School of Computer Science

COMP30640

Lab 6
Synchronisation II

Teaching Assistant:	Thomas Laurent
Coordinator:	Anthony Ventresque
Date:	Friday 19th October, 2018
Total Number of Pages:	4

1 Atomic Operations in Bash.

Semaphores are based on the concept of atomic operations - i.e., operations that cannot be split in multiple instructions by the system and cannot be interrupted while they're executed. In particular (see lecture 7) we need an atomic operation that can perform a **test** (if) and **modify/write a value**. There are a few atomic operations in Bash that we could use, as they perform something (write/modify) and give an exit value that we can use in a conditional structure. One of them is `ln`, which creates a link between two files.

1. Create a file (or use one pre-existing file, for instance one of your previous scripts, like hello.sh)
2. Create a link using the following syntax:

   ```
   $> ln hello.sh link.sh
   ```

3. Open the new file you've just created (`link.sh` in my example). What do you notice?
4. run the command `ls -l`. What difference(s) do you see between the target file and the linked (source) file?
5. Run the same command again, with a different link name, for example:

   ```
   $> ln hello.sh link2.sh
   ```

6. What is the exit value?
7. Run the exact same command again. What is the exit value now?

Solution When you open the link, you will see the exact same contents as was in the file that you were linking to. If you edit the file either via hello.sh or by the link, you will find that the changes made can be seen if the file is opened from either hello.sh or link.sh. This is because link.sh is pointing to the same part of the file system (known as an *inode*) as hello.sh. This is formally called a *hard-link*. When you type ln -l, you will notice that there is a 2 for both hello.sh and link.sh in the second column. This means that there are two links to the particular file in question, hello.sh and link.sh.

This is because the link is really just a **shortcut** to the original file.

When you run the command to create the second link, run the following directly after (if you're unsure, review last week's lab)

```
echo $?
```

The exit code will be 0, meaning the link has been created successfully. If we run the command again we get the following error:

```
ln: link2.sh: File exists
```

The link already exists and therefore can't be created again, giving us an exit code of 1.

2 Semaphores.

Now that we have an atomic operation and that we know what its exit code is, we can use it to implement our semaphores. In particular, our `P` and `V` can be used with an **existing file** given as argument. `P` would then try to create a link with `ln` and if successful the critical section would be available or the process would have to wait.

Create two scripts:

`P.sh` containing the following code:

```
#! /bin/bash

if [ -z "$1" ]; then
    echo "Usage $0 mutex-name"
    exit 1
elif [ ! -e "$1" ]; then
    echo "Target for the lock must exist"
    exit 2
else
        while ! ln "$1" "$1-lock" 2>/dev/null; do
                sleep 1
        done
        exit 0
fi
```

And `V.sh` containing this code:

```
#! /bin/bash

if [ -z "$1" ]; then
        echo "Usage $1 mutex-name"
        exit 1
else
        rm "$1-lock"
        exit 0
fi
```

8. Run `./P.sh` with one parameter: an **existing file**. For instance create a new file `sema`, using the command `touch sema`, and execute the following command: `./P.sh sema`. What happens? (look at the files in your directory)

9. Run `./P.sh` with the same parameter. What happens? Why? (look at the files in your directory)

10. kill `./P.sh` with the sequence `control + c`. Run `./V.sh`. What happened? (look at the files in your directory)

Solution If we run P.sh with an existing file as the parameter the first time, if we look into our current directory, we will see a lock under the name 'existingFileName-lock'. If we run it a second time the terminal will hang. This is because our program is sleeping for 1 second and

then trying to create the link again after that. It keeps waiting and trying but the link is never removed so the program never finishes.

When we run V.sh, we can see that the *lock* link is removed. In this way, we have simulated the behaviour of P and V:

- P is an **atomic operation** (link) that **waits** (continuously tries to make link, then sleeps) for the semaphore to become **positive** (the link is deleted) and then **decrements by 1** (creates a link)
- V which **increments** the semaphore by 1 (deletes the link), **waking up** P if there is one.

Example of Semaphores in Use.

11. Revisit the last section (section 6, questions 1 to 3) of last week's practical. Do you see the problem? Run the following command to help you:

    ```
    $> ./run_write.sh; wc -l f*
    ```

12. Identify the critical section in `write.sh` and add calls to `./P.sh` and `./V.sh` (with the correct argument) to ensure a mutual exclusion.
    Hint: You will also need to change P.sh from the first example *very* slightly - look at the first $1 in the while loop.

13. Did you synchronise the scripts?

Solution Critical section: Code inside the body of the loop:

```
if [ ! -e "$elem" ] ; then
  echo 1st $$ > $elem
else
  echo next $$ >> $elem
fi
```

This is the place where there is a possibility of two processes trying to write to a file at the same time, therefore this is the section we should lock. We place P before the *if* statement, this will create a lock for the particular file that we are about to write to. We can then write to the file/create it if it doesn't exist. V then signals that this operation is finished, this particular lock is deleted and we can write to the file again. Any process that tries to write to the file while another process is already writing to it will have to wait until V removes the lock.

write.sh

```
#! /bin/bash

if [ $# -lt 1 ] ; then
    echo "This script requires at least one parameter"
    exit 1
fi
for elem in "$@" ; do
    # If we can't write to a file because it is in use, we have to wait for V.
    ./P.sh $elem
    # Critical section: We write to files here.
    if [ ! -e "$elem" ] ; then
        echo 1st $$ > $elem
    else
        echo next $$ >> $elem
    fi
    # When we have finished writing we can 'wake up' P again.
    ./V.sh $elem
done
```

P.sh

```
#! /bin/bash

if [ -z "$1" ]; then
    echo "Usage $0 mutex-name"
    exit 1
elif [ ! -e "mutex" ]; then
echo "Target for the lock nmust exist"
exit 2
% else
    while ! ln mutex "$1-lock" 2>/dev/null; do
        sleep 1
    done
    exit 0
fi

## ALTERNATIVE VERSION
if [ -z "$1" ]; then
    echo "Usage $0 mutex-name"
    exit 1
else
    # You can just use the file itself to link to!
    while ! ln $0 "$1-lock" 2>/dev/null; do
        sleep 1
    done
    exit 0
fi
```