CS 3013 - Operating Systems

Project 2 (90 points)
Assigned: Tuesday, February 9, 2021
Checkpoint: Tuesday, February 16, 2021 at 11:59:59pm
Due: Thursday, February 19, 2021 at 11:59:59pm

Project 2: Synchronization and Concurrency

The following two problems will give students a chance to practice synchronization. The students must create threads for each of the actors and debug in a synchronous way. Each thread must use a degree of randomness in delays to test the scheduling. The random numbers should be seeded using `srand()` with the number in a `seed.txt` file (like in Project 1). This will allow students to change the seed value to test different scenarios while allowing one to create reproducible scenarios for debugging purposes.

The students must solve one of these synchronization problems with both mutexes and condition variables (e.g., `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_cond_wait`, and `pthread_cond_signal`) and the other with semaphores (e.g., `sem_wait`, `sem_post`, and `sem_init`). The students can decide which to use for each problem, but **students cannot use the same primitive type for both**. Both can be solved with either, but one way may be more straightforward than the other. All scheduling must be uncontrolled; students **may not** use a centralized coordinator that chooses which thread runs at any given time.

Problem 1: WPI's Summer Spectacular

After the "year of COVID-19," the WPI community decided it needed to celebrate the arrival of widespread vaccines and the ability to gather with lower risks. They decided to host a campus-wide event, with students, staff, and faculty providing an array of events and activities for the festivities.

After spending days failing to come up with any good event ideas, Prof. Shue knew what to do. He went over to the Foise Innovation Studio and allowed the aura of innovation to wash over him. Inspiration came to him suddenly, and he quickly enacted his plan.

Prof. Shue called a meeting with Profs. Guo and Walls and presented his idea. "As you may know, jazz improvisation is a highly regarded musical art form," he began. "But, it's been done before. We need something new. You know what else people like? Circuses. A bunch of stage acts that 'wow' an audience. We can innovate by combining these two into a single event!" After misinterpreting their polite smiles as agreement, Prof. Shue strong-armed his colleagues into joining the undertaking. And thus, "Circus Jazz, LLC" was born.

The team assembled a variety of performers and stage acts. These performers included: 1) a company of professional Flamenco dancers with boldly colored dresses, 2) a couple soloists (a magician and an opera singer), and 3) a squad of torch jugglers. In keeping with Prof. Shue's jazz motif, the performers would independently choose when to take the stage, resulting in each performance becoming an improvisational masterpiece. Or at least that was the goal.

Prof. Shue invited President Leshin to attend a Circus Jazz practice performance to convince her to add the event to the Summer Spectacular. Unfortunately, things did not go as well as he hoped. During the practice, a torch juggler and a Flamenco dancer ended up sharing the stage and, well, got a little too close together. The juggler was briefly distracted and missed a torch, which fell onto the dancer's dress, which promptly caught fire. Fortunately, with quick action from a nearby Fire Protection Engineering student, the blaze was contained without causing injury. However, President Leshin was unimpressed by the mishap and even less impressed by Prof. Shue's attempt at a "flaming-co dancer" pun. "Prof. Shue, this is completely unacceptable," she said. "We need a 'Summer Spectacular' and yet all you bring me is a 'Summer Spectacle.' You must fix this, or I will be forced to strike Circus Jazz from the event list."

Dejected, Prof. Shue called another meeting with Profs. Guo and Walls. Prof. Guo continued to protest that she never agreed to be involved in the Circus Jazz undertaking. Meanwhile, Prof. Walls had a

suggestion. "You know, we could just have each performer go on stage one-at-a-time and avoid the whole 'flaming dress' situation," he said. However, Prof. Shue was having none of that. "But my dear Prof. Walls, this is supposed to be a 'Summer Spectacular.' A single performer at a time would be, at best, 'Spectacu-lame'." The team appeared to be at an impasse.

Then, inspiration struck a second time. "At the risk of encouraging the two of you," Prof. Guo began, "I may have an idea." Prof. Guo noted that the three faculty would each be teaching an operating systems class that Spring. "Combined, we have over 150 students. Surely, one of our student teams will come up with a way to safely synchronize the performance, while still ensuring the individual improvisational decision-making." Prof. Shue enthusiastically agreed while Prof. Walls simply conceded that they lacked any other options[1].

Enabling Improvisational "Circus Jazz" Performances

The operating systems professors agreed to assign the Circus Jazz assignment to their students, which brings us to the matter at hand. They agreed upon the following constraints for the assignment:

1. **Performance Safety:** Simply put, Flamenco dancers and torch jugglers could not be on the stage at the same time. However, Flamenco dancers could share the stage with other dancers while jugglers could share the stage with other jugglers. Further, a soloist could only be on stage alone. (Prof. Guo argued "the soloist constraint doesn't really seem to belong in the 'safety' category," but Prof. Walls quickly countered "All I am saying is that you risk your own safety by promising a performer a solo and then making them share the stage.")

2. **Maximum Parallelism:** If a dancer is on stage, and other dancers are ready, they should all be able to share the stage (up to the 4 performer stage limit), leading to a more 'spectacular' performance. Likewise, jugglers who are ready should be able to join an active juggler on the stage. (Prof. Shue reiterated that this was key to the event.)

3. **Fair Performance Opportunities:** Every performer type should have an opportunity to perform without having to wait forever. The dancers could not deprive the jugglers from performing simply by continuously having a new dancer join before a current dancer departs, thus monopolizing the stage. (In private, the OS programmer types call this the "starvation" constraint, but they were afraid to use that term around the performers.)

4. **Performance Time Variability and Bounds:** Each performer improvises on the stage, leading to random execution times for each performer. However, the performance must have a fixed upperbound given the performer exertion and union contracts.

5. **Pre-performance Napping:** If a performer is ready to go on stage, they must either immediately enter the stage (if permitted under the stage limit and constraints #1 (Performance Safety) and #3 (Fair Performance Opportunities)) or they must take a nap. Importantly, a performer may not busy-wait, since that is directly correlated with performer anxiety.

In total, Circus Jazz LLC hired 15 Flamenco dancers, 8 torch jugglers, and 2 soloists. The stage can hold only up to 4 performers at a time and each of the four stage positions can hold only up to 1 performer. Students must synchronize the stage under these constraints. Students may assume that conflicts only occur on stage (i.e., the torches are only aflame during an act), so performers may congregate off-stage without issues.

Students must implement code that creates the requisite number of Flamenco dancers, torch jugglers, and soloist threads. They must use either semaphores or the combination of mutexes and condition variables to properly synchronize the stage. Each dancer, juggler, and soloist thread should identify itself (by name or number) and which stage position (e.g., 1, 2, 3, or 4) it is using when it enters or leaves the stage. Students

[1]Disclaimer: In case it was not obvious, this project narrative is a work of fiction to motivate the assignment. None of the of the events or quoted dialog actually occurred.

should simulate a dancer, juggler, or soloist's performance using the `sleep()` call with a random wait time. Students should explain how their solution avoids depriving the different performer types of the stage in a text file, `problem1_explanation.txt`, that will be submitted along with the source code.

Problem 2: Shipping Shape-Up (or Ship-Shaping Up)

The package shipping and delivery industry is quite competitive. Unfortunately, shipping company FedOops was having a bad year. They had developed a bad reputation of losing, damaging, and misdelivering packages. This even resulted in name calling, with customers causing the #FedUpsWithFedOops tag to trend on social media. The FedOops marketing team was tasked with creating a solution to the problem.

"What we really need is a way to let the customer feel greater ownership of the delivery experience," the head of the FedOops marketing team declared. "Our competitors do not allow their customers to be involved in their logistics, which creates an opportunity for us to differentiate ourselves." Multiple people in the room nodded along, so FedOops decided to enact this plan.

The FedOops Chief Morale Officer (CMO) noted that the FedOops employees were feeling demotivated given the media coverage of the company. The CMO decide that the best way to handle the situation was to make the work more exciting. He divided the logistics division into a set of groups. Each group would have a different team (color-coded into red, green, blue, and yellow). Each shift, the teams within each group would compete to see which team could process the most packages, which would yield a bonus for the day. Further, the group that had the most overall packages processed would independently receive a bonus. Accordingly, each team was incentivized to do the best within each group while simultaneously helping the group be the best within the division. Each team thus acts independently but tries to avoid negatively impacting other teams within the group.

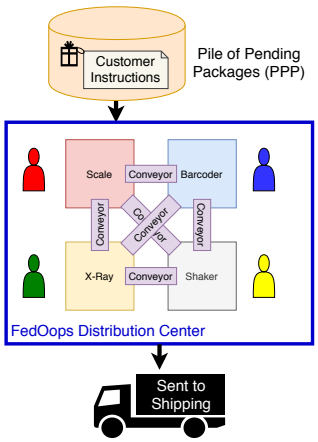

Figure 1: The FedOops Distribution Center.

The workers in the logistics division must ensure that each package proceeds through a set of steps at each distribution center. Typically, a package must go through the following steps:

1. **Weighing:** The package weight is determined and recorded on the packaging.
2. **Barcoding:** The package address must be encoded into a machine-readable barcode.
3. **X-raying:** The package must be x-rayed to ensure it does not contain hazardous materials.
4. **Jostling:** The package must be at least lightly shaken (and, if labeled "fragile", it must be violently shaken). This step is key to preserving the FedOops service reputation and brand.

With the new "customer centric" initiative, FedOops now allows customers to dictate the steps, and the order of the steps, that will be performed on their packages. This caused some consternation during an "all hands" meeting in which a logistics worker asked the CMO if this meant that customers could omit the "jostling" stage on packages. In response, the CMO sighed and replied "yes, it's possible... frankly, we just have to trust the customers know what they're doing... and if that means the occasional package is shipped without jostling, well, that's just an unfortunate consequence of this new direction." While this response left nobody happy, it was agreed that the situation demanded such desperate measures.

In Figure 1, we provide a diagram overview of the process at a typical FedOops distribution center. A pile of packages is provided to each logistics group. Four teams in the group work to process the packages. Each package will have a set of customer instructions attached which describe which steps should be taken and the order in which they are taken. Each team may have one representative working with a package at a time. Upon completing the processing for a package, the team member will "tag in" a new team member to handle the next package. Each group has 40 workers, with 10 workers per team.

Once a worker picks up a package, they must commit to processing it as the customer instructions dictate. The worker cannot put the package back just because they do not like the set of instructions or the ordering. Each set of customer instructions indicates between 1 and 4 steps (inclusively) that will be performed on the package. The worker must carry the package to the station associated with the first listed step (e.g., the scale for weighing, the barcoder for barcoding, the x-ray for x-raying, and the shaker for jostling).

Given the need for efficiency, FedOops policy states that once the package is placed on a station, it must move directly between stations using a set of conveyor belts. In other words, once an object is placed into a station, it must move from station to station without being set aside. No two objects can be in the same station. To move an object, the destination station must be empty. A movement must be atomic and only one package may move at a time. (After an unfortunate accident involving a tuba and a crated bobcat, the workers agreed to stop trying to scoot packages past each other on the conveyor belts. Safety first!)

For this portion of the project, the students must create a simulator for the FedOops distribution center. Students must create a Pile of Pending Packages (PPP), which can be represented as a queue of packages, each of which has a random set of instructions associated with it. When a worker grabs a package, the worker must read and follow the associated instructions in the specified order (again, the ordered list will be one to four actions with no action being repeated). Students must synchronize the workers (each of which is represented as a thread), the stations, and packages so that the packages are processed as quickly as possible while following all of the above rules. After all, the fastest teams and groups win bonuses!

The solution must run as efficiently as possible by maximizing parallelism while avoiding deadlock or starvation. Students can model a package being in a station by using the `sleep()` call. Students must name each worker thread (e.g., "Blue #4") working in the group. Whenever a worker grabs a package, they should shout (i.e., print out) their name, the package identifier (which can be a simple counter like "package #7"), and each of the steps in the action list that the customer requires for that package. Then, the worker should place the item in the first station once it becomes available. Each time the worker places an item in a station (or has it conveyed to a new station), they should say their name, the package identifier, and the name of the station in use. Once the action is complete, the worker should say which package is done, put it on the delivery truck, and tag in the next member of the same team. That team member should then collect the next package from the PPP queue. Workers must select packages from the front of the queue; they cannot strategically grab or reorder packages based on what else is happening in the distribution center. After loading a package onto the delivery truck, a worker returns to the distribution center and enters the queue associated with their team color, allowing them to return and process packages when tagged in by a teammate. At most four workers (one from each team) may be operating at the stations at a time. Finally, all teams should be guaranteed to be able to make progress (e.g., fairness should be ensured so teams cannot starve out other teams).

When students have completed their solution, they should explain how the solution meets the requirements described above in a text file, `problem2_explanation.txt`, accompanying their code. They must specifically describe any scenarios that are guaranteed to deadlock and any rules that can ensure a deadlock is avoided. They must discuss any decreased parallelism that may result from any deadlock avoidance mechanisms they introduce and argue how they minimized that decrease.

Checkpoint Contributions

Students must submit work that demonstrates substantial progress towards completing the project on the checkpoint date. Substantial progress is judged at the discretion of the grader to allow students flexibility in prioritizing their efforts. However, as an example, any assignment in which one of the two synchronization problems is addressed will receive full credit towards the checkpoint. **Projects that fail to submit a checkpoint demonstrating significant progress will incur a 10% penalty during final project grading.**

Deliverables and Grading

When submitting the project, please include the following:

- All of the files containing the code for all parts of the assignment.
- The `problem1_explanation.txt` explanation for the Summer Spectacular problem.
- The `problem2_explanation.txt` explanation for the FedOops Shipping problem.
- The test files or input (e.g., specific random seed values) that the team used to convince themselves (and others) that the programs actually work.
- Output from the tests showing the system's functionality.
- A document called README.txt explaining the project, any defects, and anything that the team feels the grader should know when grading the project. Only plaintext write-ups are accepted.

Please compress all the files together as a single .zip archive for submission. As with all projects, please only submit standard zip files for compression; **.rar, .7z, and other custom file formats will not be accepted**.

The project programming is only a portion of the project. Students should use the following checklist in turning in their projects to avoid forgetting any deliverables:

1. Sign up for a project partner or have one assigned (URL: `https://cerebro.cs.wpi.edu/cs3013-shue/request_teammate.php`),
2. Submit the project code and documentation via InstructAssist (URL: `https://cerebro.cs.wpi.edu/cs3013-shue/files.php`), and
3. Complete a Partner Evaluation (URL: `https://cerebro.cs.wpi.edu/cs3013-shue/evals.php`).

A grading rubric has been provided at the end of this specification to give a guide for how the project will be graded. No points can be earned for a task that has a prerequisite unless that prerequisite is working well enough to support the dependent task. Students will receive a scanned markup of this rubric as part of their project grading feedback. Students are expected to contribute equally on their teams; unequal contributions may result in different scores for the students on a team.

Project 2 – Synchronization – Grading Sheet/Rubric

Evaluation?

Grader: _____ Student Name: _____ _____

Date/Time: _____ Student Name: _____ _____

Team ID: _____ Student Name: _____ _____

Late?: _____

Checkpoint?: _____ Project Score: / 90

Earned	Weight	Task ID	Description
_____	0	0	Primitives – Students must use a different primitive for Part 1 and Part 2. If the same primitive is used for both, students may choose which part will be graded. The other part is to be assigned a score of 0.
_____	15	1	Part 1 – Threads/processes implemented correctly with a high degree of parallelism. Prerequisite: Task 0.
_____	15	2	Part 1 – Correct mutual exclusion and prevention of deadlocks. Prerequisite: Task 1.
_____	15	3	Part 1 – Solution ensures fairness/starvation prevention and good explanation in problem1_explanation.txt. Prerequisite: Task 1.
_____	15	4	Part 2 – Threads/processes implemented correctly with a high degree of parallelism. Prerequisite: Task 0.
_____	15	5	Part 2 – Correct mutual exclusion and prevention of deadlocks. Prerequisite: Task 4.
_____	15	6	Part 2 – Solution ensures fairness/starvation prevention and good explanation in problem2_explanation.txt. Prerequisite: Task 5.

Grader Notes: