Universidade Estadual de Maringá
Departamento de Informática
Curso: Informática
Disciplina: 5193 - Programação em Linguagem de Montagem
Professor: Ronaldo Augusto de Lara Gonçalves

Simulador de elevadores em linguagem GNU Assembly para plataforma 32 bits

Equipe:

Gabriel de Oliveira Conejo	RA105483
Maicon Henrique Rodrigues do Nascimento	RA102504
Raphael Franco de Lima	RA98336

Maringá
2019

Sumário

Índice de Figuras

1. Resumo

Este trabalho teve como objetivo desenvolver um software em linguagem GNU Assembly utilizando um conjunto de instruções para sistemas com arquitetura de 32 bits que simulasse um sistema de elevadores compostos por 2 elevadores com algumas funcionalidades reais.

O programa funciona da seguinte forma: Ao executá-lo será pedido o número de andares do prédio, uma probabilidade de acontecerem chamadas externas a cada iteração do elevador, o peso máximo suportado por ele e quantidade máxima de pessoas permitidas. Existe um loop infinito e a cada iteração do loop o elevador processa/executa suas determinadas chamadas externas e internas e escreve na tela as informações dos processamentos. Para o elevador prosseguir com sua execução é necessário apertar a tecla “ENTER” após cada processamento. Isso permite analisar as informações escritas na tela.

2. Divisão de tarefas

Foi descrito um pseudocódigo, sendo este dividido em 3 partes (processamento dentro de cada iteração do loop) e cada aluno se dispôs a fazer qual gostaria.

2.1. RA98336

Responsável pela escrita do pseudocódigo e sua divisão. Além disso, elaborou a rotina contida no rótulo de entrada main para leitura das variáveis e validação de entrada.

Responsável pela parte da verificação das listas (interna e externa) para verificar se existe alguma entrada ou saída de pessoas do elevador. E realizada das chamadas internas após a entrada no elevador.

Ao final das verificações mostra a quantidade de pessoas que estão saindo ou entrando no elevador.

2.2. RA102504

Responsável pela parte do processamento de chamadas externas. A cada iteração do loop o software deveria realizar o sorteio de 1 ou 2 andares e para cada andar do prédio realizar o sorteio de 1 a 3 pessoas que fizeram uma chamada externa para o elevador neste determinado andar.

Para cada andar sorteado (sorteio de 1 ou 2 andares) o programa deveria definir se as chamadas externas deveriam ocorrer ou não, baseada na probabilidade dada no início da execução do programa. Caso fossem processadas, é alocada memória para a struct “Pessoa” que contem 16 Bytes. Os 4 primeiros Bytes são preenchidos com o “andar_alvo” da pessoa. Logo após é gerado um número (4 Bytes) que preenche a “idade” da pessoa esse valor tem um máximo de 100 e um mínimo de 35. O próximo valor de pessoa a ser preenchido é o “peso” da pessoa (4 Bytes), esse valor é gerado com base em um máximo de 180 e um mínimo de 5. Para terminar a geração de pessoa é preenchido a campo “próximo” que representa um ponteiro (endereço de memória de 4 Bytes) para a próxima pessoa em uma lista encadeada.

A geração de pessoa é feita com base em um sorteio de 1 a 3 pessoas que fizeram uma chamada externa.

2.3. RA105483

Responsável pela parte de movimentação do elevador. A cada iteração do loop o programa deve realizar a chamada das funções que verifica se existe chamadas para o elevador no sentido em que ele está indo e caso houver este se movimenta em uma unidade.

Também é verificado se o elevador já se encontra no último andar ou no primeiro andar, desta forma, caso o elevador se encontre nas extremidades, é trocado a direção do elevador e realizado a verificação novamente.

Caso não exista mais chamadas para o elevador no sentido em que ele está indo, é trocado a direção.

3. Funcionamento do código

Para explicar o funcionamento do programa pode ser utilizado o pseudocódigo desenvolvido.

```
// INICIO ESTRUTURA DE DADOS



typedef struct {
  int andar_alvo;
  int idade;
  int peso;
  pessoa* proximo;
} pessoa;
```

// pessoa** significa que é uma matriz dinâmica ou seja, será um lista do tamanho

// da quantidade de andares e cada posição será uma lista das pessoas naquele andar

// seja pra entrar ou sair (lista_interna ou lista_externa)

pessoa** lista_interna_elevador_1

pessoa** lista_interna_elevador_2

pessoa** lista_externa

int peso_maximo_elevador

int quantidade_maxima_pessoas_elevador

// FIM ESTRUTURA DE DADOS

main:

lê probabilidade sorteio, peso máximo elevador, quantidade andares, quantidade máxima de pessoa

loop_infinito:

verifica lista interna na posição do andar atual (de cada elevador) para ver se alguém sai

IMPRIME X pessoas saindo do elevador

// isso é um loop x2

verifica lista externa na posição do andar que cada elevador está para ver se alguém entra

se (qtd_pessoas_elevador < quantidade_maxima_pessoas_elevador)

se ((peso_atual_elevador + pesssoa.peso) < peso_maximo)

realiza chamada interna e remove da lista_externa

senão

pula e continua percorrendo a lista_externa

IMPRIME X pessoas entrando do elevador

// fim loop x2

IMPRIME

total pessoas elevador 1 e peso atual do elevador

total pessoas elevador 2 e peso atual do elevador

realiza sorteio de um ou dois andares

para cada sorteio calcula se ele ocorre ou não

para cada andar sorteia 1 a 3 pessoas (peso e idade)

if (flag_direcao == 'SUBINDO') {

label_goto:

```
    if(tem_chamada_subindo)
      return 'SUBINDO'
    else if(tem_chamada_descendo)
        return 'DESCENDO'
    else return 'PARADO'
  } else if (flag_direcao == 'DESCENDO') {
    if(tem_chamada_descendo)
      return 'DESCENDO';
    else if (tem_chamada_subindo)
      return 'SUBINDO'
    else return 'PARADO'
  } else {
    goto label_goto;
  }
```

IMPRIME subindo, descendo ou parado

DELAY (aguarda o usuário apertar a tecla Enter para executar a próxima iteração)

Primeiramente, para o desenvolvimento do código são necessárias algumas variáveis para auxiliar o controle geral. As mais importantes para o funcionamento são apresentadas abaixo:

```
.section .data

qtd_andares:                    .int 0
qtd_pessoas_elevador_1:         .int 0
qtd_pessoas_elevador_2:         .int 0
```

```
direcao_elevador1:                .int 0 # SUBINDO (0) ou DESCENDO (1)
direcao_elevador2:                .int 0 # SUBINDO (0) ou DESCENDO (1)
andar_atual_elevador_1:           .int 0
andar_atual_elevador_2:           .int 0

.section .bss

.lcomm lista_interna_elevador1, 200 # lista de ponteiros (4 Bytes cada po
sicao)
.lcomm lista_interna_elevador2, 200 # lista de ponteiros (4 Bytes cada po
sicao)
.lcomm lista_externa, 200 # lista de ponteiros (4 Bytes cada posicao)
```

qtd_andares: representa a quantidade de andares que o elevador percorre, tal valor é informado pelo usuário.

qtd_pessoas_elevador_1: representa quantas pessoas estão dentro do elevador 1 naquele momento.

qtd_pessoas_elevador_2: representa quantas pessoas estão dentro do elevador 2 naquele momento.

direcao_elevador_1: representa para qual o elevador 1 está se deslocando no momento. Sendo 0 subindo, 1 descendo e 2 parado.

direcao_elevador_2: representa para qual o elevador 2 está se deslocando no momento. Sendo 0 subindo, 1 descendo e 2 parado.

andar_atual_elevador_1: representa em qual andar o elevador 1 se encontra no momento. O andar 0 representa o térreo.

andar_atual_elevador_2: representa em qual andar o elevador 2 se encontra no momento. O andar 0 representa o térreo.

lista_externa: lista responsável por armazenar quantas chamadas externas existem em determinado andar.

lista_interna_elevador_1: lista responsável por armazenar quantas chamadas internas do elevador 1 existem em determinado andar.

lista_interna_elevador_2: lista responsável por armazenar quantas chamadas internas do elevador 2 existem em determinado andar.

Dentro das interações do loop infinito que encontramos o funcionamento do sistema em geral, primeiramente é verificado na lista interna de ambos elevadores, se existe saída de pessoas do elevador, caso tenha, é imprimido a quantidade de pessoas que estão saindo do elevador. Logo após é verificado a lista interna para verificar se existe alguém entrando no elevador, se existir, é mostrado em tela a quantidade de pessoas entrando no elevador (importante realizar as chamadas nesta ordem). Após a entrada, cada pessoa gera uma chamada interna, a qual é impressa na tela.

Após a execução dessas operações, é mostrado a quantidade de pessoas total no elevador.

Feito isso, é realizado o sorteio de um ou dois andares e também o sorteio de uma a três pessoa. Para cada sorteio, é calculado a probabilidade de ocorrência do mesmo, caso o sorteio ocorra, o andar é sorteado e inserido na lista externa.

Logo após, é realizado a verificação se possui alguma chamada na lista externa ou na lista interna para andares que se encontram na direção em que o elevador está indo.

Caso exista alguma chamada em alguma das duas listas, o elevador se movimenta na respectiva direção. Se a direção do elevador for subindo, o andar do elevador é incrementado, se o elevador estiver descendo, o andar do elevador é decrementado, também é verificado se é possível incrementar ou decrementar a posição do elevador para que ele não passe das extremidades de andares. Neste último caso a direção do elevador é invertida. No caso em que não existam chamadas em ambas as direções o elevador fica parado.

Após a execução de todos os passos, é pedido para que o usuário aperte a tecla “enter” para continuar com a execução do sistema, isto é necessário para que seja possível verificar o que está ocorrendo no sistema e o sistema volta para o início do loop.

Para o desenvolvimento do programa foram utilizadas diversas chamadas a funções, devido a isso foi adotada como prática de desenvolvimento que toda linha de código deveria ser comentada

afim de facilitar o entendimento do código por todos os membros da equipe.

Além disso, foi utilizada a convenção de chamada x86 para funções. Sendo ela caracterizada pela seguinte estrutura:

```
rotulo_da_funcao:
  pushl %ebp
  movl %esp, %ebp
  .
  .
  # corpo do método
  .
  .
  popl %ebp
  ret
```

Tal código utiliza o registrado `%ebp` como um referencial para identificar a posição dos parâmetros passados para a função através da pilha.

Algumas das funções mais importantes para a execução do sistema são descritas abaixo.

caminha_na_lista: recebe como parâmetro uma das listas e um andar. Caminha na lista até aquele andar e incrementa o valor contido nele em uma unidade, representando assim uma nova chamada.

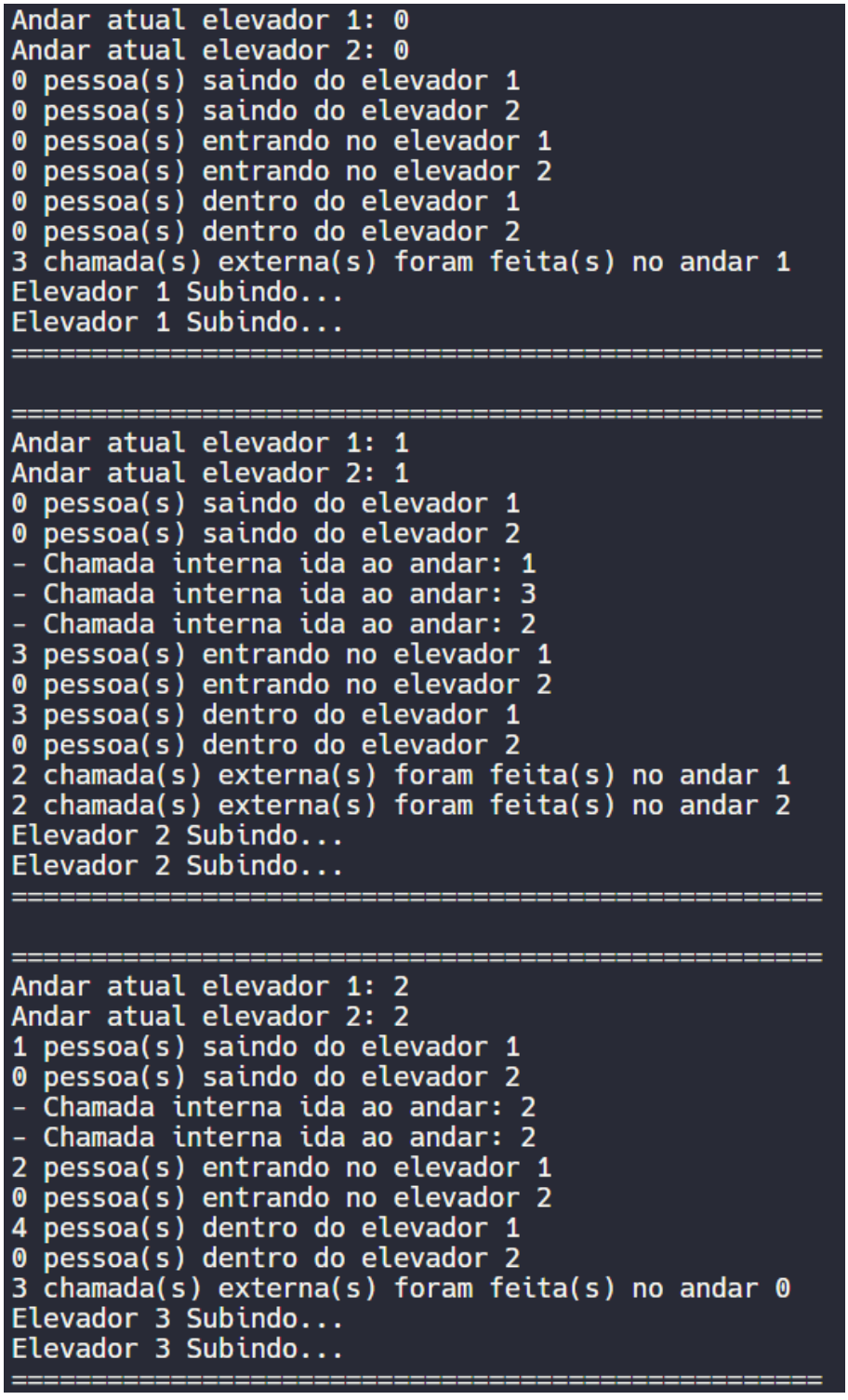
verifica_lista_externa: verifica se no andar atual existe alguém querendo entrar, caso haja, incrementa o número de pessoas no elevador e realiza um loop (baseado no número de pessoas que entraram) de chamadas internas.

verifica_lista_interna: verifica se alguém quer sair do elevador naquele andar, caso haja, atualiza a quantidade de pessoas no elevador e zera as chamadas internas naquele andar.

sorteia_andares: faz o sorteio de um a dois andares, para cada um dos andares, verifica se deve ser processado com base na probabilidade inserida pelo usuário e sorteia de um a três pessoas para fazer as chamadas externas. Modifica lista_externa com base nos sorteios de chamadas externas.

Abaixo seguem algumas imagens do programa em execução.

Figura 1 Programa em execução



Para execução do programa foi utilizado o compilador `gcc`, sendo que a sequência de comandos a serem realizadas a seguinte:

- `gcc -c -m32 elevador.s -o elevador.o`
- `gcc -m32 elevador.o -o elevador`
- `./elevador`

Com o intuito de simplificar a execução foi utilizada a ferramenta de automação `make` a qual possibilita reunir todos os comandos acima em apenas um. Sendo assim, para executar o programa, alternativamente pode-se utilizar o seguinte comando (desde que o arquivo `Makefile` esteja presente e o programa `make` instalado):

- `make run`

4. Instruções diferentes utilizadas

Não houve o uso de instruções fora do conjunto apresentado em sala de aula.

5. Restrições, limitações e problemas

O programa desenvolvido não faz o uso de números reais e nem realiza o processo de liberação de memória dinâmica alocada.

6. Bibliografia

University of Virginia. (25 de Novembro de 2019). *x86 Assembly Guide*. Fonte: Site da University of Virginia: https://www.cs.virginia.edu/~evans/cs216/guides/x86.html