Universidade Federal de Minas Gerais
DCC023: Redes de Computadores
Trabalho Prático
rmtlog: Um Sistema de Log Remoto

Introdução

Neste trabalho iremos desenvolver uma biblioteca para criação, transmissão e armazenamento de logs. A biblioteca permite que uma aplicação executando em um dispositivo d_1 gere mensagens de log e as envie para um coletor executando no dispositivo d_2. A biblioteca realiza a codificação e transmissão confiável de mensagens entre aplicações e um coletor de logs.

Mensagens

Uma aplicação cria mensagens, por exemplo capturando eventos ou seu estado atual, e as envia para um coletor. Toda mensagens de log possui cinco campos, que devem ser transmitidos em ordem:

1. **Número de sequência.** Todas as mensagens de log são identificadas por um número. Cada emissor mantém um número de sequência para numerar suas mensagens de log, que deve ser inicializado com o valor 0 e incrementado após a transmissão de cada mensagem. O número de sequência é transmitido como um inteiro de 64 bits, sem sinal, em *network byte order*.
2. **Timestamp de geração.** Todas as mensagens de log possuem um timestamp com o instante de tempo em que a mensagem de log foi lida do arquivo de entrada. O timestamp e transmitido como dois inteiros de 64 e 32 bits, sem sinal, em *network byte order*. O primeiro inteiro representa a quantidade de segundos desde a data de referência do período corrente, e o segundo inteiro a quantidade de nanosegundos desde o início do segundo.[1]
3. **Tamanho da mensagem.** Todas as mensagens possuem um contador com o número de bytes necessários para representar o texto da mensagem (ignorando os demais campos da mensagem). O tamanho da mensagem é transmitido como um inteiro de 16 bits, sem sinal, em *network byte order*.
4. **Mensagem.** A mensagem deve conter apenas caracteres ASCII e sua representação deve ser menor do que 2^{14} bytes.
5. **Código de verificação de erros.** Cada mensagem possui um hash MD5 (128 bits) para verificação de erros. O MD5 deve ser calculado sobre os quatro campos anteriores.

O diagrama a seguir ilustra uma mensagem de log. Os números indicam a quantidade de bytes até aquele ponto da mensagem.

[1] A data de referência do período corrente é 1970-01-01 00:00:00 +0000 (UTC). Ver documentação das funções [time] e [clock_gettime] da biblioteca padrão do C para maiores detalhes. Em Python, isso é obtido com a função time.time() que dá o tempo em segundos como um número de ponto flutuante.

```
DIAGRAMA 1: Formato de mensagens de log
0        8        16   20 22              22+sz            22+sz+16
+--------+--------+----+--+----/ ... /----+----------------+
|seqnum  |sec     |nsec|sz|message        |md5             |
+--------+--------+----+--+----/ ... /----+----------------+
```

Programas Cliente e Servidor

Você deverá implementar dois programas, um cliente (ou emissor) que gera mensagens de log e as transmite a um servidor (ou coletor) que as recebe, verifica, confirma e armazena.

Programa Cliente (emissor)

O programa cliente deve ler mensagens de log de um arquivo e transmiti-las para um servidor. Para cada linha do arquivo, o cliente deve realizar o montar o pacote (diagrama 1) e transmitir a mensagem ao servidor. Para transmissão eficiente dos dados, o cliente deve implementar uma janela deslizante com tamanho configurável Wtx e transmitir as mensagens de log o mais rapidamente possível. O cliente deve criar um temporizador (Tout) para detectar erros na transmissão de mensagens de log (seção 3.3). Mensagens que não forem confirmadas devem ser retransmitidas após o término do temporizador.

O cliente deve ser executado via a seguinte linha de comando:

```
./client arquivo IP:port Wtx Tout Perror
```

Onde [arquivo] é o nome do arquivo contendo as linhas de log a serem transmitidas, [IP] é o IP do servidor, [port] é o porto que o servidor está utilizando, [Wtx] é o tamanho da janela de transmissão, [Tout] é a duração do temporizador (*timeout*) de retransmissão de mensagens não confirmadas, e [Perror] é a probabilidade do cliente enviar uma mensagem com MD5 incorreto.

O parâmetro Perror e as mensagens com MD5 incorreto servirão para emular erros de transmissão. Seu cliente deve gerar um número aleatório entre [0, 1) ao enviar cada mensagem (inclusive retransmissões); se o número for menor do que Perror, seu cliente deve propositalmente colocar um MD5 incorreto na mensagem. O parâmetro Perror deve estar entre [0, 1).

Ao final da execução, o cliente deve imprimir uma linha com o número de mensagens de log distintas, o número de mensagens de log transmitidas (incluindo retransmissões), o número de mensagens transmitidas com MD5 incorreto, e o tempo total de execução. Utilize um formato de impressão equivalente ao formato [%d %d %d %.3fs] da função [printf] da biblioteca padrão do C. Um exemplo de saída possível é:

```
387 451 43 14.977s
```

Programa Servidor (coletor)

O programa servidor deve receber mensagens de log de clientes e armazená-las em um arquivo. O servidor deve ser capaz de receber e processar sessões de múltiplos clientes simultaneamente, onde uma sessão é uma sequência de mensagens enviadas por um cliente. Para cada cliente (emissor) distinto, o servidor deve armazenar as mensagens em ordem crescente de número de sequência. Ao receber uma mensagem, o servidor deve primeiro verificá-la utilizando o *hash* MD5 anexo a mensagem. Se a verificação falhar, o servidor deve descartar a mensagem. Se a verificação for realizada com sucesso, o servidor deve armazenar essa mensagem na janela deslizante e eventualmente no arquivo de saída.[2] O arquivo de saída deve conter apenas o texto das mensagens; os campos do cabeçalho e rodapé (inclusive o *timestamp*) não devem ser impressos no arquivo de saída.

O servidor deve enviar confirmações para um cliente sempre que receber uma mensagem de log com sucesso. O pacote de confirmação tem 36 bytes, como mostrado no diagrama abaixo. Os primeiros 8 bytes armazenam o número de sequência do quadro sendo confirmado, seguido do timestamp original da mensagem. O quadro de confirmação também tem um MD5 calculado sobre os três campos anteriores para fins de detecção de erros.

```
DIAGRAMA 2: Formato de mensagens de confirmação (ack)
0        8        16   20               36
+--------+--------+----+----------------+
|seqnum  |sec     |nsec|md5             |
+--------+--------+----+----------------+
```

O servidor implementa uma janela deslizante para recebimento de mensagens, e instancia uma janela para cada cliente conectado. O tamanho da janela deslizante de recebimento, [Wrx], deve ser configurável via linha de comando. Além disso, o servidor deve implementar dois mecanismos de confirmação de mensagens de log (discutidos abaixo). O servidor deve ser executado como segue:

```
./server arquivo port Wrx Perror
```

Onde [arquivo] é o arquivo onde deverão ser escritas as mensagens de log ordenadas por número de sequência, [port] é o porto onde o servidor irá receber mensagens, [Wrx] é o tamanho da janela deslizante de recepção, e [Perror] é a probabilidade de enviar uma confirmação com MD5 incorreto (equivalente ao parâmetro [Perror] do cliente).

[2] Como as mensagens precisam ser armazenadas em ordem de número de sequência, o servidor pode precisar armazenar algumas mensagens na janela temporariamente antes de armazená-las permanentemente no arquivo de saída.

Soquetes UDP: Transmissão, Retransmissão e Clientes Ativos

Os clientes e os servidores deverão comunicar através de um soquete UDP.[3] Isto implica que:

- Não existe conexão entre o cliente e o servidor. O servidor deve identificar clientes pelo endereço de origem (IP e porto) dos pacotes contendo as mensagens de log recebidas. Quando a primeira mensagem é recebida de um cliente, um novo par [IP:porto] é identificado e o servidor deve inicializar uma nova janela deslizante para controlar o recebimento de logs daquele cliente.[4]
- Mensagens e confirmações podem ser perdidas ou corrompidas. A detecção de erro deve ser feita recalculando e comparando o MD5 de mensagens e confirmações recebidas. A retransmissão deve ser realizada pelo cliente. O cliente deve manter um temporizador [Tout] (timeout) para cada mensagem de log transmitida e retransmitir mensagens caso detecte algum erro de transmissão (por exemplo, após o fim do temporizador).[5] O valor do temporizador deve ser configurável na linha de comando.

Janela Deslizante com Confirmação Seletiva

Para transmitir as mensagens de log de forma eficiente, os clientes e os servidores deverão implementar janelas deslizantes de tamanho configurável ([Wtx] e [Wrx]). Os clientes e o servidor deverão operar com janelas deslizantes utilizando confirmação seletiva. O servidor deve confirmar cada mensagem recebida individualmente. Se uma mensagem for recebida mais de uma vez, o servidor deve confirmar a mensagem para cada recebimento com sucesso.

Implementação e Interface com Usuário

Seus programas cliente e servidor devem interoperar com outros programas (teste com implementações de colegas) e inclusive com as versões do programa implementados pelo professor. Este trabalho pode ser implementado em Python, C, C++, Java, ou Rust, mas deve interoperar com emuladores escritos em outras linguagens.

- **Inicialização.** Os programas devem ser executados exatamente como descrito acima (com os mesmos parâmetros e na mesma ordem), para facilitar correção semiautomática. Note que o servidor e o cliente podem se comunicar através do IP

[3] Utilize [socket(AF INET, SOCK DGRAM, 0)], ou equivalente na linguagem utilizada, para criar o soquete.
[4] Opcional: Quando nenhuma mensagem é recebida de um cliente por 60 segundos, o servidor pode considerar que o cliente não está mais enviando logs e fechar (deletar) a janela deslizante daquele cliente.
[5] Esta é a explicação do comportamento do algoritmo de retransmissão. A implementação pode ser feita de qualquer forma, desde que atenda este requisito.

127.0.0.1 se o programa estiverem executando na mesma máquina, mas pode também ser o endereço de uma outra máquina acessível pela rede.[6]

- **Terminação.** O cliente deve terminar de executar depois que terminar de enviar todas as mensagens de log para o servidor. O servidor deve executar até receber um sinal de interrupção de teclado (ctrl-c), isto é, quando for terminado manualmente.

Avaliação

- **Grupos.** Este trabalho pode ser executado em grupos de até dois alunos (todos os alunos são responsáveis por dominar todos os aspectos do trabalho).
- **Erros na especificação e implementação.** Qualquer incoerência ou ambiguidade na especificação deve ser apontada para o professor como comentário neste documento; se confirmada a incoerência ou ambiguidade, o grupo receberá pontos extras. A mesma política se aplica a erros na implementação do professor.
- **Entrega.** Seu programa deve ser entregue de forma modular e de forma que a compilação (se necessário) e execução sejam descomplicadas. Descreva na documentação o processo de compilação e execução do seu programa.

Testes

Pelo menos os seguintes aspectos serão testados em seu servidor:

- Seu servidor deve suportar o recebimento de mensagens de vários clientes em paralelo. Teste com clientes que enviam mensagens por um longo período de tempo.
- Seu servidor deve imprimir as mensagens recebidas no arquivo de saída ordenadas por número de sequência. Teste com clientes que enviam mensagens espaçadas ao longo do tempo e com erro.
- Seu servidor deve identificar mensagens corrompidas e não confirmá-las. Teste com clientes que enviam mensagens com MD5 inválido. De forma similar, seu cliente deve identificar confirmações corrompidas e descartá-las. Teste com servidores que enviam confirmações com MD5 inválido.

Documentação

Cada grupo deverá entregar documentação em PDF de até 6 páginas (3 folhas), sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos de projeto, bem como as soluções adotadas para os problemas. Em particular, sua documentação deve:

[6] Para determinar o endereço da máquina onde o servidor vai executar você pode usar o comando [ifconfig], ou fazer o seu servidor escrever o endereço IP da máquina onde ele está executando.

- Apresentar instruções de compilação (se necessário) e execução do seu programa. Os executáveis devem ter o nome de cliente e servidor, exatamente, para fins de correção semiautomática.
- Considere o caso do servidor receber duas mensagens com o mesmo número de sequência e *hashes* MD5 corretos, mas conteúdo (texto) diferente. Este é um caso anômalo. Descreva como seu servidor se comporta nesse caso.
- Apresentar uma breve análise de desempenho, contendo pelo menos dois gráficos comparando a probabilidade de erros com (i) o número de retransmissões de mensagens e (ii) o tempo total para transferência de todas as mensagens.