Name:

Stage	1	2	3	4	Sum
Points	4	9	7	4	24
Result					

L4: Pax Romana

In the year DCCXXVII ab urbe condita Imperator Gaius Julius Caesar Augustus closed the Gates of Janus and declared the new era of Pax Romana. As a result, the Roman Empire saw a significant expansion of trade. The old Roman port in Ostia turned out to be too small, so the current Imperator, Tiberius Claudius Caesar Augustus Germanicus, decided to expand it. To ensure that a new port would have sufficient throughput, he asked you to write a simulator.

Your task is to simulate a docking place for a single ship. Various goods are unloaded from the ship by the port workers and loaded into carriages. There are $1 \leq P \leq 10$ workers and $1 \leq Q \leq 10$ carriages.

Do not use global or static variables, busy waiting, or thread cancellation at any stage. Refer to the helper functions implemented in the file `common.h`.

Stages:

1. $\boxed{\text{4 p.}}$ Implement port worker thread. Each port worker enters the ship and takes one package (sleeps 5ms). Only one worker at a time can enter the ship. Then the worker transports the package to the warehouse (sleeps 10ms) and prints the message `Worker <worker idx>: I transported <pack idx> package`. After transporting 10 packages, the worker thread ends. The main thread waits for all workers to finish.

2. $\boxed{\text{9 p.}}$ Implement ship arrivals. When the program receives `SIGUSR1`, a new ship arrives. If that is impossible (there is a ship docking already), print the message `Ship cannot dock now` to the terminal instead. Each ship holds a random number of 30 to 60 packages of trade goods and docks for as long as workers unload it (implement that logic in worker - wait 5ms and decrement package count). If there is no ship to unload, workers wait for its arrival. Use a conditional variable to implement signalling ship arrivals and workers waiting for them. Workers work until the program receives `SIGINT`. Use a separate thread for handling signals.

3. $\boxed{\text{7 p.}}$ Implement carriage driver thread. Workers carry packages to the warehouse. As workers carry packages into the warehouse, keep the warehouse package count as a semaphore. Drivers arrive at the warehouse and try to load packages into their carriage (wait on semaphore). After loading 10 packages or waiting more than 500ms without getting a package, the driver prints to the terminal: `Driver <idx>: I am departing to Rome with <number of packages> packages` and travels to Rome and back in 50ms.

   Hint: for `sem_timedwait` there is a helper function to get the correct time.

4. $\boxed{\text{4 p.}}$ To improve the security of transport, it was decided that carriages would travel together. Once a carriage is loaded, its driver waits for all other drivers, and they depart together. To simulate that behaviour, use a barrier.

   Hint: Carefully consider how to keep safe program termination with `SIGINT`. Ensure no driver thread is left hanging on the barrier during program termination. However, it is acceptable to delay program termination briefly (a few hundred milliseconds).