RELATÓRIO TRABALHO 1

Author(s): Samuel de Oliveira Krabbe

Introdução

Este relatório aborda o processo de extração de um programa de boot e um kernel de arquivos ELF, seguido pela geração de um arquivo de imagem para simular seu carregamento em memória. Essa tarefa é essencial para entender como o sistema operacional interage com o hardware durante a inicialização.

Ao longo deste relatório, serão discutidas as decisões de implementação, funcionalidades implementadas, bem como os desafios enfrentados durante o processo.

O objetivo deste trabalho é fornecer uma compreensão mais profunda do funcionamento interno e do processo de inicialização de sistemas operacionais.

Decisões de Implementação

Para a implementação da extração de arquivos ELF utilizando ferramentas de manipulação de arquivos binários. Isso incluiu o uso de funções de leitura de arquivos em C/C++, bem como a utilização de ferramentas específicas para a análise e extração de seções relevantes dentro do arquivo ELF.

Para a implementação da geração do arquivo imagem também utilizamos funções de leitura e escrita de arquivos da biblioteca em C/C++, fora isso, decidimos imprimir todas as informações que estão nas estruturas de dados de cabeçalhos de programa já definidos na biblioteca <elf.h>, tanto informações do bootblock quanto do kernel.

Para fazer o cálculo do número de setores no kernel e no bootblock criamos duas funções, assim o código ficou mais conciso e de fácil leitura, além disso, criamos mais funções tanto para ler o ELF header e program header do bootblock e do kernel separadamente quanto para escrever esses dois arquivos ELF na imagem.

Para a geração da imagem foi criado mais uma função que faz todo o processo de leitura dos headers do bootblock e do kernel e escreve na imagem, essa função só foi criada para deixar o código mais limpo e atômico.

Por fim, para armazenar todas as variáveis que iríamos utilizar, criamos uma estrutura de dados chamada pacote definida no início do código fonte, esse pacote faz com que a passagem de argumentos para as funções fique menor, deixando o código menos complexo. Além disso, para que fosse feito os testes na linha de comando da melhor maneira possível criamos um novo *target* no arquivo Makefile para podermos construir a imagem utilizando o nosso código.

Funcionalidades Implementadas

Leitura dos Headers ELF:

- As funções ***read_bootblock_ehdr*** e ***read_kernel_ehdr*** são responsáveis por ler os cabeçalhos ELF dos arquivos de boot e do kernel, respectivamente. Isso é feito alocando memória para os cabeçalhos ELF e lendo-os dos arquivos correspondentes.

Leitura dos Program Headers ELF:

- As funções ***read_bootblock_phdr*** e ***read_kernel_phdr*** são responsáveis por ler os program headers ELF dos arquivos de boot e do kernel, respectivamente. Essas funções também alocam memória para os program headers e os lêem dos arquivos correspondentes.

Escrita do Bootblock e do Kernel no Arquivo de Imagem:

- As funções ***write_bootblock*** e ***write_kernel*** são responsáveis por ler os dados do bootblock e do kernel de seus arquivos correspondentes e escrevê-los no arquivo de imagem, respeitando os deslocamentos e tamanhos especificados nos headers ELF.

Contagem de Setores do Bootblock e do Kernel:

- As funções ***count_bootblock_sectors*** e ***count_kernel_sectors*** são responsáveis por calcular o número de setores necessários para armazenar o bootblock e o kernel, respectivamente. Isso é feito com base no tamanho dos arquivos de boot e kernel e no tamanho de um setor, que é definido como 512 bytes.

Gravação do Número de Setores do Kernel no Arquivo de Imagem:

- A função ***record_kernel_sectors*** grava o número de setores ocupados pelo kernel no arquivo de imagem, após ter sido calculado na função ***count_kernel_sectors.***

Construção do Arquivo de Imagem:

- A função ***build_image*** é responsável por coordenar o processo de construção do arquivo de imagem, que inclui a escrita do bootblock e do kernel, bem como a gravação do número de setores ocupados pelo kernel.

Opção --extended:

- A função ***extended_opt*** é chamada quando o programa é executado com a opção *--extended*. Ela imprime informações detalhadas sobre os segmentos do bootblock e do kernel, incluindo seus tipos, endereços, tamanhos, flags e alinhamentos.

Novo target no arquivo Makefile:

- Para podermos fazer os testes sem muita demora e priorizando eficiência, criamos um novo target no arquivo Makefile chamado ***imagesamuel***, esse target faz a construção da imagem usando o nosso código fonte desenvolvido no trabalho e não o código ***buildimage.given*** dado pelo professor, desse modo, foi utilizado esse target no ***all*** ao invés do que estava sendo usado antes, que era o ***image***.

Problemas Enfrentados na Implementação

Um dos primeiros desafios que encontramos foi entender o que é um arquivo ELF, como ele se estruturava e como poderíamos manipulá-lo a fim de realizar o objetivo do trabalho, para isso tivemos que ler a documentação do formato ELF que foi passado no AVA.

Após o desafio inicial, tivemos uma segunda barreira, o arquivo bootblock.o não tinha program header, logo não conseguiríamos manipulá-lo com foi pedido, para resolver isso foi perguntado ao professor o que deveríamos fazer, a resposta que obtivemos foi que não era o arquivo bootblock.o que deveríamos manipular e sim o arquivo bootblock somente, resolvida esse dúvida o trabalho começou a andar mais rapidamente.

Finalizando a implementação do código fonte nos deparamos com um último problema, a impressão dos valores do bootblock e do kernel, feito na função ***extended_opt*** a partir do argumento *–extended* na linha de comando, para resolvermos esse problema tivemos que utilizar o arquivo exemplo que o professor nos concedeu para entendermos o que exatamente deveria ser impresso, feito isso foi feita uma última rodada de testes para ver se o nosso programa estava batendo com o do professor e finalizamos o trabalho.

Conclusão

Neste trabalho, abordamos o processo de extração de um programa de boot e um kernel de arquivos ELF, seguido pela geração de um arquivo de imagem para simular seu carregamento em memória. Essa tarefa foi essencial para compreender como o sistema operacional interage com o hardware durante a inicialização. Ao longo do desenvolvimento, foram tomadas diversas decisões de implementação visando eficiência, simplicidade e facilidade de manutenção.

Apesar dos desafios encontrados, como compreender o formato ELF e a estrutura dos arquivos de boot e kernel, assim como a necessidade de ajustar a abordagem quando nos deparamos com a ausência de program header no arquivo bootblock.o, conseguimos superá-los com a ajuda das referências disponíveis e do suporte do professor.

Em suma, este trabalho proporcionou uma compreensão mais profunda do funcionamento interno de sistemas operacionais e do processo de inicialização de sistemas embarcados. A experiência adquirida ao enfrentar os desafios encontrados durante a implementação contribuiu significativamente para o nosso aprendizado e desenvolvimento profissional na área de sistemas operacionais e programação de baixo nível.

Referências

- Documentação - formato ELF (PDF);
- Slides de aula sobre o trabalho;
- Bibliotecas padrão da linguagem C: <assert.h>, <elf.h>, <errno.h>, <stdarg.h>, <stdio.h>, <stdlib.h>, <string.h>;