Escola Politécnica da Universidade de São Paulo

Departamento de Engenharia de Computação e Sistemas Digitais

Gabriel Corteletti Prezotti Palassi – NUSP 11820242
Jaques Missrie – NUSP 11871326
Beatriz Pama – NUSP 11914052
João Luiz Giglio Laudissi - NUSP 11805929

GCC Playground: Um Ambiente de Desenvolvimento C no Navegador

Laboratório de Processadores – PCS3732

Prof. Bruno Basseto

Prof. Carlos Cugnasca

Sumário

Introdução

Com o avanço das tecnologias web e a crescente demanda por ferramentas educacionais interativas, ambientes de desenvolvimento online têm se tornado cada vez mais relevantes no ensino e prática da programação. Nesse contexto, o projeto GCC Playground surge como uma proposta inovadora: um compilador C moderno acessível diretamente pelo navegador, com recursos avançados de edição, compilação e análise de código, desenvolvido no escopo da disciplina *PCS3732 - Laboratório de Processadores* da Escola Politécnica da Universidade de São Paulo.

A justificativa para a criação deste ambiente reside na necessidade de fornecer uma experiência de programação prática, intuitiva e robusta, sem as barreiras tradicionais de instalação e configuração de ambientes de desenvolvimento locais. Ao integrar tecnologias como *React*, *TypeScript* e *Monaco Editor* com ferramentas de compilação nativas como o *GCC*, o projeto se alinha com práticas modernas de engenharia de software e ensino de computação.

O objetivo principal do trabalho é desenvolver uma aplicação web funcional que permita aos usuários escrever código em C, configurar opções de compilação e analisar diferentes formas de saída, incluindo mensagens de erro, código assembly e representação binária. Além disso, busca-se oferecer uma experiência fluida e segura, integrando boas práticas de desenvolvimento frontend e backend.

Desenvolvimento

1. Fundamentação Teórica

Ambientes online de desenvolvimento (Online IDEs) têm ganhado espaço por sua acessibilidade, portabilidade e integração com tecnologias modernas. Ferramentas como o *Monaco Editor*, base do Visual Studio Code, oferecem uma experiência rica de edição no navegador, enquanto bibliotecas e frameworks como *React* e *Vite* facilitam o desenvolvimento de interfaces responsivas e modulares. No backend, o uso do *Node.js* com *Express* permite a criação de APIs robustas e escaláveis.

O compilador *GCC* (GNU Compiler Collection), por sua vez, é um dos compiladores mais amplamente utilizados em ambientes Unix-like, reconhecido por sua conformidade com padrões, desempenho e flexibilidade. A integração entre GCC e aplicações web, embora desafiadora, possibilita análises detalhadas do processo de compilação, como a geração de código de máquina (*assembly dump*) e inspeção de binários (*hexdump*).

1.1 GCC e Configuração do Processo de Compilação

O GCC (GNU Compiler Collection) é uma coleção de compiladores desenvolvida inicialmente para o projeto GNU. Ele é amplamente utilizado para compilar linguagens como C, C++, Objective-C, Fortran, Ada e outras. No contexto do GCC Playground, o GCC é utilizado exclusivamente como compilador C, executado no backend da aplicação para transformar o código-fonte enviado pelo usuário em arquivos objeto e binários, bem como para extrair representações intermediárias como *assembly* e *hexdump*.

O compilador GCC é conhecido por sua portabilidade, conformidade com os padrões ISO da linguagem C e por oferecer uma grande variedade de opções de compilação e otimização, que podem alterar significativamente a forma como o código é traduzido para linguagem de máquina.

1.1.1. Processo de Compilação com o GCC

O pipeline completo do GCC envolve quatro fases principais:

1. **Pré-processamento** (*cpp*): Expansão de diretivas *#include*, *#define*, etc.;
2. **Compilação** (*cc1*): Tradução de código-fonte C para código assembly;
3. **Montagem** (*as*): Conversão de código assembly em código de máquina (objeto);
4. **Linkagem** (*ld*): União de múltiplos arquivos objeto em um executável final.

No GCC Playground, o pipeline vai até a produção do executável binário, porém sem realizar a execução.

1.1.2. Flags de Compilação Suportadas

A plataforma permite que os usuários escolham algumas das principais flags de compilação, que são refletidas diretamente no comando *gcc* executado no backend. A seguir, detalha-se o efeito de cada uma:

- ***-Wall***: Ativa uma ampla gama de avisos (*warnings*) comuns. Embora não interrompa a compilação, essa flag é fundamental para detectar práticas de programação inseguras ou bugs sutis. Inclui:
    - Uso de variáveis não inicializadas;
    - Atribuições em expressões condicionais;
    - Conversões implícitas entre tipos incompatíveis;
    - Comparações suspeitas.
- ***-Werror***: Promove todos os avisos (*warnings*) a erros (*errors*). Quando usada em conjunto com -*Wall*, impede a geração do binário caso qualquer aviso seja detectado. É comum em contextos de produção e ambientes com integração contínua (CI).
- ***-g***: Inclui informações de debug no binário gerado, permitindo que ferramentas como *gdb* (GNU Debugger) realizem depuração simbólica. Essa flag não altera a semântica do programa, mas insere metadados úteis para rastreamento de variáveis, linhas de código, nomes de funções etc.
- ***-static***: Força o link estático das bibliotecas padrão, evitando dependências dinâmicas no sistema. O binário gerado inclui todas as bibliotecas necessárias, o que aumenta seu tamanho, mas garante portabilidade e previsibilidade de execução. Pode ser necessário quando o binário será usado em sistemas sem bibliotecas compartilhadas compatíveis.

1.1.3. Níveis de Otimização

O GCC permite selecionar entre diversos níveis de otimização por meio da flag -*O*, que pode afetar desde a performance do programa até a legibilidade do código assembly gerado. No projeto, a plataforma suporta:

Nível	**Descrição**
-O0	Sem otimizações. Preserva estrutura original do código, ideal para depuração. Gera código legível e mapeável diretamente ao fonte C.
-O1	Habilita otimizações básicas de tempo de execução e espaço. Preserva tempo razoável de compilação. Remove código morto e simplifica expressões.
-O2	Otimizações mais agressivas, sem sacrificar a compatibilidade. Inclui *loop unrolling*, *strength reduction* e otimizações de fluxo de controle.

-O3	Otimizações máximas para desempenho, incluindo *function inlining* agressivo, vetorização e duplicação de loops. Pode aumentar significativamente o tamanho do binário.
-Os	Otimiza para tamanho em vez de desempenho. Baseado em -O2, mas com remoções adicionais de código redundante. Útil em sistemas embarcados.
-Ofast	Inclui todas as otimizações de -O3, mais otimizações que podem quebrar conformidade com o padrão ISO C (ex.: reordenamento de cálculos de ponto flutuante, remoção de verificações de NaN). Usado em contextos de alta performance com tolerância a imprecisões numéricas.

Essas otimizações têm impacto direto nas saídas observadas na aba de assembly do GCC Playground, permitindo ao usuário comparar a transformação do código-fonte em função das opções selecionadas.

2. Metodologia

O projeto foi desenvolvido com arquitetura *frontend-backend* separada. No frontend, foi utilizado o framework React 18 com TypeScript e Vite como ferramenta de build. A interface gráfica adota o paradigma *component-based* e faz uso extensivo da biblioteca *shadcn/ui*, Tailwind CSS para estilização e Lucide React para ícones SVG.

O editor de código foi implementado utilizando o *Monaco Editor*, com destaque de sintaxe para C, recursos de *IntelliSense* e um tema visual personalizado.

No backend, um servidor Express.js foi criado para intermediar as requisições entre o navegador e o compilador local. As principais funcionalidades do servidor incluem:

- Validação de entradas (middleware de segurança)
- Execução de comandos GCC via *child_process*
- Geração de dumps em *assembly* e *binário*
- Tratamento de erros com respostas HTTP apropriadas
- Aplicação de medidas de segurança como CORS, Helmet e *rate limiting*

A comunicação entre cliente e servidor se dá por meio de chamadas HTTP para endpoints RESTful, com destaque para o endpoint */api/compile*, que processa o código-fonte enviado e retorna as saídas esperadas.

Durante todo o desenvolvimento, utilizamos o sistema de controle de versão *Git*, tanto para registro do histórico de modificações quanto para organização do trabalho em branches e colaboração.

3. Funcionalidades

As funcionalidades principais do sistema podem ser agrupadas em três categorias:

3.1. Edição de Código

- Editor Monaco com suporte completo a C
- Sugestões automáticas e fechamento inteligente de parênteses
- Indentação e correspondência automática de blocos de código

3.2. Configuração do Compilador

- Suporte a flags do GCC como *-Wall*, *-Werror*, *-g* e *-static*
- Seleção de níveis de otimização: *O0*, *O1*, *O2*, *O3*, *Os*, *Ofast*
- Exibição do comando de compilação gerado dinamicamente

3.3. Análise de Resultados

- Saída de compilação: mensagens de erro ou sucesso
- Dump do código assembly com *objdump*
- Dump do binário gerado com *hexdump*

4. Implementação

4.1. Arquitetura Geral

O GCC Playground adota uma arquitetura dividida em duas camadas:

- **Frontend**: desenvolvido com React e TypeScript, responsável pela edição do código, configuração das opções de compilação e exibição das saídas;
- **Backend**: desenvolvido com Node.js e Express, responsável por processar o código, executar o compilador e gerar as saídas secundárias (assembly e binário).

A comunicação entre as camadas é feita via HTTP com payloads JSON, através do endpoint */api/compile*.

4.2. Implementação do Frontend

O frontend é estruturado como uma *single-page application* em React 18. O componente principal é *App.tsx*, que gerencia os seguintes estados:

- *code*: o conteúdo editado no Monaco Editor;

- *flags*: objeto com as opções de compilação definidas pelo usuário;
- *result*: objeto contendo as saídas da compilação;
- *isCompiling*: booleano de controle para estados de carregamento.

4.2.1. Editor de Código

O editor é baseado no Monaco Editor, com as seguintes funcionalidades habilitadas:

- Destaque de sintaxe C;
- Autocompletar básico;
- Fechamento automático de parênteses e aspas;
- Tema personalizado via *monacoTheme.ts.*

4.2.2. Fluxo de Compilação

Ao acionar o botão "Compilar", a função *handleCompile()* é invocada. Ela:

1. Valida o conteúdo atual;
2. Monta um objeto JSON com *code* e *flags*;
3. Envia esse objeto via *axios* para o backend;
4. Atualiza o estado *result* com a resposta.

4.2.3. Exibição de Resultados

O resultado retornado do backend é apresentado em três abas:

- **Saída**: mensagens do GCC (*stdout* e *stderr*);
- **Assembly**: resultado do comando *objdump -d*;
- **Binário**: resultado do comando *hexdump -C*.

Essas abas utilizam instâncias do Monaco Editor em modo de leitura.

4.3. Implementação do Backend

O backend é implementado com Node.js e utiliza o framework Express. Ele expõe o endpoint */api/compile*, que processa requisições POST com o código-fonte e as opções de compilação.

4.3.1. Middleware de Segurança

O servidor aplica medidas preventivas de segurança:

- **helmet**: configura headers HTTP seguros;
- **cors**: controle de origens permitidas;
- **rate-limit**: até 100 requisições por IP a cada 15 minutos;
- **body-parser**: validação do corpo da requisição;
- **errorHandler**: captura e formata erros no padrão JSON.

4.3.2. Serviço de Compilação

Toda a lógica de compilação reside no módulo *compiler.js*. A função *compileCode()* realiza as seguintes etapas:

1. Cria um diretório temporário com *uuid*;
2. Escreve o arquivo *main.c* com o código enviado;
3. Monta o comando *gcc* com as flags selecionadas;
4. Executa o compilador via *child_process.exec*;
5. Em caso de sucesso, executa:
    a. *objdump -d* para obter o código assembly;
    b. *hexdump -C* para gerar a saída binária;
6. Retorna um objeto com os campos:

Se houver erro de compilação, o campo *success* será *false* e o campo *output* conterá a mensagem de erro do GCC.

4.3.3. Gerenciamento de Arquivos Temporários

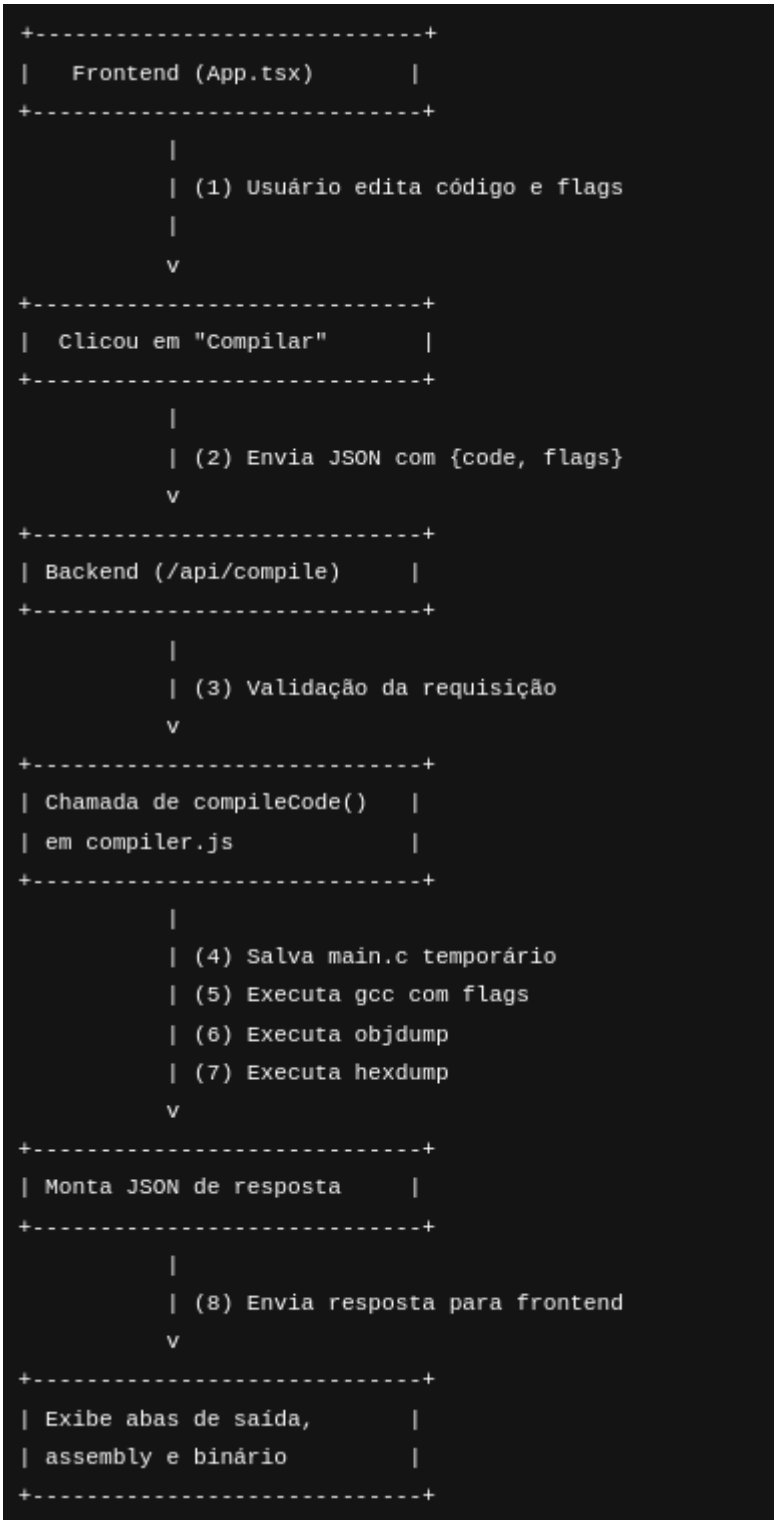
Após o término da compilação, os arquivos temporários são removidos do sistema usando o módulo *fs/promises*, garantindo segurança e limpeza do ambiente.

4.4. Comunicação Cliente-Servidor

O fluxo completo da aplicação pode ser descrito da seguinte forma:

1. O usuário edita o código no Monaco Editor;
2. Ao clicar em "Compilar", o React envia uma requisição *POST /api/compile*;
3. O backend valida os dados e executa o GCC;
4. As saídas são processadas e retornadas ao frontend;
5. O frontend exibe o resultado em três abas (Saída, Assembly e Binário).

4.7. Diagrama de Fluxo da Compilação



4.8. Diagrama de Sequência Cliente-Servidor

Conclusão

O GCC Playground se mostrou funcional em diversos cenários de uso. Os usuários podem escrever programas simples e complexos em C, aplicar diferentes níveis de otimização e visualizar os efeitos diretamente na saída gerada. A plataforma também foi eficaz em comunicar erros de compilação, além de permitir o estudo da estrutura interna dos programas gerados via dumps binários e de assembly.

A arquitetura baseada em microserviços possibilitou uma separação clara de responsabilidades entre frontend e backend, facilitando a manutenção e futura expansão do sistema. O código-fonte completo da aplicação encontra-se disponível publicamente no repositório GitHub do projeto: https://github.com/gabrielpalassi/GCCPlayground.

1. Análise Crítica

A proposta do GCC Playground apresenta claros benefícios educacionais e práticos. No entanto, alguns desafios técnicos foram identificados, tais como:

- O uso de *child_process* implica riscos de segurança se não forem aplicadas validações rigorosas nas entradas dos usuários.
- A ausência de uma sandbox completa impede o isolamento total dos processos de compilação, o que pode ser relevante em ambientes de produção.
- A plataforma não permite a execução dos binários.

Mesmo com essas limitações, o projeto alcançou seus objetivos acadêmicos e técnicos, oferecendo uma base sólida para futuras extensões.

2. Resultados

O desenvolvimento do GCC Playground representou uma aplicação prática e integrada de conceitos avançados de compilação, desenvolvimento web e segurança de sistemas. A plataforma permite que usuários escrevam, compilem e analisem código C de forma intuitiva, diretamente do navegador, sem necessidade de ferramentas locais.

Os objetivos do projeto foram plenamente atingidos, demonstrando a viabilidade técnica de criar um ambiente moderno de desenvolvimento online com suporte ao compilador GCC. Como desdobramentos futuros, sugerem-se:

- Implementação de sandbox segura para execução de binários
- Suporte a múltiplas linguagens (ex.: C++, Rust)
- Exportação de projetos

Com base em práticas modernas de engenharia de software, o GCC Playground contribui para o ensino de linguagens de programação de baixo nível, análise de código gerado e compreensão do processo de compilação em ambientes reais.