Lab 3

(1) Run JNachos with Water uncommented. It prompts you for the number of hAtoms and number of OAtoms. Run the program with several different console Inputs.

Answer:
Input: 2 Hydrogen atoms and 1 Oxygen atom
With this input, we can see that the H atoms and O atoms have been given a number associated with it to differentiate which H atom was combined which other H atom and O atom to form a molecule of Water. For example, in this input, #0 atom of Oxygen was combined with #1 atom of H and #0 of H to make water.
The total number of clock ticks and the idle state varies with each input.
For an input of 8 Hydrogen atoms and 4 Oxygen atoms, the total ticks are 910 and idle value is 20, Whereas for an input of 6 Hydrogen atoms and 6 Oxygen atoms, the total number of clock ticks was 810 and the idle value was 70. Also, an input of 4 Hydrogen atoms and 2 Oxygen atoms has 510 ticks and 60 idle time.
The total number of ticks and idle time remains the same if the same input is given again.

(2) Run JNachos with the program argument "-rs", where seed is a number. Try with different numbers, is there any difference?

Answer:
When we run the program with –rs flag, the total number of ticks and idle time changes even if the input is kept the same for a different seed value.
For, 4 Hydrogen atoms and 2 Oxygen atoms the output was 510 ticks and 60 idle time without the –rs flag. With, seed = 50, the total number of ticks and idle value changes to 555 and 85. For the same input, when the seed value is changed to 70, total ticks becomes 454 and idle becomes 14.
The –rs flag generates random numbers so as to call yield randomly but the pattern is repeated i.e. it occurs at repeatable spots.

(3) Trace through the logic of what the -rs flag does. Explain this logic here.

Answer:
The –rs flag generates random numbers so as to call yield randomly but the pattern is repeated i.e. it occurs at repeatable spots.
The –rs argument takes in as an argument a number which will set the time of or delay the next interrupt call by that number. This in turn affects the idle time needed by the machine. The interrupts would be called after those number of ticks as set by the seed value.
Main.java calls the function initialize (String args) from the JNachos.java file which contains the several different argument types. In this, argument type is checked for the -rs flag, which sets the randomYield (initialized as false in the beginning) value to be true and saves the seed value in the argument in a variable called seed. The seed value is any number that we give the program as an argument. After this, in Interrupt.initialise, mYieldOnReturn is set to false and if it is false then we call JNachos.getCurrentProcess().yield(). The interrupt is used to interrupt the CPU at regular intervals i.e., once every TimerTicks and this routine is called every time there is a timer interrupt,

when interrupts are disable. Calling yield, suspends the interrupt handler and not the process that has been interrupted.
These interrupts are generated according to the seed value given in the argument. If the seed value is 50, then the next interrupt will be generated after 50 clock ticks.

(4) There are 5 Semaphore instances, explain the purpose of each instance. For "true" Semaphores explain what the associated count represents. For a mutex explain what critical section each is protecting.

Answer:

- **Semaphore H**: After the creation of the first Hydrogen atom, it waits for the second one which is done by H.P() call. A count is kept to check if the second Hydrogen atoms is formed or not after the creation of first. It keeps track of the H atoms used to make a water molecule.
- **Semaphore O**: This waits for the creation of two Hydrogen atom so as to combine to form a water molecule. It will call O.P() to wait for their creation and once both the Hydrogen atoms are created, the second one will call O.V() to wake this process up.
- **Semaphore wait**: It waits for the message to display the information about how many Hydrogen atoms have been used to create a water molecule.
- **Semaphore mutex:** This is used to protect the critical section which updates the count variable responsible to keep track of the number of Hydrogen atoms. The first H atom waits for the creation of the second H atom in order to combine with each other for each water molecule. While the count is being updated, the mutex protects others from changing the Hydrogen count.
- **Semaphore mutex1:** This also protects the critical section which updates the Hydrogen and Oxygen atoms. The Hydrogen count is decreased by 2 whereas the Oxygen count is decreased by 1. The O atom waits for the creation of two H atoms to combine together to form a water molecule. The mutex prevents other operations from changing the H and O counts while they're being updated.

(5) Explain the logic of HAtom and OAtom.

Answer:

HAtom: Each Hydrogen atom waits for the second Hydrogen to be created so they can be grouped together with an oxygen atom.
With the number of inputs given, water calls Hatom with the associated count of the H atoms and is passed in the HAtom class. If the program is given a console input of 4, then water, loops the count and each iteration calls HAtom.
The HAtom has an id associated with it which keeps track which HAtom is used to create a water molecule. The function call(), calls the P() function which waits for the semaphore to be >0. If the value is <0, the last process is added in the queue and puts that process to sleep. If the count isn't divided by 2, i.e. the count is an odd number following steps would be performed. The second H atom has yet to be created and grouped with the first then, you increment the counter for the first H atom and end the critical section by calling V(). If the queue is not empty, this function wakes up the thread that is waiting in the queue and makes it ready to run. The first H atom thus waits

for the second H atom to be created. The critical section is protected when the H count is being updated so no other operations can update that count.

If the count however is an even number, then we increment the count of the first H atom and call V() to end the critical section and H.V() wakes up the first H atom so as to combine both the Hydrogen atoms and wakes up the Oxygen atom to combine them to form water. We wait for the message to be displayed by calling wait.P().

OAtom: The Oxygen atom too has a count associated with it similar to the H atom which is taken as a console input and iterated and sent to the constructor of OAtom. The call() function calls O.P() which waits for the creation of two H atoms and then calls makeWater(). makeWater() function, prints on the console a message saying that a water molecule has been formed. Next, wait.V() is called to wake up the H atoms and will be returned. Mutex1.P(), protects the critical section when it updates the number of H and O atoms. Here, the Hcount is decreased by 2 and the Ocount is decreased by1. The mutex is used so that these counts can be updated and no other operation can change these counts while this update is being carried out. The messages about the updates are then displayed on the console giving us information about how many H atoms and O atoms are used up to form water molecules so far and how many are left after creating water. Mutex.V1() ends the critical section.