Lab 6.1

To extract the content of a tgz file

tar vxfz xv6-sem.tgz

make clean

Compile xv6 with **make qemu** it will also run the **xv6** on **qemu**.

Try some commands (ex. **ls**).

make qemu-gdb

copy the last line of the screen, something like

qemu -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512 -S -gdb tcp::26000

on a script file **qemu.sh**

Then, run

qemu -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512

and using **cat** and redirection, create a file **myname.txt**

including **a single string: your name**.

check that a **.gdbinit** file exist that refer to the same tcp port (26000)

run **./qemu.sh** on a window

run **ddd&** on another window

Write a report that lists and comments the sequence of system calls

that are performed after issuing the command

cat myname.txt | wc

Lab 6.2

A condition variable **c** is associated with a specific lock **m**.

- Calling **cond_wait(c)** enqueues the current thread on c (suspending its execution) and unlocks **m**, as a single atomic action. When this thread resumes execution, it re-locks **m**.
- **cond_signal(c)** examines **c**, and if there is at least one thread enqueued on **c** then one such thread is dequeued and allowed to resume execution; this entire operation is a single atomic action.
- **cond_broadcast(c)** examines c and if there are any threads enqueued on c then all such threads are allowed to resume execution. Again, this entire operation is a single atomic action: the threads to be made runnable are exactly those that had called **cond_wait(c)** before this call of **cond_broadcast(c).** Of course, the runnable threads have to wait in line to acquire the lock **m**.

Since the xv6 is a kernel that does not define threads and conditions, you can use the semaphore implementation as a template for implementing the "conditions" in xv6. Since processes do not have common memory, we have to implement a particular kind of condition that merges in a single structure both the mutex, the condition, and an integer variable. A user process can allocate the condition, test in mutual exclusion the value of the variable, wait on the condition and signal the condition according to the specification given for the threads.

Write a **`cond_test.c`** main file, using as template file **`st.c,`** to test your condition implementation.

By analogy with the implementation of semaphores, you have to add statements in xv6 the following files:

- **`param.h`**
- **`user.h`**
- **`usys.S`**
- **`syscall.h`**
- **`syscall.c`**
- **`file.c`**
- **`sysfile.c`**
- **`main.c`**
- **`Makefile`**

to add the system calls:

- **`int cond_alloc()`** `//` this system call also initializes the condition
- **`void cond_set(int cond, int val)`**`//` to set to **`val`** the initial value of the **integer variable**
- **`int cond_get(int cond)`** `//` to get the value of the **integer variable**
- **`void cond_destroy(int cond)`**
- **`void cond_wait(int cond)`**
- **`void cond_signal(int cond)`**
- **`void cond_broadcast(int cond)`**

Create a compressed **`tgz`** archive file with this syntax:

Lab <LAB_number>_<Your_Id_number>_<YourLastname>. tgz

Example:

`Lab06_123456_Smith.tgz`

which includes the **`report`** for the first assignment and the **`project`** tgz file for the second assignment.

Drop your **`tgz file`** in the folder **Elaborati** of the SDP site of Portale della didattica.