MAC0422 – Sistemas Operacionais – 1s2025
EP4 (Individual)
Data de entrega: 1/7/2025 até 13:00:00

Prof. Daniel Macêdo Batista

1 Problema

A tarefa neste EP é avaliar o desempenho de servidores de echo concorrentes implementados usando as diferentes abordagens para comunicação entre processos que foram vistas em sala de aula: Internet Sockets com múltiplos processos, Internet Sockets com múltiplas threads, Internet Sockets com multiplexação de Entrada e Saída e Unix Domain Sockets. Você não precisará implementar esses servidores e nem os respectivos clientes. Todos esses códigos já estão disponíveis na área do EP no e-Disciplinas. A sua tarefa será implementar um bash script que vai executar os vários servidores, gerar gráficos utilizando o gnuplot para comparar os tempos necessários para cada servidor fornecer o serviço para clientes concorrentes e fazer um vídeo mostrando o seu script em funcionamento e analisando os resultados obtidos, explicando se os resultados coincidem com o esperado. O bash script deve ser implementado de modo a ser executado no GNU/Linux.

2 Requisitos

O seu bash script deverá receber n argumentos em linha de comando. Um primeiro argumento obrigatório que diz respeito à quantidade de clientes concorrentes que deverão ser executados. Em seguida, $n - 1$ argumentos que dizem respeito aos tamanhos dos arquivos que serão passados como entrada para o cliente que, por sua vez, enviará para o servidor ecoar de volta.

Abaixo está um exemplo de execução do bash script informando que serão executados 10 clientes concorrentes enviando arquivos de 5MB e de 10MB para os servidores. Todas as mensagens que seguem deverão ser impressas pelo seu bash script exatamente da mesma forma como apresentado. As mensagens deixam claro quais são as etapas que precisam ser realizadas pelo bash script e serão detalhadas a seguir:

```
$ ./ep4.sh 10 5 10
Compilando ep4-servidor-inet_processos
Compilando ep4-servidor-inet_threads
Compilando ep4-servidor-inet_muxes
Compilando ep4-servidor-unix_threads
Compilando ep4-cliente-inet
Compilando ep4-cliente-unix
>>>>>>> Gerando um arquivo texto de: 5MB...
Subindo o servidor ep4-servidor-inet_processos
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 5MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
```

```
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-inet_processos: 00:01
Enviando um sinal 15 para o servidor ep4-servidor-inet_processos...
Subindo o servidor ep4-servidor-inet_threads
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 5MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-inet_threads: 00:01
Enviando um sinal 15 para o servidor ep4-servidor-inet_threads...
Subindo o servidor ep4-servidor-inet_muxes
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 5MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-inet_muxes: 00:02
Enviando um sinal 15 para o servidor ep4-servidor-inet_muxes...
Subindo o servidor ep4-servidor-unix_threads
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 5MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-unix_threads: 00:01
Enviando um sinal 15 para o servidor ep4-servidor-unix_threads...
>>>>>>> Gerando um arquivo texto de: 10MB...
Subindo o servidor ep4-servidor-inet_processos
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 10MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-inet_processos: 00:03
Enviando um sinal 15 para o servidor ep4-servidor-inet_processos...
Subindo o servidor ep4-servidor-inet_threads
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 10MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-inet_threads: 00:03
Enviando um sinal 15 para o servidor ep4-servidor-inet_threads...
Subindo o servidor ep4-servidor-inet_muxes
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 10MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-inet_muxes: 00:03
Enviando um sinal 15 para o servidor ep4-servidor-inet_muxes...
Subindo o servidor ep4-servidor-unix_threads
>>>>>>> Fazendo 10 clientes ecoarem um arquivo de: 10MB...
Esperando os clientes terminarem...
Verificando os instantes de tempo no journald...
>>>>>>> 10 clientes encerraram a conexão
>>>>>>> Tempo para servir os 10 clientes com o ep4-servidor-unix_threads: 00:01
Enviando um sinal 15 para o servidor ep4-servidor-unix_threads...
>>>>>>> Gerando o gráfico de 10 clientes com arquivos de: 5MB 10MB
```

2.1 Compilação dos códigos

O arquivo `ep4-clientes+servidores.tar.gz` disponível na área do EP4 no e-Disciplinas possui o seguinte conteúdo:

- `ep4-clientes+servidores/ep4-servidor-inet_processos.c`: Servidor de echo concorrente utilizando sockets da Internet e criando um processo para cada novo cliente via `fork`;
- `ep4-clientes+servidores/ep4-servidor-inet_threads.c`: Servidor de echo concorrente utilizando sockets da Internet e criando uma thread para cada novo cliente via `pthread_create`;
- `ep4-clientes+servidores/ep4-servidor-inet_muxes.c`: Servidor de echo concorrente utilizando sockets da Internet e criando uma posição de memória para cada novo cliente via `select`;
- `ep4-clientes+servidores/ep4-servidor-unix_threads.c`: Servidor de echo concorrente utilizando sockets de Domínio Unix e criando uma thread para cada novo cliente via `pthread_create`;
- `ep4-clientes+servidores/ep4-cliente-unix.c`: Cliente de echo utilizando sockets da Internet (ele deve ser usado para conectar em todos os servidores que utilizam sockets da Internet)
- `ep4-clientes+servidores/ep4-cliente-inet.c`: Cliente de echo utilizando sockets de Domínio Unix (ele deve ser usado para conectar no servidor que utiliza sockets de Domínio Unix)

Leia cada código para entender o que cada um deles faz. Você não vai precisar modificar nenhum desses códigos mas, sendo estudante de ciência da computação, é prudente que você saiba o que você está compilando antes de executar. Em seguida, compile e teste os códigos para confirmar que eles estão funcionando corretamente. Para não se perder nesses testes, faça um a um. Primeiro rode o binário do `ep4-servidor-inet_processos.c`, monitore as mensagens dele nos logs do SO usando o comando `journalctl`, conecte um cliente usando o binário do `ep4-cliente-inet.c`, escreva linhas no cliente terminando com ENTER, verifique que as linhas são ecoadas corretamente e depois que confirmar que está tudo ok, encerre o cliente com um CTRL+d. Depois encerre o servidor enviando o sinal 15 para ele. Repita o mesmo procedimento para os demais servidores, observando que no caso do binário do `ep4-servidor-unix_threads.c`, o cliente que deve ser usado é o `ep4-cliente-unix.c`.

Uma vez que você consiga compilar e rodar tudo manualmente, automatize as compilações no seu script. Considere que todos esses arquivos terão exatamente os mesmos nomes listados acima e que todos estarão dentro do diretório com o mesmo nome do listado acima. Ou seja, ao rodar o `gcc` no seu script, não esqueça de colocar o nome do arquivo começando com `ep4-clientes+servidores/` para não receber um erro de arquivo não encontrado. Os binários devem ter o mesmo nome dos arquivos fonte sem o `.c` e devem ser criados no `/tmp`.

2.2 Geração dos arquivos que serão ecoados

O seu script terá que gerar arquivos em texto puro que serão passados para cada cliente enviar para o servidor. Cada arquivo precisará ter um tamanho específico e esse tamanho será um valor em MB informado na linha de comando para o script. Você pode usar qualquer programa ou comando existente

em uma instalação padrão do GNU/Linux para gerar esses arquivos. Uma sugestão é que você utilize os programas `base64` e `head` e o comando `echo` como no exemplo abaixo em que é gerado um arquivo de 1KB:

```
$ base64 /dev/urandom | head -c 1024 > /tmp/arquivo_de_1KB.txt
$ echo >> /tmp/arquivo_de_1KB.txt
```

Para facilitar a depuração do seu script, coloque o tamanho do arquivo no nome dele. Por exemplo 05MB, 10MB, 15MB, etc... . Todos os arquivos devem ser gerados no `/tmp`.

Como você terá que avaliar o desempenho dos servidores com diversos tamanhos de arquivos, o laço nos tamanhos dos arquivos deve ser o primeiro laço (o mais externo) do seu script.

2.3 Execução dos servidores

Cada servidor deve ser testado separadamente, um de cada vez. Certifique-se de que nenhuma execução anterior do servidor esteja em funcionamento. Como você terá que avaliar o desempenho dos vários servidores, o laço nos binários deles deve ser o segundo laço do seu script, dentro do laço anterior dos tamanhos dos arquivos.

2.4 Execução dos clientes concorrentes

Uma vez com o servidor no ar, seu script precisará executar o número de clientes informado na linha de comando. Isso exigirá um terceiro laço de 1 até essa quantidade rodando os clientes em segundo plano no shell, dentro do laço anterior dos servidores. Além disso, é importante garantir que nenhuma saída de texto, tanto para o `stdout` quanto para o `stderr` seja enviada para o terminal. Caso contrário, vai ficar bem difícil de depurar o seu script. Uma forma de rodar um cliente passando para ele um arquivo de entrada, eliminando as mensagens na saída padrão e na saída de erro e em segundo plano é por exemplo:

```
$ /tmp/ep4-cliente-inet 127.0.0.1 < /tmp/nome_do_arquivo &>/dev/null &
```

2.5 Aguardando os clientes terminarem

Depois de sair do laço do passo anterior, é necessário criar outro laço aguardando os clientes terminarem, também dentro do laço dos servidores. Se você fizer esse laço monitorando os nomes dos processos com o comando `ps`, garanta que você coloque um sleep em cada iteração para evitar que seu script vire uma espera ocupada que fique gastando CPU que deveria estar sendo gasta pelo servidor que está atendendo aos clientes.

2.6 Contabilização do tempo que o servidor levou para atender todos os clientes

Após sair do laço que aguardou todos os clientes finalizarem, você deve verificar os logs do servidor no `journald` usando o comando `journalctl` para confirmar que a quantidade correta de clientes de fato foi servida pelo servidor, qual o instante de tempo em que o primeiro cliente começou a ser atendido pelo servidor e qual o instante de tempo que o último cliente terminou de ser atendido pelo servidor. Leia o código do servidor para entender quais mensagens nos logs devem ser usadas para descobrir o início de um serviço e o encerramento de um serviço para que você contabilize corretamente o tempo para atender todos os clientes do início ao fim.

Uma dica para esta parte: ao utilizar o comando `journalctl`, passe o parâmetro `-q`, para remover várias mensagens de informação do SO que não vão te ajudar, e passe também o parâmetro `--since` com a hora em que o servidor começou a rodar, para evitar que você pegue logs de alguma execução passada do servidor de outras vezes em que você rodou o script. Para guardar a hora em que o servidor começou a rodar, use o programa `/bin/date` logo depois que você executá-lo.

Outra dica para esta parte: como você verá adiante, ao apresentar os gráficos com os resultados, o eixo y vai precisar mostrar os tempos no formato mm:ss. Para facilitar encontrar esse valor você pode usar o programa `dateutils.diff` (No Debian ele faz parte do pacote `dateutils`). Abaixo está uma execução do programa para servir de exemplo:

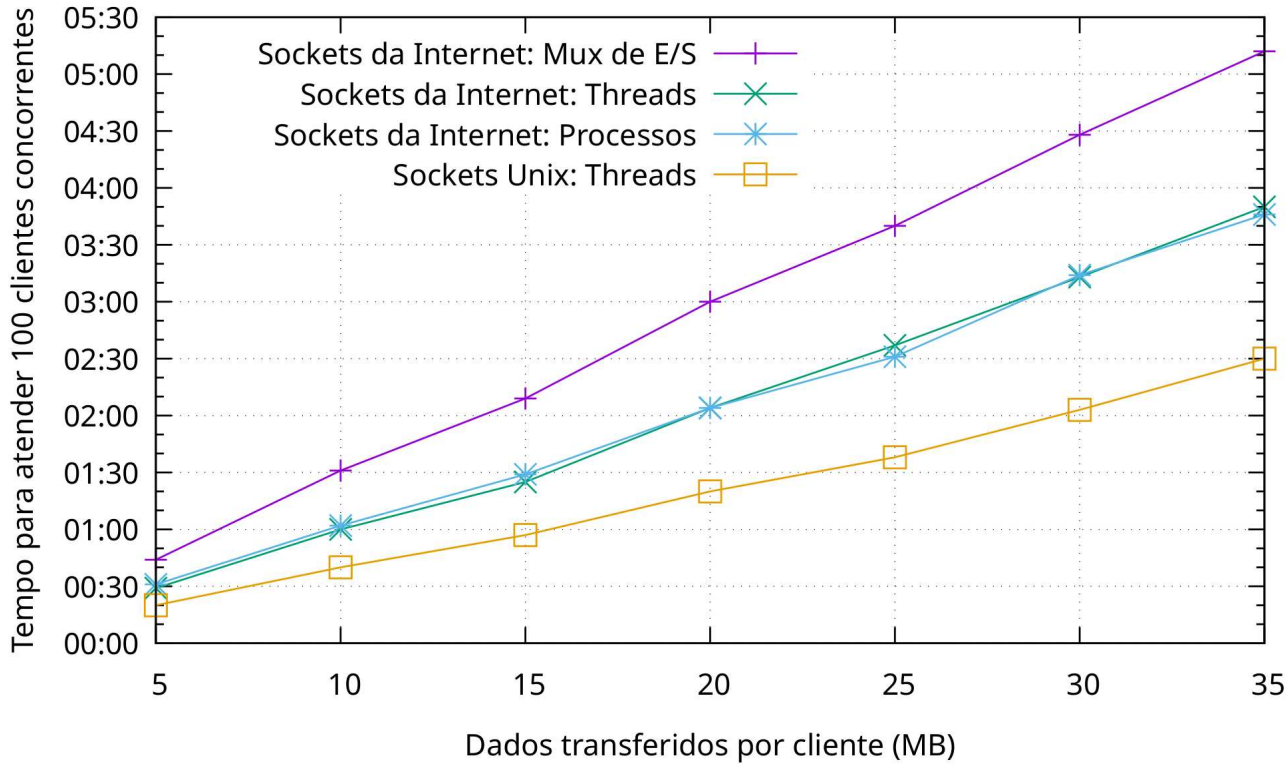
```
$ dateutils.ddiff "2025-02-18 23:43:21" "2025-02-19 00:15:35" -f "%0M:%0S"
32:14
```

O programa `dateutils.diff` estará instalado na máquina que será usada para corrigir os EPs.

Uma vez que você consiga coletar o tempo necessário para o servidor atender todos os clientes com um determinado tamanho de arquivo, guarde esse valor e encerre o servidor enviando um sinal 15 para ele. Em seguida, o laços seguirão com as iterações restantes até todas as coletas serem realizadas.

2.7 Geração do gráfico

O resultado da execução do script deve ser um gráfico em formato PDF como o gráfico abaixo que mostra os resultados da execução dos 4 servidores atendendo 100 clientes com arquivos de tamanhos 5MB, 10MB, 15MB, 20MB, 25MB, 30MB e 35MB.



O gráfico tem que ser gerado pelo programa `gnuplot` que já estará instalado na máquina que será usada para corrigir os EPs. O `gnuplot` precisa ser executado com um arquivo de configuração que define como ele deve gerar o gráfico. Para gerar o gráfico acima, foi utilizado este arquivo de configuração abaixo, salvo com a extensão `.gpi`, que é a extensão comumente usada para arquivos de configuração do `gnuplot`:

```
set ydata time
set timefmt "%M:%S"
set format y "%M:%S"

set xlabel 'Dados transferidos por cliente (MB)'
set ylabel 'Tempo para atender 100 clientes concorrentes'

set term pdfcairo
set output "ep4-resultados-100.pdf"

set grid
set key top left

plot "/tmp/ep4-resultados-100.data" using 1:4 with linespoints title "Sockets da Internet: Mux de E/S",\
     "/tmp/ep4-resultados-100.data" using 1:3 with linespoints title "Sockets da Internet: Threads",\
     "/tmp/ep4-resultados-100.data" using 1:2 with linespoints title "Sockets da Internet: Processos",\
     "/tmp/ep4-resultados-100.data" using 1:5 with linespoints title "Sockets Unix: Threads"
```

O seu script deverá criar esse arquivo antes de rodar o `gnuplot` no `/tmp`. Note que o arquivo `.gpi` acima faz referência a um arquivo `/tmp/ep4-resultados-100.data` que o script tem que gerar com os tempos coletados. O arquivo precisa ser similar a este abaixo para que o `gnuplot` consiga funcionar corretamente. Esse arquivo também precisa ser criado pelo seu script no `/tmp` (No geral, esses arquivos com os valores a serem plotados pelo `gnuplot` têm extensão `.data`).

```
05 00:31 00:29 00:44 00:20
10 01:02 01:00 01:31 00:40
15 01:29 01:25 02:09 00:57
20 02:04 02:04 03:00 01:20
25 02:31 02:37 03:40 01:38
30 03:14 03:13 04:28 02:03
35 03:46 03:50 05:12 02:30
```

Você não precisa criar o seu arquivo `.data` exatamente como apresentado acima. Você pode criar ele em outro formato ou mesmo criar mais de um arquivo, um para cada servidor. A documentação do `gnuplot` é muito rica e ajuda a bolar formatos diferentes para o arquivo `.data`. Navegue na documentação caso precise entender ou modificar algum dos parâmetros do arquivo `.gpi` que seu script vai gerar. Para navegar nessa documentação basta rodar o comando `gnuplot` sem parâmetros no shell e digitar `help`.

2.8 Término do script

Remova todos os arquivos temporários que foram gerados pelo seu script. Os únicos arquivos que não existiam antes do script executar e que devem aparecer no mesmo diretório onde o script está, são os gráficos em PDF gerados pelo `gnuplot` (Não se preocupe em apagar os fontes. Eles serão colocados e apagados no diretório por quem for corrigir). O código de saída do script deve ser 0 se tudo funcionar corretamente e ele chegar nesse ponto. Códigos diferentes de zero podem ser retornados em outras partes do script caso erros ocorram e não faça sentido prosseguir.

3 Sobre a entrega

Deve ser entregue um arquivo `.tar.gz` contendo os itens listados abaixo. EPs que não contenham **todos** os itens abaixo **exatamente como pedido em termos de formato e de nomes** terão nota ZERO e não serão corrigidos. **A depender da qualidade do conteúdo entregue**, mesmo que o EP seja entregue, **ele pode ser considerado como não entregue, o que mudará o cálculo da média final**:

- 1 único arquivo `ep4.sh` com a implementação do script;
- 5 arquivos PDF `ep4-resultados-100.pdf` `ep4-resultados-200.pdf` `ep4-resultados-300.pdf` `ep4-resultados-400.pdf` `ep4-resultados-500.pdf` resultantes da execução do script para avaliar o desempenho dos servidores atendendo 100, 200, 300, 400 e 500 clientes concorrentes transferindo arquivos de 5MB, 10MB, 15MB, 20MB, 25MB, 30MB e 35MB. Ou seja, o script precisará ser executado 5 vezes;
- 1 único arquivo `LEIAME` em formato texto puro explicando como rodar o script e com o link para um vídeo de no máximo 10 minutos com a demonstração do script em funcionamento para uma entrada específica e com a explicação dos resultados obtidos tanto nessa demonstração quanto com os 5 gráficos do item anterior. O vídeo deverá iniciar com a execução do comando `md5sum` em cada um dos arquivos dos itens anteriores (o script e os 5 gráficos) para garantir que eles são exatamente os mesmos que estão presentes no `.tar.gz` que foi enviado. Portanto, faça esse vídeo apenas quando terminar de rodar os experimentos e quando terminar o vídeo, não modifiquei mais nenhum arquivo. Se o MD5 dos arquivos do vídeo não coincidir com o MD5 dos arquivos no .tar.gz você terá nota ZERO. O comando `md5sum` deve ser rodado de fato no vídeo. O vídeo não pode começar já com a saída do programa na tela sem ele ser executado na hora. A demonstração que você precisa apresentar no vídeo é da execução do seu script com os parâmetros `10 5 10`, ou seja, o script vai rodar 10 clientes concorrentes ecoando arquivos de 5MB e de 10MB para os servidores. Para todos os gráficos, tanto o dessa demonstração quanto os 5 gráficos que já terão sido gerados, explique se os resultados foram ou não esperados, justificando com a teoria vista em sala de aula, e como foi a progressão desses resultados à medida que os arquivos aumentam de tamanho e à medida que o número de clientes aumenta. Não use slides. Faça as suas explicações usando os gráficos. Recomenda-se ampliar o ponteiro do mouse ao fazer o vídeo para que fique fácil acompanhar a explicação (Geralmente isso pode ser feito nas configurações de acessibilidade da interface gráfica do seu sistema operacional). O vídeo pode ser colocado no youtube ou no google drive. **Não grave o vídeo usando alguma câmera como a de um smartphone. Utilize algum software de captura como o OBS ou entre sozinho em uma reunião no google meet, compartilhe sua tela e grave a reunião**. Garanta que o vídeo esteja com permissão para que o professor e os monitores consigam acessar. Compartilhe o vídeo no youtube ou no google drive com o endereço de e-mail do professor que é batista@ime.usp.br. Compartilhe também com os endereços de e-mail dos monitores que serão informados no decorrer do semestre. **Não envie o vídeo no .tar.gz e nem por e-mail. Você precisa colocar no youtube ou no google drive e informar o link no LEIAME.** O vídeo vale 3,0 pontos.

Para garantir que as medições não sejam afetadas por outros programas do seu computador, evite rodar outros processos que consumam muita CPU e memória enquanto os experimentos estiverem sendo executados. O ideal mesmo é que você mantenha apenas os terminais com o código do EP rodando e, na hora de gravar o vídeo, tenha apenas o software para a gravação além dos terminais com o EP rodando.

O desempacotamento do arquivo `.tar.gz` deve produzir um diretório contendo os itens. O nome do diretório deve ser `ep4-seu_nome`. Por exemplo: `ep4-artur_avila`. Entregas que sejam tarbombs ou que, quando descompactadas, gerem o diretório com o nome errado perderão 41,0 pontosponto.

A entrega do `.tar.gz` deve ser feita através do e-Disciplinas.

O EP deve ser feito individualmente.

Obs.: não inclua no `.tar.gz` itens que não foram pedidos neste enunciado, como por exemplo, dotdirs como o `.git`, dotfiles como o `.gitignore`, saídas para diversas execuções diferentes das solicitadas, arquivos pré-compilados, os fontes dos servidores e dos clientes, o vídeo, etc.... A presença de conteúdos não solicitados no `.tar.gz` levarão a um desconto de 41,0 na nota final.