Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação de Sistemas Computacionais

Teste Global da Época de Recurso, Inverno de 2019/2020

Nas questões em que não se indiquem explicitamente outras condições, considere as características do ambiente de referência usado na unidade curricular neste semestre.

1. [2,5] Implemente em linguagem C, a função `ror128` que aplica uma rotação para a direita de `n` *bits* sobre um valor de 128 *bits*. Os ponteiros `phi` e `plo` apontam, respectivamente, para as partes alta e baixa do valor a rodar. Por exemplo, se `hi = 0x1122334455667788` e `lo = 0x9900AABBCCDDEEFF`, a invocação `ror(&hi, &lo, 16)` deixa `hi == 0xEEFF112233445566` e `lo == 0x77889900AABBCCDD`. Admita que `n` não excede 127.

   ```
   void ror128(unsigned long *phi, unsigned long *plo, size_t n);
   ```

2. [2,5] Implemente em linguagem C, a função `reverse` que inverte a sequência de caracteres compreendida entre a posição `left` e a posição `right` (inclusive), na *string* recebida em `str`. Se os parâmetros `left` ou `right` indicarem uma posição para além do fim da *string*, esses parâmetros devem assumir a última posição da *string*.

   ```
   void reverse(char *str, int left, int right);
   ```

3. [2,5] Implemente em *assembly* x86-64, a função `add_detail`, cuja definição em linguagem C se apresenta a seguir. Valorizam-se implementações que não recorram a instruções explícitas de multiplicação (`mul`, `imul`, …).

   ```
   typedef struct { unsigned int shelf; short min; short max; } Stats;

   typedef struct { long id; Stats **global; Stats *details; unsigned int ndetails; } Info;

   int add_detail(Info * info, Stats * detail) {
     int res = 0;
     Stats *pglobal = info->global[detail->shelf];
     info->details[info->ndetails++] = *detail;
     if (detail->min < pglobal->min) { res |= 1; pglobal->min = detail->min; }
     if (detail->max > pglobal->max) { res |= 2; pglobal->max = detail->max; }
     return res;
   }
   ```

4. [4] Considere a seguinte implementação da função `bsearch`, definida no âmbito da biblioteca normalizada da linguagem C.

   ```
   void *bsearch(const void *key, const void *base, size_t nelem, size_t size,
                 int (*fcmp)(const void *, const void *)) {

      while (nelem > 0) {
           const size_t pivot = nelem / 2;
           const char *q = (const char *)base + size * pivot;
           const int val = (*fcmp)(key, q);
           if (val == 0)
                 return ((void *) q);
           else if (val < 0)
                 nelem = pivot;
           else {
                 base = q + size;
                 nelem -= pivot + 1;
           }
      }
      return NULL;
   }

   typedef struct {char *registration; char *brand; char *color; } Car;
   ```

   a. [2,5] Implemente a função **`bsearch`** em *assembly* x86-64.

b. [1,5] Escreva, em linguagem C, um programa de utilização da função `bsearch` que localiza num *array* de ponteiros para instâncias do tipo `Car`, ordenado por ordem crescente da chapa de matrícula (`registration`), a informação relativa a um dado automóvel. No programa, deve explicitar a definição das estruturas de dados necessárias, a definição da função de comparação, assim como o código para mostrar na consola a informação sobre o automóvel em causa.

5. [3,0] Considere o conteúdo do ficheiro fonte `f1.c` e a tabela de símbolos resultante da compilação de `f2.c`.

```
/* f1.c */
#include <stdio.h>

float complex[2] = { 1.2, 3.4 };
extern int x;

int oper(int m, int n);
int func(int a) { return a + x; }

int main() {
     printf("%d\n", oper(3, 4));
     return 0;
}
```

```
/* f2.c */
#define N 100

typedef struct { int re; int im; } complex;

complex x = { N, N * 2 };

int func(int x, int y);

int oper(int a) {
       return func(a, 0) + 8;
}
```

a. [2] Indique o conteúdo das tabelas de símbolos dos ficheiros objecto relocáveis resultantes da compilação de `f1.c` e de `f2.c`. Para cada símbolo indique o nome, a secção e o respectivo âmbito (local ou global) (pode usar a mesma notação do utilitário `nm`).

b. [1] Se considerar que os ficheiros objecto f1.o e f2.o podem ser combinados pelo *linker* com sucesso para gerar um executável, diga o que é mostrado na consola quando se executa o executável produzido. Se, pelo contrário, considera que o *linker* falhará a combinação, diga qual a razão ou razões porque isso acontece.

6. [1] Considere um programa que manipula um *array* de valores do tipo *float*, com 1024 elementos. Se este programa executar num sistema com uma *cache* de 32KiByte de capacidade, 64 bytes de bloco e organização 16-*way*, indique a posição do *array* em que o sistema repete a utilização do mesmo *set* da *cache* que utiliza no acesso à primeira posição. Justifique.

7. [4,5] Observe as definições apresentadas abaixo, correspondentes a duas variantes de armazenamento de dados. Numa delas (`vals`) temos um *array* de ponteiros para estruturas que contêm um *array* de dados; na outra (`refs`) temos um *array* de estruturas em que cada uma aponta para um *array* de dados. As funções `refs_from_vals` e `vals_from_refs` convertem entre os dois formatos, com todos os ponteiros a referirem memória alocada dinamicamente. Das conversões não resulta qualquer partilha de dados, uma vez que todos os blocos de dados são copiados para outro espaço de memória durante a conversão. As funções `cleanup_vals` e `cleanup_refs` libertam todo o espaço alocado dinamicamente para suportar as respectivas estruturas de dados.

```
#define MAX_LEN 128
typedef struct data_val { size_t len; char data[MAX_LEN]; } DataVal;
typedef struct data_ref { size_t len; char * data; } DataRef;

DataRef  *refs_from_vals(DataVal **vals, size_t nvals);
DataVal **vals_from_refs(DataRef  *refs, size_t nrefs);

void cleanup_vals(DataVal **vals, size_t nvals);
void cleanup_refs(DataRef  *refs, size_t nrefs);
```

Implemente em linguagem C as funções:

a. [1,5] `refs_from_vals`;

b. [1,5] `vals_from_refs`;

c. [1,5] `cleanup_vals` e `cleanup_refs`.

Duração: 2 horas e 30 minutos
ISEL, 5 de Fevereiro de 2020