HINTS para el TP1

Como ya les he mencionado, es muy importante que le dediquen tiempo a las manpages, ya que ahí tienen muchas respuestas.

HINT 1:

Les presento algunas manpages que son muy útiles para lo que deben hacer para el TP1

- **fifo (7)** - first-in first-out special file, named pipe.
- **pipe (7)** - overview of pipes and FIFOs. En esta manpage tienen los detalles de si los pipes son unidireccionales o no y cuántos bytes el SO garantiza que se escriben atómicamente.
- s**em_overview (7)** - overview of POSIX semaphores.
- **shm_overview (7)** - overview of POSIX shared memory.
- **signal (7)** - overview of signals

HINT 2:

Como ya habrán visto, varias manpages tienen ejemplos simples donde pueden ver cómo utilizar la API del SO y de las librerías de C. En algunos casos tienen referencias a las manpages con ejemplos, por ejemplo, en sem_overview (7) dice:

"An example of the use of various POSIX semaphore functions is shown in sem_wait(3)."

Otra forma de buscar ejemplos en alguna sección específica es

```
$ man -K <section number> "Program source"
```

Los ejemplos en las manpages tienen como subtítulo "Program source"

HINT 3:

Para aquellos que requieran que un proceso monitoree múltiples fds, por ejemplo, que la aplicación espere por información para ser leída en varios pipes (sin hacer busy waiting), les recomiendo (fuertemente) usar select (2) y si leen esta manpage con atención encontrarán una referencia a otra manpage muy interesante para el uso de select.

HINT 4:

El docker ya tiene instaladas herramientas de análisis estático.

Análisis estático con cppcheck:

```
$ cppcheck --quiet --enable=all --force --inconclusive .
```

Análisis estático con PVS-Studio:

La imagen de docker ya está desactualizada respecto a PVS-studio, pero se resuelve ejecutando:

```
$ apt-get update
$ apt-get install pvs-studio
```

Para poder utilizar la versión gratuita de PVS-Studio deberán agregar 2 comentarios al inicio de cada source:

```
// This is a personal academic project. Dear PVS-Studio, please
check it.
// PVS-Studio Static Code Analyzer for C, C++ and C#:
http://www.viva64.com
```

Pueden hacer esto automáticamente con los siguientes comandos, que recursivamente en el directorio actual agregan los comentarios en todos los archivos .c:

CUIDADO: Esto itera sobre el directorio recursivamente, asegurarse que están parados en el proyecto y no, po ejemplo, en el home.

```
find . -name "*.c" | while read line; do sed -i '1s/^\(.*\)$/\/\/
This is a personal academic project. Dear PVS-Studio, please check
it.\n\1/' "$line"; done

find . -name "*.c" | while read line; do sed -i '2s/^\(.*\)$/\/\/
PVS-Studio Static Code Analyzer for C, C++ and C#:
http:\/\/www.viva64.com\n\1/' "$line"; done
```

Finalmente ejecutamos PVS-studio, recuerden que deben tener un Makefile y deben hacer make clean antes de ejecutar PVS-Studio ya que esta herramienta requiere recompilar todo.

```
$ pvs-studio-analyzer trace -- make
$ pvs-studio-analyzer analyze
$ plog-converter -a 'GA:1,2,3;64:1,2,3;OP:1,2,3' -t tasklist -o
report.tasks PVS-Studio.log
```

Una vez ejecutados estos comandos, encontrarán en el archivo report.tasks una lista con el resultado de la herramienta. Recuerden también que pueden visitar el sitio oficial de PVS-Studio con los códigos de cada error en busca de una explicación del mismo.

Es necesario pasarle el siguiente flag a docker run para que PVS-studio funcione correctamente

```
--security-opt seccomp:unconfined
```

Análisis dinámico:

```
$ valgrind ./a.out
```

HINT 5:

Les paso un par de aplicaciones que les pueden resultar útiles para hacer un poco de debugging

- **ipcs (1)** - show information on IPC facilities. Interesante para ver si están limpiando bien los recursos cuando terminan la app.
- **lsof (8)** (apt-get install lsof) - list open files. Interesante para comprender un poco mejor todos los archivos que un proceso está utilizando.
- **strace** - trace system calls and signals. Interesante para monitorear las syscalls que invoca un proceso en tiempo real, también es útil para ver en qué syscall se bloquea un proceso y

que retorna cada una (en caso de que no estén capturando los valores de retorno (como deberían)).

- **pidof** - find the process ID of a running program. Interesante para que el pasarle el pid de la aplicación el proceso vista.
- **pv (1)** (apt-get install pv) - monitor the progress of data through a pipe. Posiblemente no lo usen. para el TP, pero es interesante para comprender qué tan rápido pasa la información por un pipe.
- **pmap (1)** - report memory map of a process.