Universidade Federal do Rio Grande do Sul - Instituto de Informática
INF0115 Sistema Operacionais II

25/05/2024

Trabalho Prático Parte 1:

WakeOnLan

Prof. Weverton Luis da Costa Cordeiro
Henrique Carniel da Silva (*hcsilva@inf.ufrgs.br*) - 00335626
Ian Kersz Amaral (*ikamaral@inf.ufrgs.br*) - 00338368
Nathan Alonso Guimarães (naguimaraes@inf.ufrgs.br) - 00334437

Estruturas e Funções

Atomic

Classe criada com templates, para garantir que o único acesso ao recurso seja feito a partir da aquisição de um mutex.

```
template <typename T>
class Atomic
{
private:
    std::mutex lock = std::mutex();
    T resource;

public:
    Atomic() : resource(T()) {}
    Atomic(T value) : resource(value) {}

    template <class F>
    auto execute(F &&f, auto &&...args)
    {
        const std::lock_guard<std::mutex> lock_guard(lock);
        return std::invoke(std::forward<F>(f), std::ref(resource), args...);
    }

    friend std::ostream &operator<<(std::ostream &os, const Atomic &atomic)
    {
        os << atomic.resource;
        return os;
    }
};
```

Atomic Queue

Classe criada com templates, a qual cria um fila protegida com mutex, a qual pode ser produzidos itens e consumidos, para seu consumo faz uso de std::optional.

```cpp
template <typename T>
class AtomicQueue
{
private:
    std::mutex lock = std::mutex();
    std::queue<T> resources;

public:
    AtomicQueue() : resources(std::queue<T>()) {}
    AtomicQueue(std::queue<T> resources) : resources(resources) {}

    void produce(T resource)
    {
        const std::lock_guard<std::mutex> lock_guard(lock);
        resources.push(resource);
    }

    std::optional<T> consume()
    {
        const std::lock_guard<std::mutex> lock_guard(lock);
        if (resources.empty())
        {
            return std::nullopt;
        }

        auto resource = resources.front();
        resources.pop();
        return resource;
    }
};
```

Signals

Classe que usa o padrão Singleton, ou seja, todas suas referências apontam para a mesma variável. Dentro dela a classe std::atomic_bool é utilizada em diversos sinais/flags para sincronizar as threads do programa sem causar condições de corrida.

```cpp
class Signals
{
public:
    static std::atomic_bool is_manager;
    static std::atomic_bool run;
    static std::atomic_bool update;
    static std::atomic_bool manager_found;

    Signals() = delete;
    ~Signals() = delete;
};
```

Format

O namespace format foi criado para conter algumas funcionalidades para facilitar o uso de strings e enumerações.

Addresses

Essas classes contém diversas informações relacionadas a sua respectiva representação de endereço, como funções para facilitar seu uso e conversão, tanto como sua obtenção em diversas plataformas e visualização:

1. Mac Address
2. Port
3. IPV4
4. Address

```cpp
constexpr int MAC_ADDR_LEN = 6;
typedef std::array<uint8_t, MAC_ADDR_LEN> mac_addr_t;
class Mac
{
private:
    constexpr static int MAC_ADDR_STR_LEN = 17;
    constexpr static char MAC_ADDR_DELIM = ':';
    mac_addr_t m_mac_addr;

public:
    constexpr Mac() : Mac("00:00:00:00:00:00") {}
    constexpr Mac(const std::string &mac_addr);
    constexpr Mac(const mac_addr_t &mac_addr) noexcept;

    static std::optional<Mac> FromMachine(const std::string &intrfc);

    static Mac FromMachine();

    constexpr bool operator==(const Mac &other) const noexcept;

    constexpr bool operator!=(const Mac &other) const noexcept;

    constexpr bool operator<(const Mac &other) const noexcept;

    constexpr const mac_addr_t &data() const noexcept;

    constexpr size_t size() const noexcept;
    std::string to_string() const;

    friend std::ostream &operator<<(std::ostream &os, const Mac &mac);
};
```

Classe Mac Address

```cpp
typedef uint16_t port_t;

class Port
{
private:
    constexpr static port_t PORT_MIN = 0;
    constexpr static port_t PORT_MAX = 65535;
    port_t m_port;

public:
    constexpr Port() noexcept : Port(0) {}
    constexpr Port(const uint16_t &port) noexcept : m_port(port) {}
    constexpr Port(const sockaddr_in &addr) noexcept : m_port(from_network_orde

    constexpr port_t getPort() const noexcept;

    constexpr port_t to_network_order() const noexcept;

    constexpr void setPort(const port_t &port) noexcept;

    constexpr void setAddrPort(sockaddr_in &addr) const noexcept;

    std::string to_string() const noexcept;

    friend std::ostream &operator<<(std::ostream &os, const Port &port) noexcep

    constexpr static port_t from_network_order(const port_t &port) noexcept;

    constexpr static port_t to_network_order(const port_t &port) noexcept;

    constexpr bool operator==(const Port &other) const noexcept;
};
```

Classe Port

```cpp
constexpr size_t IPV4_ADDR_LEN = 4;
typedef union ipv4_t
{
    uint32_t addr;
    std::array<uint8_t, IPV4_ADDR_LEN> bytes;
} ipv4_t;

class IPv4
{
private:
    constexpr static char IPV4_ADDR_DELIM = '.';
    ipv4_t m_ipv4_addr;

public:
    constexpr IPv4() noexcept : IPv4(0) {}
    constexpr IPv4(const std::array<uint8_t, IPV4_ADDR_LEN> &ipv4_addr)
    constexpr IPv4(const uint32_t &ipv4_addr) noexcept;
    constexpr IPv4(const sockaddr_in &addr) noexcept;
    constexpr IPv4(const std::string &ipv4_addr);

    constexpr uint32_t to_network_order() const noexcept;

    constexpr void to_addr(sockaddr_in &addr) noexcept;

    std::string to_string() const;

    friend std::ostream &operator<<(std::ostream &os, const IPv4 &ipv4);

    constexpr bool operator==(const IPv4 &other) const noexcept;
};
```

Classe IPV4

```cpp
typedef sockaddr_in addr_t;

enum class IPVersion : int
{
    IPv4 = AF_INET,
    IPv6 = AF_INET6
};

class Address
{
private:
    IPv4 ip;
    Port port;
    addr_t addr;

    constexpr void init_addr() noexcept;

public:
    constexpr Address() noexcept : ip(), port() { init_addr(); }
    constexpr Address(const std::string &address);
    constexpr Address(const IPv4 &ip, const Port &port) noexcept;
    constexpr Address(const std::string &ip, const Port &port);
    constexpr Address(const IPv4 &ip, const port_t &port) noexcept;
    constexpr Address(const std::string &ip, const port_t &port);
    constexpr Address(const addr_t &address) noexcept;
    constexpr Address(const Port &port) noexcept;
    constexpr Address(const port_t &port) noexcept;
    constexpr Address(const IPv4 &ip) noexcept;

    constexpr void setIP(const IPv4 &new_ip) noexcept;
    constexpr void setIP(const std::string &new_ip);
    constexpr void setPort(const Port &new_port) noexcept;

    constexpr void setPort(const port_t &new_port) noexcept;
    constexpr void setAddr(const addr_t &new_addr) noexcept;

    constexpr IPv4 getIp() const noexcept;
    constexpr Port getPort() const noexcept;
    constexpr addr_t getAddr() const noexcept;

    std::string to_string() const;

    friend std::ostream &operator<<(std::ostream &os, const Address &a);
    constexpr bool operator==(const Address &other) const noexcept;
};
```

Classe Address - junta ipv4 com uma porta para facilitar seu uso em sockets

PCInfo

Essa classe engloba todas as informações relacionadas a um desktop conectado em uma rede, dentro dela existe o hostname, endereço mac, endereço ipv4, status atual da máquina (acordado, dormindo ou desconhecido) e se ela é ou não gerente do sistema.

Packet

Util

Esse namespace contém a enumeração do tipo de pacote, definições dos tipos utilizados no pacote, e função para conversão e desconversão de classes para bytes.

Header

Essa classe engloba um número mágico para validação de pacotes, o tipo de pacote, o número de sequência dele, tamanho do seu corpo, e um timestamp. Também contém funções para serializar e desserilizar ela mesma.

Body

Essa classe é um wrapper envolta do tipo std::variant, o qual engloba os diverso tipos de bodies que podem ser enviados, como: uma string, uma série genérica de bytes, a dupla de hostname e mac address, ou somente o mac address. Também contém funções para consultar seu próprio comprimento, e se serializar e desserilizar.

Packet

Essa classe aglutina um header e um body, assim facilitando a criação de ambos em conjunto, assim como suas serializações e desserilizações para a rede.

Socket

A classe faz uso extensivo de std::optional para efetuar todas as funções relacionados à sockets, como: criar, abrir, conectar, mudar configurações e aceitar conexões, de forma segura. Dessa forma, também é possível instanciar a classe e utilizá-la de forma abstraída do sistema operacional.

UDP

Instancia uma socket configurada para UDP, também como métodos para enviar e receber pacotes através dessa socket, e mandar broadcasts.

Primitivas de Comunicação

Como é possível observar nas classes acima, foram utilizadas as seguintes primitivas de comunicação:

1. UDP/IPV4 (em UDP)
2. Produtor/Consumidor (em Atomic Queue)
3. Sinais (em Signals)
4. Mutex (em Atomic e Atomic Queue)

Sistema

No arquivo “main.cpp” é feito o *parsing* dos argumentos da linha de comando, para saber se a estação deve ser gerente ou não, de acordo com a especificação. Seguindo isso, é feito a instalação do controlador de sinais para interceptamos o comando “CTRL+C”, e as variáveis compartilhadas entre threads são criadas, que são:

1. Uma tabela hash que relaciona um hostname ao PCInfo de cada desktop na rede, essa tabela estando atrás de um Atomic.
2. Um produtor/consumidor para novos pc’s encontrados na rede.
3. Um produtor/consumidor que recebe quais pc's devem ser enviados os pacotes de Wake On Lan.
4. Um produtor/consumidor que recebe a dupla de hostnames e status para realizar o update do status atual da tabela de pc’s.

Após isso, os subserviços são inicializados, cada um em sua respectiva thread, então o programa espera a finalização de todos os processos e faz um safe shutdown.

Portas utilizadas (10000 - 14999)

- **Porta Mágica:** 9 (utilizada para receber os pacotes mágicos)
- **Porta de Descoberta:** 10000
- **Porta de Saída:** 12345
- **Porta de Monitoramento:** 14321

Subservices

Ambos os Discovery e Monitoring Subservices compartilham do mesmo código de inicialização, que segue o seguinte molde:

Inicialmente, o serviço cria um socket UDP na porta do respectivo subserviço, o qual será usado posteriormente para receber e enviar os pacotes dessa rotina. Após isso, entra-se em um loop checando se a variável de inicialização “run” ainda é verdadeira, senão o socket é fechado, e o serviço é encerrado. No início do loop, o subserviço checa se houve uma transição de estado, se sim, existem duas possibilidades:

1. **O serviço não era gerente e virou gerente:** a variável de transição é atualizada e o queue de mensagens interno da socket é limpo, através da leitura de pacotes até ocorrer um erro por timeout.
2. **O serviço era gerente e se tornou não gerente:** a variável de transição é atualizada.

Discovery Subservice

Logo após sua inicialização, se o serviço for gerente, então ele escuta por pacotes de Sleep Service Discovery (SSD) no socket, caso o gerente receba um pacote, ele adiciona o novo desktop na fila para ser adicionado, e manda um pacote de SSD_ACK para o cliente. Se o serviço for um cliente, ele espera por um tempo determinado um pacote SSD_ACK, caso ele não receba nada, ele manda um pacote SSD para o endereço de broadcast na porta de descoberta, no caso contrário, adiciona o gerente a sua lista de desktops e modifica a variável compartilhada

“manager_found” para verdadeiro. Caso essa variável seja verdadeira, esse subserviço do cliente espera para ela se tornar falsa.

Monitoring Subservice

Logo após sua inicialização, se o serviço for gerente, ele dorme por um tempo determinado, e então entra na sua rotina de checagem de clientes. Nessa rotina, é feito uma cópia temporária de todos os clientes atuais, a qual é percorrida, e para cada cliente é enviado um pacote de Sleep Status Request (SSR) e então é feito o aguardo de cada resposta SSR_ACK. Caso um cliente não responda com o pacote de SSR_ACK, é armazenado temporariamente o status de Dormindo, e caso ele responda, o status de Acordado. Caso o status temporário do desktop defira de seu status atual na tabela, é produzido uma dupla com seu hostname e status atualizado. Nesse subserviço, o cliente simplesmente fica aguardando receber um pacote de SSR, o qual então ele responde com SSR_ACK, e volta para o aguardo.

Management Subservice

Nesse subserviço, diversas threads, as quais serão descritas abaixo, são iniciadas para efetuar todas as suas funções necessárias.

Update PC Map

É executada a tentativa de consumir um novo PC enviado por outro subserviço, até que a variável compartilhada "run" seja falsa. Caso seja bem-sucedido, esse novo PC é adicionado à tabela hash e a flag de update é ativada.

Update Status

É executada a tentativa de consumir uma atualização de status de um computador, até que a variável compartilhada "run" seja falsa. Caso seja bem-sucedido, o status é atualizado no respectivo PC na tabela hash e a flag de update é ativada.

Wakeup Sender

É executada a tentativa de consumir um hostname para o computador que deve ser acordado, até que a variável compartilhada "run" seja falsa. Caso um hostname seja obtido, o serviço adquire as demais informações do computador na tabela hash e verifica se ele está realmente dormindo. Se confirmado, um pacote mágico é criado e enviado para acordar o PC.

Exit Sender

Nessa parte do serviço, é utilizado uma primitiva do C++, a qual fica esperado para uma variável booleana atômica se tornar falsa, neste caso, a variável “run”, e quando isso ocorre, o resto da função pode prosseguir. Ao final desse serviço, é feito um broadcast para a porta de saída com um pacote de Sleep Serviço Exit (SSE).

Exit Receiver

É criado uma socket UDP na porta de saída, a qual será usada posteriormente. É então feito um loop, que roda até a variável “run” ser falsa. Nesse loop, é feito a espera por um pacote de SSE, e caso não recebido, o loop se reinicia. Quando um pacote válido é recebido, é feita a extração de seu hostname, o qual é então removido da tabela hash de PCs e a flag de update ligada. Caso o computador recebido seja o gerente do sistema, os computadores clientes voltam a procurar por um novo gerente.

Interface Subservice

Nesse subserviço são iniciadas mais duas threads, uma para a parte de leitura da tela e outra para a parte de escrita.

Escrita

A thread de escrita, roda em um loop fazendo suas operações enquanto a variável de inicialização “run” ainda é verdadeira. No loop, é primeiramente construído uma string atomicamente iterando pela tabela hash de desktops, no final disso, a variável global “update” é colocada para falso, e então é construído um

objeto de tipo std::osyncstream, o qual torna a saída para o terminal segura entre threads. No final do loop, é feito uma espera para que a variável global atômica “update” seja verdadeira novamente.

Leitura

A thread de leitura inicialmente cria uma tabela hash que liga os nomes dos comandos válidos às funções que devem ser ativadas com eles. Após isso, entra-se em loop enquanto a variável de inicialização “run” for verdadeira. No início desse loop, é feito uma tentativa de capturar uma string enviada pelo usuário, de forma assíncrona. Caso o std::optional retornado seja vazio, o loop volta para o início e espera uma entrada válida. Quando uma entrada válida é detectada, os argumentos dela são quebrados em uma lista de strings, a primeira de qual será o comando, e o restante os argumentos para este, e então o comando é feito pela aplicação.

Problemas e Soluções

Um dos principais problemas encontrados foi como testar diversos computadores em uma rede enquanto fora dos laboratórios do instituto de informática. Inicialmente, o programa era compilado na plataforma Windows, e igualmente no Windows Subsystem for Linux (WSL). Como o Windows e o WSL compartilham uma rede LAN, foi possível fazer os testes preliminares dessa forma, apesar das dificuldades do grupo em portar o código para ambas as plataformas, principalmente com os sockets em Windows, utilizando o preprocessador.

Apesar disso, essa combinação de plataforma não foi suficiente, visto que, de acordo com a especificação do trabalho, a chave primária da lista era o hostname do computador, e ambos o WSL e Windows compartilham o mesmo hostname, o que causou alguns bugs difíceis de se resolver. Além disso, por causa da interface compartilhada entre eles não ser a real ethernet do computador, às vezes mensagens eram enviadas para a rede LAN incorreta.

Para resolver os problemas citados acima, foi então implementado o uso de containers Docker, o qual, a partir de um docker compose, criava diversas máquinas

linux em uma rede compartilhada, as quais então iniciava o programa do wakeonlan. Com isso, o grupo pode finalizar o trabalho sem mais erros inesperados.