Multithread aplicada em chat na plataforma iOS

Davi Francelino, Mozart Lemos de Siqueira

Centro Universitário La Salle – Unilasalle
Caixa Postal 92.010-000 – Canoas – RS – Brasil

davifrance@gmail.com, mozarts@unilasalle.edu.br

Abstract. *This paper describes the basic operation of threads using the Objective C. Through the use of this feature is presented an application for iOS. This aims to make communication via chat between students and teachers.*

Resumo. *Este artigo descreve o funcionamento básico de threads utilizando a linguagem Objective C. Através do uso deste recurso é apresentada uma aplicação para iOS. Esta tem por objetivo realizar a comunicação, via chat, entre alunos e professores.*

1. Introdução

Os impactos das novas tecnologias da informação e comunicação assim como seu barateamento, abriram portas para a massificação de dispositivos mais potentes, portáteis e inteligentes.

Falando especificamente sobre *smartphones*, é evidente que ainda são necessários maiores cuidados em sua implementação se comparado com aplicações no ambiente *desktop*, devido principalmente a sua portabilidade, mas também por ainda contar com recursos reduzidos de memória e processamento. Normalmente seu sistema operacional (SO) é otimizado em relação aos PCs para tratar isto, ou em outras palavras, o SO móvel gerencia de forma equilibrada o potencial e memória disponível do aparelho de acordo com as tarefas que estão sendo executadas [Simonite 2011].

Todavia não apenas os SOs podem executar diversas tarefas simultaneamente como também cada aplicação, utilizando *threads*. Estas representam individualmente, uma linha de execução do código da aplicação. Todo software inicia com um *thread* principal, normalmente chamando uma função *main*. A partir de então o programa pode gerar novas *threads*, e executar paralelamente funções específicas [Silberschatz e Galvin e Gagne 2005].

Quando uma aplicação cria uma nova *thread*, esta se torna uma entidade independente dentro do espaço do processo da aplicação e possui sua própria fila de execução. Além disto, uma *thread* pode se comunicar com outras *threads* e aplicações, além de realizar operações de entrada e saída. Entretanto como executa dentro do mesmo espaço de memória da aplicação ela compartilha também, além dos dados, os direitos de acesso de quem a gerou.

Ao longo deste artigo será apresentada a linguagem *Objective* C, seu suporte e gerenciamento de aplicações *multithreading*. A seguir há um detalhamento de como fora

implementado uma versão do *chat* do sistema de gerenciamento de aprendizagem Moodle, para a plataforma móvel iOS da Apple utilizando o conceito de múltiplas *threads*.

2. Objective C e Multithreading

A linguagem tem suporte à orientação a objetos e também é a principal forma de desenvolver aplicações para o Mac OS e iOS, ambos sistemas operacionais desenvolvidos pela Apple. A diferença entre o desenvolvimento para os dois sistemas dá-se pelo ambiente final de execução: dispositivos móveis ou *desktop* e servidores. A linguagem *Objective C* é derivada diretamente do C, com algumas características de *Smalltalk*, como o uso de parâmetros dentro do nome do método ao invés de uma seção de parâmetros no mesmo [Apple 2010].

A linguagem *Objective C* implementa uma classe chamada *NSThread* que permite a criação de novas *threads* dentro de um processo. Assim como em outras linguagens, são disponibilizados mecanismos para efetuar a sincronização de dados compartilhados. Porém, isto demanda carga extra de controle que nem sempre é utilizada e por padrão seu suporte vem desabilitado. É ativada somente quando novas instâncias da *NSThread* são geradas. O fato de não habilitar este gerenciamento aumenta significativamente o desempenho de aplicações *single thread*.

Como outras linguagens derivadas de C, o *Objective C* permite a utilização de outras bibliotecas para trabalhar com *threads* como o padrão POSIX. Entretanto, devido ao suporte *multithread* vir desabilitado, ela não recebe notificações das *threads* POSIX. Isso geralmente ocasiona falta de estabilidade e erros na aplicação.

Para resolver este problema, mesmo usando bibliotecas de terceiros, é necessário fazer uma chamada da classe *NSThread* e deixá-la finalizar imediatamente. Ou seja, ela só serve para habilitar o gerenciamento *multithread* da linguagem sem que seja necessário nenhum processamento por parte deste *thread*.

2.1 Gerenciamento e sincronização de threads

O *Objective C* possui estruturas de *get* e *set* chamados *properties*. Eles funcionam como forma de encapsular o acesso aos objetos, seus atributos e relacionamentos expondo apenas uma forma de buscar ou atribuir um valor [Apple 2010].

Na *API* da linguagem eles podem ainda ter atributos adicionais que gerenciam de que forma a *property* será tratada. Somente leitura ou somente escrita, por exemplo. Por padrão as *properties* criadas são atômicas, ou seja, vários *threads* podem interagir sobre os mesmos dados sem que estes se tornem inconsistentes. Todavia, há um detalhe importante para o desenvolvedor, os atributos das propriedades são apenas aplicados pelo compilador se utilizados em conjunto com o comando *synthesize*.

O *synthesize* é responsável por gerar o código necessário de acordo com cada atributo especificado para a *property*. Entretanto se este não for utilizado, todo e qualquer atributo das propriedades será apenas informativo e não causará efeitos na compilação do programa. Nesse caso, o desenvolvedor deve se preocupar com questões de sincronização e acessos indevidos a dados compartilhados entre os *threads* [Mark; Nutting; LaMarche 2011].

3. Portando chat do Moodle para iOS

O Moodle foi construído utilizando *Hypertext Preprocessor* (PHP), logo funciona em qualquer plataforma que possua um navegador e conexão com a internet. Porém, seu maior problema é que sua interface foi pensada apenas para telas maiores, como as presentes em *tablets* ou *monitores*, não se ajustando em dispositivos menores como os *smartphones*.

Dentre os inúmeros módulos distribuídos na versão padrão do sistema um é especificamente prejudicado, o módulo de conversação (*chat*), visto que seu *layout* dificulta o acompanhamento das discussões e digitação de mensagens ao mesmo tempo. Com o objetivo de resolver estes problemas foi criada uma aplicação para a plataforma iOS que permite aos alunos e professores se comunicarem de forma mais eficiente do que a interface web. O iOS como uma plataforma bastante popular e robusta foi uma escolha natural para o desenvolvimento deste projeto.

No contexto citado é altamente recomendada a utilização de múltiplos *threads* para tornar a experiência do usuário suave. Esta técnica colabora para uma aplicação mais responsiva e eficiente tornando a comunicação dentro do *chat* dinâmica e fluída. Este, aliás, é um dos maiores desafios dos desenvolvedores na atualidade, escrever programas que possam efetuar operações complexas em resposta a uma ação do usuário enquanto ele continua o uso normal do software.

```
//Processa o envio da mensagem
- (void) enviaMensagem:(NSString *)msg {
    //Cria o pool para a Thread
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    //Monta a URL para requisição ao servidor Moodle
    //[...]
    NSString *urlPost = [navigator.urlMoodle
                         stringByAppendingFormat:@"u=%@&s=%@&sendchat&chatid=%@&idchatuser=%@&chatmessagedata=%@",
                         navigator.usuario, navigator.senha, self.mychat.id, self.idUserChat, message];
    //Requisição para o servidor Moodle
    [manageJSONRequest
        jsonSimpleUrlRequest:[NSURL
                              URLWithString:[urlPost
                                             stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]]];
    //Invoca o método reloadTable passando uma string como parâmetro
    [self performSelectorInBackground:@selector(reloadTable:) withObject:@"enviaMensagem"];
    //Libera o pool para a Thread
    [pool release];
}
//Método que atualiza as mensagens do chat
- (void) reloadTable: (id)sender {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    //Permite apenas a entrada de uma thread por vez neste bloco através
    //do bloco @synchronized
    @synchronized(self) {
        //[...]
        for (NSDictionary *arrDado in [manageJSONRequest jsonComplexUrlRequest:[NSURL URLWithString:urlMensagens]])
        {
            //[...]
            //Adiciona cada mensagem no componente da interface
            NSArray *newData = [NSArray arrayWithObjects:
                                [NSIndexPath
                                 indexPathForRow:([self.arrMensagens count] - 1) inSection:0], nil];
            [tableUIView beginUpdates];
            [tableUIView insertRowsAtIndexPaths:newData withRowAnimation:UITableViewRowAnimationNone];
            [tableUIView endUpdates];
        }
    }
    [pool release];
    //[...]
    //Finaliza a thread
}
```

Figura 1. Código parcial do envio e recebimento de mensagens na aplicação

No software desenvolvido as *threads* foram utilizadas para não bloquear a interface enquanto a tela é atualizada com mensagens do *chat,* o que tornaria impraticável o uso. Além disso, todo o envio das mensagens também é feito utilizando esta técnica.

A figura 1 ilustra uma parte da implementação do envio e atualização das mensagens na interface do dispositivo. No método enviaMensagem é criada uma *thread* de uma maneira diferente mas bastante comum através da chamada *performSelectorInBackground* (linha 16). Com ela não é necessário criar um objeto do tipo *NSThread*, configurar suas propriedades e chamar o método *start* pois o próprio compilador já se encarrega de fazer isto internamente. Em contrapartida se forem necessários configurações como prioridade, então o uso da classe *NSThread* se torna obrigatório. No exemplo da figura 1 a *thread* recebe a indicação de que executará a função reloadTable e receberá como parâmetro de entrada um objeto do tipo *string*. Após todo o processamento da *thread*, já no método reloadTable (linha 21-42), esta é então finalizada pois não tem mais trabalho algum a fazer.

Conclusão

No dia-a-dia, seja utilizando computadores ou não, as pessoas desejam liberdade para executar múltiplas tarefas simultaneamente. No desenvolvimento de sistemas, isso pode ser atingido, dentre outras formas, com a utilização de *threads*.

A linguagem *Objective C* implementa controles de manipulação de zonas de memória compartilhadas por cada *thread* de forma nativa através do uso de algumas instruções específicas. Esta facilidade pode ser responsável por um aumento da produtividade em aplicações *multithread* e direcionar as preocupações do desenvolvedor para outros fatores mais críticos.

As *threads* tem papel importante na construção da aplicação, funcionando de forma a evitar o bloqueio da interface com o usuário. Para tanto foram utilizados sempre que possível às facilidades disponíveis na linguagem para realizar esta implementação.

Referências

Apple (2010) “About Threaded Programming”, http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/Multithreading/AboutThreads/AboutThreads.html, Novembro.

Apple (2010) “Declared properties”, http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocProperties.html, Novembro.

Mark, D.; Nutting, J.; LaMarche, J. (2011). “Beginning Iphone 4 Development: Exploring the iOS SDK”, Apress, New York.

Silberschatz, A.; Galvin, P. B.; Gagne, G. (2005). Operating system concepts, Wiley Publishing, 15ª edição.

Simonite, T.; Smart-phone operating systems control more consumer electronics. Technology Review. v. 114, n. 3, p. 72-73, mai./jun. 2011.