Lineamientos de programación

Juan F. Codagnone

Revisión: 2018-02-25 12:53:42

1. Introducción

Uno de los objetivos de la materia es que “el alumno logre programar aplicaciones Cliente/Servidor de dos Capas y sus estrategias”. Para cumplir el objetivo los alumnos desarrollarán un Trabajo Práctico Especial utilizando el lenguaje de programación C sobre entornos `POSIX`.

Los alumnos ya han sido expuestos en diferentes materias previas al desarrollo de programas con el lenguaje de programación C y a las interfaces `POSIX`[1]; y es por esto que se espera que el código fuente escrito sea claro, con buen estilo de programación, con un buen manejo de errores y sin problemas de referencias de memoria. Los programas requieren manejo de entrada/salida `UNIX`, manejo de procesos, y manejo de señales.

El objetivo del presente documento es proveer lineamientos generales para la realización del Trabajo Práctico Especial y de los Ejercicios de programación introductorios al Trabajo Práctico Especial.

2. Requerimientos no funcionales

Todos los programas que desarrolle deben:

1. Estar escritos en el lenguaje de programación C, específicamente con la variante `C99` (`ISO/IEC 9899:1999`[2]).

2. Buscar ante todo la portabilidad de los programas. La Guía de Desarrollo de Git [https://git.kernel.org/pub/scm/git/git.git/plain/Documentation/CodingGuidelines?id=master] expresa buenos lineamientos a seguir:

   > Most importantly, we never say "It's in POSIX; we'll happily ignore your needs should your system not conform to it." We live in the real world.
   >
   > However, we often say "Let's stay away from that construct, it's not even in POSIX".
   >
   > In spite of the above two rules, we sometimes say "Although this is not in POSIX, it (is so convenient | makes the code much more readable | has other good characteristics) and practically all the platforms we care about support it, so let's use it".

[1] Sistemas Operativos (72.11), Arquitecturas de Computadoras (72.08), Programación Imperativa (72.31).

[2] Podrá encontrar diferentes versiones del estándar en `http://www.open-std.org/JTC1/SC22/WG14/www/standards`.

> Again, we live in the real world, and it is sometimes a judgement call, the decision based more on real world constraints people face than what the paper standard says.

En caso de utilizar interfaces provista por `POSIX`, elija las provistas por `POSIX.1-2008` (`IEEE Std 1003.1-2008, 2016 Edition`[3]).

3. Seguir los lineamientos de `IEEE Std 1003.1-2008, 2016 Edition / Base definitions / 12. Utility Conventions`[4] a menos que se especifique lo contrario: Esto se refiere a cómo manejar argumentos de línea de comandos, parámetros, etc.

4. Ejecutar correctamente en `pampero.itba.edu.ar` (lo que no significa que únicamente deba correr allí). Se recomienda utilizar **gdb**[5]para depurar los programas.

5. Evitar accesos fuera de los límites del heap, stack; use-after-free [https://www.owasp.org/index.php/Using_freed_memory], use-after-return [https://github.com/google/sanitizers/wiki/AddressSanitizerUseAfterReturn], double-free [https://www.owasp.org/index.php/Double_Free].

   Seguir referencias equivocadas lleva a que el programa termine inesperadamente, y no permite evaluar la ejecución del programa. Utilice únicamente versiones seguras de funciones (por ejemplo prefiera `snprintf` a `sprintf`). Preste atención a la especificaciones de las funciones de sistema que decida utilizar. Por ejemplo, al usar la función `strlen`, recuerde que "*returns the number of characters that precede the terminating null character*": si la utiliza para contar bytes para realizar una copia es probable que deba sumar un byte para copiar el terminador '\0'.

   valgrind [http://valgrind.org/] ayuda a detectar muchos de estos errores de programación. Tanto **gcc** como **clang** disponen opciones de instrumentación del código generado que permiten durante la ejecución del programa detectar estos errores de programación mas eficientemente que **valgrind**. En ambos compiladores `-fsanitize=address` activa la instrumentación para detectar muchos de estos problemas. Una introducción a la funcionalidad en **clang** se encuentra en `https://github.com/google/sanitizers/wiki/AddressSanitizer`. Un listado completo de instrumentaciones disponible para **gcc** está disponible en `https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html` (prestar atención en las opciones `-fsanitize=`).

Acompañe los programas con un archivo `README` que permita a un lector casual entender el marco teórico/estandares en los que se basa el trabajo, cómo es el funcionamiento del programa, como operarlo y cuales son sus limitaciones.

2.1. Estilo del código

Intente ser fiel a un único estilo de codificación. Preste atención a:

[3] Podrá acceder a una versión en linea en `http://pubs.opengroup.org/onlinepubs/9699919799/nframe.html`.
[4] `http://pubs.opengroup.org/onlinepubs/9699919799/nframe.html`.
[5] `https://www.gnu.org/software/gdb/`.

- definir y respetar un número máximo de columna. La utilización de muchas columnas puede dificultar la lectura. Ochenta columnas es el ancho mínimo que se suele aceptar, y mucha veces es suficiente (depende en general del ancho de indentación elegido: dos espacios, cuatro espacios, ocho espacios).

- definir y respetar un ancho de indentación (*sangría* si utilizamos una palabra aceptada por la Real Academia Española).

- definir y respetar la ubicación de llaves y espacios. Es bastante molesto leer códigos inconsistentes:

```
if ((valread = read( clntSocket , buffer, bufsize)) <= 0){
  return valread;
}
else if (  !err )
{
```

- ser específico con la utilización de modificadores en variables y funciones. Use el modificador `const` en toda variable que no modificará (en especial en los argumentos de las funciones) y `static` en toda función que no debe referirse desde otros módulos.

- la alineación de asignaciones. Alinear asignaciones facilita la lectura. Considere los siguientes ejemplos:

```
addr->sin_family = AF_INET;
addr->sin_addr.s_addr = INADDR_ANY;
addr->sin_port = htons( PORT );
```

```
addr->sin_family      = AF_INET;
addr->sin_addr.s_addr = INADDR_ANY;
addr->sin_port        = htons( PORT );
```

  Tenga en cuenta todos los aspectos que hagan a la buena performance, escalabilidad y disponibilidad del servidor. Se espera que se maneje de forma eficiente los flujos de información (por ejemplo no cargar en memoria mensajes muy grandes, ser eficaz y eficiente en el intérprete de mensajes).

```
char bufferAux[bufsize];
int valread = 10;
char* respStr = NULL;
```

```
char  bufferAux[bufsize];
int   valread            = 10;
char *respStr            = NULL;
```

- no abusar de la directiva `#define` para enumerados. Es preferible utilizar `enum` a `#define` ya que facilita por ejemplo la depuración. Al depurar un programa que declara enumerados de esta forma:

```
#define USER 1
#define PASS 2
#define QUIT 3
```

  se mostrará `1`, `2` o `3`; mientras que si se define como un `enum`, se verá el nombre del mismo.

Hay quienes prefieren automatizar el cumplimiento del estilo de codificación. En ese sentido **clang-format** [https://clang.llvm.org/docs/ClangFormat.html] provee una buena integración con los editores de texto (formateo automático al guardar el archivo).

2.2. Librerías Externas

Si bien las programas son pequeños podrá utilizar librerías o archivos (fragmento de código) desarrollados por terceros siempre que se cumplan los siguientes requisitos:

- La librería o fragmento NO DEBE resolver las cuestiones de fondo del Trabajo Práctico.
- La librería o fragmento DEBE tener una licencia aprobada por la Open Source Initiative [https://www.opensource.org/licenses].
- El uso de la librería o fragmento DEBE ser aprobada por la Cátedra.

Para lograr la aprobación un alumno del grupo DEBE publicar una secuencia en el foro de discusión (Sección 3, “Foro de discusión”). La secuencia DEBE describir todos aquellos datos que permitan identificar a la librería (por ejemplo la versión); su licencia de esta forma justificando porqué es válido su uso; y el propósito de su inclusión. En caso de que sea un fragmento de código debe adjuntarse.

Un caso típico de librería externa a utilizar es el de aquella librerías que facilitan las pruebas unitarias.

Está permitido utilizar código publicado por los docentes durante la cursada actual, siempre que se atribuya correctamente.

3. Foro de discusión

La materia dispone de foros de discusión alojados en `campus.itba.edu.ar` donde los alumnos podrán hacer preguntas sobre los diferentes temas de las prácticas y en particular sobre el trabajos prácticos especial. Los foros de discusión también serán utilizados por los docentes para compartir información de interés (cobertura en los medios de comunicación sobre algún tema relacionado).

Se prefiere la utilización del foro de discusión sobre otros medios de comunicación ya que todos los alumnos reciben una respuesta a una pregunta que podrían llegar a hacer otros alumnos. Para

tratar temas más sensibles (calificaciones, ausencias, etc) utilizar correo electrónico (publicado en SGA [https://sga.itba.edu.ar/]).

4. Ambigüedades

A veces existirán ambigüedades en las especificaciones o múltiples formas en como se puede resolver o implementar un problema particular. Por ser una materia de ingeniería se espera que los alumnos tomen decisiones de diseño razonables en estos casos. Los alumnos pueden basar sus decisiones en lo que conoce de ante mano de la tarea y en los objetivos enumerados en este documento o demás enunciados. Los docentes pueden darle consejos sobre las ventajas y desventajas de cada decisiones, pero los alumnos son los que en última instancia las toman.

5. Autoría y atribución

El material que los alumnos entreguen debe ser de su autoría. En el caso donde la autoría sea parcial debe declararse debidamente y mantener la atribución. Un ejemplo de esta situación es la realización de un ejercicio de programación introductorio con un alumno que no terminó formando parte del grupo del TPE. Una buena estrategia es siempre armar buenos parches de cambios, y luego al reusarlo importarlo agregar información de `Signed-off-by`.

Reconozca y atribuya tanto en el código fuente como en los informes y exposiciones aquellas ideas y estrategias tomadas de las correcciones de los *ejercicios de programación introductorio al TPE* y de lectura de *papers* o libros.