# Construyendo un daemon

## Requisitos

El daemon fue construido utilizando NodeJS versión 8.9.4. Cualquier error sintactico producido por el uso de ES6 o funciones no existentes en versiones anteriores quedan fuera del alcance del presente documento.

---

## Creando el controlador del daemon

El controlador del daemon controla la funciones básicas del mismo: Start, Stop, Restart y Status. El mismo implementa las funciones adicionales que permiten realizar el fork del proceso, controlar/operar sobre el estado del fork y parsear los comandos que recibe el controlador. Explicaremos las funciones de una en una que componen a este componente del sistema:

- **checkStatus**: Esta función controla el estado del daemon. Básicamente, controla si este esta o no en ejecución. Para esto, se apoya en un archivo que contiene el PID con el que se inicio el proceso del daemon en cada momento. Mediante este archivo, verifica si el PID almacenado corresponde a un proceso en ejecución.

```
function checkStatus(callback) {
  fs.readFile(pidFile, 'utf-8', (err, data) => {
    if(err) {
      callback(stopped);
    } else {
      ps.lookup({ pid: parseInt(data) }, (err, resultList) => {
        if(err) {
          console.log(err);
        }
        if(resultList.length) {
          callback(running, resultList[0].pid);
        } else {
          callback(stopped);
        }
      })
    }
  })
}
```

- **daemonStart**: Esta función como objetivo iniciar el proceso daemon. Para ello, en primer lugar verifica que el daemon no este en ejecución valiendose de la función checkStatus. Finalmente, si el daemon va a iniciarse, hace un fork y queda a la espera de que el proceso confirme que inicio correctamente. Si esta confirmación no llega en una ventana de tiempo de 5s, envía una señal para que el proceso aborte su ejecución.

```
function daemonStart() {
  checkStatus((status) => {
    if(status === stopped) {
      const forked = fork('sl-daemon-loop.js');
      new Promise((resolve, reject) => {
        forked.on('message', (data) => {
          if(data.start) {
```

```
            console.log(`${infoFormat}Started sl-daemon with PID:${formatClose} ${data.start}`);
            log(`Started sl-daemon with PID: ${data.start}`);
            fs.writeFile(pidFile, data.start, function(err) {
              if(err) {
                return console.log(err);
              }
              resolve();
            })
          }
        });
        setTimeout(() => {
          reject();
        }, maxTimeout);
      }).then(() => {
        process.exit();
      }).catch(() => {
        forked.send({terminate: true});
        console.log(`${errorFormat}sl-daemon timed out, start aborted.${formatClose}`);
        log(`sl-daemon timed out, start aborted.`);
        process.exit();
      });
    } else {
      console.log(`${errorFormat}sl-daemon is already running.${formatClose}`);
    }
  });
}
```

- **daemonStop**: Esta función (como lo dice su nombre se encarga de detener al daemon). Simplemente válida que el daemon se este ejecutando utilizando la función checkStatus y si el daemon esta en ejecución lo detiene.

```
function daemonStop(callback) {
  checkStatus((status, pid) => {
    if(status === running) {
      process.kill(pid);
      console.log(`${infoFormat}sl-daemon has been stopped.${formatClose}`);
      log(`sl-daemon has been stopped.`)
      if(callback){
        callback()
      }
    } else {
      console.log(`${errorFormat}sl-daemon is not running.${formatClose}`);
    }
  });
}
```

- **daemonRestart**: Esta función es una combinación de start y stop daemon. Si el daemon se encuentra en ejecución lo detiene y, utilizando el callback de la función stopDaemon, inicia nuevamente el daemon cuando el proceso ha sido detenido. En caso, que el daemon este detenido, solamente lo inicia. En todos los casos, utiliza la función checkStatus para comprobar el estado del daemon.

```
function daemonRestart() {
  checkStatus((status, pid) => {
    if(status === running) {
      daemonStop(daemonStart);
```

```
    } else {
      daemonStart();
    }
  })
}
```

- **daemonStatus**: Esta función simplemente informa el estado del daemon por consola. Para ello utiliza la función checkStatus.

```
function daemonStatus() {
  checkStatus((status, pid) => {
    switch(status){
      case running:
        console.log(`${infoFormat}sl-daemon is running with pid${formatClose}: ${pid}`);
        break;
      case stopped:
        console.log(`${errorFormat}sl-daemon is stopped...${formatClose}`);
        break;
      default:
        break;
    }
    process.exit();
  })
}
```

- **Parseo de comandos**: En última instancia el controlador debe interpretar los comandos con los que es invocado.

```
const option = process.argv[2].toUpperCase();

switch(option) {
  case START:
    daemonStart();
    break;
  case STOP:
    daemonStop();
    break;
  case STATUS:
    daemonStatus();
    break;
  case RESTART:
    daemonRestart();
    break;
  default:
    console.log(`${errorFormat}Invalid command${formatClose} -- USAGE: node sl-daemon-controller.
js {START | STOP | STATUS | RESTART}`);
    break;
}
```

## Servicios auxiliares

Podemos agregar funciones adicionales como llevar un log de los eventos que van sucediendo en el daemon. Un log sencillo sin rotación se puede lograr agregando esta función tanto en el controlador como en el loop del daemon.

```
function log(eventData) {
  const dateString = new Date(new Date() - (3*60*60*1000)).toISOString()
    .replace(/T/, ' ')
    .replace(/\..+/, '');
  const logEntry = `${dateString}: ${eventData}\n`;
  fs.appendFileSync(logFile, logEntry);
}
```

## El loop del daemon

El loop del daemon consiste de un programa aparte que será ejecutado en un proceso aparte y, posteriormente, desvinculado por el controlador del daemon. El loop contiene la funcionalidad del daemon, es decir, lo que hace. Las posibilidades son muchas, podemos hacer un programa que monitoree un archivo, un directorio, escuchar un puerto. El daemon podría monitorear constantemente o dormirse y pasado un intervalo de tiempo despertar y ver si tiene trabajo que hacer. En este caso hacemos un daemon simple que escucha el puerto 1337 y utiliza el notify-send de ubuntu para notificar de los mensajes entrantes por dicho puerto.

```
/*
Writes important events into the daemons log file
*/
function log(eventData) {
  const dateString = new Date().toISOString()
    .replace(/T/, ' ')
    .replace(/\..+/, '');
  const logEntry = `${dateString}: ${eventData}\n`;
  fs.appendFileSync(logFile, logEntry);
}

/*
If for some reason the parent aborts the daemon starting process, this process
will be terminated.
*/
process.on('message', (data) => {
    if(data.terminate) {
        process.exit();
    }
})


//Creates a TCP server to listen a port
const server = net.createServer(socket => {
    socket.on('data', (data) => {
        const execString = `notify-send -t 20 \"${notifyTitle}\" \"${data.toString().substring(0,
 data.toString().length - 2)}\"`;
        exec(execString);
        log("received message.");
    });
});

//Makes the TCP server to listen 127.0.0.1:1337
server.listen(port);

//When the TCP server is completely setup sends daemon pid to parent.
process.send({start: process.pid})
```

## Iniciando y deteniendo el Daemon

El daemon construido se ha realizado en NodeJS, con lo cual para utilizarlo debemos utilizar NodeJS (sin este no podemos ejecutar javascript fuera del browser). Para controlar el daemon podemos usar el siguiente comando:

```
node sl-daemon-controler.js <command>
```