DESENVOLVIMENTO DE JOGO JANKENPON PARA FUNCIONAMENTO EM REDE USANDO TECNOLOGIA DE SOFTWARE LIVRE

Talles Henrique Brolezzi Fagundes
IFMS - Campus Três Lagoas
Três Lagoas, Brasil
talles.makoto@gmail.com

Prof. Esp. Eduardo Hiroshi Nakamura
IFMS - Campus Três Lagoas
Três Lagoas, Brasil
eduardo.nakamura@ifms.edu.br

Prof. Me. Alex Fernando de Araujo
IFMS - Campus Três Lagoas
Três Lagoas, Brasil
alex.araujo@ifms.edu.br

Prof. Me. Vladimir Piccolo Barcelos
IFMS - Campus Três Lagoas
Três Lagoas, Brasil
vladimir.barcelos@ifms.edu.br

***Resumo*—Este projeto propõe desenvolver um jogo digital - com uso de software livre - de Jankenpon para dois jogadores em rede local. Computadores clientes interagem através de um servidor com a utilização de *socket* em sistema Linux e a parte gráfica programada com a biblioteca Allegro 5. Este jogo possui a capacidade de realizar a comunicação entre os jogadores resultando em experiência de desenvolvimento e processo, desafios e a abertura do código fonte.**

Palavras-chave - Allegro 5; Jogos; Linguagem de Programação C; Socket; Software Livre.

I. Introdução

Com o avanço dos meios virtuais de comunicação, o conhecimento do funcionamento da rede tornou-se fundamental para profissionais da área da computação. Para acompanhar este desenvolvimento os jogos digitais, em redes, podem tornar-se favoráveis na obtenção deste conhecimento no ramo.

Normalmente esse tipo de jogo apresenta a possibilidade de integrar dois ou mais dispositivos, permitindo a ocorrência de diversos processos passíveis de análise através de softwares como *wireshark*[1] e nmap[2], que serão abordados posteriormente.

Consequentemente, o presente trabalho tem como proposta o aprendizado da programação de um sistema de comunicação em redes, estruturado para um jogo digital de pedra, papel e tesoura.

A implementação é realizada com softwares livres para a distribuição *Debian GNU/Linux*[3], para um jogo que comporta dois usuários simultaneamente conectados em rede local.

A escolha de ferramentas livres é primordial para este estudo. O uso destas tecnologias - por não haver a necessidade de encargos financeiros, possuírem comunidade de desenvolvedores e documentação acessíveis - facilita o aprendizado profundo dos materiais.

Para o projeto, utiliza-se do modelo cliente/servidor, onde o computador que serve é o responsável por fornecer um serviço[4], já o cliente[5] é preparado para realizar requisições ou consumir esses serviços.

Como protocolo *TCP* (*Transmission Control Protocol*), que atua na camada de transporte[6], é que permitirá a troca segura de mensagens entre os computadores deste jogo. De acordo com Mota Filho [2] esse é um protocolo de alto grau de confiabilidade, devido a forma como foi projetado para obter controle nas transmissões, considerando que para este projeto a integridade das mensagens trocadas seja prioridade.

Para alcançar a camada de transporte é utilizado a tecnologia de *socket* em ambos os lados da conexão, o que permite esta interconexão entre computadores com uso do protocolo *TCP* ou *UDP*.

O jogo tem como base a biblioteca multimídia *Allegro* em sua versão 5, sendo a *API* (*Application programming interface*) de rotinas para construção das funcionalidades gráficas bidimensionais, áudio e captura de eventos de mouse e/ou teclado. Possuindo os atributos otimizados para o desenvolvimento de um jogo completo.

A linguagem de programação C, utilizada no desenvolvimento da própria biblioteca *Allegro* e para acessar as rotinas desta, é a mesma definida para este projeto.

São apresentados as necessidades de estudos do projeto, apresentação das tecnologias citadas acima e a lista dos materiais utilizados até o final do desenvolvimento. Em seguida o processo exigido para a realização da comunicação entre os dispositivos, problemas encontrados e soluções. Finalizando com a conclusão e *link* de acesso ao código do projeto.

[1] Para análise de tráfego
[2] Para escanear portas de rede
[3] Sistema Operacional famoso por sua estabilidade e uso como servidores
[4] centro da comunicação deste jogo
[5] os usuários do jogo que irão conectar-se ao servidor
[6] É a camada 4, tendo como base o modelo *OSI*

II. Justificativa do Trabalho

O desenvolvimento da comunicação em rede tem proporcionado grandes facilidades, como o compartilhamento de informações pelo mundo, por exemplo. Conectar-se com uma máquina no Japão e estando localizado no Brasil, isto em questões de milissegundos, é uma realidade constante nos dias atuais. Por este motivo para os programadores é essencial aprimorar o conhecimento em redes e este trabalha busca apresentar esse assunto atrelado a outro bastante visado, os jogos digitais.

Com isso realizar este projeto com *software* proprietários poderia gerar encargos que podem vir a dificultar o acesso a informação - por este motivo é utilizado *software* livre - para serem acessíveis aos desenvolvedores, sem restrição ao conhecimento e com ferramentas robustas. Sendo possível contribuir com o avanço da sociedade e da tecnológico apresentando a experiência adquirida em todo o processo de desenvolvimento.

III. Objetivo Geral

Este projeto tem com objetivo conseguir desenvolver um jogo digital, com *software* livre, que possa comunicar-se em uma rede local. Tendo como ponto mais forte, não a interface gráfica, e sim a comutação realizada entre computadores clientes com intermédio de um servidor com uso do protocolo *TCP*(*Transmission Control Protocol*)

A. Objetivos Espectficos

Descrever o processo de aprendizado e desenvolvimento com *socket* em *Linux* e do jogo *jankenpon* em um ambiente virtual. Este jogo deve ter a capacidade de comunicar-se em rede suportando dois clientes simultaneamente de forma que ambos possam interagir de acordo com as regras originais do jogo.

Este projeto deve ser desenvolvido com uso de ferramentas de *software* e fornecendo o código fonte aberto de forma a contribuir com a comunidade de desenvolvedores.

IV. Necessidades para o Desenvolvimento do Jogo

Abaixo será segmentado e listado quais foram os pontos de estudos para atender a necessidade do projeto, sejam eles materiais teóricos ou ferramentas.

A. Documento do Projeto do Jogo

Normalmente o documento de projeto do jogo, popularmente conhecido pelo seu nome em inglês *Game Design Document* ou simplesmente *GDD*, é a estrutura inicial de um jogo. Como o coração do projeto, este contém desde a história e o roteiro do jogo à características de jogabilidade e necessidades a serem atendidas.

Como simplifica o autor Jesse [1], o documento do jogo possui as propostas de, primeiro ser o centro de acesso aos requisitos do projeto, e segundo auxiliar a comunicação entre as áreas aplicadas no desenvolvimento.

Esta documentação circula por todas a áreas do projeto. Servindo de base para o desenvolvimento e programação, administração, pois nele é possível abstrair o que será necessário ou não de ser implementado.

B. Arquitetura de Computadores

Ao desenvolver um projeto de jogos digitais, principalmente sem o uso de um motor gráfico de jogos[7], conhecer sobre a arquitetura do hardware empregados pode interferir diretamente quando o assunto é comutação entre computadores.

Um fator que deve ser observado é ordenação dos *bytes*. Esta representa a ordem que os *bytes* são lidos em arquiteturas de processadores, sendo dois tipos utilizados, *big-endian* ou *little-endian*. A ordem de leitura para um é inverso do outro e isto pode impactar em como serão processados os dados.

Para enviar dados de um *host*[8] de arquitetura que faça a leitura da direita para esquerda - do conjunto de *bytes* no lado emissor - para um receptor que faça leitura contraria, o resultado será invertido.

No exemplo abaixo, utilizando mapeamento da tabela *ASCII*[9], com isso é possível compreender esta diferença de ordenação.

Tabela I
ORDEM DE LEITURA DE *bytes*

	BIG-ENDIAN				***LITTLE-ENDIAN***		
	Leitura de direita para esquerda				Leitura da esquerda para a direita		
Endereço	x	x+1	x+2		x+2	x+1	x
Hexadecimal	61	6C	65		61	6C	65
ASCII	a	l	e		a	l	e

.

Na tabela I, cada arquitetura, ao imprimir os caracteres da palavra ***ale***, o fará de forma diferente e se isso fosse um ponto crítico para obtenção de resultados como senhas, traria efeitos negativos no funcionamento do programa.

No cenário de desenvolvimento deste jogo, foi utilizado arquitetura *little-endian*. Mesmo assim caso a aplicação dependa que o dado transmitidos sejam tratados ou invertidos, para serem recebidos em outra arquitetura, o autor Beej [8] apresenta soluções e uma delas são as funções abaixo.

- ***uint16_t*** *htons*(...) - ***unsigned integer*** de 16 *bits*; da ordem do *host* para rede.
- ***uint32_t*** *htonl*(...) - ***unsigned integer*** de 32 *bits*; da ordem do *host* para rede.
- ***uint16_t*** *ntohs*(...) - ***unsigned integer*** de 16 *bits*; da ordem da rede para *host*.
- ***uint32_t*** *ntohl*(...) - ***unsigned integer*** de 32 *bits*; da ordem da rede para *host*.

Com estas funções é possível converter a ordem de *bytes* antes que saia para rede e no recebimento destes, sendo usado para tratar valores numéricos inteiros.

[7]um conjunto de ferramentas e rotinas bem definas que permite focar-se melhor no desenvolvimento do jogo

[8]Um máquina conectada a rede

[9]Tabela de codificação de caracteres 8 *bits*

C. Sistema Operacional

O ambiente de desenvolvimento é o *Debian*, sistema operacional da família *Linux*, livre e conhecido por priorizar a estabilidade e confiabilidade dos seus componente. Esta distribuição também detêm de diversos pacotes contendo programas e códigos facilmente acessíveis com o uso da ferramentas *apt*, que gerencia a instalação e desinstalação de pacotes dentro do computador entre outras funções.

Todos os pacotes fornecidos diretamente pelo *Debian* são livres, isso é base da filosofia de liberdade de conhecimento adotado no desenvolvimento deste sistema, no entanto não é proibitivo o uso de *software* proprietário[10].

Cada sistema operacional possui características próprias e para melhor aproveita-se destas é necessário o estudo para acessar as rotinas deste sistema e compreender como funcionam.

Para acessar estas rotinas o *Linux* provém, através do projeto *GNU C Library* [9], de um conjunto de bibliotecas padronizadas e organizadas conhecida como *glibc*. Com essa *API* o programador consegue ter acesso às rotinas do núcleo do sistema com uma *interface* de alto nível[11].

D. Linguagem C

Não é de hoje a importância da linguagem C, que de acordo com [5], foi criada em meados da década de 70, e mesmo antiga é uma das mais populares até hoje, considerada excelente para programação de sistemas operacionais sendo os mais modernas escritos em C ou C++[12].

De acordo também com o autor acima, o desempenho de C é citado como uma das razões de sua popularidade, explicando que um programa bem escrito nela só poderia ser equiparada com o mesmo programa escrito em *assembly* ficando de 1% a 2% mais lento apenas que este.

E. Biblioteca Allegro

Atualmente em sua versão 5, *Allegro* [7] é uma opção livre para desenvolvimento de jogos em 2D(duas dimensões), sendo possível em sua versão atual o desenvolvimento para plataformas mobile e outros sistemas operacionais.

Esta biblioteca está disponível para download no repositório do *Debian*, a mesma usada neste projeto, uma versão estável, mesmo não sendo a mais atual. Está também é acessível para *download* do código fonte direto do site de desenvolvimento da biblioteca.

Esta *API* permite implementar projetos em C e C++ de forma nativa, disponibilizando acesso a rotinas de baixo nível, além da parte gráfica, a mídia de áudio, eventos de teclado e mouse.

Allegro também possui a própria implementação de *threads*, semáforos e *mutex*[13], o que melhora a portabilidade para outras plataformas. Rotinas como essas podem variar no como são implementadas em cada sistema operacional e seu modo de usar.

Suas funções são todas padronizadas e intuitivas e seu funcionamento opera como uma máquina de estado, operando seus estado de forma sequencial. Além disso, no documento de licença da *Allegro* [6] informa que na biblioteca está incluso o uso totalmente livre e aberto a qualquer desenvolvedor, para âmbito comercial até projetos coletivos, ficando a cargo de responsabilidade o próprio criador do jogo.

F. Comunicação em Redes

Para o jogo, a rede é o intermédio da comunicação entre os usuários e seu coração. Em outras palavras, nesta possui um servidor e através dele se conectam os computadores clientes dos quais apenas o servidor tem o controle e a visão direta do canal de comunicação entre eles. Para os usuários, outros computadores são transparentes não sendo possível realizar o jogo sem intermédio com servidor.

Para trabalhar esta interconexão entre computadores, pode-se utilizar da tecnologia de ***socket***, que permite estabelecer um ponto final e comunicação entre os processos de computadores na rede.

De acordo com Tanenbaum, em seu livro de sobre redes de computadores [4], essa tecnologia trata-se de um conjunto de rotinas primitivas que são aplicadas à camada de transporte, sendo usadas também no *UNIX* de *Berkeley* para o modelo *TCP/IP*[14] [4]. Estas rotinas são um dos motores da Internet através do protocolo TCP.

De acordo com Akimichi [18] *socket* é uma forma de entrada e saída de dados pelo usuário, permitindo que o mesmo se comuniquem através de escrita e leitura destes dados da rede. No *Debian* é possível trabalhar com esta rotina através da biblioteca de *socket*[15].

Os itens seguintes apresentam 5 funções utilizadas para se estabelecer uma conexão entre computadores. Existem outras que permitem manipular opções específicas do *socket*, no entanto estas são as principais rotinas da biblioteca sendo possível compreendê-las através do manual do *Linux* sobre a biblioteca [12].

1) ***int*** *socket*(...) - Define pontos de comunicação e retorna o descritor.
2) ***int*** *bind*(...) - Vincula um endereço ao *socket* gerado; através do indice do descritor.
3) ***int*** *listen*(...) - Torna-o passivo e o permite receber conexão.
4) ***int*** *accept*(...) - Recebe uma conexão através do *socket* criado e gera um novo ponto de conexão vinculado ao ponto principal.
5) ***int*** *connect*(...) - Estabelece uma conexão com *socket* do servidor.

As rotinas acima, do número 1 até a 4, operam do lado servidor e a rotinas de número 1 e 5 operam no lado cliente.

[10]Que detêm direitos de *copyright*
[11]que aproxima-se mais da linguagem humana
[12]Linguagem de programação
[13]Para sincronizado a recurso
[14]Arquitetura de referência para comunicação em rede
[15]Localizado no *Debian* em : sys/socket.h

Estas funções são utilizadas para estabelecer uma conexão via *TCP*, variando a forma de aplicação para o protocolo *UDP*, no entanto para este não possuindo a garantia na entregada dos dados transmitidos.

G. Nmap

Para os protocolos, *TCP* e *UDP*, existe para cada 65535 portas de acesso a rede disponível, e elas podem estar sendo usadas por algum serviço ou aplicação, além de abertas ou fechadas. Neste ponto entra o ***nmap***, que de acordo com o site oficial do *software* [15] essa é uma ferramenta para segurança e auditoria de rede, permitindo determinar quais computadores[16] estão disponíveis na rede, suas portas e quais aplicações possam estar sendo executadas através delas.

Antes de realizar troca de mensagem, pela rede, é importante atentar se as rotinas de *socket* estão funcionando devidamente. Para isso é feito a varredura na rede assegurando que todas as portas necessárias para comunicação do jogo, tanto do computadores clientes quanto do servidor estejam abertas, ou se nenhuma outra aplicação está em execução na mesma porta no momento.

H. Wireshark

Wireshark de acordo com Sharpe [17] é uma ferramenta de análise de pacotes na rede, e busca apresentar os dados capturados de forma gráfica.

Com esse software é possível capturar dados de pacotes que trafegam pela rede e inspecionar seus elementos: de cabeçalho de protocolo, número de bytes transmitidos, tipo do protocolo, endereços de destino e origem além de outras informações. O autor complementa também que é semelhante ao uso de um voltímetro para examinar eletricidade que está passando por um cabo.

Em se tratando de redes de computadores ele é como um *debugger*[17] e pode ser usado para verificar se os pacotes estão sendo enviados corretamente, permitindo a detecção de possíveis falhas, como, por exemplo, *Malformed Packet Error*(Erro de pacote mal formado).

V. Materiais Empregados no Projeto

Para o desenvolvimento deste projeto, utilizou-se as ferramentas listadas abaixo:

- *Inter(R) Core(™)* i5-4200 *CPU* @ 1.60 GH.z, 4GB de memória *RAM*.
- Sistema Operacional *Debian GNU/Linux* 3.16.0-4-amd64, x86_64
- Compilador *GCC* versão 4.9.2 (*Debian* 4.9.2-10)
- *Debugger GNU gdb* (*Debian* 7.7.1+dfsg-5) 7.7.1
- *Wireshark* 1.12.1 (Analisador de pacotes trafegados na rede)
- *Nmap* 6.47 (Inspecitionar abertura e fechamento das portas utilizadas pelo jogo)
- Biblioteca de Multimídia *Allegro* - liballegro5.0 - versão 2:5.0.10+b1
- *Software* de pintura digital - *Krita*

VI. Metodologia

O *Jankenpon*, pronunciado no Brasil como joquempô conhecido também como pedra, papel e tesoura, foi o jogo escolhido para a elaboração do projeto. Sua mecânica tradicional oferece aos jogadores três possibilidades de jogadas, cada uma tendo um ponto forte e um ponto fraco. Com exceção em casos que os jogadores joguem os mesmos elementos resultando em um empate.

Regra gerais:

- Pedra perde para Papel
- Papel perde para Tesoura
- Tesoura perde para Pedra

Estes elementos são simbolizados com o uso das mãos na hora de jogar. O jogo ocorre quando os dois usuários pensam em um dos 3(três) elementos e apresenta de forma simultânea suas jogadas, uma tentativa de evitar que o adversário obtenha vantagens.

Para que esta mecânica fosse mantida, um usuário não poderia descobrir a operação realizada pelo seu outro, por este motivo há um servidor intercambiando a comunicação dos jogadores.

Toda a partida é administrada pelo servidor, sendo todas as operações e resultados calculados neles. Esta forma evita que um computador cliente possa, de forma antecipada, receber a mensagem do outro cliente contendo a jogada, assegurando que estas mensagens serão processadas sem intervenção de nenhum usuário.

O jogador utiliza-se das tecla enter do teclado toda vez que desejar executar uma ação e as teclas direcionais para escolher qual ação antes que seja executada.

Na figura 1 observa-se um diagrama de sequência[18] demonstrando o funcionamento do jogo, da conexão ao encerramento da partida.

Nela o segundo cliente, a direita, apresenta o mesmo processo de conexão e jogabilidade do cliente da esquerda, apenas difere o tempo que cada um executa essas ações.

A leitura é realizada na sequência numérica, iniciado pelo boneco na imagem, percorrendo entre os processos do servidor e finalizando na parte inferior da imagem.

Para aplicar o uso do protocolo foi estudado *socket* que progrediram vagarosamente sem o devido entendimento sobre o *Linux*. Portanto foi notado que compreender o sistema operacional é quase tão importante quanto a rede, para este projeto. Comunicação inter-processos, processos, *threads*, semáforos e *mutex* são exemplos de estudos realizados e isto formou uma base para conseguir programar com *socket*.

As primeiras análises realizadas foram com uso da ferramenta *nmap*, para verificar se haveria disponibilidade de uso da porta e se era exatamente o processo do servidor do jogo operando nesta porta.

Para teste, com erros a nível de aplicação foi verificando com impressões em terminal e o depurador (neste caso o *GDB*), para buscar falhas ou comportamento inesperado.

[16] Qual sistema operacional, versões, endereço *IP* entre outros

[17] *Software* para realizar testes do código de algum programa

[18] Apresenta a sequência dos processos.

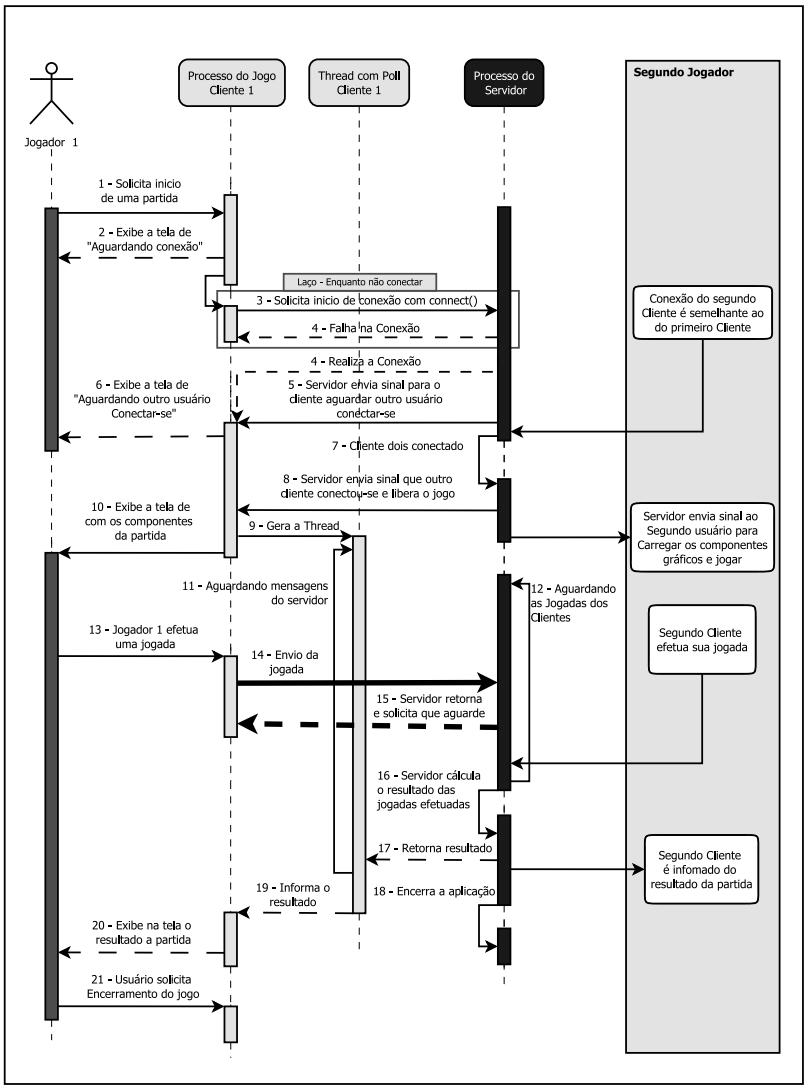

Figura 1. Esta imagem representa a sequência de funcionamento do jogo

Também foi usado o *wireshark*, que apresenta muita informações capturadas do tráfego da rede o que necessita de um tempo para analisar e compreender o que esta ocorrendo.

Quando iniciou o envio mensagens exigiu paciência, principalmente com ocorrência de erros, pode ser do lado cliente, do lado servidor, alguma falha na meio do caminho.

Quanto estes testes foram realizados, foi incluso *threads* tanto no cliente e servidor, ainda não era o jogo propriamente dito, era um estudo. Trabalhar com várias *threads*, pode trazer benefícios como ganho de desempenho, porém administrá-las devidamente levaria mais tempo do que o projeto permitia, no final ficou implementado apenas no cliente.

Nesta etapa já foi consumido um pouco mais da metade do tempo do projeto. Ainda restava a estudo da parte gráfica do jogo e fazer integrar a comunicação.

Ocorreram diversas falhas com captura de evento em implementações com *poll*, este problema evitava que todos os *hosts* conectados recebessem mensagens e só foi corrigido próximo ao final do projeto.

Este jogo precisou de 2 anos de projeto, aproximadamente, para ser desenvolvido. Foi levando em consideração desde o aprendizado da linguagem C(um dos requisitos básicos) até o jogo concluído em sua versão atual.

VII. Resultados do Desenvolvimento do Jogo

São apresentadas duas imagens do gráfico do jogo. Elas representam cenas que desenvolvidas com o uso da biblioteca *Allegro* e o software de pintura digital *Krita*.

A figura 2 é a cena de espera do usuário. Foi toda programada com o uso de formas primitivas, que são funções que permitem gerar na tela formas geométricas como, quadrados ou círculos.

Sendo na figura 2 o momento que já ocorreu a conexão com o servidor, com apenas do primeiro cliente, aguardando o momento que próximo realize a conexão com sucesso. Diferente da próxima figura:

Quando o usuário visualiza a figura 3, todo o processo de conexão dos dois usuários estão completos, a partir deste

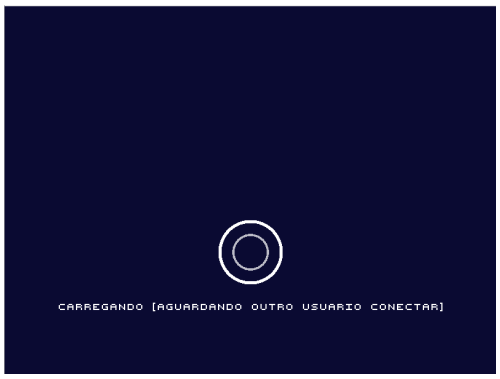

Figura 2. Aguardando o outro usuário

Figura 3. *interface* jogável

ponto efetuam as jogadas (o elemento envolto pelo argola e selecionando com *ENTER*), enviá-las ao servidor e receber como retorno a resultado. No final é apresentado no mesmo local que consta o nome do elemento escolhido pelo usuário.

A. Estudos do Sistema Operacional

Como a proposta é gerar um jogo para dois jogadores, com comunicação em rede através do modelo *TCP/IP* na distribuição *Debian*, foi realizado um estudo em comunicação inter-processos, visando compreender como um processo comunica-se com outro, além do estudos de *threads*. Através desta base foi possível compreender melhor o funcionamento da rede.

B. Comunicação Cliente/Servidor

O servidor foi desenvolvido suportando 2 usuários simultaneamente através do modelo cliente/servidor em cima do protocolo *TCP*. Sendo definido como porta de conexão ao servidor, no entanto a porta 22222[19] foi definida apenas a nível de teste local.

Para a porta do lado cliente não há necessidade de definir um padrão. Nos testes realizados o sistema escolheu de aleatoriamente portas que entre 1024 à 65535[20].

[19]está dentro da faixa de portas registradas pelo *IANA* [10]

[20]de 49152-65535 são portas dinâmicas ou privadas, não possuindo atribuição de serviços

1) Aplicação do Cliente: Cliente é a parte do sistema que irá se conectar ao servidor, consumindo ou requisitando algo.

Utilizando-se das primitivas *socket()* e *connect()*, o cliente já conhece o endereço do servidor e porta ao qual deverá se conectar.

Primeiramente chama-se o *socket()*, caso obtenha sucesso vai para a função ***connect()***. Esta tenta efetuar uma conexão com o servidor, porém caso retorne valor **-1** o cliente tentará conectar-se outra vez.

Realizada a conexão com sucesso, o jogo fará todos os procedimentos e instâncias necessárias ao suporte gráfico e iniciará um *thread* que paralelamente ficará responsável por escutar, isto é, preparada para receber mensagens vindas do servidor.

Para capturar este evento (as mensagens) é utilizado da rotina ***poll()*** [13], que permite esperar determinadas atividades associados a um arquivo descritor[21].

Esta função recebe do servidor o resultado das jogadas e devolve ao processo principal do jogo. Enquanto não receber uma mensagem que atenda aos requisitos dela o cliente continua escutando até receber.

Mesmo que a função acima seja projetada para receber o resultado, ainda possuí erros. Como exemplo, todo dado escrito no *buffer* de entrada do *poll()* é considerado evento e ocorrem erros, como por exemplo ao receber segmentos[22] com falhas ou desconexão vindas do servidor.

O exemplo acima acontecia quando recebimento de um segmento de **12** ***bytes*** era enviado do servidor aos clientes conectados. Isto ocorria quando o segundo usuário conetava, causando instabilizado no *poll*.

Para ilustrar o envio desde erro, abaixo um trecho das análises realizadas com *Wireshark* que apresenta o envio de um segmento do servidor (usando porta 22222) a o cliente (usando porta 36348 definida aleatoriamente). Esta mensagem enviada automaticamente contêm 12 *bytes* e correspondente ao erro que ocorria no *poll()* a partir da segunda conexão.

```
Transmission Control Protocol,
 Src Port: 22222 (22222),
 Dst Port: 36348 (36348),
 Seq: 1, Ack: 1, Len: 12
 Data (12 bytes)
```

Para uma melhor compreensão é apresentado os campos do cabeçalho do protocolo *TCP* correspondes ao contidos no trecho acima. Esta correlacionado com o autor Mota Filho [2], em seu livro sobre análise e tráfego com uso de *tcpdump*[23] e estes elementos são apresentados da seguinte forma:

- *Src Port* - Porta de origem, a máquina que realiza o envio
- *Dst Port* - Porta de destino, a máquina que irá receber a mensagem
- *Seq* - Número de sequência do segmento transmitido,

[21]determina um número inteiro que identifique um descritor de arquivo

[22]Nome utilizado para a unidade de dado do protocolo enviada pela camada de transporte

[23]*API* de captura de tráfego em modo texto

- *Ack* - Número de confirmação que indica o próximo número da sequência ser enviado
- *Len* - Tamanho do segmento *TCP*
- *Data* - O dado transmitido

2) Aplicação do Servidor: Agora abordando o servidor, mesmo iniciando com a função *socket()* já não utiliza a primitiva ***connect()*** sendo substitui por ***bind()***, responsável em associar um ip a uma porta.

Em seguida prepara-se o ***listen()*** [12], que de acordo com Tanenbaum [4] é a rotina responsável em escutar qualquer requisição de conexão e controlar a fila[24] de pedidos que dos computadores clientes definindo quantos usuários poderão ficar esperando.

O *Debian* por padrão tem pré-definido um número máximo de conexões, localizado no arquivo do sistema *somaxconn*[25] ou pela definição na biblioteca de *socket* como *SOMAXCONN* tendo ambos o número máximo de 128, indicando o limite de requisições na fila. Não houve a necessidade de alterar o limite padrão deste arquivo para um valor numérico maior.

Para verificar a abertura da porta, utilizou-se do *nmap* através do terminal *Linux* para obter esta informação. O objetivo foi buscar diretamente na porta 22222 do protocolo *TCP* para o endereço *IP* pré-definido ao servidor.

```
$ nmap -sT -p 222222 127.0.0.1

Starting Nmap 6.47
(http://nmap.org) at 2017-08-15 13:11 -04
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000024s latency).
Other addresses for
  localhost (not scanned): 127.0.0.1
PORT      STATE  SERVICE
22222/tcp closed unknown

Nmap done: 1 IP address
  (1 host up) scanned in 0.03 seconds
```

Através deste trecho do exame, observa-se que para a porta *TCP* encontra-se aberta(*STATE open*), o serviço desconhecido ao lado é a aplicação do servidor.

Para efetuar a conexão foi necessário utilizar-se da rotina *accept()*, que fica responsável em receber o pedido e gerar um novo descritor de arquivo para este novo ponto. A cada chamada desta função, um novo usuário pode realizar uma conexão, sendo usada no máximo duas[26] vezes.

O servidor só permite iniciar o jogo quando os dois jogadores estiverem conectados, até que ocorra este evento é enviado ao cliente de espera e só o libera a conexão do adversário. Se este requisito for concluído é aberto ao usuário realizar sua jogadas.

O servidor realizar uma abertura de captura de eventos com *poll*, semelhante ao do cliente, ele ficará responsável por receber as mensagens vindas dos dois clientes. Como não se sabe qual jogar irá efetuar uma jogada primeiro o que fizer será processado antes, este fato ocorre devido ao arquivo descritor gerado pelo *accept()* para cada cliente.

Quando *buffer* de entrada do descritor de arquivo destinado a cliente, dentro do servidor, é escrito por algum dado vindo do cliente o evento é capturado e destinado as variáveis deste cliente.

No trecho de código acima apresenta o servidor capturando as jogadas enviadas pelos clientes. Cada volta no laço principal a função *poll()* reinicia o evento de capturar dos dados, prossegue verificando se foi ou não capturado algo, se foi vai até o *while* interno e percorre os clientes até encontrar quem realizou o envio.

Sendo assim, o momento que os dois usuários efetuarem suas jogadas o servidor calcula o vencedor e devolve o resultado a cada um e encerra a partida.

C. Programando a Interface Gráfica

Foi tomado por base na construção do código o material do autor George [16] que segmenta o jogo em 3 etapas bem definidas, inicialização(estático), laço principal do jogo (dinâmico) e encerramento.

Para o segmento do código de encerramento, são funções chamadas no final do jogo para que seja encerrado as estruturas de dados de forma correta. Para cada inicialização, que for explicada, será necessário encerrá-la ao final.

1) Segmento Estático: A parte estática é a inicialização das estrutura, configurações delas e instalações de componentes[27] oferecidos pela própria biblioteca.

Configurado com sucesso estes componentes tornam possível o uso de captura de evento do teclado, gerar imagens na tela em formato como *JPEG*[28], formas primitivas[29] e até mesmo exibir tipos diferentes de fontes de texto.

Todas as funções de inicialização ou de instalação estão padronizadas primeiramente com *al_*, que identifica ser da biblioteca *Allegro*, seguidas da sua ação executada pela função. É necessárias chamar as funções de inicialização antes de usar os componentes.

2) Segmento Dinâmico: O segmento dinâmico do código é tratado como o laço principal do jogo, sendo responsável por manter o usuário, teoricamente, executando em processo infinito e apresentando na tela as imagens processadas ao final de cada volta do laço. A Cada volta é chamado diversos objetos que contêm imagens, fontes ou primitivas sendo calculado suas posições na tela. Sempre o último objetivo a ser chamado é o último a ser mapeado na tela, sobrepondo os anteriores se necessário.

Por fim, não é apenas a parte gráfica que ocorrem na área dinâmica, sendo subdividido esta parte em, captura de eventos, cálculos e processamento e por ultimo a imagem.

Primeiro verifica se o usuário realizou algum evento pelo teclado, em uma fila de eventos.

[24] O primeiro que chegar será o primeiro a ser processado
[25] Caminho do arquivo: /proc/sys/net/core
[26] Apenas dois jogadores simultaneamente conectados
[27] Componentes como, teclado, mouse, eventos etc.
[28] Método de compressão de imagem
[29] ponto, linha, círculo, quadrado etc

A próxima etapa é o cálculo e validação destes eventos. Se houver a necessidade de processar algo, será feito aqui, vinculados a eventos ou não, como exemplo o relógio.

Finalizando as etapas anteriores é mapeado na tela os objetos que precisam ser exibidos dentro da janela do jogo. Este evento deve ocorrer tão rápido quanto os olhos do jogador puder acompanhar, pelo fato de tratar-se de diversas sobreposições de imagens ocorrendo em um único segundo. Isto fornece ao usuário uma percepção de algo contínuo.

Sempre no final são chamadas as funções *al_clear_to_color()* e *al_flip_display()*, que trabalham como uma equipe de limpeza, sendo a primeira como se retirasse os móveis do local e a segunda a que as devolve (nas mesmas posições ou outras).

A primeira sobrepõem qualquer objetivo na tela com uma cor única. Esta ação é para evitar que algum objeto anterior, que vem a ser calculado em outra posição, cause algum efeito indesejável na visão final da tela ao ficar por trás.

Por fim a segunda função faz com que seja exibida todos as figuras, primitivas ou fontes que tenham sido mapeadas.

VIII. Conclusão

Desenvolver um jogo que comunica-se em rede é um trabalho complexo que exige atuação de diversas áreas da computação e da arte.

As tecnologias empregadas atenderam muito bem as necessidades do projeto, são boas aliadas do programador e sem a necessidade de encargos financeiros.

A respeito da comunicação ela exige atenção e cuidados com diversos detalhes[30]. O uso de ferramentas como ***nmap***, ***wireshark*** ou ***tcpdump*** são essenciais, para análise da rede.

Com o uso destas rotinas do sistema operacional possibilitou o trabalho com ***socket*** em **linguagem C**, uma tecnologia incrível e prática. Além do mais, aplicá-las no *Linux* é acessível, pois detêm de manuais robusto e bem explicado para o programador, aliado a uma forte comunidade de desenvolvedores.

Para o jogo, a biblioteca ***Allegro*** conseguiu atender bem as necessidades do projeto, sendo uma tecnologia completa para produzir um jogo do começo ao fim. Seu ponto fraco é apenas sua documentação não ser tão auto explicativa quanto o da manual de desenvolvedor do ***Linux***, mas novamente a comunidade pode ser citada como uma forte aliada.

Por fim, todo o processo foi uma experiência completa e permitiu navegar por algumas áreas dentro da computação e deixando como resultado o código aberto sob licença *MIT* [14].

Todo o código pode ser encontrado no endereço **github.com/KoureiMakoto/MakotoGameServer** .

Agradencimentos

Agradeço primeiro a Grande Deus SU, que me permitiu realizar este maravilhoso estudo, além disso dia após dia tem me abençoado com sua Divina Luz e dado discernimento para trilhar com felicidade e amor esta vida.

Deixo agradecimentos a minha família que me proporcionou, com as graças de Deus, o bens materiais necessários para estar realizando este curso, a todos meus amigos que me apoiaram neste empreitada, principalmente, meu mestre em computação professor Eduardo Hiroshi Nakamura, que acreditou em mim desde de primeiro semestre e me auxiliou até o final. Muito Obrigado a Todos.

[30]Se ocorreu a comunicação, se o dado foi enviado corretamente etc

Referências

[1] S. Jesse, *The Art of Game Design : A Book of Lenses*, Taylor and Francis Group LLC - EUA, 2015.

[2] E. M. F. João, *Análise de Tráfego em Redes TCP/IP : Utilize tcpdump na análise de tráfegos em qualquer sistema operacional*, São Paulo : Novatec Editora Ltda - BR 2013.

[3] A.S. Tanenbaum, *Sistemas Operacionais Modernos*, 3rd ed. São Paulo, Pearson Education, BR, 2010.

[4] A.S. Tanenbaum, *Redes de Computadores*, 5th ed. São Paulo, Pearson Education, BR, 2011.

[5] U. Oliveira, *Programando em C – Volume 1 : Fundamentos*, Rio de Janeiro, Editora Ciência Moderna Ltda, BR, 2008.

[6] S. Hargreaves, *Allegro Copyright*, http://liballeg.org/license.html(Current Jun. 21, 2017).

[7] S. Hargreaves, *Welcome to Allegro*, http://liballeg.org/index.html(Current Jun. 21, 2017).

[8] B.B.J. Hall, *Beej's Guide to Network Programming*, https://beej.us/guide/bgnet/output/print/bgnet_A4_2.pdf(Current Jun. 21, 2017).

[9] *What is glibc?*, Free Software Foundation, https://www.gnu.org/software/libc/(Current Jun. 21, 2017).

[10] *Service Name and Transport Protocol Port Number Registry* , Inst. Electrical Electronics Engineers, https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=22222(Current Jun. 21, 2017).

[11] M. Kerrisk et al., *Linux Man Pages Online*, http://man7.org/linux/man-pages/(Current Jun. 21, 2017).

[12] M. Kerrisk et al., *Linux Programmer's Manual - SOCKET(7)*, http://man7.org/linux/man-pages/man7/socket.7.html(Current Jun. 21, 2017).

[13] M. Kerrisk et al., *Linux Programmer's Manual - POLL(2)*, http://man7.org/linux/man-pages/man2/poll.2.html(Current Jun. 21, 2017).

[14] *The MIT License*, Masschusetts Inst. Technology, https://opensource.org/licenses/MIT(Current Jun. 21, 2017).

[15] G. Lyon, *Introduction*, https://nmap.org/book/toc.html(Current Jun. 21, 2017).

[16] G. Foot, *Allegro Vivace*, https://wiki.allegro.cc/index.php?title=Allegro_Vivace(Current Jun. 21, 2017).

[17] R. Sharpe, *Wireshark User's Guide : Introduction*, https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs(Current Jun. 21, 2017).

[18] O. Akimichi, *Desenvolvendo com Socket*, http://www.geekpage.jp/programming/linux-network/socket.php(Current Jun. 21, 2017)(in Japanese).