# Práctica de Sistemas Operativos

*v1.0*

## 1. Descripción

Se trata de implementar un framework de testing en C llamado *cunit*. Una ejecución del programa se ocupa de ejecutar todos los tests del directorio actual. Se considerará que todos los ficheros acabados en `.tst` son tests a realizar. En cada fichero de test tiene que haber líneas con un nombre de comando y sus argumentos. Un fichero de test representa un pipeline con todos los comandos que hay en el fichero. Por ejemplo:

```
$ cat x.tst
ps x
grep bash
```

Se corresponde con el pipeline:

```
ps x | grep bash
```

El programa ejecutará todos los tests concurrentemente y dejará la salida de cada test en un fichero con el mismo nombre del test pero con la extensión `.out`. Por ejemplo, el fichero correspondiente a `bla.tst` es `bla.out`. El fichero `.out` debe contener la salida estándar y la salida de errores del test. La entrada del test tiene que estar redirigida a `/dev/null`.

Si existe un fichero con el mismo nombre con extensión `.ok` entonces se debe comparar con la salida. El test se considera fallido si el fichero `.out` y el fichero `.ok` difieren. Si no existiese el fichero `.ok`, se creará con el contenido del fichero `.out`. El test también se considera fallido si el pipeline falla. En general, un pipeline se debe considerar que ha terminado cuando ha terminado el último comando, se considera que ha fallado cuando ha fallado el último comando, etc.

Los comandos del fichero de test podrán ser, en este orden de prioridad, un builtin del programa de test, un fichero ejecutable que se encuentre en el directorio de trabajo o un fichero ejecutable que se encuentre en alguno de los directorios de que se encuentre en cualquiera de los directorios de la variable PATH.

El programa de test debe entender la notación `$nombre` y reemplazar `$nombre` por el valor de la variable de entorno `nombre`.

El comando builtin `cd` cambia el directorio de trabajo para todos los comandos del test. Si el test usa el builtin, siempre debe estar en la primera línea del fichero. Por ejemplo, al ejecutar el siguiente test en nuestro directorio home:

```
cd /tmp
ps
tee a.txt
grep hola
```

se crearía un fichero `/tmp/a.txt` (lo crearía el comando `tee`, ya que su directorio de trabajo debe ser `/tmp`).

No se permite utilizar pexec, system o cualquier llamada al sistema o función de biblioteca similar. Sólo se permite ejecutar comandos externos para ejecutar los comandos de los ficheros de test. Es imprescidible que el programa esté íntegramente programado en C.

Para poder obtener más de un 7.0 en la práctica, se deberán implementar partes opcionales.

### 2. Parte opcional: timeout

Con el flag -t, se establece un timeout en segundos para los tests. Si un test tarda más de los segundos especificados en ejecutar, se considera fallido. Por ejemplo:

```
$ cunit -t 10
```

### 3. Parte opcional: clean

Con el flag -c, el programa borrará todos los ficheros `.ok` y `.out` del directorio de trabajo. En este caso, no se ejecutará ningún test. Este argumento es incompatible con `-t`.

```
$ ls *.ok *.out
a.ok
a.out
b.ok
b.out
$ cunit -c
$ ls *.ok *.out
ls: *.ok: No such file or directory
ls: *.out: No such file or directory
$
```

### 4. Parte opcional: ejecución condicional

El programa procesará ficheros `.cond`, que tendrán la misma sintaxis que los `.tst`. Para estos ficheros, el programa ejecuta sus líneas una tras otra (sin pipeline) hasta que alguna ejecute con éxito. En este momento, se considerará que el test es correcto. Si al final todos los comandos fallan, se considerará fallido. Respecto a los ficheros de salida, se debe comportar igual que en el caso básico.

Por ejemplo:

```
$ cat x.cond
false
true
grep holA /tmp/a
```

En ese ejemplo, se ejecutaría `false` y `true`, pero no se ejecutaría `grep`.

**5. Entrega**

La entrega del proyecto es presencial y se realizará el mismo día que el test final de la asignatura.