Nome: Daniel Kenji Saito

**Relatório – Case da Cromai**

1. Modelo conceitual
   1. Diagrama do código em Shell

Uma imagem contendo texto, mapa

Descrição gerada automaticamente

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

* 1. Diagrama do código em python3

Uma imagem contendo texto

Descrição gerada automaticamente

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

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

Figura 3 – Código da função Le\_PID.sh

1. Código da Função em Python3
2. # -\*- coding: utf-8 -\*-
3. """
4. Autor: Daniel Kenji Saito
5. Descrição: O programa a seguir tem como objetivo escrever seu próprio PID em um
6. arquivo para que seu status seja monitorado por outro programa. Além disso, esse
7. deve sinalizar por 3 iterações que está funcionando, aguardando x segundos entre
8. elas. Por fim, o programa sinaliza que está sendo finalizado.
9. """
10. import os #importa biblioteca OS para usar funçoes relacionadas a PID
11. import time #importa biblioteca time para usar função sleep
12. pid\_path = "PID.pid" #arquivo
13. flags = (os.O\_CREAT | os.O\_RDWR ) # cria o arquivo se não existe ou lê/escreve
14. mode = 0o755 #permissões do arquivo
15. pid\_fd = os.open(pid\_path, flags,mode) #local do arquivo PID
16. pidfile = os.fdopen(pid\_fd, 'w') #arquivo PID
17. pid = os.getpid() #obtém o próprio PID e salva na variável pid
18. pidfile.write("%s\n" % pid) #Escreve o PID da função no arquivo
19. pidfile.close() #fecha o arquivo
20. wait\_time = 1 #Tempo de espera (x) em segundos entre os prints
21. for i in range(3): #loop de 0 a 2 (3 iterações)
22. print("2: I am alive") #sinaliza que programa está em execução
23. time.sleep(wait\_time) #aguarda os x segundos
24. print("2: I gonna die now, bye") #sinaliza fim da execução do programa

Figura 4 – Código da função Escreve\_PID.py

1. Integração dos códigos
   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).

* 1. Resultados

Uma imagem contendo captura de tela

Descrição gerada automaticamente

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.

1. 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.