Examen Mejoramiento Programación de Sistemas

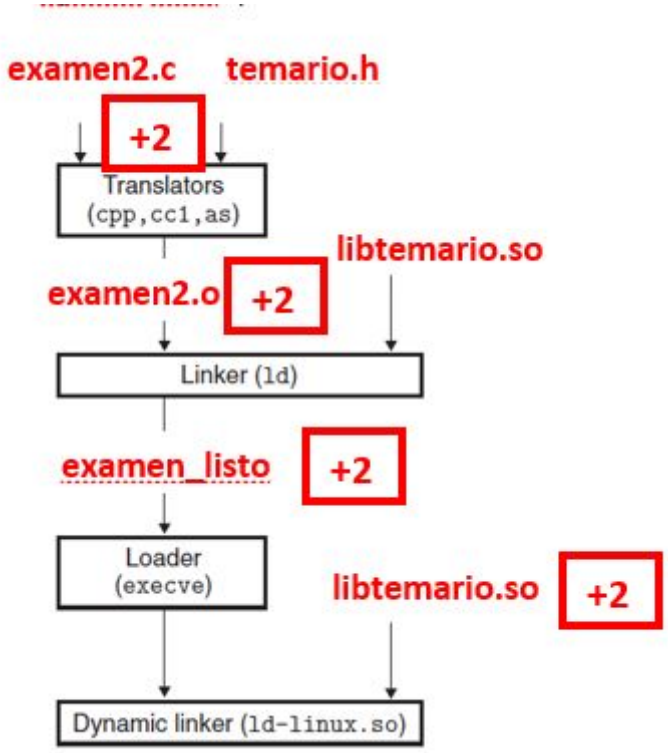
1er Término 2019

Nombre: _____ Paraleo: _____

Pregunta 1 (20 puntos)

Dado un programa escrito en lenguaje C, llamado "`examen2.c`" y que requiere una librería llamada "`libtemario.so`" con su respectiva cabecera "`temario.h`".

1. Complete el siguiente diagrama para obtener el ejecutable "`examen_listo`", indicando las entradas y salidas de cada una de las fases mostradas. (8 puntos)



2. Explique, ¿por qué es requerida la fase del "Dynamic linker" y cuál es el resultado de la misma? (7 puntos)

La fase del dynamic linker es requerida para acceder y usar el código ejecutable en libtermario.so en tiempo de ejecución. La librería libtemario.so es dinámica y por lo tanto su contenido no está incluido en el ejecutable *examen_listo*. + 7

3. Indique el comando completo de compilación usando gcc. (5 puntos)

gcc -o examen_listo examen2.c -ltemario +5

Pregunta 2 (25 puntos)

1. Considere la siguiente sentencia:

```
int var = -19;
```

Muestre el contenido de la memoria de esta variable (dirección y contenido, en hexadecimal). Asuma que la máquina es little-endian, de 32 bits. La dirección de la variable es 0x2000. (10 puntos)

0x2000	**0x2001**	**0x2002**	**0x2003**
0xED	**0xFF**	**0xFF**	**0xFF**

2. Ahora considera la siguiente sentencia:

```
unsigned char p = 300;
```

¿Hay algún problema con esta sentencia? Muestre cómo se guarda este variable en memoria (contenido y dirección en hexadecimal). Asuma la misma máquina, y la variable desde la dirección 0x1000. (7 puntos)

Hay overflow. Valor máximo de unsigned char es 255 (0xFF). 300 es 0x012C

0x1000

0x2C

3. Implemente la siguiente rutina, que reemplaza el byte **i** de la variable **val** con el byte **byte.** El tamaño de la variable **val** está dado por **tam.** (8 puntos)

```
void reemplazar_byte(void *val, int tam, int i, unsigned char
                                                    byte)
```

```
void reemplazar_byte(void *val, int tam, int i, unsigned char byte){
   if(i >= 0 && i < tam - 1){
      unsigned char *p = (unsigned char *)val;
      p[i] = byte;
   }
}
```

Pregunta 3 (25 puntos)

Considere el siguiente código multihilo de un servidor:

```
#define QUEUE_SIZE 5
void *fn(void *arg){
        int fd = (int)arg;
        char buf[100] = {0};
        int res = read(fd, buf,100);
        if(strcmp("GET", buf) == 0){
                memset(buf, 0, 100);
                snprintf(buf,100,"%d", fd);
                write(fd, buf, strlen(buf));
        }
        else write(fd, "ERROR",5);
        close(fd)
}

int main(){
        struct sockaddr_in direccion_servidor;
        direccion_servidor.sin_family = AF_INET;
        direccion_servidor.sin_port = htons(8779);
        direccion_servidor.sin_addr.s_addr = inet_addr("186.14.249.22") ;

        int fd;
        int err = 0;

        if((fd = socket(direccion_servidor.sin_family, SOCK_STREAM, 0)) < 0){
                //... manejar error
        }

        if(bind(fd, (struct sockaddr *)&direccion_servidor,
                sizeof(direccion_servidor)) < 0){
                //... manejar error
        }
        if(listen(fd, QUEUE_SIZE) < 0){
                //... manejar error
        }

        while(1){
            pthread_t id;
            int connfd = accept(fd, NULL, 0);
            pthread_create(&id, NULL, fn, (void*)connfd);
            pthread_detach(id);
         }
}
```

Al correr este código, el programa eventualmente no puede aceptar más conexiones. ¿A qué se puede deber esto? ¿Tiene algún otro problema el código? Indíque los problemas y corríjalos.

1. **No se cierra socket al finalizar hilo** **+10**
2. **No se envía el descriptor de socket apropiado** **+10**
3. **El tamaño del backlog es muy pequeño.** **+5**

Pregunta 4 (30 puntos)

En el código a continuación la estructura *client_connections* representa una lista enlazada de clientes conectados a un servidor en una aplicación cliente - servidor. En esta lista *connfd* representa el descriptor del socket del cliente y *user* el nombre del cliente.

```
typedef struct client_connections {
  int connfd;
  char user[80];
  struct client_connections *next;
}clients_t;

//Inserta un cliente en la lista
void new_client(clients_t **head,
  clients_t *new_conn)
{
  new_conn->next = *head;
  *head = new_conn;
}

void remove_client(clients_t **head, int connfd)
{
//Busca al cliente con descriptor "connfd",
//lo remueve de la lista y libera la memoria usada
}

void printClients(clients_t *head)
{
  clients_t *temp = head;
  while(temp){
    printf("%d ", temp->connfd);
    printf("%s\n", temp->user);
    temp = temp->next;
  }
}
```

```
//Simulación uso de la lista
int main(int argc, char **argv)
{
  clients_t *connected_clients_list = NULL;

  int arr[] = {22, 17, 2, 9, 11, 5, 13};
  int n = sizeof(arr)/sizeof(arr[0]);
  char usuario[] = "usuarioX";

  for(int i = 0; i < n; i++){
    clients_t *nueva =
      (clients_t *) malloc(sizeof(clients_t));
    nueva->connfd = arr[i];
    sprintf(usuario, "usuario%d", i);
    memcpy(nueva->user, usuario, strlen(usuario));
    new_client(&connected_clients_list, nueva);
  }

  remove_client(&connected_clients_list, 11);
  printClients(connected_clients_list);
}
Salida:
```

1. ¿Cuál es la salida del programa? (10 puntos)

Muestra el orden correcto +5

Muestra correctamente usuarioX +2

Omite 11 usuario4 +3

13 usuario6
5 usuario5
9 usuario3
2 usuario2
17 usuario1
22 usuario0

2. Implemente la función *remove_client* (20 puntos)

```
void remove_client(clients_t **head, int connfd)
{

  clients_t *current = *head; //Maneja correctamente doble puntero +5
  clients_t *ant = NULL;
  while(current != NULL){
    if(current->connfd == connfd){   //Compara correctamente +2
          if(ant == NULL)
            *head = current->next;      //Verifica caso límite +3
          else
            ant->next = current->next; //Borra correctamente +5
          free(current);     //Libera correctamente el nodo +5
          return;
    }
    ant = current;
    current = current->next;
  }
}
```

puntero current	+2
recorre current->siguiente	+2
Valida current->connfd == connfd	+1
Valida si current es cabecera de lista	+5
actualiza saca elemento y actualiza cabecera de lista	+5
Si no es cabecera, saca elemento de lista y actualiza punteros	+3
libera bloque de memoria de elemento removido	+2