DonutOS: Um Sistema Operacional Portátil para Monitoramento de Redes

Bernardo M. Fernandes[1], Marcos A. Lommez C. R.[1], Saulo de Moura Zandona F.[1], Fabio Freire K.[1], Bruno Santiago de Oliveira

[1]Pontifícia Universidade Católica de Minas Gerais (PUC-MG)

Abstract. *In recent years, the exponential growth of device connectivity to networks has generated an increasing demand for effective network monitoring tools. Especially in corporate and academic environments, the ability to detect and analyze network traffic has become essential to ensuring the security, performance, and integrity of systems. In response to this demand, this article proposes the development of a portable operating system equipped with network sniffer functionalities. The goal is to offer a practical and efficient solution for network monitoring, leveraging its potential in environments conducive to its use.*

Resumo. *Nos últimos anos, o crescimento exponencial da conectividade de dispositivos à rede tem gerado uma demanda cada vez maior por ferramentas de monitoramento de rede que sejam eficazes. Especialmente em ambientes corporativos e acadêmicos, a capacidade de detectar e analisar o tráfego de rede tornou-se essencial para garantir a segurança, o desempenho e a integridade dos sistemas. Em resposta a essa demanda, este artigo propõe o desenvolvimento de um sistema operacional portátil dotado de funcionalidades de Sniffer de rede. O objetivo é oferecer uma solução prática e eficiente para o monitoramento de redes, extraindo seu potencial em ambientes favoráveis a sua utilização.*

1. Introdução e tema

No cenário atual, a segurança e o monitoramento de redes são aspectos cruciais na gestão de sistemas e infraestrutura de T.I. Com o crescimento exponencial do uso da internet e o surgimento de novas tecnologias, existe uma demanda contínua por ferramentas de monitoramento de rede mais eficazes e acessíveis. Junto a essa necessidade crescente do mercado, faz-se necessário a criação de ferramentas especializadas para analise de trafego de rede que muitas vezes acabam se tornando refém das funcionalidades fornecidas nos programas, sem a capacidade de se aprofundar na análise da rede.

Diante deste contexto, o trabalho apresentado visa o desenvolvimento de um sistema operacional portátil com funcionalidades de sniffer de rede, destinado a ser não apenas uma ferramenta de análise de tráfego eficiente e independente da máquina hospedeira, mas também um ponto de partida para futuras expansões e aplicações personalizadas de código aberto para qualquer necessidade que envolva controle a nível de sistema operacional e baixo custo operacional.

Faz-se importante frisar o contexto no qual a produção do sistema é criado, que nasce da motivação de criação e estudo da produção de sistemas operacionais, assim

como a escrita de bibliotecas base de sistema assim como a preparação completa de um ambiente para rodar aplicações simples, sendo a motivação inicial que levou a este projeto ser rodar o código do Donut 3d emulado em letras ascii no terminal, assim levando ao mesmo de maneira legada o nome de DonutOS.

1.1. Objetivos e hipótese

Este projeto tem como objetivo geral desenvolver um sistema operacional portátil focado em monitoramento de rede, que sirva como uma base sólida para futuras extensões e personalizações. Especificamente, buscamos: (i) demonstrar a viabilidade técnica e prática de tal sistema; (ii) discutir o processo de implementação e necessidades do sistema; e (iii) analisar resultados a partir da criação do protótipo e suas características.

2. Referências bibliográficas

Para o desenvolvimento do sistema operacional e implementação das funcionalidades específicas necessárias para o projeto, diversas fontes foram utilizadas: desde playlists educativas até repositórios com referências de códigos.

O sistema operacional em si tem sua divisão explicitada na criação do sistema bootável, desenvolvimento do kernel e funções de biblioteca padrão assim como syscalls. Para um primeiro momento, a playlist "Making an OS [x86]" foi utilizada para controle da bios e no sistema de start-up para controle direto das peças para o boot do sistema. Após as configurações iniciais, a estruturação do código, entendimento e planejamento dos próximos passos foram feitas com base em diversos tutoriais disponibilizados na OSDev Wiki. Ademais, o livro de Sistemas Operacionais de Oliveira et al. [2010] também foi utilizado.

Além disso, foram utilizadas referências de códigos para implementação de drivers para peças da máquina, não só para funcionalidades do sistema mas também para permitir interação do usuário. Para isso, algumas referências de códigos disponíveis no GitHub foram utilizadas, como o Nanvix de Penna [2017], OS-Tutorial de Dubbelboer e o OS utilizado em explicações da OSDev de Zhao, assim como um vídeo explicativo sobre a implementação do TetrisOS.

Possuindo um sistema operacional utilizável para o proposito inicial, pode-se continuar para a criação da sessão especifica de redes. Utilizamos o sniffer de rede Wireshark como base para idealização no contexto de implementação a ser apresentado abaixo. O programa faz análise do tráfego de rede e é tratado comercialmente como referência de aplicação devido a sua base sólida e confiável para tal avaliação. Ao compreender os padrões e comportamentos das redes, pode-se desenvolver soluções mais eficientes e seguras.

A partir disso, serão implementados os drivers da placa de ethernet RTL8139. Sua implementação foi baseada no OSDev Wiki que é um manual simplificado criado pela OSDev para auxiliar na implementação de placas de redes em sistemas operacionais. Para mais além disso foi utilizada a implementação do szhou42 como referência. Por fim para a aplicação das camadas de redes utiliza-se a referencia do livro clássico de redes de Redes de Computadores de Tanenbaum and Wetherall [2011], assim como o próprio sistema utilizado na implementação da placa.

Por fim faz-se valido mencionar outros códigos importantes para o desenvolvimento do sistema operacional, que são o gerador de números pseudo-aleatórios Itamaracá de Henrique Pereira [2022]. E não menos importante o código que motivou a criação do projeto o qual o intitulou, sendo o mascote do próprio sistema, o Donut de Sloane [2021].

3. Metodologia e Cronograma

Este projeto visa o desenvolvimento de um sistema operacional portátil equipado com funcionalidades de monitoramento de rede. A metodologia foi adaptada para se encaixar em um cronograma organizado em três sprints principais, cada um com objetivos específicos.

3.1. Desenvolvimento

O desenvolvimento do projeto sera dividido em 3 sprints principais sendo estas:

Sprint 1: Implementação base do sistema, Planejamento e Design - O projeto inicia com a preparação da base de implementação do sistema, criando-se assim um dispositivo bootável, a partir deste ponto inicia o desenvolvimento de funções bases do kernel assim como a implementação de uma biblioteca padrão de C para o sistema. Com a base do sistema pronto e seu entendimento de arquitetura é possível planejar a implementação do driver de rede assim como a implementação das camadas para a criação do sniffer.

Sprint 2: Drivers de rede, Paginação - Com a implementação base do sistema concluída, a segunda sprint focará na implementação da base necessária para o controle das funcionalidades do SO. A partir desse ponto, o sistema será capaz de receber pacotes através da placa de rede, assim como configurações iniciais de paginação para o escalonamento de processos.

Sprint 3: Escalonamento, Testes e Avaliação - A última sprint será dedicada a implementação da pilha de protocolos (sendo essa focada exclusivamente apenas na leitura dos dados) assim como dar ao sistema a capacidade de abrir e gerenciar processos.

3.2. Avaliação

O trabalho tem como principal foco fornecer conhecimento público, seja com o próprio resultado desenvolvido como ferramenta e código ou como fonte de referencia pública para produção comercial ou educativa. Assim sendo, será verificado se o sistema atende aos requisitos pré-definidos, avaliando a precisão da captura e análise de dados de rede. A avaliação será feita através de máquinas virtuais com dados controlados, sendo enviados e recebidos pelas mesmas através de uma rede externa com entrada encapsulada pelo sistema hospedeiro.

3.3. Cronograma

O projeto será executado em três sprints ao longo de três meses, conforme descrito a seguir:
1º Mês: Implementação base do sistema, Planejamento e Design.
2º Mês: Drivers de rede e paginação.
3º Mês: Escalonamento de processos e pilha de protocolos.

4. Aspectos de implementação

Para a criação de um ambiente de teste, foi utilizado a máquina virtual QEMU que implementa a simulação de diferentes peças de hardware (*e.g.* processadores), permitindo uma virtualização completa de um sistema PC dentro de outro. Incluindo tanto as configurações do QEMU quanto decisões de implementação relacionadas ao sistema, as mesmas estão listadas adiante:

- QEMU
- GCC Cross Compiler
- NASM
- Processador 80386
- 32 bits - protected mode
- 128 MB ram
- PS/2 Keyboard
- 8259 PIC
- PIT
- RTL8139
- TAP - Terminal Access Point

4.1. Boot Loader

O primeiro bloco de código a ser carregado pela BIOS do computador é o bootloader, que deve ser compilado como binário puro e estar localizado nos primeiros bytes na memória secundária. Isso implica também na necessidade de ter exatamente 512 bytes de tamanho, que é justamente o tamanho de um setor de disco. A BIOS, então, carrega aquele setor na memória principal e o bootloader precisa ser capaz de executar o restante do sistema.

Uma vez que o bootloader é executado, ele lê o restante do kernel do disco e leva para um endereço de memória através de interrupts providos pela BIOS, enquanto está em modo 16 bits (real mode). Todo processador da linha x86 é iniciado em 16 bits e cabe ao sistema operacional configurá-lo como necessário. A troca dos modos é feita através de uma GDT (Global Description Table), separando cada sessão da memória a ser utilizada com seus respectivos metadados como: nível de privilégio, endereços de ínício e fim, entre outros. No DonutOS as sessões rodam em modo kernel, permitindo acesso à toda memória com mais altos níveis de privilégio.

Feita a configuração do processador, um *far jump* é criado ou um pulo para outra sessão da memória que, no caso, é o endereço de memória onde a função de entrada do kernel foi carregada previamente. O kernel e o restante do código são compilados na formatação ELF.

4.2. Interrupts e exceções

Para permitir a interação com o usuário, é necessário a implementação de funções para tratamento de interrupções de hardware e, ao longo disso, tratamento de exceções. O tratamento de interrupções funciona com uma IDT (Interrupt Description Table), que consiste de um vetor com entradas armazenando metadados, assim como ponteiros para funções que fazem o tratamento de cada interrupção.

Cada função contida nas entradas da IDT precisa, quando chamada, salvar o estado de todos os registradores de propósito geral no stack antes de fazer o tratamento da

interrupção e, depois, restaurá-los aos valores originais. Isso garante que interrupções possam ser lançadas em qualquer instante da execução sem comprometê-la.

Por fim, são enviados todos os handlers de interrupts para a tabela e ela pode ser instalada salvando seu endereço no registrador dedicado a isso e cada handler para interrupts que vem de peças de hardware devem ser habilitados aplicando sua respectiva flag no PIC (Programmable Interrupt Controller), que gerencia quais interrupt requests estão habilitados.

4.3. Drivers básicos

Com uma tabela de interrupções criada, é necessário mais um passo para interação com o usuário, que são os drivers das várias peças de hardware. Foram implementados os seguintes drivers:

4.3.1. Programmable Interval Timer

O PIT, ou Programmable Interval Timer é uma peça que opera em precisamente 1.193182 MHz. Nessa frequência, o mesmo subtrai um número salvo em um de seus registradores internos indefinidamente. Uma vez que o número chega em 0, uma interrupção é lançada e o registrador é reiniciado ao mesmo número para ser decrementado novamente.

4.3.2. Teclado PS/2

Também foi implementado um driver de teclado PS/2 para permitir um prompt de terminal. Toda vez que uma tecla é pressionada, o teclado solta uma interrupção onde o sistema operacional pode ler qual o código da tecla pressionada que está armazenada em um buffer no teclado.

4.3.3. RTL8139

Para a placa de rede foi utilizada a RTL8139, que é uma placa de ethernet da RealTek. Essa placa foi fundamental para o desenvolvimento do sniffer de rede, sendo explicada e demonstrada sua funcionalidade ao decorrer do trabalho.

4.4. I/O e TTY

O terminal funciona através de uma integração com a placa de VGA, onde é reservado um endereço específico de memória para representar os dados que vão ser exibidos na tela. Cada entrada de caracteres consiste em 2 bytes: o primeiro representa o caractere, o segundo possui 4 bits para definir a cor da fonte e outros 4 bits para a cor do fundo.

4.5. Paginação

Foi implementado paginação com páginas de 1kb, onde um alocador de páginas armazena informações de páginas disponíveis através de um bitmap. A partir disso, cada bit indica se uma página está sendo usada ou não. A paginação é usada para alocação de memória para processos do usuário.

4.6. Processos, Escalonamento e sessões críticas

Cada processo pode ocupar várias páginas e a troca entre processos é feita pelo escalonamento round-robin com um quantum de 10 ms. As operações de I/O feitas entre os processos são controladas por sessões críticas, isso garantindo que vários processos não concorram pelos mesmos dados, prevenindo efeitos colaterais. Processos também tem visibilidade da área baixa de memória (Low memory) para garantir que as funções básicas localizadas próximo ao kernel (como interrupts, drivers) possam continuar sendo acessadas. Ainda assim, cada processo acessa apenas seu próprio endereço de memória.

4.7. Sistema de redes

Para as configurações de rede do sistema operacional foi utilizada a placa RTL8139, uma placa antiga com suporte de 10 a 10Mbit que se caracteriza por ser bastante simples, se tornando ideal para um ambiente de desenvolvimento de sistema operacional.

Para controle da placa foi criado um driver baseado em implementações livres e publicadas na 'os.dev' a partir de sistemas hobbies e compartilhados publicamente, o mesmo foi feito de maneira simplificada respeitando os principios de leveza, legibilidade e escalabilidade.

Para se enviar dados do sistema Host para o sistema Guest, estão sendo utilizadas duas técnicas: Primeiro fazendo apenas um port forwarding para envio de dados para teste da placa e da pilha de protocolos. Segundamente, a partir de determinado momento onde dados mais complexos puderem ser processados, será utilizado uma porta tap com uma bridge(*e.g.* ponte) entre a entrada ethernet do host diretamente para o QEMU.

A partir deste ponto o sistema carrega nativamente a função de listar e abrir todo e qualquer pacote recebido em modo promiscuo para analisar a rede. O mesmo é feito com o intuito de permitir que qualquer pessoa seja capaz de continuar o projeto para uma aplicação personalizada com o contexto que for necessário, mas mantendo a simplicidade para permitir que o sistema seja capaz de continuar possuindo leveza máxima.

5. Métricas e avaliação

Para avaliação do sistema realizamos um teste de utilização de quantidade de memoria ram e analisamos que o mesmo chegou a um gasto máximo de 1056kb, como pode ser obsevado na figura 1. Destes 1056kb, 1024kb são fixos necessários para controle de drivers, além de 34kb de uso do heap do próprio sistema.

Figure 1: Gasto de ram

Outra métrica utilizada é a velocidade do sistema, para isso medimos a quantidade de telas printadas por ciclo do sistema operacional ou simplesmente o fps (frames por

segundo) do sistema, chegando a um total de 1561 atualizações de tela por segundo como mostrado na figura 2.

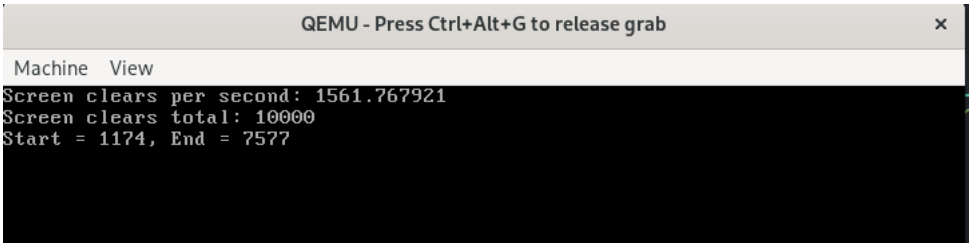

Figure 2: telas por segundo

Para a parte de redes fizemos uma análise da quantidade de pacotes capazes de serem processados por segundo, e com isso foram encontrados que a partir de 10 pacotes o crescimento se torna exponencial explodindo a capacidade de processamento do sistema, levando a perda de pacotes como mostrado na figura 3. O mesmo acontece devido a necessidade de modularização dos pacotes recebidos pela placa, separando a lógica de interrupt do sistema da lógica de processamento dos mesmos. Devido a possibilidade de que um interrupt ocorra durante o processamento de outro, o mesmo deve ser o mais breve possível, o que não ocorre ao momento.

Figure 3: Perda de pacotes por segundo x porcentagem de perda

6. Trabalhos Futuros

É notável que uma gama de aprimoramentos e atualizações serão bem-vindas ao longo do desenvolvimento de Donut OS, que não termina neste trabalho. A partir disso, propomos alguns trabalhos futuros que são prioridade a fim de satisfazer a experiência do usuário e tornar o sistema operacional notável e expressivo: **Portabilidade**, **sistema de acessos remotos ao sistema**, **syscalls e compiladores**, **processador em 64bits mode** e **acelerar o processamento de quadros**.

A fim de justificar a usabilidade do Donut OS, permitir uma portabilidade do sistema operacional para incentivar o usuário a carregar seu ambiente de trabalho para qualquer lugar, se torna justificável. A partir disso, viabilizar acessos remotos ao sistema cria uma maior adequação do sistema a diferentes contextos que o usuário pode encontrar no dia a dia. Syscalls, compiladores e um processador em 64bits mode são propostos para tornar o sistema operacional mais versátil e permitir que o mesmo acompanhe o avanço da tecnologia, visto que 32bits é pouco utilizado nos dias de hoje. Contudo, acelerar o processamento de quadros torna sua usabilidade satisfatória e entrega tudo que é esperado de um sistema operacional para o dia-a-dia.

7. Conclusão

Neste trabalho, nós apresentamos a criação de um sistema operacional escalável. Propomos um ambiente com uma análise detalhada sobre camadas de rede e seus protocolos, além de um escalonamento que torna o sistema operacional leve e viável para diversos processos. As métricas propostas e utilizadas comprovam que a entrega do projeto construído é satisfatória até o momento de escrita do artigo, reconhecendo possíveis melhorias a serem feitas. Criar um panorama como o apresentado demonstra os desafios e detalhes na criação de um sistema operacional, contribuindo com a comunidade da computação que busca por incentivos e aprendizados nas mais diversas áreas de aprendizado.

References

Erik Dubbelboer. Os tutorial. https://github.com/erikdubbelboer/OS-Tutorial/. Acessado em: 14 de março de 2024.

D. Henrique Pereira. Itamaracá: A novel simple way to generate pseudo-random numbers. *Cambridge Open Engage*, 2022. doi: 10.33774/coe-2022-zsw6t. Este conteúdo é um preprint e não foi submetido à revisão por pares. Acessado em: 14 de março de 2024.

Making an OS (x86). Playlist: Making an os (x86). https://www.youtube.com/playlist?list=PLm3B56ql_akNcvH8vvJRYOc7TbYhRs19M. Acessado em: 14 de março de 2024.

Rômulo Silva De Oliveira, Alexandre Da Silva Carissimi, and Simão Sirineo Toscani. *Sistemas Operacionais*. Bookman, 4 edition, 2010.

OSDev Wiki. Rtl8139. https://wiki.osdev.org/RTL8139. Acessado em: 02 de maio de 2024.

OSDev Wiki. Osdevwiki. https://wiki.osdev.org/Main_Page. Acessado em: 14 de março de 2024.

Pedro Henrique Penna. Nanvix operating system. https://github.com/nanvix/nanvix, 2017. Acessado em: 14 de março de 2024.

Andy Sloane. donut.c without a math library. https://www.a1k0n.net/2021/01/13/optimizing-donut.html, 2021. Acessado em: 14 de março de 2024.

szhou42. edu os. https://github.com/szhou42/osdev/tree/master. Acessado em: 02 de maio de 2024.

Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall, Boston, 5 edition, 2011. ISBN 978-0-13-212695-3. Acessado em: 14 de março de 2024.

TetrisOS. I made an entire os that only runs tetris. https://www.youtube.com/watch?v=FaILnmUYS_U. Acessado em: 14 de março de 2024.

Yanhui Zhao. Osdev repository. https://github.com/yzz127/osdev/tree/master. Acessado em: 14 de março de 2024.