BCC264 **2015/2**

Email client - Sistemas Operacionais

27 de janeiro de 2016

Professor: Carlos Frederico *Alunos: Hannon Queiroz, Marcos Vinícius*

Neste trabalho, foi implementado um sistema *web* capaz de ler *emails* de duas contas: uma do provedor *gmail* e outra do *yahoo*. Para realizar tal tarefa, o *framework web Django* foi utilizado, a fim de reduzir o trabalho necessário para atender os requisitos de uma aplicação *web*.

1 Implementação

Ambas as contas de *email* são verificadas simultaneamente. Para isso, foi utilizada a biblioteca ***threading***, da biblioteca padrão da linguagem de programação *Python*.
A *thread* foi implementada da seguinte forma:

```
class EmailThread(threading.Thread):
    def __init__(self, username, password, imap_id):
        """
        EmailThread constructor.
        :param username: username to log in the email account
        :param password: password for the given account
        :param imap_id: an ID to identify which imap code to
           use: 1 = gmail, 2 = yahoo
        """
        threading.Thread.__init__(self)

        self.username = username
        self.password = password
        self.imap_id = imap_id

    def run(self):
        """
        Runs this thread
        """

        print("Starting thread ", self.username)
        print()
        self.imbox = login(self.username, self.password, self
           .imap_id)

        load_already_seen()  # Loads messages that have
           already been seen in previous runs of this code so
            that they don't appear as duplicate in the view
           for the user
        self.check_for_new_emails()

    def check_for_new_emails(self):
        """
        Checks for new emails that match the pattern "[BBC423
           ][xx.x.xxxx] Agenda dd/mm/aaaa hh:mm"
```

```
        """
        print("Checking for new emails on ", self.username)
        sleep_for = threading.Event()

        while True:
            unread_messages = self.imbox.messages()

            for uid, message in unread_messages:
                if  pattern.match(message.subject) and
                   message.message_id not in already_seen:
                    EmailId.objects.create(
                        email_id=message.message_id
                    )
                    already_seen.add(message.message_id)  #
                       workaround to avoid repeated emails

                    print("Subject: ", message.subject)
                    save_email(message)
                    add_to_calendar(message)

        print("%s is going to sleep for %d seconds"%(self.
           username, timeout))
        sleep_for.wait(timeout=timeout) # sleeps for n
           seconds
        print("%s is waking up"%(self.username))
```

Quando o usuário loga com as duas contas de *email*, duas instâncias da *thread* mostrada no código anterior são criadas.

2 Condição de corrida

Na implementação deste trabalho, não foram tratados casos onde condições de corrida possam ocorrer. O motivo para a não observação destes casos é que, apesar de teoricamente possíveis, condições de corrida são **extremamente** improváveis dadas as características da *Internet*. Considere o seguinte cenário onde poderia haver condição de corrida:

1. O email do Gmail recebe uma mensagem agendando um compromisso para o dia 30/01/2016 às 10:00.
2. O email do Yahoo recebe este mesmo email, com a mesma data e hora, mas para um compromisso diferente.
3. O cliente de email desenvolvido neste trabalho recupera os dois emails **ao mesmo tempo** e tenta adicionar os eventos no Google Calendar **ao mesmo tempo.**

Mesmo que isso ocorresse, a própria latência da Internet faria com que cada requisição chegasse separadamente no servidor do Google Calendar, evitando que os dois eventos competissem entre si.