is not in the top of $QUEUE_i \rightarrow$ **Send**$(wait)_{i,j}$

- **Receive(j)$_{j,i}$**

Pre: State = WS
Eff: $(j) \in AGREE_i$
if, $\exists Q \subseteq AGREE_i \rightarrow$ State = CS
else, State = WS

- **Receive(*wait*)$_{j,i}$**

Eff: State = WS
Request(Ts_i, P_i)$_{i,j}$, $\forall P_j \in Q'$, $Q' \subset C$
remove (wait)

- ***Reclaim(j)$_{j,i}$***

Pre: **Receive**$(j)_{j,i}$
Eff: Relinquish$(j)_{i,j}$

- ***Relinquish(i)$_{j,i}$***

Pre: Reclaim$(i)_{i,j}$
Eff: $\exists (Ts_k, P_k) < max\ QUEUE_i \rightarrow$ **Send**$(i)_{i,k}$,

- **Receive(*exit* P_j)$_{j,i}$**

Eff: remove (Ts_j, P_j) in $QUEUE_i$
if, $\exists (Ts_k, P_k) < max\ QUEUE_i \rightarrow$ **Send**$(i)_{i,k}$,
else, *do nothing*

- ***Exit$_i$***

Pre: State = CS
Eff: State = ES
remove (Ts_i, P_i) in $QUEUE_i$
Send(*exit* $P_i)_{i,j}$, $\forall P_j \in Q$, $\forall P_j \in Q'$, $Q, Q' \subset C$

Fig. 3. Input Transition of process P_i in *k*-mutex

Fig. 3 shows the input transition of an I/O automaton process P_i in *k*-mutex. If there is an input $Trying(Ts)_i$ to process P_i , it means the user is asking to process P_i to access the resource. That input will makes process P_i to changes its status from *R* to *WS* and sends $Request(Ts_j, P_j)_{i,j}$ to quorum *Q* with $\forall P_j \in Q$, ($i=j$ its means P_i also one of P_j) . If there is an input $Request(Ts_j, P_j)_{j,i}$, it means process P_j is asking the permission to process P_i. That input will make process P_i put (Ts_j, P_j) into $QUEUE_i$ and see the status of the set. There are four possibilities, (1) $(Ts_j, P_j) <$ max $QUEUE_i$, (2) $(Ts_j, P_j) <$ max $QUEUE_i$ and $\exists (k) \in PERM_i$, (3) $(Ts_j, P_j) <$ max $QUEUE_i$ and $\exists$ $(i) \in PERM_i$, and (4) (Ts_j, P_j) is not in the top of $QUEUE_i$. If (1) then process P_i **Send**$(i)_{i,j}$. If (2) then process P_i sends $Reclaim(i)_{i,k}$, $(k) \in PERM_i$. If (3) then P_i removes (i) in $PERM_i$ and **Send**$(i)_{i,j}$. if (4) then P_i **Send**$(wait)_{i,j}$. If there is an input **Receive**$(j)_{j,i}$, it means process P_i in *WS* status and get the permission from process P_j. Process P_i will put (j) into $AGREE_i$. If there is $Q \subseteq AGREE_i$ then process P_i will changes its status from *WS* to *CS* which means process P_i successfully access the resource, if else then process P_i stay in *WS* status. If there is an input **Receive**$(wait)_{j,i}$, it means process P_j is asking to process P_i to wait. This input will make process P_i stay in *WS* status and then send $Request(Ts_j, P_j)_{i,j}$ to other quorum Q'. If there is an input $Reclaim(j)_{j,i}$, then P_i have to send $Relinquish(j)_{i,j}$. If there is an input **Receive**(*exit* $P_j)_{j,i}$ it means P_j is releasing the resource and then process P_i may remove P_j in $QUEUE_i$. The last input is $Exit_i$, it is a message from user to make process P_i releases the resource.

Output Transition (*trans*(P_i)):

- ***Request(Ts_i, P_i)$_{i,j}$***

Pre: State = WS
$\forall P_j \in Q$ **if** $Trying(Ts)_i$
$\forall P_j \in Q'$ **if Receive**$(wait)_{j,i}$,
Eff: $(Ts_i, P_i) \in QUEUE_i$
case,
$\exists P_i \in Q$ and $(Ts_i, P_i) < max\ QUEUE_i \rightarrow$
$(i) \in AGREE_i$ and $(i) \in PERM_i$
$\exists P_i \in Q$, $(Ts_i, P_i) < max\ QUEUE_i$ and $\exists (k) \in PERM_i \rightarrow$
Reaclaim $(i)_{i,k}$

- **Send(i)$_{i,j}$**

Pre: $Request(Ts_j, P_j)_{j,i}$ and $(Ts_j, P_j) < max\ QUEUE_i$
Eff: $(j) \in PERM_i$

- **Send(*wait*)$_{i,j}$**

Pre: $Request(Ts_j, P_j)_{j,i}$ and (Ts_j, P_j) is not in the top of $QUEUE_i$
Eff: do nothing

- ***Reclaim(i)$_{i,j}$***

Pre: $\exists (Ts_k, P_k) < max\ QUEUE_i$ and $(j) \in PERM_i$
Eff: do nothing

- ***Relinquish(j)$_{i,j}$***

Pre: Reclaim$(j)_{j,i}$
Eff: remove (j) in $AGREE_i$

- **Send(*exit* $P_i)_{i,j}$**

Pre: State = ES
$\forall P_j \in Q$, $\forall P_j \in Q'$, $Q, Q' \subset C$
Eff: remove (j) in $AGREE_i$,

- ***Critical(Ts)$_i$***

Pre: State = CS
Eff: do nothing

- ***Reminder$_i$***

Pre: State = ES
$AGREE_i = \emptyset$
Eff: State = R
$\exists (Ts_k, P_k) < max\ QUEUE_i \rightarrow$ **Send**$(i)_{i,k}$,

Fig. 4. Output Transition of process P_i in *k*-mutex

Fig. 4 shows the output transition of an I/O automaton process P_i in *k*-mutex. If there is an output $Request(Ts_i, P_i)_{i,j}$ to process P_j , it means process P_i is requesting to process P_j the permission to access the resource. Process P_i should be in *WS* status to send this message. If there is an input $Trying(Ts)_i$ before then the request is sending to quorum Q. If there is an input **Receive**$(wait)_{j,i}$ before then the request is sending to quorum Q'. After sending $Request(Ts_i, P_i)_{i,j}$, then process P_i will put (Ts_i, P_i) into $QUEUE_i$ and see the status of the set. There are two possibilities, (1) process P_i is one of the quorum and (Ts_i, P_i) is in the top of $QUEUE_i$ and (2) process P_i is one

of the quorum, (Ts_i, P_i) is in the top of $QUEUE_i$ and there is (k) in $PERM_i$. If (1) then process P_i put (i) into $AGREE_i$ and $PERM_i$. So (i) became element of $AGREE_i$ and $PERM_i$. If (2) then process P_i sends $Reaclaim\ (i)_{i,k}$ to process P_k. If there is an output **Send(***i***)**$_{i,j}$, it means process P_i gives its permission to process P_j. If process P_i gives its permission to process P_j then process P_i put (j) into $PERM_i$. If there is an output **Send(***wait***)**$_{i,j}$, it means process P_i is asking to process P_j to wait. If there is an output $Reclaim(i)_{i,j}$, it means there is another process in the top of $QUEUE_i$ besides process P_j. So process P_i reclaims its permission from process P_j. If there is an output $Relinquish(j)_{i,j}$, it means process P_i receives reclaim message before. So process P_i removes (j) in $AGREE_i$. If there is an output **Send(***exit* P_i**)**$_{i,j}$, it means process P_i is releasing the resource. This message will make process P_i removes (j) from $AGREE_i$. If there is an output $Critical(Ts)_i$, it is a message from process P_i to the user as an announcement that process P_i successfully access the resource. The last output is $Reminder_i$. This is the message from process P_i to the user. There are two condition to sending this message, process P_i in *ES* status and $AGREE_i$ in initial state.

Task (*task*(P_i)):
Critical(Ts)$_i$,

Fig. 5. Task of process P_i in k-mutex

Fig. 5 shows the task of an I/O automaton process P_i in k-mutex. This task makes the systems sustainable. Ts value in this task will be recorded so that the fairness among all the process guaranteed.

B. I/O Automata of k-mutex in Time Line

Input transition in Fig. 4 and output transition in Fig. 5, can be simulated in time line. Fig. 6 is an example of k-mutex with k = 2 (2-mutex). There are six processes competing to access the single resource but only two of them can access it at the same time. Every process has a different value of Ts. Value of Ts determined how many times the process successfully access the resource. Process P_1 with Ts_1 = 3, process P_2 with Ts_2 = 1, process P_3 with Ts_3 = 0, process P_4 with Ts_4 = 2, process P_5 with Ts_5 = 4, and process P_6 with Ts_6 = 3. The simulation in Fig. 6 shows that three processes tried to access the resource, P_2, P_3 and P_4. But only P_2 and P_3 will successfully access the resource.

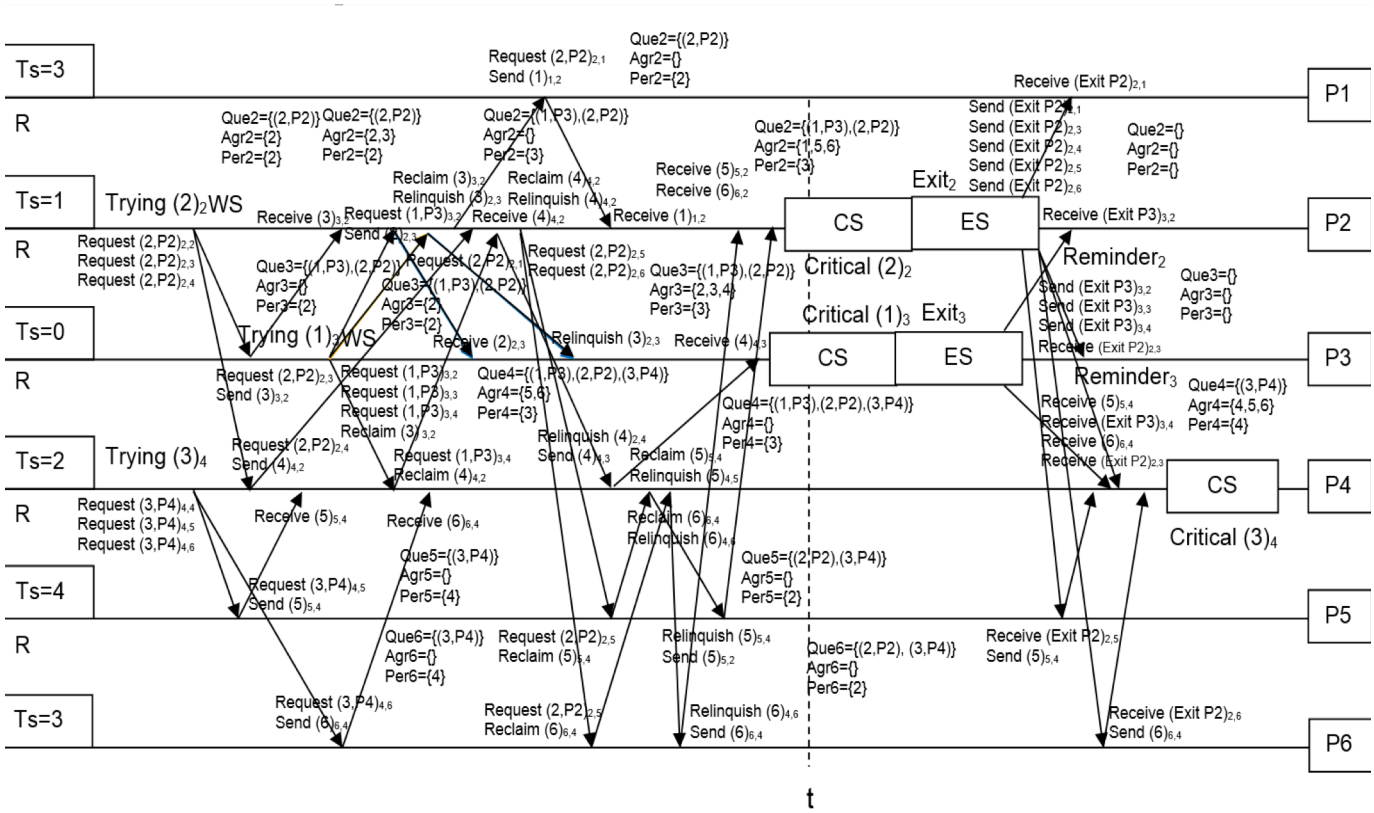

Fig. 6. Simulation of I/O automaton P_i of k-mutex in time line.

Execution of Process P_3:
[R, $QUEUE_3$={}, $AGREE_3$={}, $PERM_3$={}],
$Request(2,P2)_{2,3}$,
[R, $QUEUE_3$={(2,P2)}, $AGREE_3$={}, $PERM_3$={}],
$Send(3)_{3,2}$,
[R, $QUEUE_3$={(2,P2)}, $AGREE_3$={}, $PERM_3$={2}],
$Trying(1)_3$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={}, $PERM_3$={2}],
$Request(1,P3)_{3,2}$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={}, $PERM_3$={2}],
$Request(1,P3)_{3,3}$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={}, $PERM_3$={2}],
$Request(1,P3)_{3,4}$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={}, $PERM_3$={2}],
$Reclaim(3)_{3,2}$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={}, $PERM_3$={2}],
$Receive(2)_{2,3}$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={2},$PERM_3$={2}],
$Relinquish(3)_{2,3}$,
[WS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={2,3},
$PERM_3$={3}],
$Receive(4)_{4,3}$,
[WS,$QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={2,3,4},
$PERM_3$={3}],
$Critical(1)_3$,
[CS, $QUEUE_3$={(1,P3),(4,P1)}, $AGREE_3$={2,3,4},
$PERM_3$={3}],
$Exit_3$,
[ES, $QUEUE_3$={(4,P1)}, $AGREE_3$={2,3,4}, $PERM_3$={3}],
$Send(exit\ P3)_{3,2}$,
[ES, $QUEUE_3$={(4,P1)}, $AGREE_3$={3,4}, $PERM_3$={3}],
$Send(exit\ P3)_{3,3}$,
[ES, $QUEUE_3$={(4,P1)}, $AGREE_3$={4}, $PERM_3$={}],
$Send(exit\ P3)_{3,4}$,
[ES, $QUEUE_3$={(4,P1)}, $AGREE_3$={}, $PERM_3$={}],
$Reminder_3$,
[R, $QUEUE_3$={(4,P1)}, $AGREE_3$={}, $PERM_3$={}].

IV. Conclusion

A quorum based *k*-mutex algorithm can be modeled by using I/O automata. The model of quorum based *k*-mutex algorithm using I/O automata focused on process as an automaton consists of five components: signature ($sig(P_i)$), states ($states(P_i)$), initial states ($start(P_i)$), transition ($trans(P_i)$), and task ($task(P_i)$). The transition of the model can be proved by simulation on a time line. The execution of I/O automata can be obtained from the simulation on a time line. The simulation of the model on a time line solved *k*-mutex problem that given.

References

[1] S. Giro, P.R. D'Argenio, and L. M. F. Fioriti, "Distributed Probabilistic Input/Output Automata: Expressiveness, (un)Decidability and Algorithms," Theoretical Computer Science, vol. 538, pp. 84-102, 2014. (*http://www.sciencedirect.com/*)

[2] K.J. Goldman, "Distributed Algorithm Simulation Using Input/Output Automata," MIT Laboratory for Computer Science, MIT/LCS/TR-490, pp. 15-16, 1990.

[3] L. Lamport, "Time, clocks and the ordering of events in a distributed system," Communications of the ACM, vol. 21(7), pp.558–565, 1978.

[4] A. Lawi and M. Yamashita, "A quorum based *m* group (*h, k*)-exclusion algorithm," In Proc. International Symposium on Information Science and Electrical Engineering (ISEE2003), pp. 405–408, 2003.

[5] A. Lawi, K. Oda, and T. Yoshida, "A Quorum based (m,h,k)-Resource Allocation Algorithm," Conf. on Parallel Dist. Proc. Tech & Appl, PDPTA'06 + RTCOMP'06, pp. 399–405, 2006.

[6] N.A. Lynch, Distributed Algorithms. San Fransisco: Morgan Kaufmann Publishers, 1996.

[7] N.A. Lynch and M.R. Tuttle, "An Introduction to Input/Output Automata", CWI Quarterly, vol. 2(3), pp. 219-246, 1988.

[8] H.E. Jensen and N.A. Lynch, "A proof of burns *N*-process mutual exclusion algorithm using abstraction," 4th International Conference, TACAS'98 Held as Part of the Joint European Conferences on Theory and Practice of Software, vol. 1384, pp.409-423, March 28 – April 4, 1998.

[9] H.G. Molina and D. Barbara, "How to assign votes in a distributed system, " Journal of the ACM, vol. 32(4), pp. 841–860. 1985.