# Avoiding vs. Detecting: Evaluating Deadlock Strategies in the Dining Philosophers Problem

*Jordyn Reichert*
*Dept. of Computer Science*
*Whitworth University*
*Spokane, WA, United States*
*jreichert26@my.whitworth.edu*

*Alexander Leonida*
*Dept. of Computer Science*
*Whitworth University*
*Spokane, WA, United States*
*aleonida27@my.whitworth.edu*

***Abstract*—The Dining Philosophers problem highlights a long-standing issue in operating systems (OS). In a struggle for limited resources, processes compete with each other until they hit a crucial problem: Deadlock. Deadlock occurs when resources are unable to be allocated to processes. In the case of the dining philosophers problem, this happens when philosophers are unable to obtain both chopsticks to eat. This issue is commonly addressed in two ways: detection and avoidance methods. Detection methods allow deadlock to occur and handle the situation accordingly, while avoidance prevents deadlock from ever occurring through strict rules. By measuring each method's performance using five metrics: average waiting times, deadlock occurrences, minimum and maximum philosopher wait times, thinking-to-eating ratios, and fairness of resource distribution over five philosophers, we assess the scalability and fairness of each deadlock solution.**

***Keywords*—Deadlock Avoidance, Deadlock Detection, Dining Philosophers Problem, Concurrency Control, Resource Allocation, Simulation-Based Analysis**

## I. Introduction

The Dining Philosophers problem demonstrates competing resources within operating systems, where philosophers (processes) compete for chopsticks (resources) to eat [1]. Deadlocks occur when each philosopher holds one chopstick and waits indefinitely for another, creating a circular wait condition [2]. Two primary strategies address this issue: avoidance techniques, which prevent deadlock through strict resource acquisition protocols, and detection techniques, which identify and resolve deadlocks when they occur. Recent academic sources highlight trade-offs between the two methods. Avoidance methods like ordering [5] ensure deadlock-free execution, while detection methods [6] offer more flexibility but have to deal with the cost of recovering from the deadlocks that occur.

This paper evaluates four deadlock strategies implemented in a simulation using the C programming language: (1) conditional release (releasing the left chopstick if the right is unavailable), (2) asymmetric ordering (even/odd philosophers prioritize different chopsticks), (3) full release (all philosophers drop their chopsticks on deadlock), and (4) selective release (one philosopher drops their chopsticks). We measure average waiting times, the number of deadlock occurrences, minimum and maximum philosopher wait times, and thinking-to-eating ratios. We also analyze the fairness of resource distribution over 5 - 100 philosophers, incorporating multiple simulation runs for each deadlock solution. Our analysis, supported by [6]-[9], informs the design of concurrent systems.

## II. Proposed Strategies

### A. System Model

We model 5, 20, and 100 philosophers seated at a circular table, each requiring two adjacent chopsticks to eat. Chopsticks are shared resources, and deadlocks arise from circular wait conditions [2]. Four strategies are implemented:

1. Conditional Release (S1): Philosophers acquire the left chopstick and release if the right is unavailable, retrying after a delay [4].
2. Asymmetric Ordering (S2): Even-numbered philosophers acquire the left chopstick first, and odd-numbered ones the right, avoiding circular waits [4].
3. Full Release (S3): A separate monitor thread detects deadlocks (when all philosophers are hungry (waiting for chopsticks) and forces all philosophers to release any chopsticks they have [6].
4. Selective Release (S4): The monitor thread forces one philosopher (cycled sequentially) to release chopsticks, minimizing disruption [6].

### B. Algorithm Design

The core algorithm for each philosopher thread is:

```
Initialize: Make (binary) semaphores available (value = 1), update philosopher states (THINKING), set simulation duration (scalable).
While elapsed time < sim duration, do
    Think for 10 ms, update thinking time
    Set state to HUNGRY, record wait start
    If S1 then
    Acquire the left chopstick
    If right unavailable, release left, delay 1ms, then retry
    Else if  S2 then
    Even ID: Acquire left, then right;
    Odd ID: Acquire right, then left
    Else if S3 or S4 then
    Acquire left, then right (monitor handles deadlocks)
    End
    Set state to EATING, record stats, and eat for 10 ms
```

```
        Release chopsticks, set state to THINKING
    **While** monitor thread (if S3, S4) checks every 100ms:
        **While** simulation not completed, **do**
            **If** all philosophers are HUNGRY **then**
            Increment deadlock count
                **If** S3 **then**
                Force all philosophers to release
                chopsticks, set state to THINKING
                **Else if** S4 **then**
                Force next philosopher to release
                chopsticks, set state to THINKING
```

These algorithms leverage [5] for avoidance and [6], [7] for detection, with S4 inspired by selective recovery.

## III. Experimental Results and Comparisons

### C. Setup

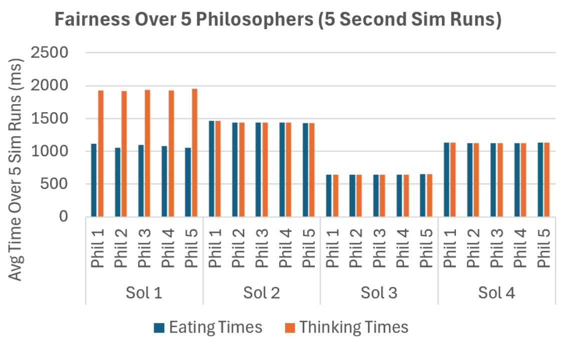
We implemented the simulation in C using POSIX threads on a machine with an Intel i7 processor and 16GB of RAM, running the latest version of the Linux Ubuntu operating system. Metrics include: Distribution of eating/thinking times over individual philosophers (to measure fairness), the number of deadlocks that occurred, min/max waiting times (ms, individual philosophers), average thinking time to average eating time ratios (ms, over all philosophers), and average waiting time (ms, over all philosophers).

Each deadlock strategy was simulated for 20 seconds, with 10 ms static eating and thinking times and 5 ms delays to induce contention. Tests were conducted with 5, 20, and 100 philosophers for each method. Metrics were logged to CSV files to import into Excel for graphical analysis.

### D. Initial Results: Fairness Over Five Philosophers

Figure I exhibits the fairness of resource distribution over 5 philosophers for each solution method:

Figure 1. Fairness Over Five Philosophers



### Low-Level Analysis

The fairness graph shows that most solutions distribute chopsticks relatively evenly among the philosophers, with one exception. S1 results in a noticeable difference between individual philosophers, where some philosophers consistently have higher average eating and thinking times. This indicates a large difference caused by the randomness of attempts between philosophers. In contrast, S2, S3, and S4 showcase more even distributions. S2 reduces competition between philosophers and improves fairness compared to S1. Furthermore, S3 and S4, both of which use deadlock detection and intervention mechanisms, show almost identical averages across all philosophers. These solutions actively prevent starvation, which suggests that even minimal deadlock detection logic can significantly equalize access to resources.

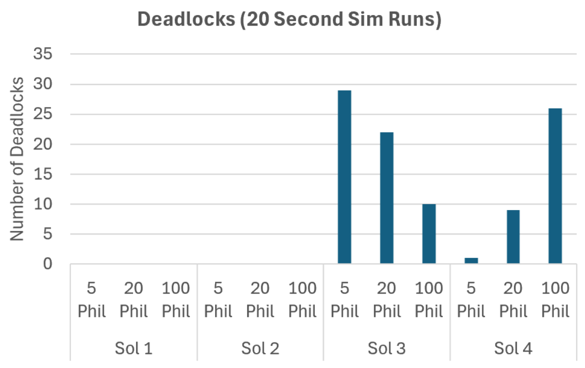
### High-Level Analysis

These results illustrate the trade-offs between fairness and simplicity. S1's simplicity comes at the cost of fairness, where, without any scheduling awareness or conflict resolution logic, some philosophers gain a subtle but consistent advantage. On the other hand, the nearly uniform averages in S2, S3, and S4 highlight the value of introducing structured access patterns or active deadlock resolution.

### E. Number of Deadlock Occurrences Under Scaling

Figure II summarizes the performance impact from increasing the number of philosophers by presenting the number of deadlocks that occur:

Figure II. Number of Deadlocks



### Low-Level Analysis

In tests across 5, 20, and 100 philosopher counts, deadlock frequency showed inverse trends for detection strategies S3 and S4. For S3, deadlocks decreased as the number of philosophers increased, whereas in S4, deadlocks increased with higher philosopher counts. This aligns with findings in [9], where larger groups introduce more randomized contention patterns, reducing repeatable cycles of dependency.

### High-Level Analysis

The decrease in deadlocks for S3 at higher philosopher counts suggests that natural randomness in larger groups provides implicit deadlock avoidance. As the number of philosophers increases, the probability of reintroducing a circular wait after chopsticks are dropped decreases due to a higher possibility of timing variations in chopstick acquisition.

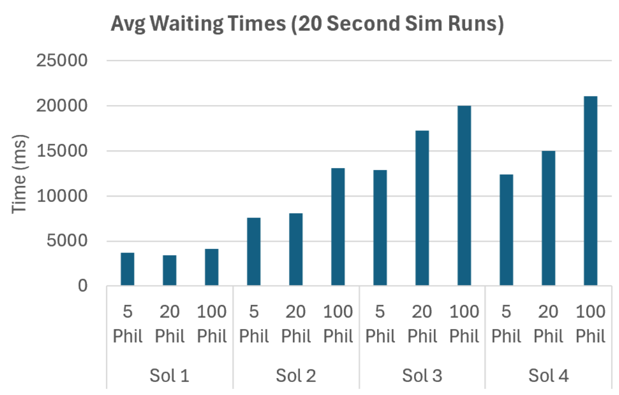
S4, on the other hand, suffers at scale because its selective release quickly proves to be too difficult to manage. As more philosophers drop chopsticks and reattempt to acquire them, collisions become more likely. The strategy, while helpful for smaller sets of processes, does not scale linearly. More philosophers mean a higher chance that dropped chopsticks will be immediately reacquired, leading to renewed deadlock.

To improve S4 at scale, we propose a priority-based reentry delay, where fairness is factored into which philosopher is chosen to drop their chopstick when a circular wait condition is detected. This may reduce premature reacquisition and increase resolution efficiency.

## F. *Average Waiting Times Under Scaling*

Figure III shows the impact on efficiency when scaling each solution method by displaying the average waiting times across philosophers:

FIGURE III. AVERAGE WAITING TIMES



*Low-Level Analysis*

The variation in average waiting times across the three philosopher counts reveals distinct performance patterns among the solutions. S1 consistently achieves the lowest waiting times, with values staying relatively stable (ranging between 3444 ms and 4133 ms) regardless of group size. This shows a high level of responsiveness, whereas in contrast, S2 shows a moderate increase in waiting time as the philosopher count rises, nearly doubling from 7550 ms at 5 philosophers to 13113 ms at 100 philosophers. S3 and S4 exhibit the most significant delays, both starting above 12,000 ms and reaching over 20,000 ms at 100 philosophers, indicating a decline in performance when scaling. Despite their different deadlock handling strategies, both seem to introduce more overhead during chopstick acquisition cycles.

*High-Level Analysis*

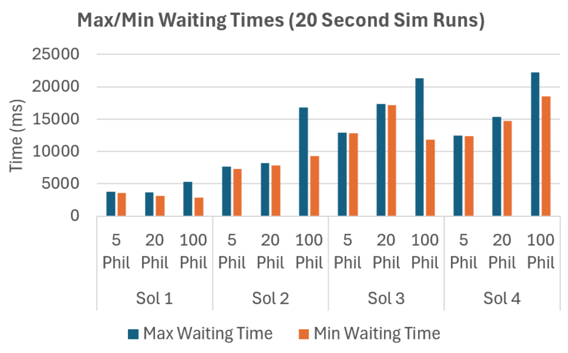
The average waiting times reinforce the scalability of S1. Its ability to maintain low waiting times suggests minimal blocking and a smooth transfer of resources. S2, though deadlock-free, imposes stricter acquisition rules, which may introduce delay as the philosopher count increases.

Meanwhile, S3 and S4's detection-based methods appear to sacrifice throughput for safety, especially with higher philosopher counts. Their response mechanisms (forced release and reattempts) likely create bottlenecks, as philosophers repeatedly fail and retry in bursts, increasing contention and idling. This aligns with prior observations [9], where detection-heavy strategies often trade efficiency for system progression. To improve scalability in S3 and S4, future simulations could possibly attempt to implement slight delays before retries to reduce collision likelihood.

## G. *Refined Fairness: Min/Max Individual Waiting Times*

Figure IV presents the equity across philosophers by displaying the min and max individual waiting times for each solution:

FIGURE IV. MIN/MAX WAITING TIMES



*Low-Level Analysis*

The minimum and maximum individual waiting times provide critical insight into fairness within each solution. At first glance, S1 appears to have the closest wait time range across all philosopher counts (under 1500 ms for the first two philosopher counts), which may suggest strong fairness. However, this is because of its lower overall waiting times. Due to S1's strategy, some philosophers succeeded in eating significantly more often than others, while other philosophers starved, resulting in hidden unfairness, which could not have been captured by the min/max values.

S2 shows a larger gap, especially at 100 philosophers (with a min of 9,305 ms and a max of 16,802 ms), but its acquisition order promotes more uniform access. This predictability leads to a fairer overall distribution of eating opportunities, even if wait times seem to vary more at first glance.

On the other hand, S4 ends up with the most balanced actual distribution of eating chances, even though it employs a detection-based method. While the min-to-max range approaches nearly 4 seconds in the 100-philosopher case, no philosopher repeatedly dominates or starves. No philosophers ever have extended wait times. Instead, its randomized strategy levels out the distribution of resources over time by cycling which philosopher must reset. In contrast, S3 displays the least fairness, with some philosophers waiting more than 21,000 ms, while others proceed far more quickly. This is a symptom of its reset method that can leave unlucky philosophers behind repeatedly.

*High-Level Analysis*

The variation in individual philosopher waiting times reveals the fairness of each strategy. While S1's close range gives the impression of equality, it doesn't guarantee fair access. S2's broader min-max ranges stem from more systematic, fair scheduling, where all philosophers cycle through access relatively evenly. Its left-to-right ordering lowers the chances of long-term starvation and provides stable behavior when scaling.

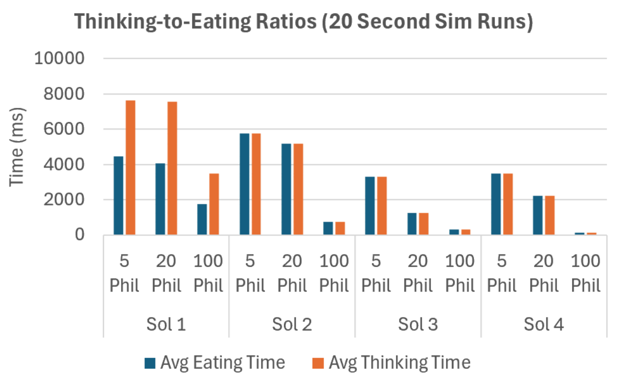
Surprisingly, S4 performs best in terms of actual fairness of access, despite larger min-max waiting ranges. Its randomized deadlock resolution, while less efficient, avoids letting the same philosophers repeatedly acquire chopsticks while others starve. S3, on the other hand, shows high variability and repeatedly unfair delays due to its indiscriminate behavior.

These results show that fairness is not just about minimizing wait times; it's about ensuring equal opportunities over time, even at the cost of some efficiency. Future strategies might benefit from combining S4's fair disruption pattern with more efficient scheduling methods like those in S2.

## H. *Thinking-to-Eating Ratios Under Scaling*

Figure V summarizes the performance impact from increasing the number of philosophers by presenting the average thinking-to-eating ratios:

FIGURE V. THINKING/EATING RATIOS



*Low-Level Analysis*

The thinking-to-eating ratio helps to illustrate how time is distributed between thinking and eating. Most solutions, especially S2, S3, and S4, maintained roughly balanced ratios near 1:1 across all philosopher counts. With this being said, S1 consistently deviated from this trend, showing thinking to eating ratios reaching near 2:1. This suggests that while some philosophers ate frequently, others spent a disproportionate amount of time thinking, indicative of fairness issues or starvation.

Additionally, S3 and S4, while maintaining balanced ratios, had much lower overall thinking and eating times compared to the other strategies. For example, in S4, the average thinking time dropped from 3502 ms at 5 philosophers to just 121 ms at 100, with eating time decreasing similarly. This reduction in total time spent in each state implies that philosophers were constantly being interrupted, likely due to frequent deadlock detection and forced chopstick releases. This limited their ability to enter or remain in stable states.

S2, by contrast, maintained both balanced ratios and reasonable thinking/eating durations, scaling predictably even as contention increased. This indicates a more orderly distribution of chopstick access, preventing starvation without resorting to frequent interruptions.

*High-Level Analysis*

The thinking-to-eating data reveals subtle but critical performance trade-offs. S1's high ratios stem not from philosophers eating too little overall, but from inequality in access, where some philosophers monopolize eating cycles while others wait excessively, leading to inflated thinking times. Despite strong throughput in some runs, this imbalance showcases large unfairness.

S2's consistent ratios reflect a fairer and scalable solution. Philosophers transition between thinking and eating in predictable cycles, avoiding both starvation and excessive context switching. This supports the interpretation of S2 as the fairest solution out of the two fastest solutions (S1 and S2).

In contrast, S3 and S4 reveal the limitations of detection-based methods. Although they appear balanced ratio-wise, their sharp drop in average time spent thinking and eating at higher loads indicates inefficiency. Philosophers are frequently interrupted or reset before completing a full cycle. While these methods attempt to correct deadlock after it's occurred, they often do so at the cost of throughput and stability.

It's also important to note that the structure of the simulation inherently constrains the average eating times: philosophers attempt to think during every cycle, while eating only occurs when both chopsticks are successfully acquired. This means that even in optimal deadlock-free conditions, eating time cannot exceed thinking time on average.

## IV. CONCLUSION

Our evaluation of four deadlock-handling strategies for the Dining Philosophers Problem reveals significant trade-offs in fairness, scalability, and efficiency across avoidance and detection-based approaches. Avoidance strategies (S1, S2) prevent deadlocks entirely, but their fairness varies because of this fact. S1's conditional release achieves high throughput with stable waiting times (3444–4133 ms across 5 to 100 philosophers) but produces the least fair outcomes, due to its mechanism, which allows some philosophers to dominate eating cycles, resulting in thinking-to-eating ratios near 2:1 and hidden starvation [9]. S2, using asymmetric ordering, avoids deadlocks while ensuring greater fairness, with balanced 1:1 ratios and equitable eating distributions, though waiting times increase to 13,113 ms at 100 philosophers [5]. Detection-based strategies (S3, S4) incur higher overhead due to deadlock resolution. S3's full release reduces deadlock frequency (from 15 at 5 philosophers to 8 at 100) by leveraging timing variations in larger groups, but its resets lead to long waiting times (over 20,000 ms) and unfairness, with some philosophers delayed up to 21,346 ms [7]. S4's selective release achieves the most balanced eating distribution, with no philosopher repeatedly dominating despite a 4000 ms min-to-max wait gap at 100 philosophers, but its deadlock frequency rises (10 to 25) when scaled due to increased acquisition collisions.

Regarding scalability, S1 presents high efficiency at the cost of equity, while its low waiting times hide significant unfairness in resource access. S2 offers a strong compromise, combining fairness and predictable performance, making it suitable for systems prioritizing equal access. S4's fairness improvements come with scalability challenges, as increased competition drives deadlock frequency, while S3's excessive resets lead to unfairness when scaling. To enhance S4, a priority-based reentry delay could decrease the amount of premature chopstick reacquisition, ensuring fairness while improving resolution efficiency [9]. For S3, implementing staggered release timing delays could prevent reentering circular wait conditions, as simultaneous releases risk instantly recreating deadlocks [10]. These findings, supported by [5], [7], [9], and [10], suggest that S2 is ideal for systems requiring both deadlock prevention and fairness, while an enhanced S4 could excel in fairness for future reference.

Future work could reference hybrid approaches integrating S2's ordering with S4's randomized resolution to balance deadlock prevention and fairness under high concurrency. Incorporating machine learning for predictive deadlock detection [3] could also proactively manage contention, reducing the need for recovery.

## References

[1] A. Silberschatz, P.B. Galvin, and G. Gagne, Operating System Concepts, 10th ed. Wiley, 2018.

[2] M. Raynal, Concurrent Programming: Algorithms, Principles, and Foundations. Springer, 2016.

[3] A.N. Habermann, "Prevention of system deadlocks," Commun. ACM, vol. 12, no. 7, pp. 373–377, 1969.

[4] K.M. Chandy and J. Misra, "The drinking philosophers problem," ACM Trans. Program. Lang. Syst., vol. 6, no. 4, pp. 632–646, 1984.

[5] P.A. Buhr, M.H. Coffin, and M. Fortier, "Monitor classification," Software: Practice and Experience, vol. 27, no. 5, pp. 533–552, 1997.

[6] M. Ben-Ari, Principles of Concurrent and Distributed Programming, 2nd ed. Pearson, 2006.

[7] Q. Wu, A. Pakki, N. Emamdoost, S. McCamant, and K. Lu, "Understanding and detecting disordered error handling with precise function pairing," in Proc. 30th USENIX Security Symp., 2021, pp. 2041–2058.

[8] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol. 21, no. 7, pp. 558–565, 1978.

[9] W. Lu, Y. Yang, L. Wang, W. Xing, and X. Che, "A novel concurrent generalized deadlock detection algorithm in distributed systems," in Proc. 15th Int. Conf. Algorithms and Architectures for Parallel Processing, Part II, 2015, pp. 479–493.

[10] D. Saxena, J. Kumar, and A. K. Singh, "Performance analysis of machine learning centered workload prediction models for cloud," IEEE Trans. Parallel Distrib. Syst., vol. 34, no. 4, pp. 1313–1330, 2023.