

Tarea 1

CC51C Comunicación de Datos

30 de Abril de 2001 - Plazo: 2 semanas (vence el 14 de Mayo de 2001)

1 Descripción

Se deberán implementar 3 algoritmos de control de errores, para lo cual se cuenta con una interfaz definida que es necesario seguir.

Se proveen varios archivos en el directorio `~cc51c/T1` de anakena, los que son dependientes de la arquitectura tienen una versión para solaris/sparc y otra para linux/x86:

- `tester.o`: contiene el código para realizar pruebas, y deberá ser linkeado con otro archivo de objetos que provea la implementación del control de errores. Contiene una función `main` que compara los resultados obtenidos de la implementación del control de errores con los resultados esperados, y dará el detalle en caso de error. Este detalle se da en notación binaria, para poder hacer las comparaciones en forma fácil y detectar los errores. En la sección 2 se dan más detalles del uso del programa que resulta de linkear este object file con el archivo `tareal.c`.
- `checksum.h`: interfaz que deberá implementar su programa. También define la estructura `struct t_cc` que deberán retornar las funciones que se implementen.
- `tareal.c`: incluye una implementación de las funciones que deberá proveer, pero que no calculan nada. Este es el único archivo que se deberá modificar y finalmente entregar.
- `Makefile`: descripción de cómo se debe compilar para obtener un ejecutable útil. Para usar, simplemente basta con ejecutar el comando `make`.

Lo que el alumno debe proveer finalmente es un archivo `tareal.c` que implemente las tres funciones que se definen en el archivo `checksum.h`.

Las tres funciones a implementar tienen la misma forma: reciben parámetros iguales y entregan un puntero a una estructura del mismo tipo en los tres casos. Cada función es responsable de pedir la memoria necesaria para almacenar la estructura y cada uno de sus campos, pero no libera nunca esa memoria.

La estructura que se retorna es la siguiente:

```
struct t_cc {
    short size;
    unsigned char *buf;
};
```

En el campo `buf` se debe crear una secuencia de caracteres (no tiene por qué estar terminada en `'0'` como es el caso de los strings en C). Esta secuencia estará compuesta por el buffer que se le entrega a la función como parámetro, pero agregando la redundancia del checksum en la forma que corresponda. Esta forma se indica más adelante en la explicación detallada de cada función a implementar. El campo `size` indica la cantidad de bytes que tiene la secuencia `buf` en total (datos + redundancia).

Los algoritmos a implementar son los siguientes:

1.1 Internet Checksum

Este algoritmo ya fue visto en clase y está en los apuntes, por lo cual su inclusión en el esquema de la tarea es bastante directa. Se sugiere comenzar por este método, para así disipar la mayoría de las dudas relacionadas con la compilación y el uso de los archivos provistos. El buffer que se deberá retornar dentro de la estructura `t_cc` se arma a partir del buffer que se recibe como parámetro seguido de los dos bytes de checksum.

1.2 Parity Bit

El buffer de entrada consta de caracteres con valor entre 0 y 127. Esto significa que no se ocupa el bit más significativo. Si en la secuencia existe algún valor $v > 127$, se deberá ignorar el bit más significativo (se toma como si el valor fuera $v - 128$). Por cada byte se deberá calcular un bit de paridad que vale 0 si la cantidad de bits en uno del byte es par y vale 1 si la cantidad de bits en uno es impar (*even parity*). En el resultado, todos los bits originales se corren una posición hacia la izquierda (con esto se pierde el bit más significativo), y se agrega el bit de paridad en la posición menos significativa. Ejemplo: 01111110 en el buffer de entrada pasa a ser 11111100 en la salida, 01111101 pasa a ser 11111011.

1.3 Paridad 2D

La entrada consta de grupos de 7 bytes. La matriz para calcular las paridades es de 8×7 (8 filas y 7 columnas) sin incluir el checksum. Al incluir el checksum por filas y por columnas, se llega a una matriz de resultado de 9×8 .

Cada fila contiene 7 bits de información del buffer de entrada y un bit de paridad. Por esto, los bits (usando notación C, contando desde 0 a n-1) del 0 al 6 van en la primera fila, del 7 al 13 en la segunda, y así sucesivamente. Nótese que cada fila corresponde a 8 bits (o sea, 1 byte), y el octavo bit es el de paridad.

Finalmente corresponde calcular un bit de paridad por cada columna y almacenar los resultados en el último byte (son 8 columnas, así que a cada columna le corresponde un bit con para almacenar la paridad dentro del último byte).

Ejemplo:

La entrada (7 bytes escritos en binario) 10110100 01100110 11001110 01010111 00000100 01101001 00001011 se escribe a la matriz, quedando como:

1011010	0
0011001	1
1011001	0
1100101	0
0111000	1
0010001	0
1010010	1
0001011	1
0001111	0

donde la última fila y la última columna es información de paridad. El bit compartido por la última fila y la última columna se calcula como paridad por columna. La salida queda entonces: 10110100 00110011 10110010 11001010 01110001 00100010 10100101 00010111 00011110.

2 Evaluación y Uso

La implementación del bit de paridad y del internet checksum tienen un puntaje de 1.5 cada una. La implementación de la paridad 2D vale 3 puntos. La nota será proporcional a la cantidad de pruebas correctas al ejecutar el programa resultante, y se indica al final de la ejecución del programa. Si se salta alguno de los algoritmos, se asume para fines del cálculo de la nota que falló todas las pruebas.

El ejecutable que se genera admite tres argumentos opcionales, que determinan qué pruebas se realizarán. Si se usa el argumento '-1', se saltan las pruebas de bit de paridad. Con el argumento '-2' se saltan las pruebas de Internet Checksum y finalmente con '-3' se saltan las pruebas de paridad 2D. Para cada algoritmo, se genera un stream aleatorio y de largo aleatorio (pero que cumple los requisitos descritos para cada algoritmo), y se compara el resultado calculado con el esperado. Esto se repite una cantidad (casi) aleatoria de veces.

3 Entrega

La entrega de la tarea se hará por mail a `cc51c@dcc.uchile.cl`, con Subject "Entrega Tarea 1 CC51C", y atachando (usando MIME) solamente 1 archivo (`tarea1.c`), que implemente las tres funciones necesarias.

4 Recomendaciones

- no usar funciones de `<strings.h>` (`strcpy`, `strncpy`, `strcmp`, etc), excepto `bcopy` de ser necesario, y sólo comparar byte a byte.
- no usar `printf("... %s ...", buf)`; ni similares, porque `buf` puede contener como dato válido un `'0'`, que es considerado fin de string por estas funciones.