Rúbrica Examen Mejoramiento Programación de Sistemas

2do Término 2018

Nombre: _____ Paraleo: _____

Pregunta 1 (25 puntos)

Escriba la salida del siguiente programa:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  char blocks[5] = {'A','B','C','D','E'};
  int bloques[6] = {10 , 20, 30, 40 ,50, 100};
  long *mis_bloques[5];
  char *ptr1 = &blocks[0];
  int *ptr2 = bloques;
  long **ptr3 = mis_bloques;
  char *temp;

  for(int i=0;i<5;i++){
     *ptr3 = calloc(1,sizeof(long *));
     ++ptr3;
  }

  temp = ptr1 + 2;
  **(--ptr3) = (long) (ptr2 + 2) - (long) &bloques[4];

  printf("%c\n",*(blocks + 3));
  printf("%ld\n",*mis_bloques[2]);
  printf("%ld\n",*mis_bloques[4]);
  printf("%c\n",*temp);
  printf("%ld\n",(ptr1 + 4) - &blocks[2]);

  return 0;
}
```

La salida de cada printf vale 5 puntos:

D

0

-8

C

2

Pregunta 2 (25 puntos)

Considere el siguiente código, ejecutado en una máquina de 32 bits, little endian. Asuma que las variables w, z, q, m, a, b y c se almacenan de manera consecutiva en memoria desde la dirección 0xA0. Muestre la salida del programa (las direcciones y valores deben estar en **hexadecimal, formato 0x##**).

```
void mostrarContendioDireccion(unsigned char *num, size_t size){
	int i = 0;
	for(i = 0; i < size; i++){
		printf("%p: 0x%02x\n", num+i, *(num+i));
	}
}

int main(){
	int w = 200;
	int z = 300;
	int q = w - z;
	int m = z - w;

	char a = 100;
	char b = 159;
	char c = b - a;

	printf("w: \n");
	unsigned char *p = (unsigned char *)&w;
	mostrarContendioDireccion(p, sizeof(w));

	printf("z: \n");
	p = (unsigned char *)&z;
	mostrarContendioDireccion(p, sizeof(z));

	printf("c: \n");
	p = (unsigned char *)&c;
	mostrarContendioDireccion(p, sizeof(c));

	printf("q: \n");
	p = (unsigned char *)&q;
	mostrarContendioDireccion(p, sizeof(q));

	return 0;
}
```

```
w: (+7 puntos, 5 puntos representación, 2 puntos dirección)
0xA0: 0xc8
0xA1: 0x00
0xA2: 0x00
0xA3: 0x00
z: (+7 puntos, 5 puntos representación, 2 puntos dirección)
0xA4: 0x2c
0xA5: 0x01
0xA6: 0x00
0xA7: 0x00
c: (+4 puntos, 1 puntos representación, 3 puntos dirección)
0xB2: 0x3b
q: (+7 puntos, 5 puntos representación, 2 puntos dirección)
0xA8: 0x9c
0xA9: 0xff
0xAa: 0xff
0xAb: 0xff
```

Pregunta 3 (25 puntos)

Ud ha desarrollado un nuevo programa, generado a partir de varios archivos de código fuente. Los archivos de código objeto generados tienen un tamaño de 1.5 MB. Ud decide enlazar su programa estáticamente con las librerías libc.a (8 MB), libpga.a (2.5 MB) y libpgb.a (5 MB). Responda las siguientes preguntas:

1. ¿Cuál es el tamaño aproximado del ejecutable?

Tamaño: 1.5MB + 8MB + 2.5MB + 5MB = 17MB **(+5 puntos)**

2. Ud. decide poner a ejecutar 20 instancias de su programa. ¿Cuánta memoria ocupan aproximadamente las 20 instancias del programa? Explique detalladamente. Asuma que el tamaño del heap y stack es despreciable comparado con el tamaño del ejecutable.

Cada instancia ocupa aproximadamente 17MB de memoria, por lo tanto 20 instancias ejecutándose de manera simultánea ocuparían 17*20 = 340MB. **(+5 puntos)**

3. ¿Hay alguna forma de reducir el uso de memoria del ejecutable? Sí así lo cree, explique cómo.

Usando linking dinámico las librerías libc, libpga y libpgb no formarían parte del ejecutable y serían compartidas en memoria por todas las instancias del ejecutable. **(+5 puntos)**

- *Menciona linking dinámico (+3)*
- *Explica porque linking dinámico ahorraría memoria (+2)*

4. Ud. se da cuenta que la librería libpgb.a tiene un bug. Ud necesita usar la versión más actualizada de la librería. ¿Qué necesita hacer con su programa? ¿Por qué? Si necesita hacer algo ¿Hay alguna forma más fácil de hacerlo? Explique.

Debido a que se usa linking estático sería necesario volver a construir y probablemente compilar el programa usando la nueva versión de libpgb.a. **(+5 puntos)**

- *Menciona volver a compilar o construir (+2) con versión nueva de libpgb (+1)*

- *Conecta volver a compilar/construir con linking estático (+2)*

Si usamos linking dinámico, en la mayoría de los casos es suficiente actualizar la librería en el sistema operativo sin necesidad de reconstruir o recompilar los programas que usan la librería. **(+5 puntos)**

- *Menciona linking dinámico (+1) y por lo tanto no es necesario compilar/construir (+2)*
- *Menciona actualizar la librería en el sistema (+2)*

Pregunta 4 (25 puntos)

En una aplicación cliente - servidor de transferencia de archivos, la siguiente función envía el contenido de un archivo con tamaño ***size*** almacenado previamente en el buffer ***archivo*** en el lado del servidor. La función retorna *false* al más mínimo error y *true* si el archivo fue enviado con éxito.

```
bool enviar_archivo(int connfd, char *archivo, size_t size)
{
   char ack[MAXCMDSIZE];
   ssize_t s = read(connfd, ack, 6);
   if(s == 6){
      if(strcmp(ack, "READY") != 0)
         return false;
   }else
      return false;

   s = write(connfd, &size, sizeof(size_t));
   if(s < sizeof(size_t))
      return false;

   s = write(connfd, archivo, size);
   if(s < size)
      return false;

   return true;
}
```

Escriba la correspondiente función *recibir_archivo* en el lado del cliente la cual debe de leer, usando un socket previamente conectado, el archivo enviado por *enviar_archivo*. La función debe retornar el número de bytes del archivo leído y almacenarlo en el buffer de salida ***archivo***. La función debe retornar 0 ante el más mínimo error. Es necesario también que la función reserve memoria para ***archivo***, este es el prototipo de la función:

```
size_t recibir_archivo(int connfd, char **archivo)
{
```

```
        size_t size;
        ssize_t s = write(connfd, "READY", 6);
        if(s < 6)
                return 0;

        s = read(connfd, &size, sizeof(size_t));
        if(s < sizeof(size_t))
                return 0;

        *archivo = malloc(size);
        if(*archivo == NULL)
                return 0;
        s = read(connfd, *archivo, size);
        if(s < size)
                return 0;

        return size;
}
```

- Envío de “READY” (**+5 puntos**)
  - Envío al principio antes que lecturas (+3 puntos)
  - Uso correcto de write o send (+2 puntos)
- Lectura del tamaño (**+5 puntos**)
  - Lectura inmediatamente después de envío de “READY” (+2 puntos)
  - Almacenamiento correcto en variable tipo size_t usando read o recv (+3 puntos)
- Reserva de memoria para archivo (**+5 puntos**)
  - Uso correcto de doble puntero (+3 puntos)
  - Uso correcto de malloc o calloc (+2 puntos)
- Lectura de archivo (**+5 puntos**)
  - Lectura al final y después del malloc (+2 puntos)
  - Uso correcto de doble puntero (+1 punto)
  - Uso correcto de read o recv (+2 puntos)
- Retorna tamaño archivo (**+2 puntos**)
- Chequeo de errores (**+3 puntos**)
  - En el ejemplo hay 4 chequeos de error pero se adjudica un punto por cada chequeo hasta el tercer chequeo.