The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 23, No. 2, pp.73-78, Apr. 30, 2023. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2023.23.2.73
JIIBC 2023-2-10

일반화된 철학자 만찬 문제의 교착상태 예방 알고리즘

Algorithm for Deadlock Prevention of Generalized Philosophers' Dining Problem

이상운*

Sang-Un Lee*

요 약 식사하는 철학자 문제는 5명의 철학자(프로세서)들이 원형 탁자에 둘러 앉아 함께 스파게티(또는 국수) 식사를 하는데 있어 자신의 양쪽에 있는 젓가락(자원) 한 쌍(2개)을 모두 가져야만 식사가 가능한 경우로 모든 철학자가 우측의 젓가락 1개씩 모두 가진 경우 아무도 식사를 못하는 교착상태(deadlock)를 해결하는 문제이다. 교착상태는 병행 시스템(concurrent system)에서 빈번히 발생하는 문제로 현행 운영체제(OS)에서는 이를 예방하는 방법은 채택되지 않고 있다. 본 논문은 $2 \leq n \leq \infty$의 모든 프로세서들이 다중 병행(parallel concurrency)처리 능력을 갖고 있는 OS에서 교착상태를 전혀 유발하지 않는 묘책을 제안한다. 제안된 방법은 $\lfloor n/2 \rfloor$개의 홀수 프로세서들이 그룹을 형성하여 동시에 수행하는 방법으로 실행이 종료되면 다음 프로세서로 우측 이동(shift right)시키는 그룹 라운드-로빈 방법이다. 제안된 방법은 1-라운드의 모든 프로세서를 실행시키려면 짝수 프로세서인 경우 2회, 홀수 프로세서는 3회를 수행하면 되고, n회를 수행하면 짝수 프로세서인 경우는 $n/2$회, 홀수 프로세서는 $(n-1)/2$회를 수행하는 방식이다.

Abstract The dining philosophers problem(DPP) is that five philosophers sit around a round table and eat spaghetti(or noodles) together, where they must have a pair of chopsticks(two) on both sides of them to eat, and if all philosophers have one chopstick on the right, no one can eat because the deadlock occurs. Deadlocks are a problem that frequently occur in parallel systems, and most current operating systems(OS) cannot prevent it. This paper proposes a silver bullet that causes no deadlock in an OS where all processors of $2 \leq n \leq \infty$ have multiple parallel processing capabilities. The proposed method is a group round-robin method in which $\lfloor n/2 \rfloor$ odd processors form a group and perform simultaneously, and shift right to the next processor when execution ends. The proposed method is to perform two times for even processors, three times for odd processors per one round-robin. If the proposed method is performed n times, even-numbered processors perform $n/2$ times and odd-numbered processors perform $(n-1)/2$-times.

Key Words : Dining philosophers problem, Deadlock, Round-robin, Group round-robin

*정회원, 강릉원주대학교 과학기술대학 멀티미디어공학과
접수일자 2022년 10월 22일, 수정완료 2023년 3월 6일
게재확정일자 2023년 4월 7일

Received: 22 October, 2022 / Revised: 6 March, 2023 / Accepted: 7 April, 2023
*Corresponding Author: sulee@gwnu.ac.kr
Dept. of Multimedia Eng., Gangneung-Wonju National University, Korea

Ⅰ. 서 론

철학자들 만찬 문제(dining philosophers problem, DPP)는 운영체제(OS)에서 동시성과 교착상태를 설명하는 예시로 활용하기 위해 제기된 문제이다.[1] DPP는 n명의 철학자가 원형 탁자에 빙 둘러앉아 국수(스파게티) 만찬 식사를 하는 경우 총 n개의 젓가락(chopsticks)이 각 철학자 사이(각 철학자의 좌우측)에 하나씩 놓여 있고, 국수(또는 스파게티)를 식사하려면 한 쌍의 젓가락을 가져야만 식사가 가능한 경우이다. 이는 철학자를 프로세서(processor, P)로, 젓가락을 자원(resource, R)으로, 젓가락 한 쌍을 병행(concurrency)처리로 치환하면 여러 명의 철학자가 동시에 식사하는 다중 병행(parallel concurrency) 처리 운영체제 환경으로 볼 수 있다.[2-5] 이 문제는 식사를 못해 굶는 철학자가 발생하지 않도록 병행 알고리즘인 행위 원칙을 어떻게 설계할 수 있는가이다. 즉, 다른 철학자가 언제 식사 또는 사색에 잠기는 시기를 알 수 없는 상태에서 교착상태(deadlock)에 빠지지 않고 모든 철학자가 식사와 사색에 잠기는 행위를 영원히 반복할 수 있는 알고리즘 개발이 주 목적이다.[1]

DPP에서 모든 철학자들이 자신의 좌측에 있는 젓가락을 먼저 갖는다고 가정하면 어는 누구도 한 쌍의 젓가락을 갖지 못해 식사를 할 수 없는 교착상태가 발생한다.

$n=5$ DPP에 대한 교착상태 해결책을 1965년 Dijkstra[2]가 처음으로 자원인 젓가락에 번호를 부여하고, 모든 자원은 순서대로 요청되며, 순서에 의해 연관되지 않는 두 자원은 동시에 하나의 작업 단위(철학자)에게 사용되지 못하도록 하는 자원 계층 해결책(resource hierarchy solution, RHS)을 제시하였다. 이후, 중재자(웨이터)가 한 명의 철학자에게 두 개의 젓가락을 잡거나 잡지 못하게 하는 권한을 부여하는 방법인 중재자 해결책(arbitrator solution, AS)[6]과 Chandy와 Misra[7]의 철학자에게 번호를 부여하고, 각자는 이웃(인접)한 철학자에게만 음료수병 사용 여부 요청을 하는 분산형 방식이 있다. 이외에도 다양한 방법들이 제안[8-14]되었음에도 불구하고 현존하는 운영체제에서 교착상태를 예방할 수 있는 방법은 채택되지 않고 있는 실정이다.[15]

본 논문에서는 그룹 라운드-로빈(group round-robin method, GRR) 방식을 채택하여 교착상태를 원천적으로 봉쇄하면서도 여러 명이 동시에 식사(다중 처리)할 수 있는 방법을 제안한다. 2장에서는 철학자들 만찬문제에 대해 기존에 널리 알려진 대표적인 방법을 고찰한다. 3장에서는 그룹 라운드-로빈(GRR) 방식을 제안한다. 4장에서는 $2 \le n \le \infty$에 대해 일반적으로 적용되는 GRR의 일반화된 규칙을 유도한다.

Ⅱ. 관련연구와 문제점

Dijkstra[2]가 제안한 RHS는 철학자가 대기상태(사색, thinking), 허기(hungry), 식사(eating)의 3가지 상태 변수(state variable)를 갖으며, 하나의 Mutex(mutual exclusion object)와 젓가락 각각이 하나의 세마포어(semaphore)를 갖도록 하였다. 또한 젓가락(포크)에는 번호 f_i를 부여하였으며, 철학자 P_i 좌측에는 f_i가, 우측에는 f_{i+1} 젓가락이 위치한 상태이다.

$P_1, P_2, \cdots, P_{n-1}$은 자신의 왼쪽 젓가락(낮은 순서) f_i를 먼저 잡고, 다음으로 오른쪽 젓가락(높은 순서) f_{i+1}을 집는 방법이다. 만약 5명 중 4명이 동시에 $f_i, i=1,2,3,4$ 젓가락을 집은 경우, 가장 높은 순서인 $f_n, n=5$은 식탁에 남게 되며, 이 경우 P_5는 어떠한 젓가락도 집지 못하게 된다. 더군다나 단지 1명의 철학자만이 가장 높은 번호를 가진 젓가락을 집어 식사를 할 수 있다. RHS는 교착상태는 회피할 수 있지만 항상 실용적이지는 못한 문제점을 갖고 있다. 특히 원하는 자원(젓가락)의 목록을 사전에 완전히 알지 못하는 경우가 대표적인 사례이다. 예를 들면 철학자가 f_3, f_5를 들고 있는 상태에서 작업을 수행하려면 f_2가 필요하다는 것을 결정하면 항상 낮은 순서의 젓가락을 먼저 잡아야 하는 원칙에 의거 f_2를 집기 전에 f_5, f_3 순서대로 내려놓아야만 한다. f_2를 집은 다음에 다시 f_3과 f_5에 대한 요청을 해야만 한다. 또한 RHS는 공정(fair)하지 않다. 만약 철학자 P_1이 나머지 하나의 젓가락 f_{12}을 느리게 집으려 하는 사이에 P_2가 빠른 사색을 끝내고 P_1이 집어야 할 젓가락 f_{12}을 먼저 집은 경우, P_1은 f_{12}을 결코 집을 수 없게 된다. 공정 해법(fair solution)은 한 철학자는 다른 철학자에 비해 상대적으로 느리게 움직이든지에 상관없이 결국에는 식사를 할 수 있음을 보증해야만 한다.[1]

AS는 중재자(웨이터)가 존재하여 단지 한 명의 철학자에게 2개의 젓가락을 모두 집거나 집지 못하게 하는 권한을 부여하는 방법이다. 젓가락을 집고자 하는 철학자는 웨이터에게 허락을 득해야만 한다. 웨이터는 젓가락을 집을 권한을 허락한 철학자가 2개의 젓가락을 모두 집을 때까지 다른 철학자 요청은 허락하지 않는다. 철학자가 젓가락을 내려놓는 것은 본인의 자유로 웨이터의

허락을 득할 필요가 없다. 웨이터는 Mutex로 구현될 수 있다. 이 방법은 병렬성(parallelism)을 감소시키는 단점을 갖고 있다. 예를 들면, 한 철학자 P_i가 식사를 하고 있는 도중에 이웃인 P_{i-1}, P_{i+1}중 어느 한 명이 젓가락 집는 것을 요청한 경우, 다른 철학자들은 자신들에게 필요한 젓가락 2개를 모두 집을 수 있음에도 불구하고, 요청한 철학자가 해당 젓가락을 집을 때까지 계속적으로 대기해야만 한다. 결국 웨이터는 하나의 요청을 충족한 이후에 다음 요청을 받아들이는 한계로 인해 다중처리가 불가할 수 있다.[1]

Chandy와 Misra[7]의 분산형 방법은 P_i는 P_{i-1}, P_{i+1}에게만 젓가락을 집을 수 있는지 요청하는 방식이다. 따라서 P_{i-1}과 P_{i+1}의 식사가 모두 끝날 때까지 대기해야만 한다. 극단적인 경우 P_1 부터 한 명씩 차례대로 식사를 하는 라운드-로빈(round-robin, RR) 방식의 비효율성으로 인해 병렬성을 상실하기도 한다.

3장에서는 병렬성, 공정성을 꾀하면서도 교착상태를 원천적으로 차단하는 방법으로 중재자(웨이터)가 $\lfloor n/2 \rfloor$ 명의 그룹을 관리하는 그룹 중재자 해결책(group AS, GAS)을 제안한다.

Ⅲ. 그룹 라운드-로빈 알고리즘

n명의 철학자들 중 동시에 자신의 좌우측 젓가락을 갖고 식사를 할 수 있는 최대 인원은 $\lfloor n/2 \rfloor$ 명이다. 즉, 짝수의 경우 $n/2$명으로 $n=6$인 경우 $\{P_1, P_3, P_5\}$의 홀수와 $\{P_2, P_4, P_6\}$의 짝수인 2-그룹으로 분할시킬 수 있다. 반면에 홀수인 경우 $(n-1)/2$명으로 $n=5$인 경우 $\{P_1, P_3\}$과 $\{P_2, P_4\}$가 그룹을 형성할 수 있고, $P_n = P_5$은 R_{45}, R_{51}을 사용해야 하기 때문에 두 그룹 중 어디에도 포함시킬 수 없어 그룹을 형성하지 못한다. 왜냐하면 $\{P_1, P_3\}$의 P_1과 R_{51}의 공유할 수 없고, $\{P_2, P_4\}$의 P_4와 R_{45}를 공유할 수 없기 때문이다. 따라서 분리 가능한 그룹 집합 내의 프로세서들 간에는 인접하지 않는 독립집합(independent set, IS)이 되어야만 한다. 이러한 동시처리 가능한 그룹 형성 속성을 반영하여 본 장에서는 최대 동시 처리 프로세서 그룹 G를 형성하는 방법을 제안한다. 한 그룹이 작업을 종료하면 다음 그룹으로 권한을 이관하는 방식은 라운드 로빈 방식(RR)을 채택한다.

아메리카 로빈(american robin) 새는 새끼들에게 모이를 줄 때 10마리 새끼가 있으면 조금씩 나눠서 조금 주고, 또 조금 주고, 또 조금 주고 하여 한 라운드를 다 돌면 그 다음에 다시 처음 새끼부터 조금씩 주는 행위를 반복한다고 하여 이 새의 새끼에게 먹이를 주는 방식을 본떠 한 라운드에 시분할로 조금씩 전부 분배하고 다시 반복하는 만든 스케줄링 방식을 라운드 로빈 방식이라 한다.

기존의 RHS나 AS는 어느 하나의 프로세서 작업이 종료되기 전에는 인접한 2개 프로세서는 무한정 기다려야 하는 문제가 있다. 이를 해결하는 방법으로는 RR과 같이 시분할 방식을 채택할 수도 있고, 그룹의 모든 프로세서들이 작업이 종료되는 시점에 다음 그룹으로 이관하는 일괄처리 방식을 적용할 수도 있다.

제안된 알고리즘은, 일단 홀수 번호 프로세서 그룹 $G = P_1, P_3, \cdots, P_k, k = \lfloor n/2 \rfloor$을 형성하여 각자 자신의 좌우측에 놓여있는 젓가락을 집어 식사를 한다.

이 그룹이 식사를 마치면 G의 각 번호 프로세서는 다음 번호 프로세서에게 식사 권한을 양도한다. 이 과정을 n회 수행하면 전체적으로는 1-라운드이지만 실제적으로는 각 프로세서들이 $\lfloor n/2 \rfloor$-라운드를 수행한 결과를 얻는다. $n=5$인 경우를 예로 들면 $(P_1, P_3) \rightarrow (P_2, P_4) \rightarrow (P_3, P_5) \rightarrow (P_4, P_1) \rightarrow (P_5, P_2)$로 n회의 1-라운드 로빈을 수행하면 각 프로세서는 2회씩 식사를 한 결과를 얻는다. $n=7$인 경우를 예로 들면 $(P_1, P_3, P_5) \rightarrow (P_2, P_4, P_6) \rightarrow (P_3, P_5, P_7) \rightarrow (P_4, P_6, P_1) \rightarrow (P_5, P_7, P_2) \rightarrow (P_6, P_1, P_3) \rightarrow (P_7, P_2, P_4)$로 n회의 1-라운드 로빈을 수행하면 각 프로세서는 3회씩 식사를 한 결과를 얻는다. 결국 n회의 1-라운드 로빈을 수행하면 모든 철학자는 $\lfloor n/2 \rfloor$회의 식사를 하는 공정성과 더불어 병렬성도 갖고 있는 방법이라 할 수 있다.

만약 1-라운드(각 철학자가 단 1회 식사)를 수행하는 경우 n=짝수이면 2회를, 홀수이면 3회째에는 P_n만 식사를 하는 방법을 취한다. 만약 n회 수행하면 모든 철학자는 $\lfloor n/2 \rfloor$-라운드(식사)를 하게 된다. 제안된 알고리즘을 그룹 라운드-로빈 알고리즘(group round-robin algorithm, GRRA)라 하며 그림 1과 같이 수행된다.

Ⅳ. 적용 및 결과 분석

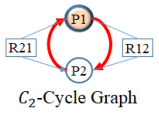
GRRA를 짝수는 2회, 홀수는 3회의 1-라운드(식사)를 수행하는 방식과 n회 수행으로 $\lfloor n/2 \rfloor$-라운드(식사)를 수행하는 경우에 대해 $2 \le n \le 7$에 대해 그림과 행렬로

표현하면 그림 2와 그림 3과 같다. $\lfloor n/2 \rfloor$의 철학자 그룹이 동시에 식사를 하고 난 후 각자는 다음 번 철학자에게 식사 권한을 이관시키는 방식으로 모든 철학자가 1회 식사를 마치는 공정성을 유지하려면 n=홀수인 경우 3번째는 P_n 단독 식사를 하면 된다.

- 프로세서 $P_i, i=1,2,\cdots,n$
- $k=\lfloor \frac{n}{2} \rfloor$, 홀수 프로세서 $P_i, i=1,3,\cdots,k$ 그룹 G 형성.
- s : processing sequence */

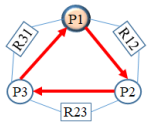
For $s=1$ to 3 /* 1-Round Eating */
  if (n=짝수), ($s=3$) then exit
  else if (n=홀수), ($s=3$) then P_n 식사
  else G 집합 프로세서 동시 식사,
  G의 모든 프로세서 : $P_i \rightarrow P_{i+1}$로 식사 권한 이관 (Shift right.).
End
For $s=1$ to n /* $\lfloor n/2 \rfloor$-Round Eating */
  G 집합 프로세서 동시 식사.
  G의 모든 프로세서 : $P_i \rightarrow P_{i+1}$로 식사 권한 이관(Shift right.).
End

그림 1. 그룹 라운드-로빈 알고리즘
Fig. 1. Group round-robin Algorithm



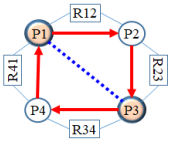
C_2-Cycle Graph

GRR		Philosopher's Eating	
Round	Seq.	P1	P2
1 Round	1	1	
	2		1
No. of Eating		1	1



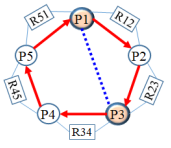
C_3-Cycle Graph

GRR		Philosopher's Eating		
Round	Seq.	P1	P2	P3
1 Round	1	1		
	2		1	
	3			1
No. of Eating		1	1	1



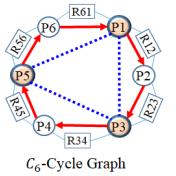
C_4-Cycle Graph

GRR		Philosopher's Eating			
Round	Seq.	P1	P2	P3	P4
1-Round	1	1		1	
	2		1		1
No. of Eating		1	1	1	1



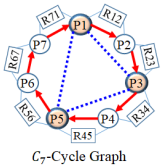
C_5-Cycle Graph

GRR		Philosopher's Eating				
Round	Seq.	P1	P2	P3	P4	P5
1-Round	1	1		1		
	2		1		1	
	3					1
No. of Eating		1	1	1	1	1



C_6-Cycle Graph

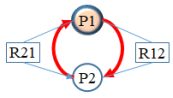
GRR		Philosopher's Eating					
Round	Seq.	P1	P2	P3	P4	P5	P6
1-Round	1	1		1		1	
	2		1		1		1
	3						
No. of Eating		1	1	1	1	1	1



C_7-Cycle Graph

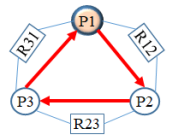
GRR		Philosopher's Eating						
Round	Seq.	P1	P2	P3	P4	P5	P6	P7
1-Round	1	1		1		1		
	2		1		1		1	
	3							1
No. of Eating		1	1	1	1	1	1	1

그림 2. 1-라운드 GRR
Fig. 2. One-round GRR



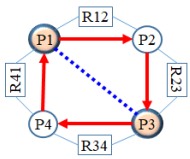
C_2-Cycle Graph

GRR		Philosopher's Eating	
Round	Seq.	P1	P2
1 Round	1	1	
	2		1
No. of Eating		1	1



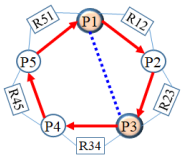
C_3-Cycle Graph

GRR		Philosopher's Eating		
Round	Seq.	P1	P2	P3
1 Round	1	1		
	2		1	
	3			1
No. of Eating		1	1	1



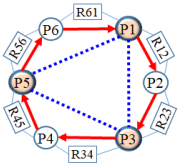
C_4-Cycle Graph

GRR		Philosopher's Eating			
Round	Seq.	P1	P2	P3	P4
2-Round	1	1		1	
	2		1		1
	3	1		1	
	4		1		1
No. of Eating		2	2	2	2



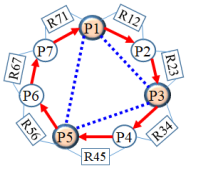
C_5-Cycle Graph

GRR		Philosopher's Eating				
Round	Seq.	P1	P2	P3	P4	P5
2-Round	1	1		1		
	2		1		1	
	3			1		1
	4	1			1	
	5		1			1
No. of Eating		2	2	2	2	2



C_6-Cycle Graph

GRR		Philosopher's Eating					
Round	Seq.	P1	P2	P3	P4	P5	P6
3-Round	1	1		1		1	
	2		1		1		1
	3	1		1		1	
	4		1		1		1
	5	1		1		1	
	6		1		1		1
No. of Eating		3	3	3	3	3	3



C_7-Cycle Graph

GRR		Philosopher's Eating						
Round	Seq.	P1	P2	P3	P4	P5	P6	P7
3-Round	1	1		1		1		
	2		1		1		1	
	3			1		1		1
	4	1			1		1	
	5		1			1		1
	6	1		1			1	
	7		1		1			1
No. of Eating		3	3	3	3	3	3	3

그림 3. $\lfloor n/2 \rfloor$-라운드 GRR
Fig. 3. $\lfloor n/2 \rfloor$-round GRR

이를 일반화하면 $2 \le n \le \infty$의 어떠한 멀티프로세싱에 대해서도 $\lfloor n/2 \rfloor$-라운드 GRR을 적용하면 교착상태

없이 모든 프로세서가 동일한 횟수의 실행을 할 수 있음을 알 수 있다.

DPP 해결책의 주 목적은 식사를 못해 굶는 철학자가 발생하지 않는 공정성 확보, 모든 철학자들이 전혀 식사를 하지 못하는 교착상태 원천 봉쇄, 동시에 여러 철학자들이 식사를 할 수 있는 병행성 확보를 갖고 있으면서도 모든 철학자가 식사와 사색에 잠기는 행위를 영원히 반복할 수 있는 알고리즘 개발이다.

결국, $\lfloor n/2 \rfloor$−라운드 GRR을 무한 반복 수행하면 DPP 해결책 알고리즘의 목적을 달성할 수 있다.

V. 결 론

본 논문은 다중 병행(parallel concurrency) 처리 운영체제에서 교착상태 해소, 공정성, 병행성과 무한 반복성을 갖는 방법을 연구하였다. 기존에는 5명의 철학자들 만찬문제(DPP)에 한해 해결책들이 제시되었지만 본 논문은 $2 \leq n \leq \infty$로 일반화된 해결책을 제시하였다.

제안된 방법은 $\lfloor n/2 \rfloor$명이 한 조를 이루어 식사를 하고 각 철학자는 다음 번호 철학자에게 식사권한을 양도하는 간단한 그룹 라운드-로빈 방법이다. 제안된 방법을 n회 수행하면 각 철학자는 $\lfloor n/2 \rfloor$회의 식사를 하는 공정성과 병행성 뿐 아니라 교착상태도 원천 봉쇄할 수 있는 방법으로 n회의 라운드-로빈을 무한 반복하면 모든 철학자가 식사와 사색에 잠기는 행위를 영원히 반복할 수 있어 DPP가 추구하는 모든 목적을 달성한 알고리즘이라 할 수 있다

References

[1] P. McClanahan, “6. Concurrency: Deadlock and Starvation, 6.4: Dining Philosopher Problem,” Operating System: The Basics, https://eng.libretexts.org/Courses/Delta_College/Operating_System_The_Basics_Deadlock/6.4_Dining_Philosopher_Problem, Jan. 2021.

[2] E. W. Dijkstra, "Hierarchical Ordering of Sequential Processes," Acta Informatica, Vol. 1, No. 2, pp. 115-138, Jun. 1971, https://doi.org/10.1007/BF00289519

[3] C. A. R. Hoare, "Communicating Sequential Processes," Communications of the ACM, Vol. 21, No. 8, pp. 666-677, Aug. 1978, https://doi.org/ 10.1145/359576.359585

[4] J. Díaz and I. Ramos, "Formalization of Programming Concepts," Proceedings of International Colloquium on the Formalization of Programming Concepts, pp. 323-326, Apr. 1981.

[5] D. J. Lehmann and M. O. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem," Proceedings of the 8th ACM SIGPLAN- SIGACT symposium on Principles of Programming Languages, pp. 133-138, Jan. 1981, https:// doi.org/10.1145/567532.567547

[6] D. Bhargava and S. Vyas, "Agent Based Solution for Dining Philosophers Problem," International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions), pp. 563-567, Dec. 2017, https://doi.org/10. 1109/ICTUS.2017.8286072

[7] K. M. Chandy and J. Misra, "The Drinking Philosophers Problem," ACM Transactions on Programming Languages and Systems, Vol. 6, No. 4, pp. 632-646, Oct. 1984, https://doi.org/10.1145/1780.1804

[8] T. A. Cargill, "A Robust Distributed Solution to the Dining Philosophers Problem," Journal of Software: Practice and Experience, Vol. 12, No. 10, pp. 965-969, Oct. 1982, https://doi.org/10.1002/spe.4380121009

[9] J. Gutknecht, "The Dining Philosophers Problem Revisited," Proceedings of the 7th joint conference on Modular Programming Languages, pp. 377-382, Sep. 2006, https:// doi.org/10.1007/11860990_23

[10] D. Ruiz, R. Corchuelo, J. A. Pérez, and M. Toro, "An Algorithm for Ensuring Fairness and Liveness in Non-Deterministic Systems Based on Multiparty Interactions," Parallel Processing, Proceedings of 8th International Euro-Par Conference, pp. 563-572, Aug. 2002, https://doi.org/10.1007/3- 540-45706-2_77

[11] A. P. L. Ferreira, L. Foss, and L. Ribeiro, "Formal Verification of Object-Oriented Graph Grammars Specifications," Electronic Notes in Theoretical Computer Science, Vol. 175, No. 4, pp. 101-114, Jul. 2007, https://doi.org/10.1016/j.entcs.2007.04. 020

[12] K. S. Cheung, “The Dining Philosophers Problem,” Augmented Marked Graphs, pp. 81-92, May 2014, https://doi.org/10.1007/978-3-319-06426-4.6

[13] A. K. Bandyopadhyay, “Fairness and Conspiracy Concepts in Concurrent Systems,” AGM SIGSOFT Software Engineering Notes, Vol. 34, No. 2, pp. 1-8, Feb. 2009, https://doi.org/10.1145/1507195. 1507203

[14] Z. Ramadhan and A. P. U. Siahaan, “Dining Philosophers Theory and Concept in Operating System Scheduling,” IOSR Journal of Computer Engineering, Vol. 18, No. 6, pp. 45-50, Dec. 2016.

[15] A. Silberschatz, G. Gagne, and P. B. Galvin, “Operating System Principles (7th ed.),” Wiley India Pvt., pp. 237, 2006, ISBN: 9788126509621.

저 자 소 개

이 상 운(정회원)

- 1987년 : 한국항공대학교 항공전자공학과 (학사)
- 1997년 : 경상대학교 컴퓨터과학과 (석사)
- 2001년 : 경상대학교 컴퓨터과학과 (박사)
- 2003년 : 강원도립대학 컴퓨터응용과 전임강사
- 2004년 ~ 2007.2 : 국립 원주대학 여성교양과 조교수
- 2007.3 ~ 2015.3 : 강릉원주대학교 멀티미디어공학과 부교수
- 2015.4 ~ 현재 : 강릉원주대학교 멀티미디어공학과 정교수
- 관심분야 : 소프트웨어 프로젝트 관리, 개발 방법론, 분석과 설계 방법론, 시험 및 품질보증, 소프트웨어 신뢰성, 인공지능과 빅데이터분석, 최적화 알고리즘
- e-mail : sulee@gwnu.ac.kr