CS39002 Operating Systems Laboratory
Spring 2026

Lab Assignment: 6
Date posted: 06-Mar-2026

Design of a synchronization primitive

Think of a situation when a customer wants a service at a counter. The service room is small, and can accommodate only a few customers (perhaps only one). However, there is a waiting room outside, which is sufficiently large for accommodating many customers. If a customer seeking the service sees the service room full or busy, (s)he comes to the waiting room, and waits there. When the service room can accommodate new customers, one or more waiting customers are taken from the waiting room and admitted to the service room. There are two types of waits involved here: a wait in the service room for getting the service, and a wait in the waiting room for accessing the service room. The second wait is called a *conditional wait* (because it depends on the condition of the service room). Conditional waits are frequently used in synchronization applications. In this assignment, you implement conditional waits using semaphores, and write a sample application to utilize your constructs.

Implementation of condition variables

A condition variable *CV* consists of two semaphores and a shared integer count. The first semaphore mtx is used for the mutual exclusion of count (and other external items), and the second semaphore cnd is the waiting place of all processes/threads when the condition is not right. Finally, count stores how many processes/threads are waiting conditionally. Both mtx and cnd should be operated in atomic fashions, so the use of semaphores to implement them is justified. Define a cond_t structure consisting of two int variables storing the semid of the set of two semaphores and the shmid of the shared-memory segment storing count. Implement the following operations on cond_t variables.

`CV = cond_create(token1, token2)`

Two tokens are needed for the associated `semget` and `shmget` calls. These may be IPC_PRIVATE or tokens generated by `ftok()`. Upon the creation of a semaphore set of size two and a shared-memory segment capable of storing one `int` variable, the two IDs are returned to *CV*. All future operations on this condition variable will pass *CV* as a parameter.

`cond_init(CV)`

This is used (after creation) for the initialization of the three items of *CV*. The semaphore mtx should be initially unlocked (otherwise no process can enter the critical section involving count and other application-specific items). The other semaphore cnd is used as a waiting place, that is, so long as some condition (external to *CV*) does not hold, the relevant processes must wait. Therefore cnd should be initialized to 0. Finally, count should be initialized to 0. Using an uninitialized condition variable may lead to unexpected results.

`cond_lock(CV)`

Lock the semaphore mtx of *CV*. This is the usual `P()` operation on this semaphore. The wait, signal, and broadcast functions (explained below) should be called with this semaphore locked beforehand. This ensures mutually exclusive accesses to the shared count (and other application-specific shared items).

`cond_unlock(CV)`

Unlock the semaphore mtx of *CV*. This is the usual `V()` operation on this semaphore. When a process is not accessing shared data (including count), mtx should be left unlocked.

`cond_wait(CV)`

This function implements the wait on the condition semaphore cnd of *CV*. The semaphore mtx should be locked before calling this function. The function first increments count by 1 (so this is a critical section). Then mtx is released (the critical section is over, so other processes are allowed to enter their critical sections one by one). This is followed by a `P()` operation on cnd. Because cnd has the value 0, this call blocks until some other process signals this semaphore. When the blocking wait is over, the function again locks mtx, decrements count by 1 (this is again a critical section), and leaves. It is the duty of the user program to release mtx after this function returns.

The release of mtx and the beginning of the wait on cnd should be done together as a single atomic operation. A kernel-level implementation can achieve that (for example, using a spinlock). Your implementation of condition variables is in the user level. Apparently, the atomicity of the combined release-and-wait operation cannot be

guaranteed at the user level. (Simulate this by a one-second sleep between the two operations. Suppose that the API designers specify this limitation for user programs.) Doing `nsops` = 2 semaphore operations using `semop()` is allowed but does not have the desired effect. Multiple semaphore operations by a single `semop()` call is carried out *as a complete unit, or not at all*, so both the release (+1) of `mtx` and the wait (–1) on `cnd` are performed together when neither involves a wait. But then, `mtx` remains locked until a signal comes to `cnd`.

`cond_signal(CV)`
: This function again should be called with `mtx` of *CV* locked. The function reads `count`, and if `count` > 0, it sends a signal to the semaphore `cnd` thereby waking up one process/thread waiting conditionally. After a waiting process wakes up, it decrements `count`, so signaling must not update `count` a second time. Finally, if the function finds `count` = 0 at the beginning (nobody is waiting), it does nothing. Notice that the signal to `cnd` (in the case `count` > 0) is sent with `mtx` locked, so the process that is woken up cannot immediately lock it. It is the responsibility of the user to unlock `mtx` after the call of `cond_signal()` returns (so that the woken up process can proceed).

`cond_broadcast(CV)`
: This is similar to conditional signaling with the exception that all processes waiting on `cnd` are woken up. The function assumes that `mtx` is locked beforehand. It reads `count`. If it is 0 (nobody is waiting on `cnd`), the function returns. Otherwise, it increments `cnd` by `count` (a single `V()` operation). You may also try to increment `count` by 1 in a loop (with `count` iterations). Like conditional signaling, the woken up processes cannot proceed immediately, because `mtx` is locked by the function sending the broadcast signal. When `mtx` is unlocked (after return from the function), a single woken up process locks it. Upon return from `cond_wait()`, `mtx` is unlocked by the first woken up process, so a second woken up process can lock it, and so on. In other words, the conditional waits of `count` processes finish sequentially one after another. A single broadcast call initiates this process.

`cond_destroy(CV)`
: Release the IPC resources (the shared-memory segment storing `count`, and the semaphore set consisting of `mtx` and `cnd`) to the system.

Write the above functions in a standalone `main()`-less file `cond.c`. Any application program that wants to use this implementation of condition variables can include it (or link a library generated from this). Do **not** use any application-specific code in the functions of `cond.c`. Moreover, in the application program, use only the above API calls; make **no** attempts to access the internal structure of *CV* in any manner.

A sample application

Several demons and several nomads (these are processes, not humans or ghosts) use a deserted house in the jungle, as a temporary shelter. However, they cannot tolerate the smell of each other. That is, when demon(s) are inside the house, any nomad willing to enter the house must wait. Only when the last demon in the house leaves, the waiting nomads can enter. Similarly, if nomad(s) are in the house, all demons willing to enter the house must wait until the last nomad leaves. There is however no restrictions on how many nomads or demons can enter, leave, or stay in the house so long as the house is occupied by entities of the same kind. When the house is empty, either a demon or a nomad can enter. If a demon enters, other demons can enter (but no nomad can enter). Likewise for nomads.

When the last demon leaves, it marks the house as empty, and allows a single nomad to enter (if any is waiting). The first nomad upon entering allows all other waiting nomads to enter. Nomads are not so mean as demons. When the last nomad leaves, it too marks the house as empty, but allows all the waiting demons (if any) to enter. The pseudocodes for a demon and for a nomad are given below.

```
Demon ()
{
    while (1) {
        So long as the house is occupied by nomads, wait (while statement here, not if).
        Enter the house.
        If this is the first demon to enter, mark the house as occupied by demons.
        Use the house (delay for a random time in the range 0.5 – 1 second).
        Leave the house.
        If this is the last demon to leave, mark the house as empty,
            and if there is any waiting nomad, wake up a single waiting nomad.
        Wait for some time (random delay between 1 and 5 seconds) before coming back to the house.
    }
}
```

```
nomad ( )
{
    while (1) {
        So long as the house is occupied by demons, wait (while statement here, not if).
        Enter the house.
        If this is the first nomad to enter, mark the house as occupied by nomads,
            and if there are other waiting nomads, admit them all.
        Use the house (delay for a random time in the range 0.5 – 1 second).
        Leave the house.
        If this is the last nomad to leave, mark the house as empty,
            and if there are waiting demons, wake up all waiting demons.
        Wait for some time (random delay between 1 and 5 seconds) before coming back to the house.
    }
}
```

This is akin to the reader-writer problem except that like reads, writes can also run together. Only a reader and a writer cannot be in their critical sections at the same time. Implement this protocol using your implementation of condition variables. Use a single condition variable *CV*, on which both demons and nomads will wait.

You need a shared-memory segment *H* for this application in order to store the following information: the state of the house (this is one of EMPTY, DEMON_INSIDE, and NOMAD_INSIDE), the counts of demons or nomads inside the house (DEMON_COUNT and NOMAD_COUNT, both cannot be positive at the same time), and the counts of waiting demons or nomads (DEMON_WAIT_COUNT and NOMAD_WAIT_COUNT, both cannot be positive at the same time). Use the lock and unlock calls of *CV* for the mutual exclusion of *H*. There is no need to use additional semaphores. Note that when some entities are waiting, DEMON_WAIT_COUNT or NOMAD_WAIT_COUNT can be obtained from (indeed is equal to) the value of `count` in *CV*. But no application program may look into the internal details of API implementations, so these wait counts must be maintained externally.

The wait of a demon is conditioned on that the state of the house is NOMAD_INSIDE. Likewise, the wait of a nomad is conditioned on that the state of the house is DEMON_INSIDE. These two situations cannot happen together, so demons and nomads can both wait on the same condition variable *CV*. Waking up a single waiting nomad (by a demon) is sending a single conditional signal to *CV*, whereas waking up all waiting nomads or all waiting demons is sending a conditional broadcast to *CV*. Since the release-and-wait operation in `cond_wait` is not atomic, write a `while` loop (**not** an `if` statement which checks only once whether the house is occupied by others).

The delays in the pseudocodes of demons and nomads are to be implemented by `usleep()` with random parameters. Make sure that your program runs neither too fast nor too slow. Moreover, during a run of a few seconds, the arrivals of demons and nomads should be interleaved (so conditional waits, signals, and broadcasts are amply tested).

Write three source files.

`manager.c(pp)` The manager creates and initializes *CV* and *H*, and waits for the user to terminate it (like by hitting return). The user should keep it running throughout the time when demons and nomads are still running. Before exiting, the manager should remove the IPC resources *CV* and *H* from the system.

`demon.c(pp)` A parent process forks *n* demon processes (the default value of *n* is 10). The parent waits for all these *n* child processes to terminate. Each demon (child) runs the function demon() as stated above.

`nomad.c(pp)` A parent process forks *n* nomad processes (the default value of *n* is 10). The parent waits for all these *n* child processes to terminate. Each nomad (child) runs the function nomad() as stated above.

The demons and nomads run infinite loops. In order to terminate them, the user should interrupt them (Control-C). So each demon, each nomad, and their parents should catch signal handlers for SIGINT. This is to ensure that they detach from the shared-memory segment *H* before terminating.

Run manager in a terminal. Run demon and nomad in two separate terminals (or in a single terminal with at least one in the background). After terminating demons and nomads, terminate manager.

Submit a single zip/tar/tgz archive with four source files (cond.c and the three c/cpp files of the application), a makefile, and any other source file that you have used.

Sample Output

Compile the application sources (which `#include "cond.c"`).

```
$ make compile
gcc -Wall -o manager manager.c
gcc -Wall -o demon demon.c
gcc -Wall -o nomad nomad.c
$
```

On a terminal, we run `./manager`. On a second terminal, we run both `./demon` and `./nomad`. The output of the second terminal is given below.

```
$ make demorun
./demon 4 &
./nomad 5
Demon 0 arrives (D_CNT = 0, N_CNT = 0, state = EMPTY)
Demon 0 enters [house empty] (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
                                        Nomad 0 arrives (N_CNT = 0, D_CNT = 1, state = D_INSIDE)
                                                                          Nomad 0 waits (NW_CNT = 1)
Demon 1 arrives (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 1 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
                                        Nomad 1 arrives (N_CNT = 0, D_CNT = 2, state = D_INSIDE)
                                                                          Nomad 1 waits (NW_CNT = 2)
Demon 2 arrives (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 2 enters [other demons present] (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 3 arrives (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 3 enters [other demons present] (D_CNT = 4, N_CNT = 0, state = D_INSIDE)
                                        Nomad 2 arrives (N_CNT = 0, D_CNT = 4, state = D_INSIDE)
                                                                          Nomad 2 waits (NW_CNT = 3)
                                        Nomad 3 arrives (N_CNT = 0, D_CNT = 4, state = D_INSIDE)
                                                                          Nomad 3 waits (NW_CNT = 4)
                                        Nomad 4 arrives (N_CNT = 0, D_CNT = 4, state = D_INSIDE)
                                                                          Nomad 4 waits (NW_CNT = 5)
Demon 3 leaves (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 0 leaves (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 1 leaves (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 2 leaves (D_CNT = 0, N_CNT = 0, state = EMPTY)
                              Nomad 0 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                      Nomad 2 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                      Nomad 3 enters [other nomads present] (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                      Nomad 1 enters [other nomads present] (N_CNT = 4, D_CNT = 0, state = N_INSIDE)
                      Nomad 4 enters [other nomads present] (N_CNT = 5, D_CNT = 0, state = N_INSIDE)
                                       Nomad 3 leaves (N_CNT = 4, D_CNT = 0, state = N_INSIDE)
                                       Nomad 4 leaves (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                                       Nomad 2 leaves (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                       Nomad 0 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                          Nomad 1 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
Demon 0 arrives (D_CNT = 0, N_CNT = 0, state = EMPTY)
Demon 0 enters [house empty] (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 1 arrives (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 1 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 2 arrives (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 2 enters [other demons present] (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 0 leaves (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
                                        Nomad 3 arrives (N_CNT = 0, D_CNT = 2, state = D_INSIDE)
                                                                          Nomad 3 waits (NW_CNT = 1)
Demon 1 leaves (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 2 leaves (D_CNT = 0, N_CNT = 0, state = EMPTY)
                                 Nomad 3 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                                   Nomad 3 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
                                                  Nomad 1 arrives (N_CNT = 0, D_CNT = 0, state = EMPTY)
                                 Nomad 1 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                                   Nomad 0 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                      Nomad 0 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
Demon 3 arrives (D_CNT = 0, N_CNT = 2, state = N_INSIDE)
Demon 3 waits (DW_CNT = 1)
                                        Nomad 1 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                        Nomad 4 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                      Nomad 4 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                        Nomad 0 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                        Nomad 2 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                      Nomad 2 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                        Nomad 4 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                           Nomad 2 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
Demon 3 enters [house empty] (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
                                        Nomad 3 arrives (N_CNT = 0, D_CNT = 1, state = D_INSIDE)
                                                                          Nomad 3 waits (NW_CNT = 1)
Demon 0 arrives (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 0 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 3 leaves (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 2 arrives (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 2 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
                                        Nomad 1 arrives (N_CNT = 0, D_CNT = 2, state = D_INSIDE)
                                                                          Nomad 1 waits (NW_CNT = 2)
Demon 0 leaves (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
                                        Nomad 0 arrives (N_CNT = 0, D_CNT = 1, state = D_INSIDE)
                                                                          Nomad 0 waits (NW_CNT = 3)
Demon 2 leaves (D_CNT = 0, N_CNT = 0, state = EMPTY)
                                 Nomad 3 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
```

```
                    Nomad 1 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                    Nomad 0 enters [other nomads present] (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                                          Nomad 2 arrives (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                    Nomad 2 enters [other nomads present] (N_CNT = 4, D_CNT = 0, state = N_INSIDE)
Demon 1 arrives (D_CNT = 0, N_CNT = 4, state = N_INSIDE)
Demon 1 waits (DW_CNT = 1)
                                          Nomad 4 arrives (N_CNT = 4, D_CNT = 0, state = N_INSIDE)
                    Nomad 4 enters [other nomads present] (N_CNT = 5, D_CNT = 0, state = N_INSIDE)
                                           Nomad 0 leaves (N_CNT = 4, D_CNT = 0, state = N_INSIDE)
                                           Nomad 1 leaves (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                                           Nomad 3 leaves (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 2 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
Demon 2 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 2 waits (DW_CNT = 2)
                                             Nomad 4 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
Demon 1 enters [house empty] (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 2 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 0 arrives (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 0 enters [other demons present] (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 3 arrives (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 3 enters [other demons present] (D_CNT = 4, N_CNT = 0, state = D_INSIDE)
Demon 1 leaves (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 2 leaves (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
                                          Nomad 2 arrives (N_CNT = 0, D_CNT = 2, state = D_INSIDE)
                                                                Nomad 2 waits (NW_CNT = 1)
Demon 3 leaves (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 0 leaves (D_CNT = 0, N_CNT = 0, state = EMPTY)
                              Nomad 2 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                          Nomad 1 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                    Nomad 1 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                          Nomad 3 arrives (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                    Nomad 3 enters [other nomads present] (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                                           Nomad 1 leaves (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 2 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                              Nomad 3 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
                                            Nomad 0 arrives (N_CNT = 0, D_CNT = 0, state = EMPTY)
                              Nomad 0 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                          Nomad 4 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                    Nomad 4 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 0 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
Demon 1 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 1 waits (DW_CNT = 1)
Demon 0 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 0 waits (DW_CNT = 2)
Demon 2 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 2 waits (DW_CNT = 3)
                                          Nomad 1 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                    Nomad 1 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                          Nomad 2 arrives (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                    Nomad 2 enters [other nomads present] (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                                           Nomad 4 leaves (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 1 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
Demon 3 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 3 waits (DW_CNT = 4)
                                             Nomad 2 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
Demon 1 enters [house empty] (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 0 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 2 enters [other demons present] (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 3 enters [other demons present] (D_CNT = 4, N_CNT = 0, state = D_INSIDE)
Demon 3 leaves (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 1 leaves (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
                                          Nomad 3 arrives (N_CNT = 0, D_CNT = 2, state = D_INSIDE)
                                                                Nomad 3 waits (NW_CNT = 1)
Demon 2 leaves (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 0 leaves (D_CNT = 0, N_CNT = 0, state = EMPTY)
                              Nomad 3 enters [house empty] (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                          Nomad 1 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                    Nomad 1 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 3 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                          Nomad 0 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                    Nomad 0 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 1 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
Demon 2 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 2 waits (DW_CNT = 1)
Demon 3 arrives (D_CNT = 0, N_CNT = 1, state = N_INSIDE)
Demon 3 waits (DW_CNT = 2)
                                          Nomad 4 arrives (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                    Nomad 4 enters [other nomads present] (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                          Nomad 2 arrives (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                    Nomad 2 enters [other nomads present] (N_CNT = 3, D_CNT = 0, state = N_INSIDE)
                                           Nomad 0 leaves (N_CNT = 2, D_CNT = 0, state = N_INSIDE)
                                           Nomad 4 leaves (N_CNT = 1, D_CNT = 0, state = N_INSIDE)
                                             Nomad 2 leaves (N_CNT = 0, D_CNT = 0, state = EMPTY)
Demon 2 enters [house empty] (D_CNT = 1, N_CNT = 0, state = D_INSIDE)
Demon 3 enters [other demons present] (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 1 arrives (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
Demon 1 enters [other demons present] (D_CNT = 3, N_CNT = 0, state = D_INSIDE)
Demon 2 leaves (D_CNT = 2, N_CNT = 0, state = D_INSIDE)
                                          Nomad 3 arrives (N_CNT = 0, D_CNT = 2, state = D_INSIDE)
                                                                Nomad 3 waits (NW_CNT = 1)
^C
```

After this, we terminate the manager.