Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Inverno de 2017/2018

Série de Exercícios 2 - Individual

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. O teste das funções deve ser realizado no contexto de um programa escrito em linguagem C. Não se esqueça de testar devidamente o código desenvolvido, 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 de outros grupos, mas salienta-se que a partilha directa de soluções leva, no mínimo, à anulação das entregas de todos os envolvidos.

1. Escreva em *assembly* a função **`rotate_left`** que roda para a esquerda o valor a 128 *bit*, que recebe no argumento **`value`**, o número de posições indicadas no argumento **`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*.

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

2. Escreva em *assembly* a função **`memset`** tal como está definida na biblioteca normalizada da linguagem C. Esta função preenche **`len`** *bytes* de memória a partir do endereço definido por **`ptr`** com o valor **`value`**. Procure minimizar o número de acesso à memória efetuando acessos alinhados a palavras com múltiplos *bytes*.

   ```
   void *memset(void *ptr, int value, size_t len);
   ```

3. Considere a estrutura **`Worker`** usada para o registo dos períodos de trabalho de um funcionário. A estrutura inclui os dados de identificação do funcionário e uma colecção de instâncias do tipo **`CheckPoint`** onde cada um representa a hora de entrada e de saída do serviço num dia. Os tempos são definidos com um valor numérico que representa o número de segundos desde uma data de referência (e.g., 1 de Janeiro de 1970). Escreva em *assembly* a função **`get_total_work_time_after`** que retorna, para o funcionário especificado com o argumento **`worker`**, o total do tempo de serviço realizado em data posterior à definida com o argumento **`start_time`**.

   ```
   typedef struct personal_info {
     char genre;
     int age;
     const char *name;
   } PersonalInfo;

   typedef struct check_point {
     long entry_time;
     long exit_time;
   } CheckPoint;

   typedef struct worker {
     PersonalInfo identity;
     size_t check_points_len;
     CheckPoint check_points[];
   } Worker;

   long get_total_work_time_after(Worker *worker, long start_time);
   ```

4. Apresenta-se abaixo uma implementação do algoritmo *quicksort* segundo a definição da função **qsort** da biblioteca normalizada da linguagem C. Implemente as funções **quick_sort** e **memswap** em linguagem *assembly*.

```
static void memswap(void *one, void *other, size_t width) {
  char tmp[width];
  memcpy(tmp, one, width);
  memcpy(one, other, width);
  memcpy(other, tmp, width);
}

void quick_sort(void *base, size_t nel, size_t width,
                int (*compar)(const void *, const void *)) {
  char *limit, *last, *ptr;

  if (nel < 2)
    return;
  limit = (char *)base + (nel * width);
  memswap(base, (char *)base + (nel >> 1) * width, width); /* pivot at middle */
  /* do partition */
  for (last = base, ptr = (char *)base + width; ptr < limit; ptr += width)
    if ((*compar)(ptr, base) < 0)
      memswap(ptr, last += width, width);
  memswap(base, last, width);
  quick_sort(base, (last - (char *)base) / width, width, compar);
  quick_sort(last + width, ((limit - last) / width) - 1, width, compar);
}
```

Data limite de entrega: 26 de Novembro de 2017

ISEL, 26 de Outubro de 2017