Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação de Sistemas Computacionais

Inverno de 2021/2022

Série de Exercícios 2

Nos exercícios seguintes é proposta a escrita de funções em *assembly* para a arquitetura x86-64, usando a variante de sintaxe AT&T, e seguindo os princípios básicos de geração de código do compilador de C da GNU. A resolução de cada exercício (código *assembly*) deve ser apresentado em conjunto com o respectivo programa de teste escrito em linguagem C. Tenha em consideração que os exercícios que não forem demonstrados a funcionar serão considerados como não tendo sido realizados. Não se esqueça de testar devidamente o código desenvolvido, **invocando as funções escritas em *assembly* com parametrizações diferentes**, bem como de o apresentar de forma cuidada, apropriadamente indentado e comentado. Não é necessário relatório. Contacte o docente se tiver dúvidas. Encoraja-se também a discussão de problemas e soluções com colegas, mas salienta-se que a partilha direta de soluções leva, no mínimo, à anulação das entregas de todos os estudantes envolvidos.

1. Escreva em *assembly* a função **`rotate_right`** que desloca para a direita (no sentido de maior peso para o de menor peso) o valor a 128 *bit*, que recebe no parâmetro **`value`**, o número de posições indicadas no parâmetro **`n`**. O valor numérico de 128 *bit* é formado pela concatenação de dois valores a 64 *bit* armazenados num *array* com duas posições, segundo o formato *little-endian*. Os *bits* que saem da posição de menor peso entram, pela mesma ordem, na posição de maior peso.

   ```
   void rotate_right(unsigned long value[], size_t n);
   ```

2. Escreva em *assembly* a função **`my_strlen`** segundo a definição de *strlen* tal como está definida na biblioteca *standard* da linguagem C. Esta função determina a dimensão da *string* referenciada pelo parâmetro **`str`**. Na programação, procure um bom desempenho em tempo de execução. Por exemplo, minimizar o número de acessos à memória realizando, sempre que possível, leituras de palavras de 64 *bits* em endereços alinhados (i.e., endereços múltiplos de 8) e minimizando o número de iterações, mesmo que para isso tenha que aumentar a dimensão de memória ocupada por código.

   ```
   size_t my_strlen(const char *str);
   ```

3. A seguinte definição de tipos permite criar uma estrutura de dados para agendamento de compromissos. **`struct appointment`** permite registar um compromisso com descrição e horário; **`struct appointments`** permite registar os compromissos de um dia, na forma de um *array* de compromissos individuais; **`struct agenda`** permite registar o agendamento de um mês inteiro. A função **`get_appointment`** recebe como primeiro argumento um array de agendamentos mensais e devolve o ponteiro para o compromisso de ordem **`index`** do dia definido pelos argumentos **`month`** e **`day`**.

   Reescreva a função **`get_appointment`** em linguagem *assembly* x86-64.

   ```
   typedef struct appointment {
       char *description;
       char *schedule;
   } Appointment;

   typedef struct appointments {
       int length;
       Appointment *events;
   } Appointments;

   typedef struct agenda {
       char *name;
       Appointments *calendar[31];
   } Agenda;

   Appointment *get_appointment(Agenda agenda[], size_t month, size_t day, size_t index) {
       return agenda[month].calendar[day] != NULL
                   && index < agenda[month].calendar[day]->length
              ? &(agenda[month].calendar[day]->events[index])
              : NULL;
   ```

```
}
```

No programa de teste, escrito em C, defina uma estrutura de dados estática que suporte o agendamento de um ano. Por razões práticas preencha os dados apenas para dois meses, em cada mês um dia e em cada dia dois compromissos.

4. Traduza para *assembly* x86-64, a função **`selective_copy`**, que recebe em **`src`** a referência para um *array* de objetos com **`src_size`** elementos, aplica a função referida por **`predicate`** a cada um desses objetos e, se retornar verdadeiro, copia o objeto para o array referido por **`dst`**. O *array* **`dst`** tem a dimensão indicada por **`dst_size`**. A função retorna verdadeiro se o *array* **`dst`** tiver dimensão suficiente para copiar todos os elementos selecionados. Em ambos os casos a variável apontada por **`elem_found`** é atualizada com o número de elementos selecionados.

```
int selective_copy(void *dst, size_t dst_size,
                   void *src, size_t src_size,
                   size_t elem_size,
                   int (*predicate)(void *ptr, void *ctx), void *context,
                   size_t *elem_found) {
    size_t dst_index = 0;
    for (size_t src_index = 0; src_index < src_size - 1; ++src_index)
          if (predicate(src + src_index * elem_size, context)) {
                if (dst_index < dst_size)
                      memcpy(dst + dst_index * elem_size,
                            src + src_index * elem_size, elem_size);
                 ++dst_index;
          }
    *elem_found = dst_index;
    return dst_index <= dst_size;
}
```

Recomenda-se que realize o teste da função em duas situações distintas. Por exemplo: para um conjunto de nomes de pessoas duplicar os nomes que terminam por “Silva”; para um conjunto de registos de dados pessoais, duplicar os registos que correspondam a pessoas com mais de 65 anos.

Data recomendada para conclusão: 11 de Dezembro de 2021

ISEL, 8 de Novembro de 2021