# Project #4. Synchronization

In this homework, you will practice the implementation and use of synchronization locks. The homework environment is available as a Docker Image or a Virtualbox virtual machine image. The homework files are placed under the HOME directroy.

.

* Docker (Linux Kernel)

Docker download ( ubuntu )

|  |
| --- |
| $ sudo apt-get update  $ sudo apt-get install docker-compose  $ sudo docker run -it --cpus=2 pandaft/os\_project /bin/bash |

* VirtualBox Image

VirtualBox virtual machine image <https://bit.ly/2DQXdEd>

VirtualBox official site <https://www.virtualbox.org/>

VM\_User: os\_project Password: 0

\*\* If the virtual machine (VM) does not boot properly, you may need to go check your computer’s UEFI/BIOS settings and make sure that the Virtualization Technology / VT-X support is enabled.

After downloading, import the image file to VirtualBox, and launch it.

Note: The host machine (your computer) should have at least two CPU cores.

### Spin-lock Implementation

Spin lock is a busy waiting synchronization mechanism, which can be used to prevent threads from entering a critical section at same time. In this part, you will practice the implementation of spinlock with atomic test-and-set (or compare-and-swap) instructions.

In this part, we supply the following files.

[Makefile]: For compiling the project with `make` command and generating the ***spinlock*** executable file.

[main.c]: Create two threads and a shared counter, which is wrapped in the critical section marked by the call to **spin\_lock(&mutex)** and the call to **spin\_unlock(&mutex)**.

[check.sh]: Execute ***spinlock*** for 100 times, and auto-grade your code.

[spinlock.s]: Implement your spinlock here in **AT&T style** assembly !!

* Hint
  + Using AT&T style, MOV, CMPXCHG, JZ, JNZ instructions.
  + In AT&T style x64 assembly,  movl %ecx, %eax  ( assigned ecx value to eax )
  + You can use the following command to see more AT&T assembly coding style after compiling spinlock part.

  $ objdump -d ~/os\_project/spinlock

* + The value of &mutex will be place on “rdi” register.

Reference: [Assembly Reference](https://www.felixcloutier.com/x86/index.html) , [x64 register](https://wiki.cdot.senecacollege.ca/wiki/X86_64_Register_and_Instruction_Quick_Start), [linux x64 calling convention](https://www.systutorials.com/240986/x86-64-calling-convention-by-gcc/)

Note: You may use ‘check.sh’ to verify your spinlock implementation.

SingleTest:

|  |
| --- |
| $ ./spinlock 1000 |

### Producer-Consumer Problem

When we download a file from a server on the Internet, the client program has to carry out two I/O operations. One is the I/O for receiving the data from the network interface card, and the other is for saving the data from the memory to the hard disk. As both the I/O bandwidths and the I/O latencies can be quite different between the network interface card and the hard disk, it would be advantageous to decouple the I/O operations so that a delay in a disk write would not affect the receiving of packets from the network interface card, and vice versa. The decoupling can be achieved by running the network I/O and the disk I/O in two different child processes within the client program. However, this would require proper synchronization between the child processes. In this part, you are asked to use semaphore to synchronize the child processes, so that the data received from the network will be written properly to the disk (We will use *sha256 hash* to verify the integrity of the data). Figure 1 is the Server/Client architecture diagram. You need to start server.py first to wait for network connections from the client. Figure 2 shows the operation flow chart of the client program.

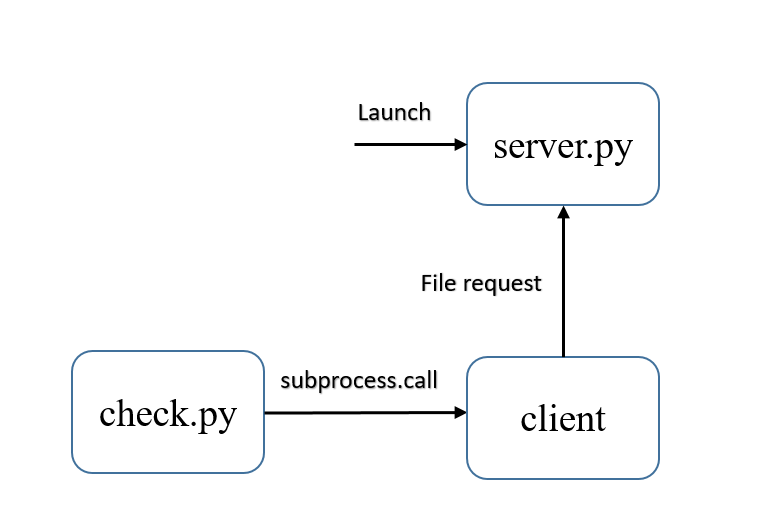


圖1. Server/Client Architecture

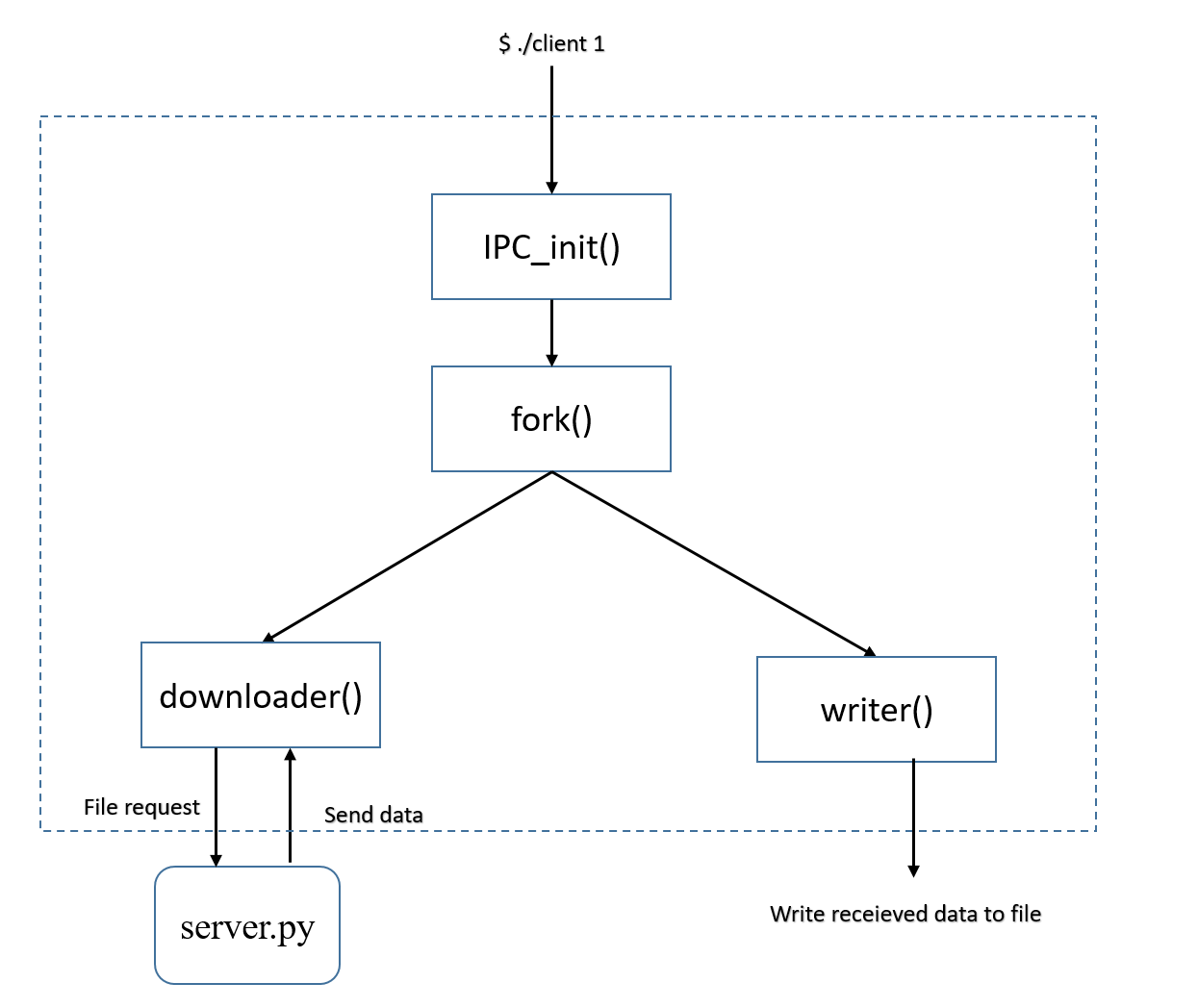


圖2. Client Operation Flow Graph

In this part, we supply the following files.

[client.c]: Create the **downloader** process for receiving data from the server and the **writer** process for saving the data to a file. You need to add the synchronization code (i.e., use of semaphores) to synchronize the two processes.

[Makefile]:Compile the project with `make` command to create the ***client***

executable file.

[server.py]: The file server.

[check.py]: Will Execute ***client*** for 10 times, and auto-grade your code.

* Hint
  + Using System V semaphore ( semget, semctl, semop, etc )

Note: Execute ***[server.py]***, ***[check.py]*** sequentially for verifing your work. (python2 script)

## Tasks

### Implement [spinlock.s].

1. Implement two functions, ***spin\_lock*** and ***spin\_unlock*** with atomic test-and-set (or compare-and-swap) and make sure that it passes the check of ***check.sh.* Submit [spinlock.s] only!!!** [30%]
2. Briefly explain how you implement the spinlock. Write down your answer in a pdf file. **Submit [spinlock.pdf]** [20%]

### Implement [client.c].

* 1. Fill up the ***“TODO”*** sectionand use **semaphore** to synchronize ***downloader*** and ***writer***. **Submit [client.c] only!!!** [50%]

Note: Put **[spinlock.s], [client.c], [spinlock.pdf]** in the folder named your student\_id and compress as zip file. Submit Student\_ID.zip, or you will get 5 point penalty.