Trabalho Prático
Sistemas Operacionais

RELATÓRIO 3

Relatório Final

RTL (Real-Time Loading)

Adler Melgaço Ferreira

Hiago de Souza Cruz Alves e Silva

Pedro Elias Valadares Castanheira

Ronald Davi Rodrigues Pereira

1 Introdução

É fato que, atualmente, os computadores pessoais estão bastante populares na sociedade. Cada usuário tem um perfil e utiliza seu sistema de acordo com seus interesses. Grande parte desses usuários segue uma “rotina”, isto é, ao iniciar o sistema, abre um conjunto padrão de processos que usa com frequência, sejam eles navegadores, jogos, softwares de comunicação, editores, entre outros.

A partir disso, viu-se a possibilidade de tornar essa rotina mais eficiente. A proposta de alteração do kernel é uma ferramenta chamada Real-Time Loading (RTL), que visa identificar o conjunto de processos que cada usuário utiliza com frequência e inicializá-los previamente de forma automática e prioritária, garantindo uma melhor eficiência para o usuário.

Esse relatório contém a conclusão do desenvolvimento dessa aplicação e os detalhes sobre ela, como o seu funcionamento, os problemas e desafios que foram encontrados e o que foi feito para superá-los.

2 Objetivo

Este trabalho teve o objetivo de aumentar os conhecimentos dos participantes sobre o sistema operacional em questão, por meio da proposta de uma alteração no mesmo. Para que essa alteração fosse feita, no entanto, foi necessário que os integrantes do grupo entendessem o funcionamento do sistema como um todo, para conseguir minimizar os impactos em lugares indesejados gerados pela alteração.

Ao mesmo tempo, buscou-se desenvolver uma aplicação viável ao sistema alterado, mesmo que em áreas bem específicas. Dessa forma, houve uma discussão entre o grupo para elegermos uma alteração que, aos nossos olhos, pudesse melhorar a experiência do usuário com o sistema operacional. A partir disso, surgiu a proposta tema deste trabalho.

Também vale ressaltar, que, ao longo de todo o desenvolvimento da ferramenta, o grupo tentou fazer com que a aplicação pudesse não só servir para este trabalho, mas também

para ser utilizada pelos integrantes e demais colegas futuramente. Portanto, nossas prioridades foram sempre uma melhor ergonomia e customização da ferramenta, de modo que ela possa ser realmente aproveitada como benefício para o sistema operacional como um todo e não somente como trabalho prático.

---

3 Desenvolvimento

Em inicialização de processos, um dos maiores gargalos de eficiência é a troca de contexto de um processo em execução para um outro prestes a ser iniciado. Desse modo, ao realizar o carregamento prévio de partes essenciais de um processo, facilita-se a troca de contexto e, consequentemente, diminui-se o tempo para a sua inicialização.

Portanto, para suprir essa necessidade, está sendo utilizada uma ferramenta desenvolvida pelo grupo para inicializar esses processos de uma maneira customizada pelo usuário. Essa ferramenta, depois de testada separadamente, foi integrada ao kernel para ser inicializada automaticamente.

3.1 Real-Time Loading (RTL)

A ferramenta consiste em uma estrutura de dados “Command”, que é constituída de uma string para armazenar o comando para inicialização do processo e um contador de eventos (quantas vezes o processo foi executado/incluído). Esse contador é utilizado para priorização e customização de inicialização de processos, de forma que os processos que possuem um maior número de eventos sejam prioritários frente a outros com um menor número de eventos.

A ferramenta possui 3 modos distintos, diferenciáveis pelos parâmetros passados por linha de comando:

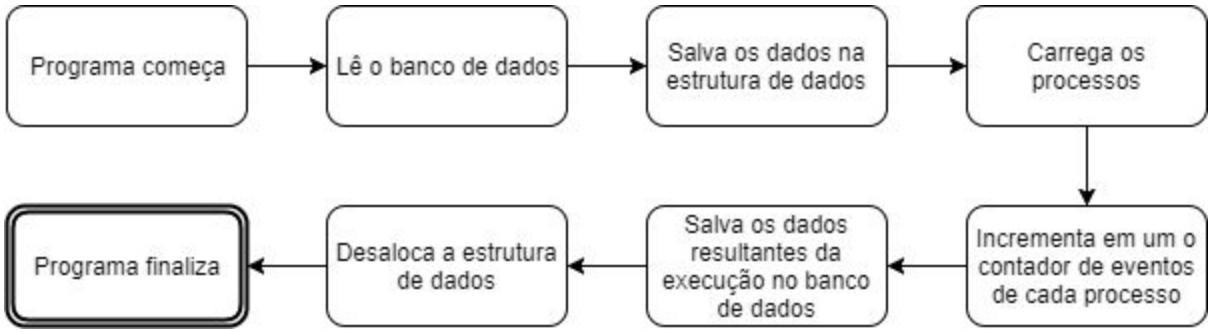
1. ./realtimeloading.exe init → Executa a criação de um banco de dados vazio “.rtldatabase.db”, onde serão armazenados os processos, de forma a manter a informação de configuração e execução posteriormente à execução da ferramenta.
2. ./realtimeloading.exe config → Modo de customização do usuário por meio de um menu de configuração. Esse modo contém opções de impressão dos processos armazenados

no banco de dados do programa, bem como a inclusão manual, exclusão ou a modificação do tamanho máximo de processos que podem ser armazenados.

3. ./realtimeloading.exe → Modo padrão e automático de inicialização de processos contidos no banco de dados. Esse modo será executado automaticamente a cada inicialização do sistema operacional.

A ferramenta também possui um gerador de log de erros que a execução do programa pode encontrar, como por exemplo tentar inserir um novo processo com o banco de dados lotado. Desse modo, caso algum erro seja detectado em tempo de execução do programa, ele será guardado em um arquivo de texto, evitando perda de informações.

A execução sequencial dessas tarefas acontece na seguinte ordem:



3.2 Integração ao kernel

Primeiramente, foi criado uma ferramenta externa ao kernel, de modo que pudéssemos realizar compilações e testes no software de maneira mais rápida e eficiente. Essa ferramenta foi desenvolvida na linguagem C e, inicialmente, não possuía nenhuma ligação ou função integrada com o kernel do Linux.

Após a finalização do desenvolvimento do programa externo, realizamos alguns testes somente para verificar se a ferramenta criada realmente possuía resultados satisfatórios de acordo com as expectativas dos integrantes do grupo. Esses testes serão anexados na seção “Testes”.

Finalmente, após algumas dificuldades enfrentadas pelo grupo, foi possível integrar a ferramenta ao kernel. Isso se deu por meio de um guia de criação de módulos do kernel [1] , que explica passo-a-passo de como inserir um novo módulo do kernel.

Portanto, para conseguirmos inserir a ferramenta como um módulo interno ao kernel, tivemos que realizar algumas modificações no código fonte do programa. A primeira modificação foi incluir as bibliotecas:

```
1   // Real-Time Loading
2
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6   #include <time.h>
7   #include <unistd.h>
8   #include <linux/module.h> // Included for all Linux Modules
9   #include <linux/kernel.h> // Included for KERN_INFO
10  #include <linux/init.h> // Included for __init and __exit macros
```

As 5 primeiras bibliotecas foram utilizadas no desenvolvimento do programa. A biblioteca <linux/module.h>, é uma biblioteca padrão para todos os módulos do Linux, contendo algumas funções de organização externa, como por exemplo:

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ronald Pereira");
MODULE_DESCRIPTION("Real-Time Loading Module");
```

A segunda biblioteca importada, serve para importar informações sobre o kernel, utilizando da palavra chave KERN_INFO. A terceira biblioteca importada contém macros indispensáveis para a inicialização de um módulo interno ao kernel, de modo que seja automático. Essas macros são utilizadas em funções de inicialização e finalização do módulo:

[1] http://www.thegeekstuff.com/2013/07/write-linux-kernel-module/

```
394  static int __init rtl_init(int argv, char *argc[])
395  {
396      if(argc[1] == NULL)
397      {
398          printk(KERN_INFO "\nExecuting Real-Time Loading. Please wait...\n");
399          rtl();
400          printk(KERN_INFO "\nReal-Time Loading ended\n");
401      }
402
403      else if(strcmp(argc[1], "init") == 0)
404      {
405          printf("\n\n");
406          printf("********************\n");
407          printf("*                  *\n");
408          printf("* Real-Time Loading *\n");
409          printf("*                  *\n");
410          printf("********************\n");
411
412          printf("\nInitializing Real-Time Loading database file...\n");
413          init();
414          printf("Real-Time Loading ready!\n");
415      }
416
417      else if(strcmp(argc[1], "config") == 0)
418      {
419          printf("\n\n");
420          printf("********************\n");
421          printf("*                  *\n");
422          printf("* Real-Time Loading *\n");
423          printf("*                  *\n");
424          printf("********************\n");
425
426          printf("\nWelcome to Real-Time Loading configuration.\n");
427          config();
428      }
429
430      return 0;
431  }
```

```
433  static void __exit rtl_cleanup()
434  {
435      printk(KERN_INFO "Cleaning up module\n");
436  }
437
```

E esses módulos são chamados novamente por funções dessa biblioteca, de modo a executar o módulo a cada inicialização do sistema:

```
438    module_init(rtl_init);
439    module_exit(rtl_cleanup);
```

Para uma melhor organização dos dados no kernel, de modo a evitar vazamentos de memória, foi utilizada uma estrutura de dados simples, porém eficiente, de modo a armazenar todos os dados dos comandos contidos no banco de dados externo do programa:

```
typedef struct
{
    char cmd[100000];
    unsigned int events;
} Command;
```

Cmd → String para guardar o nome exato do comando a ser executado
Events → Contador de quantas vezes o processo foi executado e inserido no banco de dados.

O próximo passo foi criar um Makefile compatível com um módulo de kernel, de modo a pegar o objeto do programa criado pela compilação externa da ferramenta:

```
1  obj-m += rtl.o
2
3  all:
4      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6  clean:
7      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
8
```

Com isso, a compilação do módulo é feita conjuntamente com os módulos dependentes dele, gerando um arquivo rtl.ko (kernel object).

A partir desse ponto, só foi necessário inserir esse objeto criado na pasta do kernel do linux, criando um novo diretório rtl/, contendo somente esse arquivo.

Para uma melhor utilização da ferramenta, foi inserido também um README contendo o comando: ln -s /linux-kernel/kernel/modules/rtl/rtl.ko /usr/local/bin, de modo a criar um link binário no sistema para executar as opções de configuração mais facilmente.

3.3 Resolução de problemas anteriores

Ao longo do desenvolvimento da ferramenta, foram descobertos alguns possíveis problemas com que teríamos que lidar durante o restante do trabalho. O primeiro desses problemas foi o de mensurar o tempo que cada processo demora para ser inicializado, ou seja, o tempo para acontecer a troca de contexto ou, pelo menos, a alocação dos recursos necessários à inicialização dele. Esse problema foi resolvido com uma criação de um script em shell (que pôde ser colocado no programa em C):

- ➢ until pids=$(pidof <processo>); do sleep 0.1; done

Esse programa utiliza de uma variável local “pids”, de modo que enquanto ela não receber o PID do processo a ser inicializado, o loop continua, deixando um tempo de ociosidade de 0.1 segundos, através do comando sleep 0.1. Com isso, esse loop é interrompido assim que o processo é inicializado no sistema, podendo continuar com a execução correta do programa, independentemente da velocidade do sistema.

O segundo problema era o de integrar a ferramenta ao kernel do Linux sem que haja qualquer tipo de interferência no funcionamento do sistema operacional. Para isso, fizemos pesquisas e leituras de documentações do kernel, de modo a proporcionar ao grupo uma visão mais específica e especialista do assunto de integração com o kernel. Essas documentações, no entanto, são complexas e estão espalhadas, o que configurou mais uma barreira à compreensão do funcionamento do kernel e como alterá-lo. Porém, com o guia já citado acima, foi possível inserir essa ferramenta como um módulo do kernel.

Portanto, o grupo pôde concluir o trabalho com sucesso, de modo que os únicos problemas restantes se resumem a limitações e possíveis complicações que poderão ser encontradas no futuro.

3.4 Problemas futuros

Os possíveis problemas futuros que a nossa ferramenta poderá enfrentar são os seguintes:

I. O banco de dados é feito por meio de um documento de texto. Portanto, caso o banco de dados cresça muito, pode ser que fique muito pesado para o módulo carregar, podendo acarretar em uma inicialização mais lenta do sistema.
II. Pode ser que, no futuro, exista uma otimização de hardware para essa finalidade (otimização de inicialização de processos) e essa ferramenta fique obsoleta, acarretando em uma nova alteração no kernel, apenas para retirar essa ferramenta.

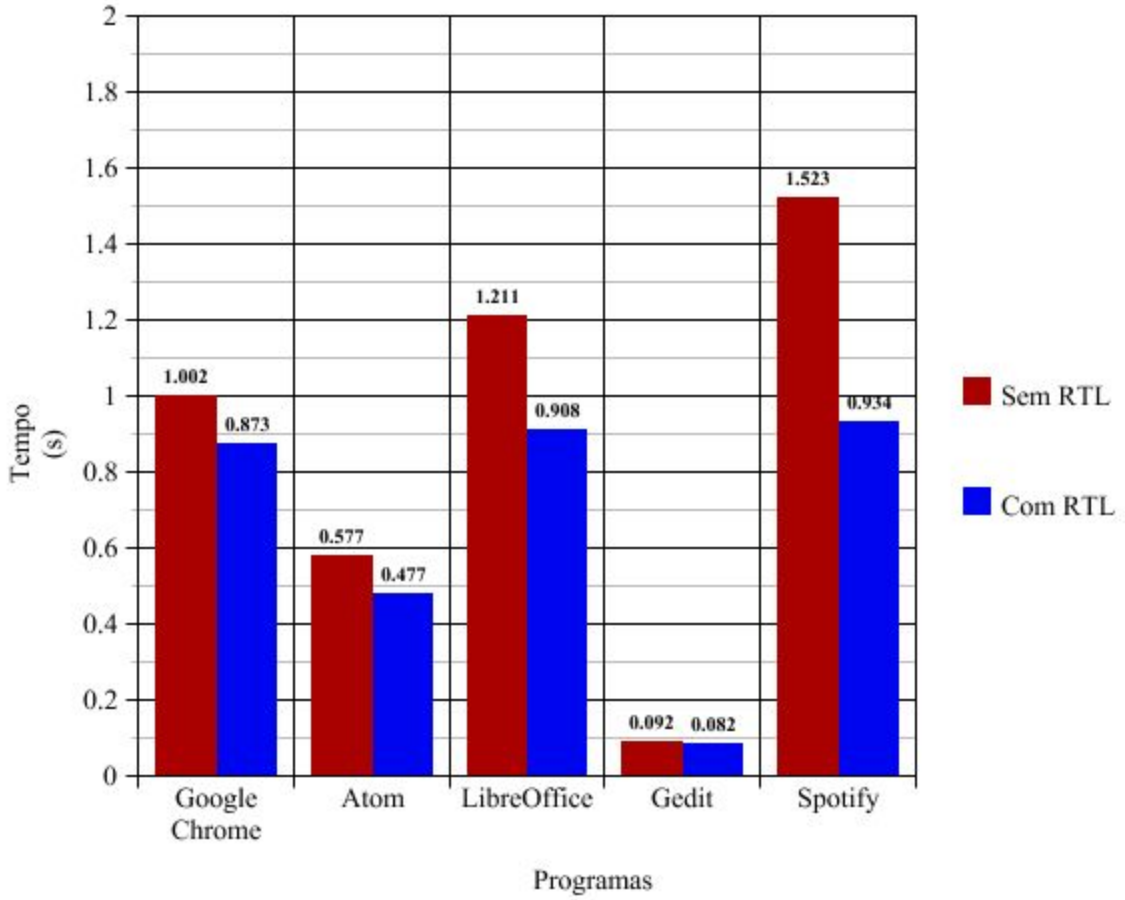
4 Testes

Não foram utilizados benchmarks para realização destes testes. Isso se deve ao fato de que benchmarks são um conjunto de programa pré-definidos para testes, o que não está nem um pouco alinhado com o objetivo da nossa ferramenta, que é a otimização de tempo de inicialização de processos recorrentes de cada usuário. Portanto, preferimos utilizar programas que possuem uma maior utilização por parte de usuários comuns ao nosso meio acadêmico para esses testes, de modo a expressar melhor os resultados frente ao nosso objetivo principal.

Os testes foram realizados por meio de um comando já contido no Linux: time. Esse comando serve para medir o tempo que um processo demora para ser inicializado, contendo também outras informações, que não são relevantes para o nosso objetivo. Com isso, foram feitos dois tipos de testes:

1. Inicializa o sistema e abre os programas por linha de comando, precedido por “time”.
2. Inicializa o sistema com o kernel alterado, executando automaticamente o Real-Time Loading e, após isso, abre os programas por linha de comando, precedido por “time”.

Com esses passos citados anteriormente, pudemos obter tempos de inicialização de processos, conforme o gráfico expresso abaixo:



5 Resultados

Conforme os resultados obtidos acima, podemos observar uma melhora de , aproximadamente, 26% no tempo de inicialização dos processos utilizados no teste. Com isso, o grupo alcançou o objetivo esperado com essa ferramenta, que era obter um mínimo de melhoria no tempo de inicialização de processos. Portanto, o resultado foi satisfatório, levando em consideração o amadorismo dos integrantes do grupo desenvolvendo ferramentas para o kernel do Linux e o tempo hábil de entrega do trabalho prático.

6 Conclusão

A proposta inicial era desenvolver uma modificação no kernel do linux que melhorasse o tempo médio de inicialização dos processos mais utilizados pelo usuário a partir de uma pré-inicialização, gerando uma experiência de ganho na eficiência. Os resultados alcançados nesse contexto foram bastante satisfatórios, já que os tempos de inicialização desses processos foram reduzidos com sucesso utilizando o RTL.

O trabalho agregou bastante experiência ao grupo em relação ao funcionamento dos processos no kernel, principalmente sobre inicialização e manipulação dos mesmos. Além disso, as dificuldades encontradas para acoplar as alterações ao restante do sistema também foram proveitosas, já que nos proporcionaram uma maior familiarização com a compilação do kernel e sua estrutura em geral.

7 Referências Bibliográficas

https://en.wikipedia.org/wiki/Linux

https://www.linux.com/what-is-linux

https://en.wikipedia.org/wiki/Kernel_(operating_system)

https://github.com/torvalds/linux

http://linuxcommand.org/man_pages/ps1.html

http://www.thegeekstuff.com/2013/07/write-linux-kernel-module/