CT213
Computing Systems & Organisation

University of Galway
2022/2023

Lab

Synchronisation (Semaphores)

Semester 1

1 Synchronisation Issues

1.1 Symbolic Links

A symbolic link (also known as soft link), is a special kind of file that points to another file. This is similar to a shortcut in Windows or a Macintosh alias. You can create a symbolic link to a file using the command `ln` (e.g., `ln -s file1.txt linkFile1.txt`).

Unlike a hard link, a symbolic link does not contain the data in the target file. It simply points to another entry somewhere in the file system.

1. Run the command `ln -s "$0" "$1"` in the terminal, with `"$0"` being any file (this file should exist) and `"$1"` being any random name (not the name of an existing file). What do you see?

2. Now, write the following script `test_ln.sh` and check what happens if `"$1"` is an existing file (Answer: the loop should never finish and you don't have the prompt any more). Don't stop the script now.

```
#! /bin/bash

# the 'link' (ln) system call is supposed to be atomic on a local file system
while ! ln -s "$0" "$1" 2> /dev/null; do
    sleep 1
done
```

Listing 1: test_ln.sh

4. Now open another terminal (or another tab in the same terminal) and remove the file (symbolic link) that was used as `"$1"` in question 2. What do you observe?

5. What does the commented line in the script says? Why does the atomicity of the `ln` command matter?

1.2 Symbolic Links as Locks

The atomicity guaranties that both commands: (i) checking whether a link with the specified name exists and (ii) the creation of a new link with that name cannot be interleaved (e.g., another program cannot create link with that same name in between). Therefore, we could use the existence/absence of a symbolic link with the given name as a locking mechanism as follows:

- If the link already exists, then the lock is taken by another process and we have to wait until the link no longer exists, then this process will create the link (i.e., acquire the lock). See script `acquire.sh`.
- The process holding the lock deletes the link it created when it wants to release the lock. See script `release.sh`.

```
#! /bin/bash

if [ -z "$1" ]; then
    echo "Usage $0 mutex-name" >&1
    exit 1
else
    # the 'link' (ln) system call is supposed to be atomic on a local file system
    while ! ln "$0" "$1" 2>/dev/null; do
        sleep 1
    done

    exit 0
fi
```

Listing 2: acquire.sh

```
#! /bin/bash

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

Listing 3: release.sh

1.3 Synchronisation Issue

Here is the code for the script `write.sh`:

```
#! /bin/bash

if [ $# -lt 1 ] ; then
  echo "This script requires at least one parameter"
  exit 1
fi
for elem in "$@" ; do
  if [ ! -e "$elem" ] ; then
    echo 1st $$ > $elem
  else
    echo next $$ >> $elem
  fi
done
```

Listing 4: write.sh

1. run the following command twice (or more): `./write.sh a b c` and explain the content of the files. Read the files using less (for instance. the Variable `$$` corresponds to the PID of the process running a script.

2. Now run the script `run_write.sh` which exhibits synchronisation problems (two processes accessing files concurrently). Check how many lines are there in the files (there should be two but because of synchronisation issues there may be only one in some of the files)? You can check the number of lines using `wc -l file_name`: `wc -l f1 f2 f3` in our case. Try several times if you see 2 lines in each files – until you see the problem. Why does the problem occur?

```
#! /bin/bash

rm -f f1 f2 f3

./write.sh f1 f2 f3 & ./write.sh f1 f2 f3
```

Listing 5: run_write.sh

4. Identify the *Critical Section* in `write.sh`

5. Use the two scripts `acquire.sh` and `release.sh` given above and modify `write.sh` to ensure mutual exclusion on the critical section.
6. Make sure that the solution you proposed does not block processes which do not access the same file.