

Publicidade



ASSINE 0800 703 3000

BATE-PAPO

E-MAIL

SAC

Voip

E-Mail Grátis

Shopping

ÍNDICE PRINCIPAL

PROCURAR:

no site

OK

Terça, 24/02/2009

- » Introdução
- » Programação
- » Administração
- » Hardware
- » Aplicativos
- » Jogos
- » Segurança
- » Editorial
- » Entrevistas

ARTIGOS

- » Fórum
- » Links
- » Notícias
- » Pegue o Linux
- » Documentação

OLINUX SERVIÇOS COMUNIDADE

- » Programas
- » Dúvidas
- » Oportunidades
- » Sobre
- » Contato
- » Publicidade

Powered By:
DEBIAN
GNU/LINUX

English Version

Linux Solutions
Shopping
OLinux



Programação

Tutorial de sockets - Parte VI

Por: [Frederico Perim](#)

▶ Técnicas Avançadas

Não são realmente avançadas, mas não são tão básicas como os tópicos que já abordamos. Na verdade, se você chegou até aqui considere-se com bons conhecimentos básicos de programação de redes Unix.

▶ Blocking

Você provavelmente notou quando executou o programa _____, acima, que ele espera até que um pacote chegue. O que aconteceu é que ele chamou a função `recvfrom()`, e não havia dados, então `recvfrom()` está em estado "sleep", esperando pacotes.

Várias funções utilizam esse procedimento, tais como: `accept()` e todas as variantes de `recv()`. A razão pela qual realizam isso é porque a elas é permitido tal comportamento. Quando você cria um socket com a função `socket()`, o kernel vai ajustá-la para blocking. Se você não quiser que um socket realize isso, então você deve chamar a função `fcntl()`:

```
sockfd = socket (AF_INET, SOCK_STREAM, 0);
fcntl (sockfd, F_SETFL, O_NONBLOCK);
.
.
.
```

Ao setar um socket para o estado non-blocking, você pode efetivamente "consultar" o socket para obter informações. Se você tentar ler um socket em estado non-blocking e não houver dados, ele retorna -1.

Genericamente falando, no entanto, este tipo de "consulta" é uma má idéia. Se você mantiver seu programa ocupado esperando por dados em um socket, você vai consumir sua CPU ao ponto de tornar as coisas complicadas. Uma solução mais elegante para checar se existem dados esperando para serem lidos vem na próxima seção sobre `select()`.

▶ `select()`

Esta função é um tanto estranha, mas muito útil. Analise a seguinte situação: você é um servidor e deseja escutar por conexões e ao mesmo tempo continuar recebendo dados das que já existem.

Sem problema, apenas uma chamada `accept()` e algumas `recv()`. Opa, não tão rápido!. E se a chamada `accept()` estiver em blocking? Como você vai receber dados ao mesmo tempo? "Use sockets non-blocking!" De forma alguma! Você não quer comprometer sua CPU. E então?

`select()` lhe dar o poder de monitorar várias conexões ao mesmo tempo. Ela lhe dirá quais estão prontas para receber, quais estão prontas para enviar, e quais leventaram exceções, se você realmente deseja saber.

Sem mais demora, aqui está a sinopse:

ENQUETE

Com qual frequência você acessa o site Olinux?

- ☐ Todos os dias
- ☐ Uma vez por semana
- ☐ Cinco vezes aos mês
- ☐ Poucas vezes ao mês
- ☐ Outra

VOTAR

NEWSLETTER

Inscreva-se e receba as últimas notícias, programas, artigos, novidades e tudo do mundo Linux que aconteceu na semana.

Digite seu email:

OK

Relógio

de Pulso em até
12x.

Brinquedos

das Meninas Super
Poderosas. Clique!

Filmadora

Multilaser CR-518
Digital.
Compare!

Esteira

Entre em forma
antes do verão.

COMPARE PREÇOS


```
int select (int numfds, fd_set *readfds, fd_set *writefds,  
            fd_set *exceptfds, struct timeval *timeout);
```

A função monitora conjuntos de descritores de [arquivos](#), em particular readfds, writefds e exceptfds. Se você deseja saber se pode ler da entrada padrão e algum socket, sockfd, acrescente o descritor de arquivo 0 e sockfd ao conjunto readfds. Ao parâmetro numfds deve ser atribuído os maiores valores do descritor de arquivo mais um. Nesse exemplo, ele deve ser sockfd+1, já que seguramente é maior que a entrada padrão (0).

Quando select() retorna, readfds será modificado para refletir qual descritor você selecionou que está pronto para receber. Você pode testá-los com o macro FD_ISSET(), abaixo.

Antes de continuar, irei abordar a forma de manipular estes conjuntos. Cada conjunto é do tipo fd_set. Os seguintes macros operam nesse tipo:

- FD_ZERO (fd_set *set) -- limpa o conjunto
- FD_SET (int fd, fd_set *set) -- adiciona fd ao conjunto
- FD_CLR (int fd, fd_set *set) -- remove fd do conjunto
- FD_ISSET (int fd, fd_set *set) -- verifica se fd está no conjunto

[Próximo»](#)

▶ **Técnicas Avançadas**
▶ **Blocking**
▶ **select()**



Enviar para um amigo



Imprimir



Índice de artigos

[Publicidade](#) / [Sobre OLinux](#) / [Entre em Contato](#) / [Privacidade](#)
Copyright (c) 2000-2007, OLinux - O Portal de Linux do Brasil.
Desenvolvido por: [Linux Solutions](#)
Todos os Direitos Reservados.