Um modelo de computação multiusuário baseado em computadores pessoais

Ander Conselvan de Oliveira[1], **Tiago Vignatti**[1], **Daniel Weigaertner**[1], **Fabiano Silva**[1], **Marcos Castilho**[1], **Marcos Sunye**[1]

[1] Departamento de Informática – Universidade Federal do Paraná
Caixa Postal 19.081 – CEP 81.531-980 – Curitiba – PR – Brasil

{ander,tv02,danielw,fabiano,marcos,sunye}@c3sl.ufpr.br

Abstract. *This paper presents a new interaction model that reduces the costs and the idleness of computer systems applied to educational environments.*

Resumo. *Este trabalho apresenta a implementação de um novo modelo de interação dos usuários com o sistema que visa diminuir a ociosidade dos equipamentos e reduzir os custos de implantação de sistemas computacionais voltados para inclusão digital e ambientes de ensino.*

1. Introdução

Um dos primeiros modelos de computação era o de processamento centralizado em equipamentos de grande porte e de elevados custos, onde a interação entre o sistema e o usuário se dava através de terminais sem capacidade de processamento, responsáveis apenas pela visualização e entrada de dados. O uso de vários terminais de baixo custo permitiam o uso destes computadores por um grande número de usuários concorrentemente.

Na década de 1980, o surgimento do computador pessoal possibilitou, devido ao seu baixo custo, que um novo modelo de computação se estabelecesse. O usuário passa a interagir com o sistema no mesmo equipamento onde o processamento ocorre. Ainda assim, os sistemas operacionais desses equipamentos mantiveram a capacidade de uso simultâneo pelos usuários, porém somente era possível fazer isso através de conexões de redes de comunicação.

Com o crescimento da capacidade de processamento dos computadores pessoais, um grande de número de tarefas podem ser executadas concorrentemente por mais de um usuário do sistema. Este cenário remonta ao citado inicialmente, um ambiente multitarefa e multiusuário. Entretanto, apenas um usuário tem acesso aos recursos de interação fornecidos pelo equipamento, o que limita o aproveitamento de todo o sistema, que fica grande parte do tempo ocioso.

Neste artigo apresentamos um modelo baseado em software livre, usando Linux, que aproveita a capacidade ociosa dos computadores pessoais modernos: o multiterminal. O multiterminal é um computador pessoal em que vários conjuntos de dispositivos de interação (vídeo, teclado e mouse) são conectados, permitindo seu uso simultâneo por vários usuários em um mesmo local.

A próxima seção apresenta o modelo conceitual do multiterminal e algumas definições importantes. A seção 3 faz uma retrospectiva dos modelos semelhantes propostos anteriormente pela comunidade. Na seção 4 são apresentadas duas implementações do multiterminal. E finalmente são apresentadas as conclusões do trabalho.

2. O Modelo Multiterminal

Um terminal é um conjunto de equipamentos que faz a interface entre o usuário e computador. No modelo multiterminal, vários desses conjuntos (mouse, teclado e monitor) são conectados a um único computador. A camada de software desse sistema precisa fornecer a independência entre os terminais (figura 1).

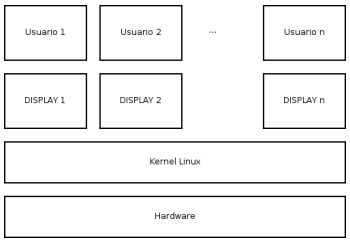

Figura 1. Estrutura do modelo multiterminal

Nos sistemas operacionais derivados do UNIX, como o Linux, a interação com o usuário se dá através do Sistema de Janela X (X Window System). Esse sistema é baseado na arquitetura cliente-servidor, em que o cliente envia requisições de desenho para o servidor e recebe desse os eventos de entrada (teclado e mouse) (figura 2) [Quercia and O'Reilly 1991]. Os servidores X têm o conceito de recursos, como um dispositivo de entrada ou uma janela, que são disponibilizados aos seus clientes. Tais recursos estão associados a um Display, que pertence a um usuário. Portanto, um sistema multiterminal baseado em UNIX deve prover um Display para cada usuário.

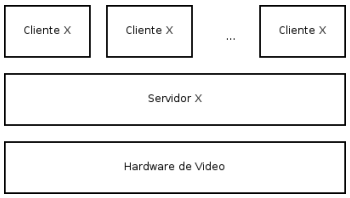

Figura 2. Arquitetura do Sistema de Janela X

O Xorg, implementação mais recente do servidor X, não tem suporte a múltiplos *Displays*. Seguindo o modelo do computador pessoal, ele é projetado sobre o pressuposto de que apenas um usuário utiliza a máquina de cada vez. A sua camada de entrada é implementada sobre o método padrão de entrada do kernel, os terminais virtuais (VT). Eles recebem esse nome pois simulam o método de entrada dos antigos mainframes. O terminal virtual é implementado inteiramente em software, mas simula o tty, um dispositivo que era conectado através de portas seriais. O kernel Linux tem suporte a vários terminais virtuais, mas apenas um deles pode receber eventos de teclado por vez. Caso mais de um teclado estivesse conectado ao computador, os eventos de todos os teclados seriam enviados ao VT ativo. Isso impede a execução concorrente de dois ou mais servidores X, pois apenas um pode estar ativo num determindado momento, mesmo que utilizem hardware de vídeo distintos.

Recentemente, uma nova camada de entrada foi introduzida ao kernel Linux. A nova camada permite o acesso separado a cada dispositivo de entrada conectado. Cada dispositivo corresponde a um *character device* no diretório */dev/input* e os eventos são

reportados através de um protocolo padronizado. As versões mais novas do Xorg (6.9 e 7.0) têm suporte a este método de entrada, porém o método padrão ainda é utilizar os ttys.

Apesar de tirar proveito dos métodos de entrada do kernel, o Xorg implementa uma camada independente para controle de hardware de vídeo. Isso acontece porque o kernel não tem uma arquitetura de entrada e saída robusta o bastante para se obter um bom desempenho do servidor X. Porém, essa abordagem causa problemas de concorrência, pois um servidor X e o kernel podem tentar controlar um mesmo dispositivo ao mesmo tempo, causando falha em seu funcionamente e instabilidade.

Outro conflito surge devido a arquitetura do computador pessoal IBM/PC. Nessa arquitetura, os periféricos são acessados por endereços previamente fixados. Por exemplo, a COM1 tem endereço 0x3F8. O acesso ao VGA (Video Graphic Array), interface de controle gráfico do IMB/PC, também tem essa limitação de endereçamento. Quando temos mais de uma placa de vídeo no sistema, como no multiterminal, as placas tentarão disputar o mesmo endereço do barramento. Uma mesma instância do Xorg consegue trabalhar com vários adaptadores VGA, usando um módulo de controle de acesso a recursos que coordena o acesso a diferentes dispositivos VGA, possibilitando que um Display possua várias telas. Porém, quando executamos uma instância do Xorg para cada placa de vídeo, este controle de acesso não funciona, pois cada instância considera-se a única a controlar dispositivos de vídeo na máquina.

Na próxima seção, veremos algumas implementações do multiterminal e como elas superam os problemas citados acima.

3. Implementações do Multiterminal

A idéia de utilizar mais de uma instância do servidor X num mesmo computador e, com a adição de teclados e mouses, permitir o uso simultâneo por mais de um usuário, apareceu na Internet em 2001, na página do brasileiro Miguel Freitas [Freitas 2005]. Basicamente, o que ele fez foi modificar a implementação do XFree86 (implementaçao do X Window System em que o Xorg foi baseado) de forma que este pudesse acessar um teclado que não fosse o VT do kernel. O acesso ao teclado era feito através da interface event do kernel (/dev/input/event), de maneira semelhante ao protocolo evdev atualmente suportado pelo Xorg. Nesta solução, eram utilizados dois binários distintos do XFree86, um com a modificação e outro sem, que rodavam simultaneamente no computador, cada um controlando um terminal.

O principal problema desta solução é a incompatibilidade das placas de vídeo devido à concorrência no acesso ao barramento PCI, de forma que apenas poucas combinações de placas de vídeo funcionam. Outro problema era a necessidade de modificar o XFree86 e de manter sua modificação atualizada. Mesmo assim, esta solução inspirou as que vieram posteriormente.

A partir da solução de Miguel de Freitas, surgiu a proposta de modificar o kernel para que este forneça um VT para cada teclado encontrado no sistema, de forma que se pode iniciar diversas instâncias do XFree86, uma para cada VT, sem a necessidade de alterar seu código. A primeira destas soluções foi desenvolvida por James Simmons, Vojtech Pavlik e Aivils Stoss e está descrita em [Slavtchev 2004, Centro de Computação Científica e Software Livre 2006a]. Ela utiliza um kernel modificado chamado de Backstreet Ruby que cria VTs adicionais, associando-os aos teclados

existentes no sistema. Um parâmetro passado ao kernel (dumbcon=?) configura a quantidade de VTs adicionais a serem criados, e a associação dos teclados aos VTs é feita através da escrita em arquivos do diretório /proc depois da inicialização do sistema.

Esta solução é bastante estável para algumas combinações de placas de vídeo, especialmente chipsets da NVidia e SIS315. Outros chipsets também funcionam desde que sejam as placas primárias do sistema. Mas ocorre o mesmo problema de concorrência no barramento PCI, o que limita muito as placas que podem ser utilizadas. Além disso, a modificação no kernel para implementar os múltiplos VTs é bastante grande, o que a torna difícil de ser atualizada a cada nova versão do kernel.

Devido a este problema, Aivils Stoss criou uma solução alternativa que cria "falsos VTs´´ que controlam os teclados adicionais e que podem ser utilizados pelas instâncias adicionais do X. Esta solução, descrita em [Centro de Computação Científica e Software Livre 2006b, Stoss 2006], tem como principal vantagem em relação à anterior o fato de utilizar um módulo do kernel (faketty) que pode ser carregado dinamicamente, não exigindo modificações no kernel em si. Desta forma, torna-se muito mais simples o processo de atualização para novas versões do kernel, embora os problemas de incompatibilidade de placas de vídeo continuem os mesmos, uma vez que existem diversas instâncias do X concorrendo no sistema.

Outra solução, muito parecida com a idéia original de Miguel de Freitas, é a utilização do protocolo evdev do kernel através da modificação do Xorg que permite a configuração de teclados e mouses de forma a utilizar diretamente o protocolo evdev. Nesta solução, da mesma forma que nas anteriores, utiliza-se uma instância do servidor X para cada terminal, mas os teclados não são mais configurados através dos VTs, e sim acessados pela interface evdev através do endereço físico fornecido pelo kernel. Esta solução foi incorporada ao Xorg, de forma que não necessita de manutenção e recompilação do servidor X. Novamente, sofre dos mesmos problemas das versões anteriores.

Tendo em vista os problemas expostos até aqui, apresentaremos na próxima seção as soluções que se baseiam em servidores X aninhados (*nested X servers*).

4. Solução proposta

4.1. Implementações baseadas em servidores X aninhados

Um servidor X aninhado é um servidor X que roda dentro de outro servidor X. O funcionamento dos servidores aninhados diferem dos servidores tradicionais nos métodos de entrada e saída. Enquanto um servidor tradicional utiliza o hardware de vídeo e interação com o usuário, um servidor aninhado utiliza uma janela em outro servidor X para exibição. Os eventos de entrada também são recebidos através dessa janela, utilizando o protocolo X11.

4.1.1. multiXnest

A solução multiXnest que aproveita a capacidade de controle de acesso a recursos, embutida no servidor X. Nessa solução, um servidor X (servidor base) controla todos os

dispositivos de vídeo e sobre este servidor base é executado um servidor X aninhado modificado para receber os eventos de entrada diretamente do kernel, ao invés do servidor base.

O módulo de controle de acesso a recursos do servidor base permite que qualquer placa de vídeo suportada pelo X seja utilizada em sistemas multiterminal com estabilidade.

O primeiro servidor aninhado a ser modificado foi o Xnest, para o qual foi implementado um novo driver de entrada, que lê os eventos de entrada a partir da interface event do kernel, e um cursor de mouse próprio (o Xnest original aproveita o cursor do servidor base). Apesar de resolver em grande parte os problemas de estabilidade e compatibilidade, essa solução apresenta alguns problemas: a falta de desenvolvimento do Xnest implica em assumir sua manutenção; e a falta de suporte à extensões muito utilizadas nos ambientes desktop, como as extensões Render, Xv e GLX; sobrecarga no processamento das requisições gráficas, pois o Xnest é um servidor proxy, ou seja, ele recebe dados do cliente e os repassa para o servidor base, sobrecarregando-o.

Os problemas apresentados pelo multiXnest motivaram a criação de uma nova solução, que foi o servidor Xephyr com algumas modificações semelhantes ao do Xnest.

4.1.2. Xephyr modificado

O Xephyr, diferente do Xnest, não atua como um servidor proxy, mas utiliza-se de uma extensão do protocolo X11 para acessar a janela do servidor base como um *framebuffer*. Isso permite que as requisições sejam tratadas no próprio Xephyr, diminuindo a sobrecarga de rodar servidores aninhados. Essa abordagem permite também que algumas extensões não implementadas no servidor base possam existir no servidor aninhado.

O Xephyr foi construído sobre a arquitetura kdrive, que simplifica o processo de criação de um novo servidor X e também facilita a implementação de extensões. Por isso, o servidor Xephyr implementa completamente as extensões Render e Shm. A extensão Shm permite a reprodução de vídeos com qualidade satisfatória, o que não era possível com o multiXnest. A extensão Render é utilizada por grande parte dos Desktops modernos, e sua presença permite o bom funcionamento dos mesmos.

A modificação necessária no Xephyr foi a criação do novo driver de entrada, que, devido à arquitetura kdrive, ficou mais simples que a do Xnest. Uma documentação mais detalhada sobre as modificações feitas no Xephyr e sobre o multiXnest pode ser encontrada em [Centro de Computação Científica e Software Livre 2006d, Centro de Computação Científica e Software Livre 2006c].

4.2. Estabilidade

Como visto na seção 2., o ponto crucial para ter um multiterminal funcional e estável seria isolar os eventos de entrada e conseguir a independência entre os terminais. Com o evdev o isolamento ficou fácil. A estabilidade foi garantida pelo servidor X base. A primeira possibilidade estudada de modificação para um multiterminal foi modificar o Xorg, de modo que uma única instância do servidor pudesse atender vários terminais, ou seja, um servidor X com vários displays. Porém a complexidade dessa proposta e a facilidade de

embutir eventos vindos do kernel (através do evdev) nos servidores X aninhados fez com que implementássemos essas soluções.

5. Conclusão e desenvolvimentos futuros

O multiterminal baseado no Xephyr tem desempenho inferior ao das soluções não aninhadas. Em contrapartida, ele é indiscutivelmente a solução mais estável e compatível com uma grande variedade de hardware. Esses fatores permitiram sua adoção em larga escala na UFPR e no projeto Paraná Digital, que irá instalar cerca de 44.000 pontos de trabalho nas escolas da rede estadual do Paraná. Algumas empresas do Brasil e do mundo demonstraram, também, interesse na solução.

Ainda assim, a falta de suporte a gráficos 3d acelerados por hardware é uma barreira na adoção em maior escala do sistema. O desenvolvimento de um novo servidor X, o Xgl [freedesktop.org 2006], cria perspectivas de uma nova solução multiterminal estável e com suporte a gráficos 3d. Atualmente em desenvolvimento, o Xgl é composto de um servidor X aninhado construído sobre os recursos de aceleração 3d dos equipamentos de vídeo modernos. Além disso, uma solução baseada nele iria proporcionar uma qualidade ainda maior na reprodução de vídeos de alta resolução. Porém, a solução definitiva para esse problema deve ser a alteração do sistema operacional e sua arquitetura de vídeo, tendo em vista o melhor aproveitamento dos recursos computacionais existentes.

Referências

Centro de Computação Científica e Software Livre (2006a). Multiterminal com patch backstreet ruby. http://www.c3sl.ufpr.br/multiterminal/howtos/ruby/howto-ruby-pt.html.

Centro de Computação Científica e Software Livre (2006b). Multiterminal usando o módulo faketty. http://www.c3sl.ufpr.br/multiterminal/howtos/howto-faketty-pt.html.

Centro de Computação Científica e Software Livre (2006c). Multiterminal usando xephyr. http://www.c3sl.ufpr.br/multiterminal/howtos/howto-xephyr-pt.html.

Centro de Computação Científica e Software Livre (2006d). multixnest - multiterminal usando xnest. http://www.c3sl.ufpr.br/multiterminal/howtos/howto-xnest-pt.html.

freedesktop.org (2006). Xgl. http://www.freedesktop.org/wiki/Software_2fXgl.

Freitas, M. (2005). Multiple local xfree users under linux. http://cambuca.ldhs.cetuc.puc-rio.br/multiuser/.

Quercia, V. and O'Reilly, T. (1991). *X Window system user's guide: OSF/Motif edition*, chapter An Introduction to the X Window System, pages 19–20. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Slavtchev, S. (2004). Xfree local multi-user. http://www.tldp.org/HOWTO/XFree-Local-multi-user-HOWTO/.

Stoss, A. (2006). Faketty kernel module. http://www.ltn.lv/ aivils/?proj_id=faketty.