

物聯網與微處理機系統設計

Internet of Things and

Microprocessor System Design

Lecture 12 - Advanced RPi

Lecturer: 陳彥安 Chen, Yan-Ann

YZU CSE

Outline

- Auto-start Program
- System Watchdog
- Threading

Outline

- Auto-start Program
- System Watchdog
- Threading

Autostart Program

- IoT or embedded system is a computing system for a special purpose.
- Usually, it should be ready to use after powering on.



- We have to **start our programs** after the system is booted up **automatically**.

Methods

- rc.local
 - The easiest and simplest way
 - Tasks started with rc.local happen before the X win starts.
- autostart
 - Automatically run your programs once LXDE (graphical desktop environment used by Raspbian) starts.
 - Graphical support
- systemd
 - The new and popular way to automatically start programs in Linux.
 - Wait until you have access to other processes (e.g., networking, graphical desktop).
 - Restart your program over and over again until it works.

Preparing Test Program

\$ cd ~

\$ nano helloWeb.py

```
from flask import Flask

app = Flask(__name__)

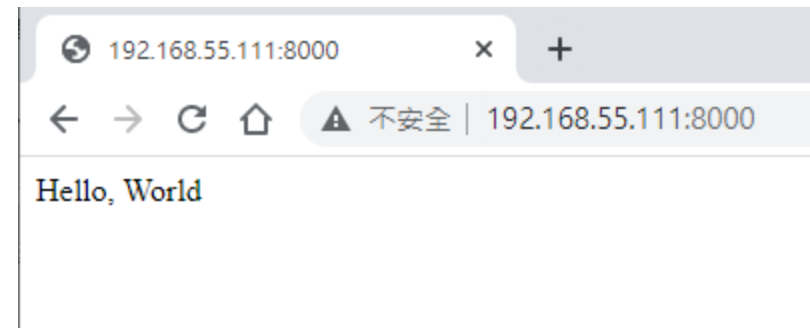
@app.route('/')
def hello():
    return 'Hello, World'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

\$ python3 helloWeb.py

- Open browser on your desktop and view <http://140.138.145.158:8xxx>
 - For “192.168.88.240”, browse <http://140.138.145.158:8240>

```
pi@rpi4-A00:~/iot/lec13 $ python3 helloWeb.py
* Serving Flask app "helloWeb" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
```



Method 1 - rc.local

```
$ cp helloWeb.py ~/
```

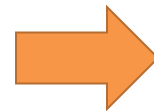
```
$ sudo nano /etc/rc.local
```

- Add the following line just before “exit 0” and save this modification.

```
python3 /home/pi/helloWeb.py &
```

- Reboot your RPi

```
$ sudo reboot
```



```
GNU nano 3.2 /etc/rc.local

#
# By default this script does nothing.

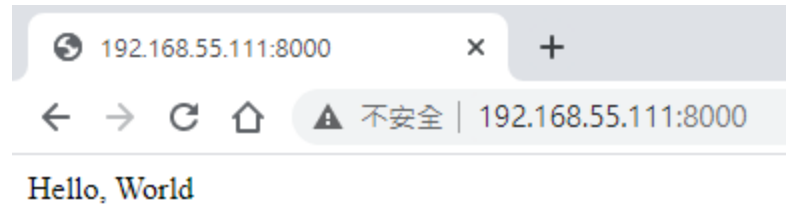
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

python3 /home/pi/helloWeb.py &

exit 0
```

Verification (1/2)

- After RPi is rebooted, you don't have to reconnect the ssh or VNC.
- Check the website by <http://140.138.145.158:8xxx>



- You will see that the web program starts automatically after rebooting.

Verification (2/2)

- Log into RPi

\$ ps -aux | grep helloWeb.py

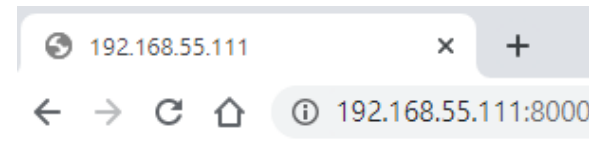
```
pi@rpi4-A00:~$ ps -aux | grep helloWeb.py
root      504    0.4  0.4  36212 18476 ?        S      18:36   0:00 python3 /home/p
i/helloWeb.py
pi        1350    0.0  0.0   7480   524 pts/0    S+     18:38   0:00 grep --color=au
to helloWeb.py
```

- Terminate the process

- sudo kill -9 504

\$ sudo kill -9 xxx

- Check the webpage again.



無法連上這個網站

Log Info

- We cannot get the log of the auto-started program.
- Output the log to a file.

`$ sudo nano /etc/rc.local`

- Modify the previous added command.

`sudo bash -c '/usr/bin/python3 /home/pi/helloWeb.py > /home/pi/helloWeb.log 2>&1' &`

- Reboot the RPi

`$ sudo reboot`

- Log in and show the log after rebooting

`$ cat helloWeb.log`

```
GNU nano 3.2 /etc/rc.local Modified
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

sudo bash -c '/usr/bin/python3 /home/pi/helloWeb.py > /home/pi/helloWeb.log 2>&1' &
exit 0
```

```
pi@pi4-A00:~ $ cat helloWeb.log
* Serving Flask app "helloWeb" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
192.168.55.54 - - [19/Dec/2020 18:46:10] "GET / HTTP/1.1" 200 -
```

Disable Method 1

\$ sudo nano /etc/rc.local

- Comment out the execution command.

```
# sudo bash -c '/usr/bin/python3 /home/pi/helloWeb.py > /home/pi/helloWeb.log 2$
```

Another Option

- rc.local is a good place to start your program whenever the system boots up (before users can log in or interact with the system).
- If you would like your program to start whenever a user logs in or opens a new terminal, consider adding a similar line to [/home/pi/.bashrc](#).

Method 2

- Example code: clock.py
- Get the code by the following link.

<https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec12/clock.py>

- Or, In RPi,

```
$ cd ~
```

```
$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec12/clock.py
```

Method 2 - autostart

- Create an autostart directory

```
$ mkdir /home/pi/.config/autostart
```

```
$ nano /home/pi/.config/autostart/clock.desktop
```

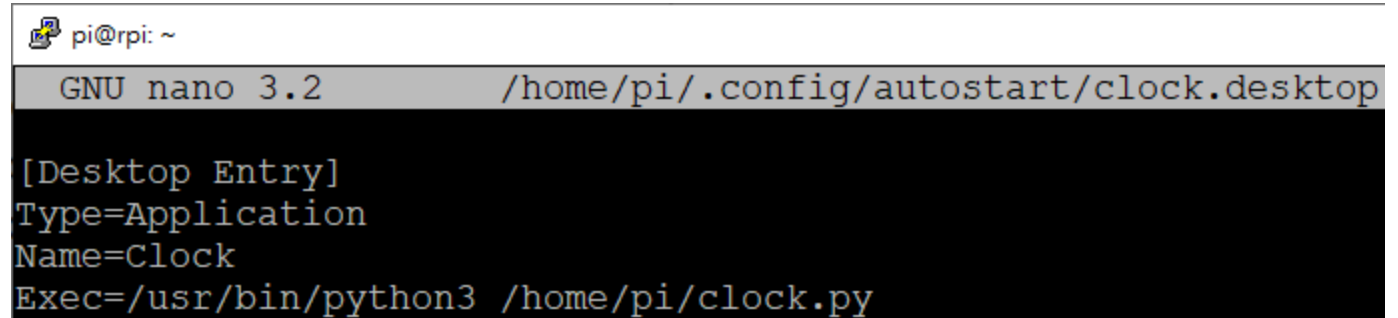
- clock.desktop

```
[Desktop Entry]
```

```
Type=Application
```

```
Name=Clock
```

```
Exec=/usr/bin/python3 /home/pi/clock.py
```



```
pi@rpi: ~  
GNU nano 3.2 /home/pi/.config/autostart/clock.desktop  
[Desktop Entry]  
Type=Application  
Name=Clock  
Exec=/usr/bin/python3 /home/pi/clock.py
```

- Reboot RPi

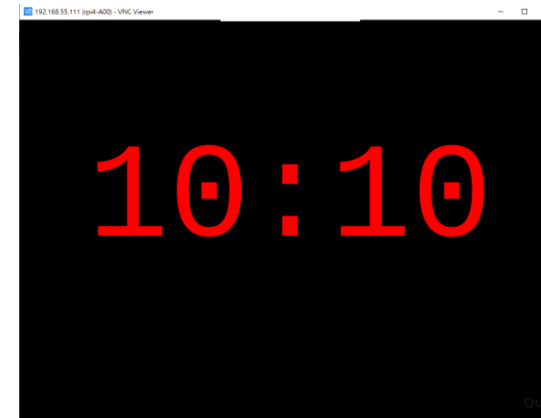
```
$ sudo reboot
```

Verification

- In the terminal,
`$ ps -aux | grep python`

```
pi@rpi4-A00:~ $ ps -aux | grep python3
pi          954  1.2  0.4  35732 16744 ?        S1   22:09   0:00 /usr/bin/python
3 /home/pi/clock.py
pi          958  4.4  1.6 185060 64552 ?        S1   22:09   0:01 /usr/bin/python
3 /usr/bin/blueman-applet
pi         1356  0.0  0.0   7348   544 pts/0    S+   22:09   0:00 grep --color=au
to python3
```

- Connect to your RPi via VNC viewer.



- Click “Quit” on the bottom right corner to exit.

Disable Method 2

- Rename to desktop file

```
$ cd /home/pi/.config/autostart/
```

```
$ mv clock.desktop clock.desktop.bak
```

- Or, delete the file directly.

Method 3 - systemd

- With `systemd`, you have the benefit of being able to tell Linux to start certain programs only after certain services have started.
- Create a unit file
 - Give information to systemd about a service, device, mount point, etc.
 - Start the program as a service (a process that runs in the background)

```
$ sudo nano /lib/systemd/system/hello.service
```

hello.service

[Unit]

Description=Hello Web

After=network.target

[Service]

ExecStart=/usr/bin/python3 /home/pi/helloWeb.py

[Install]

WantedBy=network.target

```
GNU nano 3.2 /lib/systemd/system/hello.service

[Unit]
Description=Hello Web
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/pi/helloWeb.py

[Install]
WantedBy=network.target
```

Special System Units

- <https://www.freedesktop.org/software/systemd/man/systemd.special.html>
- network.target
 - This unit is supposed to indicate when network functionality is available.
- multi-user.target
 - A special target unit for setting up a multi-user system (non-graphical).

Enable Service

- Tell system to recognize the service

\$ sudo systemctl daemon-reload

- Enable the service

\$ sudo systemctl enable hello.service

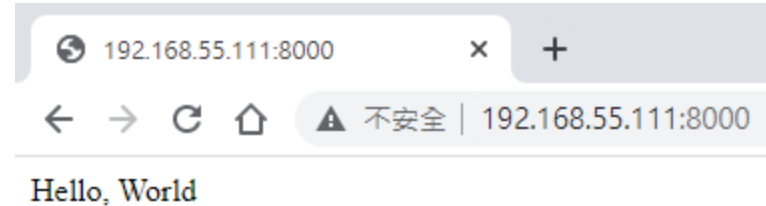
- Reboot RPi

\$ sudo reboot

```
pi@rpi4-A00:~ $ sudo systemctl daemon-reload
pi@rpi4-A00:~ $ sudo systemctl enable hello.service
Created symlink /etc/systemd/system/network.target.wants/hello.service → /lib/systemd/system/hello.service.
pi@rpi4-A00:~ $ sudo reboot
```

Verification

- Check the webpage after rebooting.



- Service status

\$ sudo systemctl status hello.service

```
pi@rp4-A00:~ $ sudo systemctl status hello.service
● hello.service - Hello Web
   Loaded: loaded (/lib/systemd/system/hello.service; enabled; vendor preset: en
   Active: active (running) since Sat 2020-12-19 22:24:05 CST; 1min 9s ago
     Main PID: 489 (python3)
        Tasks: 2 (limit: 4915)
      CGroup: /system.slice/hello.service
              └─489 /usr/bin/python3 /home/pi/helloWeb.py
```

Restart After Failure

```
$ sudo nano /lib/systemd/system/hello.service
```

```
[Unit]
```

```
Description=Hello Web
```

```
After=network.target
```

```
[Service]
```

```
ExecStart=/usr/bin/python3 /home/pi/helloWeb.py
```

```
Restart=always
```

```
RestartSec=10s
```

```
KillMode=process
```

```
TimeoutSec=infinity
```

```
[Install]
```

```
WantedBy=network.target
```

Update Setting

```
$ sudo systemctl disable hello.service
```

```
$ sudo systemctl daemon-reload
```

```
$ sudo systemctl enable hello.service
```

```
$ sudo reboot
```

Verification

- Check the process status

\$ ps -aux | grep python3

```
pi@rpi4-A00:~$ ps -aux | grep python3
root    501    1.9  0.4  26736 18352 ?        Ss   22:27   0:00 /usr/bin/python
3 /home/pi/helloWeb.py
pi      963    7.2  1.6 185056 64400 ?        Sl   22:27   0:01 /usr/bin/python
3 /usr/bin/blueman-applet
root   1296    1.9  0.6  49508 26472 ?        S    22:27   0:00 /usr/bin/python
3 /usr/lib/blueman/blueman-mechanism
pi     1321    0.0  0.0   7348   556 pts/0    S+   22:27   0:00 grep --color=au
to python3
```

- Check the availability of the web page.
- Kill the process by PID.

\$ sudo kill -9 {PID}

- The program is restarted after killing.

\$ ps -aux | grep python3

```
pi@rpi4-A00:~$ ps -aux | grep python3
pi      970    3.0  1.6 185060 64392 ?        Sl   22:30   0:01 /usr/bin/python
3 /usr/bin/blueman-applet
root   1336    4.0  0.4  26736 18476 ?        Ss   22:31   0:00 /usr/bin/python
3 /home/pi/helloWeb.py
pi     1338    0.0  0.0   7348   552 pts/0    S+   22:31   0:00 grep --color=au
to python3
```


Disable Service

```
$ sudo systemctl disable hello.service
```

- Manually start/stop the services
 - sudo systemctl stop hello.service
 - sudo systemctl start hello.service

More

- Daemontools is a tool to monitor the status of your process and restart after failure.
- <https://gist.github.com/connorjan/01f995511cfd0fee1cf9e2387024b54a>

Outline

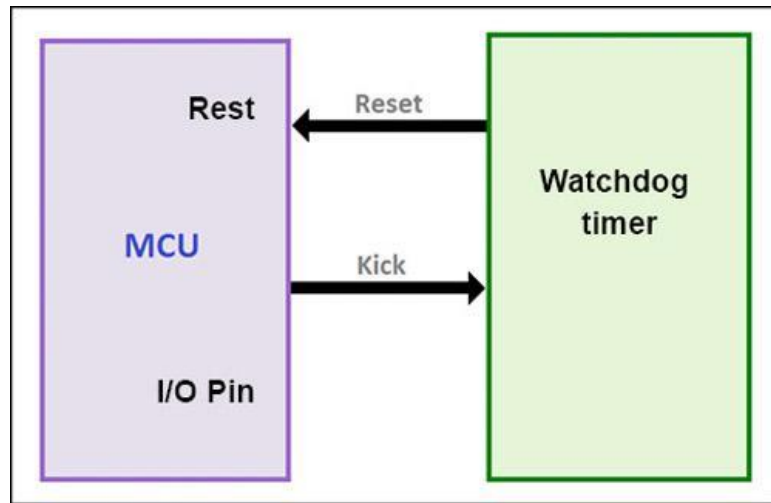
- Auto-start Program
- System Watchdog
- Threading

System Watchdog

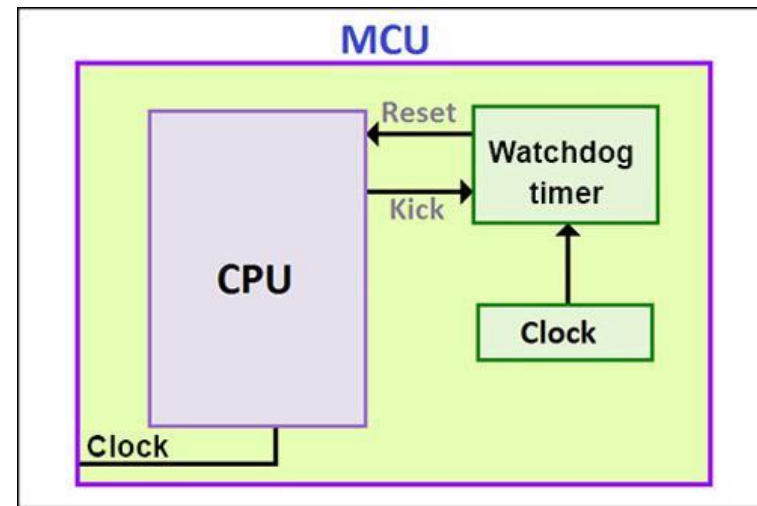
- System should keep alive without user intervention.
- We need a mechanism to find out failure events.
- Watchdog mechanism maintains a countdown timer and reset by others for proving that the system is alive.
 - Hardware watchdog
 - Software watchdog

Hardware Watchdog

- External



- Internal



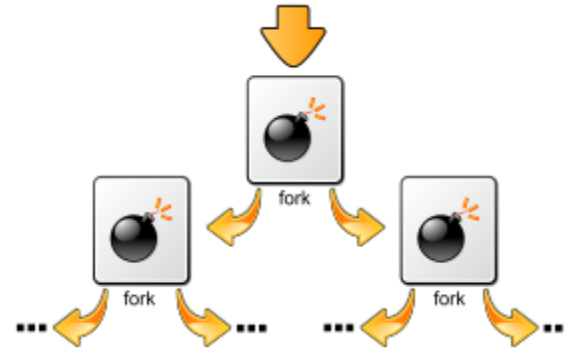
Crash RPi

- Use fork bomb to crash your system

\$:(){ :|:& };;

```
pi@rpi4-A00:~ $ :(){ :|:& };;  
[1] 1364
```

- Semicolon is a function name.



- Observe the results
 - The system is too busy to answer your input.
- Unplug the power for 5 seconds and plug the power.

Installation

- Enable the watchdog module

```
$ sudo modprobe bcm2835_wdt
```

- Add the module to the list for loading at boot time.

```
$ echo "bcm2835_wdt" | sudo tee -a /etc/modules
```

- Install the watchdog package.

```
$ sudo apt-get install watchdog
```

Configurations

- Uncomment the following settings

\$ sudo nano /etc/watchdog.conf

```
# Uncomment to enable test. Setting one of these values to '0' disables it.  
# These values will hopefully never reboot your machine during normal use  
# (if your machine is really hung, the loadavg will go much higher than 25)  
max-load-1          = 24  
max-load-5          = 18  
max-load-15         = 12
```

```
#retry-timeout      = 60  
#repair-maximum     = 1  
  
watchdog-device = /dev/watchdog
```

- Reboot RPi

\$ sudo reboot

max-load-1 = <load1>

Set the maximal allowed load average for a 1 minute span. Once this load average is reached the system is rebooted. Default value is 0. That means the load average check is disabled. Be careful not to this parameter too low. To set a value less then the predefined minimal value of 2, you have to use the -f commandline option.

max-load-5 = <load5>

Set the maximal allowed load average for a 5 minute span. Once this load average is reached the system is rebooted. Default value is $3/4 * \text{max-load-1}$. Be careful not to this parameter too low. To set a value less then the predefined minimal value of 2, you have to use the -f commandline option.

max-load-15 = <load15>

Set the maximal allowed load average for a 15 minute span. Once this load average is reached the system is rebooted. Default value is $1/2 * \text{max-load-1}$. Be careful not to this parameter too low. To set a value less then the predefined minimal value of 2, you have to use the -f commandline option.

Verification

- Crash your system by fork bomb again.

```
$ :(){ :|:& }::
```

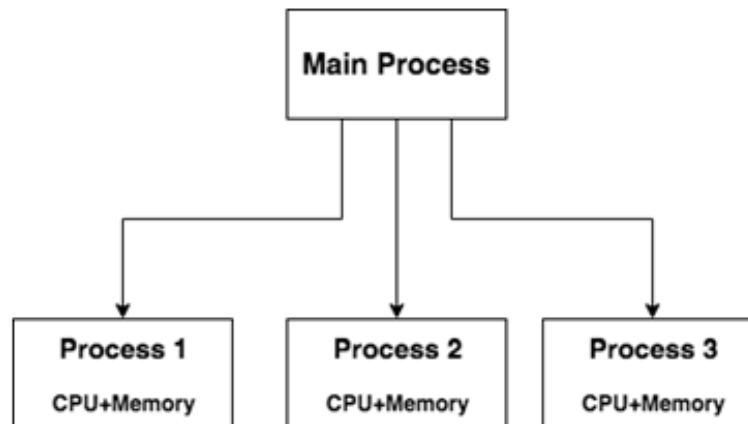
- See if it recovers from the failure by rebooting.

Outline

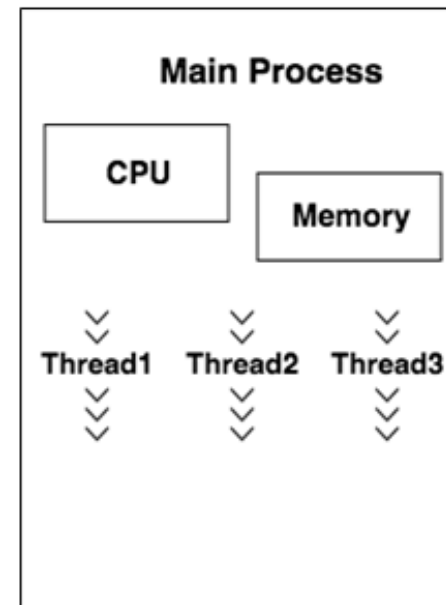
- Auto-start Program
- System Watchdog
- Threading

Threading

Multiprocessing



Multithreading



threads.py

\$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec12/threads.py

\$ python3 threads.py

```
import logging
import threading
import time

def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)
if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO, datefmt="%H:%M:%S")
    logging.info("Main    : before creating thread")
    x = threading.Thread(target=thread_function, args=(1,))
    logging.info("Main    : before running thread")
    x.start()
    logging.info("Main    : wait for the thread to finish")
    # x.join()
    logging.info("Main    : all done")
```

■ Results

```
pi@rpi4-A00:~/iot/lec13 $ python3 threads.py
22:51:38: Main    : before creating thread
22:51:38: Main    : before running thread
22:51:38: Thread 1: starting
22:51:38: Main    : wait for the thread to finish
22:51:38: Main    : all done
22:51:40: Thread 1: finishing
```

daemon_thread.py

- A daemon thread will shut down immediately when the program exists.

\$ nano threads.py

```
if __name__ == "__main__":  
    format = "%(asctime)s: %(message)s"  
    logging.basicConfig(format=format, level=logging.INFO, datefmt="%H:%M:%S")  
    logging.info("Main      : before creating thread")  
    x = threading.Thread(target=thread_function, args=(1,), daemon=True)  
    logging.info("Main      : before running thread")  
    x.start()  
    logging.info("Main      : wait for the thread to finish")  
    # x.join()  
    logging.info("Main      : all done")
```

- Results

```
pi@pi4-A00:~/iot/lec13 $ python3 threads.py  
22:54:29: Main      : before creating thread  
22:54:29: Main      : before running thread  
22:54:29: Thread 1: starting  
22:54:29: Main      : wait for the thread to finish  
22:54:29: Main      : all done
```

join()

- To tell one thread to wait for another thread to finish.
- Uncomment “x.join()” and disable the daemon thread.

\$ nano threads.py

```
x = threading.Thread(target=thread_function, args=(1,))
```

```
x.join()
```

- Results

```
pi@rpi4-A00:~/iot/lec13 $ python3 threads.py
22:55:36: Main      : before creating thread
22:55:36: Main      : before running thread
22:55:36: Thread 1: starting
22:55:36: Main      : wait for the thread to finish
22:55:38: Thread 1: finishing
22:55:38: Main      : all done
```

multithreads.py

\$ wget https://raw.githubusercontent.com/yachentw/yzucseiot/main/lec12/multithreads.py

\$ python3 multithreads.py

```
import logging
import threading
import time
def thread_function(name):
    logging.info("Thread %s: starting", name)
    time.sleep(2)
    logging.info("Thread %s: finishing", name)

if __name__ == "__main__":
    format = "%(asctime)s: %(message)s"
    logging.basicConfig(format=format, level=logging.INFO, datefmt="%H:%M:%S")
    threads = list()
    for index in range(3):
        logging.info("Main      : create and start thread %d.", index)
        x = threading.Thread(target=thread_function, args=(index,))
        threads.append(x)
        x.start()
    for index, thread in enumerate(threads):
        logging.info("Main      : before joining thread %d.", index)
        thread.join()
        logging.info("Main      : thread %d done", index)
```

Results

```
pi@rpi4-A00:~/iot/lec13 $ python3 multithreads.py
22:58:33: Main      : create and start thread 0.
22:58:33: Thread 0: starting
22:58:33: Main      : create and start thread 1.
22:58:33: Thread 1: starting
22:58:33: Main      : create and start thread 2.
22:58:33: Thread 2: starting
22:58:33: Main      : before joining thread 0.
22:58:35: Thread 0: finishing
22:58:35: Main      : thread 0 done
22:58:35: Thread 1: finishing
22:58:35: Main      : before joining thread 1.
22:58:35: Main      : thread 1 done
22:58:35: Thread 2: finishing
22:58:35: Main      : before joining thread 2.
22:58:35: Main      : thread 2 done
```


Synchronization

- <https://docs.python.org/3/library/asyncio-sync.html>
- Lock
- Event
- Condition
- Semaphore
- BoundedSemaphore

Message Passing

- <https://docs.python.org/3/library/queue.html>
- The queue module implements multi-producer, multi-consumer queues.
- It is especially useful in threaded programming when information must be exchanged safely between multiple threads.
- The Queue class in this module implements all the required locking semantics.

```
import threading, queue

q = queue.Queue()

def worker():
    while True:
        item = q.get()
        print(f'Working on {item}')
        print(f'Finished {item}')
        q.task_done()

# turn-on the worker thread
threading.Thread(target=worker, daemon=True).start()

# send thirty task requests to the worker
for item in range(30):
    q.put(item)
print('All task requests sent\n', end='')

# block until all tasks are done
q.join()
print('All work completed')
```

Lab

- Blink two LEDs by **threading**.
 - One thread blinks one LED every 1 second.
 - Another thread blinks another LED every 2 seconds.
- **Automatically** run your program after the system is booted on.
 - You can use one of these 3 methods for the implementation.
- Use fork bomb to crash your system to **see if your system will be rebooted and the blink program will be executed**.
 - Enable watchdog mechanism.