Nome: Daniel Kenji Saito

Relatório – Case da Cromai

1. MODELO CONCEITUAL

1.1 Diagrama do código em Shell

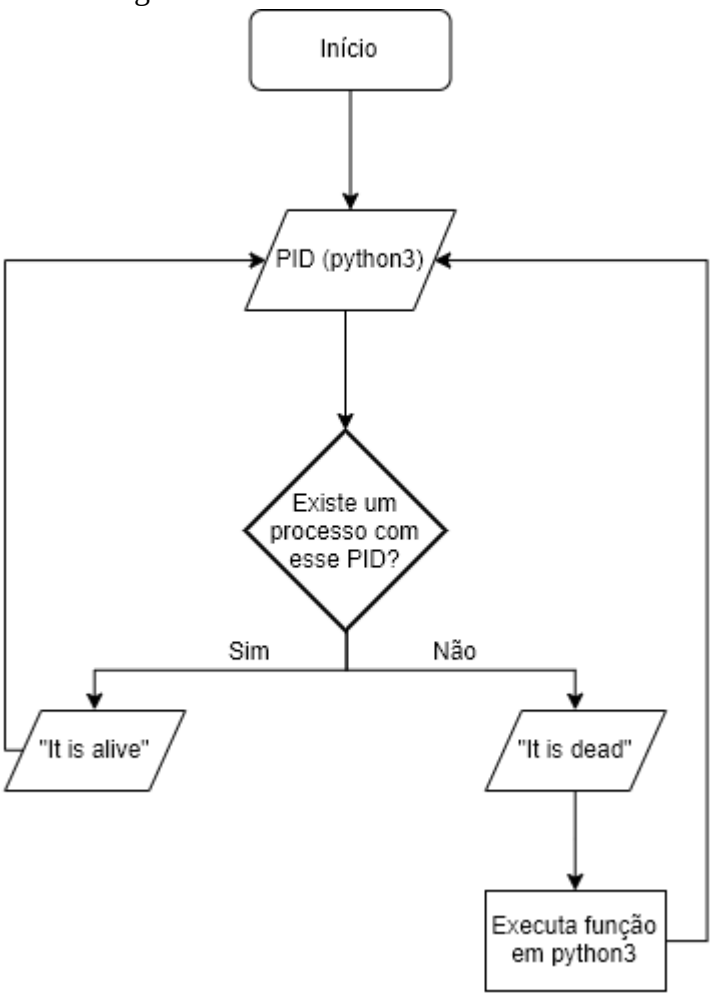

Figura 1 – Representação em diagrama de blocos da função em Shell

1.2 Diagrama do código em python3

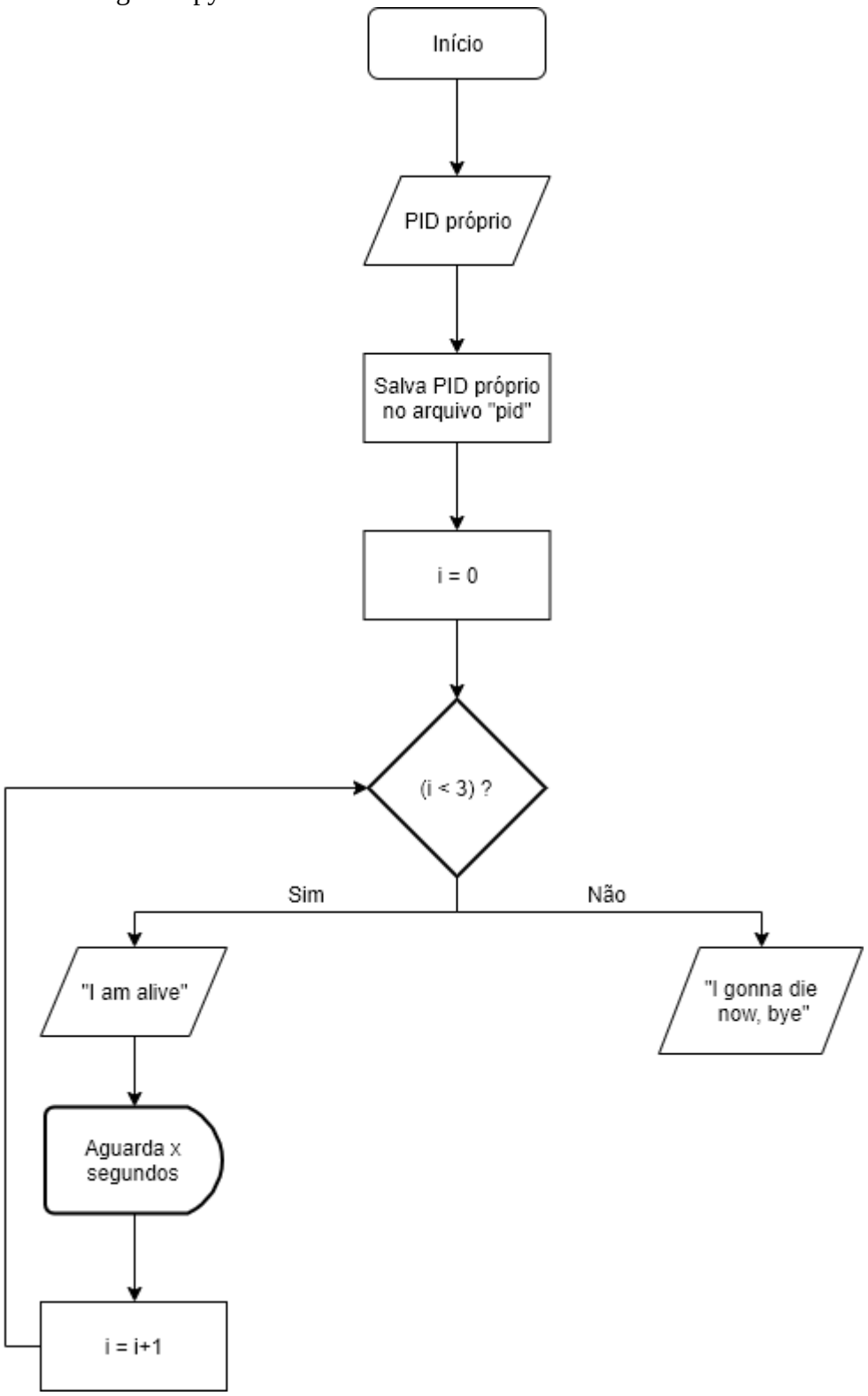

Figura 2 – Representação em diagrama de blocos da função em python3

2. CÓDIGO DA FUNÇÃO EM SHELL

```
2  #!/usr/bin/env bash
3
4  file="PID.pid" #Arquivo
5
6  while : #Loop infinito
7  do
8      while i= read -r line #Lê arquivo
9      do
10        PID=line #Salva na variável PID o pid contido no arquivo
11     done < "$file" #até o fim do arquivo
12
13
14
15     if [ -n "$PID" -a -e /proc/$PID ] #Se estiver rodando o programa
16     then
17         echo "1: It is alive" #Sinaliza que está rodando
18     else
19         echo "1: It is dead" #Sinaliza que não está rodando
20         python3 Escreve_PID.py #Executa função em python3
21
22     fi
23
24 done
```

Figura 3 – Código da função Le_PID.sh

3. CÓDIGO DA FUNÇÃO EM PYTHON3

```
2  # -*- coding: utf-8 -*-
3  """
4
5  Autor: Daniel Kenji Saito
6
7  Descrição: O programa a seguir tem como objetivo escrever seu próprio
   PID em um
8  arquivo para que seu status seja monitorado por outro programa. Além d
   isso, esse
9  deve sinalizar por 3 iterações que está funcionando, aguardando x segu
   ndos entre
10 elas. Por fim, o programa sinaliza que está sendo finalizado.
11 """
12
13 import os #importa biblioteca OS para usar funçoes relacionadas a PID
14 import time #importa biblioteca time para usar função sleep
```

```
15
16 pid_path = "PID.pid" #arquivo
17
18 flags = (os.O_CREAT | os.O_RDWR ) # cria o arquivo se não existe ou lê
   /escreve
19 mode = 0o755 #permissões do arquivo
20
21 pid_fd = os.open(pid_path, flags,mode) #local do arquivo PID
22 pidfile = os.fdopen(pid_fd, 'w') #arquivo PID
23
24 pid = os.getpid() #obtém o próprio PID e salva na variável pid
25 pidfile.write("%s\n" % pid) #Escreve o PID da função no arquivo
26 pidfile.close() #fecha o arquivo
27
28 wait_time = 1 #Tempo de espera (x) em segundos entre os prints
29
30 for i in range(3): #loop de 0 a 2 (3 iterações)
31     print("2: I am alive") #sinaliza que programa está em execução
32     time.sleep(wait_time) #aguarda os x segundos
33
34 print("2: I gonna die now, bye") #sinaliza fim da execução do programa
```

Figura 4 – Código da função Escreve_PID.py

4. INTEGRAÇÃO DOS CÓDIGOS

4.1 Descrição de como integrar os códigos

A integração dos códigos descritos nas seções 2 e 3 é simples, basta que o arquivo pid acessado pelas duas funções seja o mesmo. Dessa forma, garante-se que o pid lido pela função em Shell é o mesmo gravado pela função em python3. Por fim, é necessário executar a função “Escreve_PID.py” dentro da função “Le_PID.sh”, caso essa não esteja sendo executado (linha 20 da Figura 3).

4.2 Resultados

```
root@LAPTOP-63T97OTL:/mnt/c/Users/daniel/Desktop/Case_Cromai# bash Le_PID.sh
1: It is dead
2: I am alive
2: I am alive
2: I am alive
2: I gonna die now, bye
1: It is dead
2: I am alive
2: I am alive
2: I am alive
2: I gonna die now, bye
1: It is dead
2: I am alive
2: I am alive
```

Figura 5 – Resultados após a integração das funções

Percebe-se a partir dos resultados demonstrados acima que a função em Shell é executada em loop infinito. Quando essa função detecta que a função em python3 não está mais sendo executada ela imprime “It is dead” e executa novamente a função em python3, que quando é executada imprime “i am alive” 3 vezes em um intervalo de “x” segundos entre elas. Ou seja, a função em Shell monitora a função em python3 para garantir que essa esteja sempre sendo executada.

5. DIFERENÇAS ENTRE OS MODELOS

A partir dos modelos conceituais e dos códigos implementados, percebe-se que os diagramas representam bem o que foi implementado nos códigos, porém não incluem ou simplificam algumas etapas do algoritmo. No diagrama da função em Shell, o PID é apenas uma entrada, ou seja, o loop de leitura do arquivo PID não é representada. Já no diagrama da função em python3, a etapa de salvar o PID é apresentada em apenas um bloco de maneira simplificada, ou seja, não é representado o passo a passo de abertura do arquivo, obtenção do PID com a função “os.getpid()”, escrita do PID e fechamento do arquivo.