ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

PCS3216 – SISTEMAS DE PROGRAMAÇÃO

Gabriel da Cunha Rodrigues - 8992930

PROJETO - MOTOR DE EVENTOS

Prof. João José Neto

2019

Sumário

[1. Introdução 3](#_Toc11973379)

[2. Objetivo 3](#_Toc11973380)

[3. Metodologia 3](#_Toc11973381)

[4. Especificações do motor de eventos 3](#_Toc11973382)

[4.1 Processador hospedeiro 3](#_Toc11973383)

[4.2 Hardware a simular 4](#_Toc11973384)

[4.3 Memória 5](#_Toc11973385)

[4.4 Registradores 5](#_Toc11973386)

[4.5 Conjunto de instruções 5](#_Toc11973387)

[4.6 Equipamentos 5](#_Toc11973388)

[4.7 Blocos funcionais 5](#_Toc11973389)

[4.8 Montador 5](#_Toc11973390)

[5. Entradas do simulador 5](#_Toc11973391)

[6. Testes 5](#_Toc11973392)

# 1. Introdução

Este relatório consiste na documentação do projeto semestral desenvolvido para a disciplina de Sistemas de Programação (PCS3216), ministrada pelo Professor João José Neto no ano de 2019.

# 2. Objetivo

O objetivo deste projeto é criar um motor de eventos para simulações determinísticas de sistemas reativos, definidos como conjunto de regras. Sua aplicação deve consistir na simulação de um subconjunto simples de instruções em um hospedeiro a ser definido. Os códigos das instruções, que são as entradas do simulador de eventos, deverão estar armazenados na memória do processador simulado, podendo ter diferentes ordens de ocorrência de uma simulação para outra.

# 3. Metodologia

O motor de eventos foi desenvolvido em uma máquina virtual usando a linguagem Python 3.7, uma linguagem de alto nível para acelerar o processo de desenvolvimento e permitir um código mais legível e intuitivo. Além disso, fez-se o uso das bibliotecas Python Numpy, Pandas e Tabulate para ajudar na manipulação e visualização de tabelas de dados, tornando o código facilmente escalável com a adição de novas instruções e programas predefinidos de simulação.

# 4. Especificações do motor de eventos

## 4.1 Processador hospedeiro

O processador hospedeiro é uma máquina real, utilizando-se para a execução desse simulador um processador Intel® Core i5™-7200U, com unidade de processamento computacional de 2.5GHz de velocidade de clock, operando em um sistema Windows 10.

Para a codificação do projeto, assim como para a execução dos testes, utilizou-se o software Visual Studio Code, da Microsoft.

O simulador do motor de eventos inicia fornecendo a lista de programas inclusos para teste, solicitando do usuário a seleção do programa desejado para execução, como mostra a Figura 1.

Esse trecho de código está disponível na função *main* do arquivo motorEventos.py, arquivo principal do projeto.

## 4.2 Hardware a simular

Para simplificar o desenvolvimento do projeto, optou-se pela execução de um *loader* absoluto, preenchendo a memória simulada com o código a ser simulado a partir de um arquivo *.txt* que o contém, ao selecionar o programa desejado. O *loader* está escrito no arquivo de texto *loader.txt*, utilizando o conjunto de instruções definidas no item 4.5, sendo lido e executado através da função *loader()* do arquivo *motorEventos.py.*

A codificação do arquivo *loader* está em ASCII hexadecimal, tornando o código mais simplificado do que com o uso de códigos binários diretamente. O programa *loader* possui as sub-rotinas START, DATA e EXE. A subrotina START obtém do arquivo carregado pelo *loader* os endereços e bytes a serem armazenados na memória. A subrotina DATA analisa os dados do arquivo, armazenando um byte no endereço de carga predefinido e decrementando o número de bytes restantes no carregamento a cada iteração. Enquanto houver bytes para serem carregados, a subrotina DATA é executada novamente. Ao esgotar o número de bytes do arquivo a ser carregado, esse número se anula, dando início à subrotina EXE, que recebe o endereço de execução inicial do programa e modifica o contador de instruções para essa posição, dando início à execução do programa que está sendo carregado.

Da mesma forma, utilizou-se a simplificação de um *dumper* absoluto, que grava o conteúdo da memória simulada em um arquivo *dump.txt* após a simulação do código, que pode ser carregado pelo *loader* em uma próxima execução do simulador. Assim como o *loader,* o *dumper* está escrito no arquivo *dumper.txt*, em codificação ASCII hexadecimal. Ele possui três subrodinas: START, DUMP e END. A primeira subrotina acontece de forma semelhante à subrotina de mesmo nome do programa *loader*, obtendo um endereço e um número de *bytes*. A subrotina DUMP recebe o dado do endereço de carga e o armazena no arquivo de texto *dump.txt*, decrementando o número de bytes a serem armazenados. Da mesma forma, a mesma rotina é repetida até não haver mais bytes para serem armazenados, pois a partir do momento que esse número chega a zero, dá-se início à subrotina END. Nessa última subrotina, obtém-se o endereço de execução inicial do programa, armazenando-o no arquivo de saída *dump.txt*, que já fica apto para ser carregado pelo *loader* em uma execução futura.

## 4.3 Memória

Para a implantação da memória simulada do motor de eventos, utilizou-se uma única matriz global, acessada por todos os programas. Como facilidade de construção da linguagem adotada, o Python não exige que suas matrizes sejam alocadas previamente, podendo aumentar de acordo com a necessidade. Entretanto, para estruturar o projeto de forma correta, definiu-se uma memória de 16 bits por palavra, ou seja, cada palavra possuindo 2 bytes. O endereçamento é dado por 4 dígitos hexadecimais, que constituem 16 bits, ou 2 bytes de endereçamento.

É importante notar que há uma alocação fixa do espaço de memória ocupada pelo programa *loader*, definida pela função loader() do programa *motorEventos.py.*

## 4.4 Registradores

## 4.5 Conjunto de instruções

Todas as instruções possuem um formato de 4 bytes, sendo os dois primeiros bytes correspondentes ao código da operação (*opcode*)



## 4.6 Equipamentos

## 4.7 Blocos funcionais

## 4.8 Montador

# 5. Entradas do simulador

# 6. Testes