sqtpm

Guilherme P. Telles

13 de dezembro de 2022

O sqtpm é um sistema para recepção via web, compilação e verificação de correção de trabalhos de programação. Permite a recepção de programas em C, Fortran, C++, Pascal, Python 3, Java e Octave, e de arquivos PDF.

A primeira versão do sqtpm foi escrita no fim de 2003 quando assumi duas turmas de programação na USP. Organizar e verificar a correção dos trabalhos de programação consumia muito tempo. O sqtpm foi colocado no ar pela primeira vez em janeiro de 2004. O sistema evoluiu um pouco com o passar do tempo, mas a filosofia continua a mesma desde a primeira versão: o sistema organiza os trabalhos por diretórios, arquivos e links e o responsável pelos trabalhos gerencia o sistema na linha de comandos. Não há uma interface web de administração, nem há uma interface específica para avaliação de aspectos não-funcionais dos trabalhos.

O sqtpm teve sua inspiração inicial no sistema MC102 construído pelo Zanoni Dias na Unicamp e principalmente nas conversas com o próprio Zanoni, mas não tem código do MC102. Não obstante, a forma de limitar os recursos via `ulimit` durante a execução dos programas submetidos é igual à usada pelo MC102. À época da primeira versão também tomei conhecimento acerca da existência de outros sistemas do gênero, mas independentemente da existência deles achei melhor fazer do zero do que usar ou adaptar outros sistemas. Desta forma eu teria o prazer de programar, poderia tomar as decisões que eu bem entendesse e não precisaria lidar com programas de outras pessoas.

Este documento descreve o sqtpm. Foi escrito inicialmente para amadurecer as idéias, antes de programar, e agora serve para ajudar a manter, a instalar e a configurar o sistema. É uma mistura de documento de requisitos, de arquitetura e manual, que não é nenhum deles. Este documento descreve a versão 10.

Conteúdo

1 Organização do sistema

O sqtpm está escrito em Perl. Usa Session de Perl-CGI e ExpireSessions de Perl-Session, make, diff, tar, moss (`theory.stanford.edu/~aiken/moss`) e opcionalmente indent, além dos compiladores. Também empacota o google-code-pretiffier. O sistema nunca foi testado em um sistema diferente de GNU/Linux com Apache.

O sqtpm é organizado em uma hierarquia de diretórios onde os componentes dele, as definições de trabalhos e os programas enviados ficam armazenados. O sistema funciona em um diretório que chamaremos **diretório-raiz**. O diretório-raiz contém:

- os programas e arquivos do sistema,
- os arquivos de grupos de usuários, com sufixo `.pass` e
- um subdiretório para cada trabalho.

O comportamento do sqtpm quando nomes de diretórios e nomes de arquivos contêm espaços não foi testado.

1.1 Usuários e senhas

No diretório-raiz deve haver um ou mais arquivos de usuários com nome que tenha sufixo `.pass`. O formato de um arquivo de usuários é:

```
identificador:senha-criptografada
identificador:senha-criptografada
...
identificador:senha-criptografada
```

Os arquivos de usuários devem ser criados apenas com os identificadores seguidos de dois-pontos, um por linha. As senhas são adicionadas pelos usuários através da interface do sistema. O caractere `#` faz com que o conteúdo de uma linha seja ignorado a partir da primeira ocorrência dele.

Os identificadores de usuários são cadeias de letras e dígitos, opcionalmente precedidos por um `*` (asterisco) ou por um `@` (arroba). Cada identificador deve ser único na união de todos os arquivos de senhas. Não deve haver usuários com identificador backup, extra-files, include ou com prefixo config. O tamanho de um identificador ou senha é limitado a 20 símbolos nos formulário html.

O sistema tem três tipos de usuários: professor, monitor e aluno.

- Usuários que têm um `*` precedendo o identificador no arquivo de senhas são do tipo professor. Um professor pode ver tabelas consolidadas com as submissões para todas as turmas, pode submeter trabalhos antes do início do prazo ou depois do fim do prazo e sem limite no número de submissões e pode comparar trabalhos pelo Moss.
- Usuários que têm um `@` precedendo o identificador no arquivo de senhas são do tipo monitor. Um monitor pode ver as submissões de cada trabalho, submeter trabalhos antes do início do prazo ou depois do fim do prazo e sem limite no número de submissões.
- Os demais usuários são alunos. Um aluno pode ver apenas trabalhos e envios dele mesmo.

Cada usuário cadastra sua senha sem intervenção, através da interface do sistema. O sistema foi construído supondo que a lista de usuários possa ser extraída facilmente de algum sistema acadêmico e que cada arquivo contenha usuários de uma mesma turma, ou de outro grupo que faça sentido nas disciplinas.

1.2 Trabalhos

Todo subdiretório do diretório-raiz cujo nome não começa com ponto e que contém um arquivo chamado `config` é um trabalho. No diretório de um trabalho ficam os seguintes arquivos criados pelo professor:

- o arquivo de configuração do trabalho, chamado `config`,
- links simbólicos para arquivos de usuários,
- opcionalmente, os casos-de-teste do trabalho,
- opcionalmente, o programa verificador,
- opcionalmente, arquivos de configuração do trabalho por grupo com prefixo `config-`,
- opcionalmente, o diretório `include`,
- opcionalmente, o arquivo `include.tgz` e
- opcionalmente, o diretório `extra-files`,
- opcionalmente, o arquivo `casos-de-teste.tgz` e
- opcionalmente, arquivos `.html` e `.png` que compõem o enunciado.

Além desses arquivos, no diretório do trabalho o sistema pode criar:

- um diretório para cada usuário que enviou o trabalho,
- o diretório `backup`,
- arquivos para comparação de saídas de programas, com sufixo `.lc`,
- arquivos com tabelas de acertos de grupos de usuários, com sufixo `.grades` (sobre esses arquivos veja também a Seção 4.4).

Ao submeter um trabalho, um usuário deve enviar um ou mais programas-fonte. O trabalho pode ser configurado para limitar a quantidade e nomes dos arquivos.

1.2.1 Permissão para submeter

Um usuário enxerga e pode enviar um certo trabalho `T` se o identificador dele estiver em algum arquivo de usuários para o qual existe um link dentro do diretório `T`. Essa estrutura permite administrar as autorizações através da criação e remoção de links simbólicos.

Por exemplo, suponha que há três turmas de alunos que submeterão trabalhos da seguinte forma:

Trabalho	Turmas que devem fazer o trabalho
busca em largura	123
árvore geradora	123, 456
caminhos mínimos	456, 789

Os arquivos de usuários e respectivos links simbólicos podem ser construídos da forma como aparecem na Figura 1.

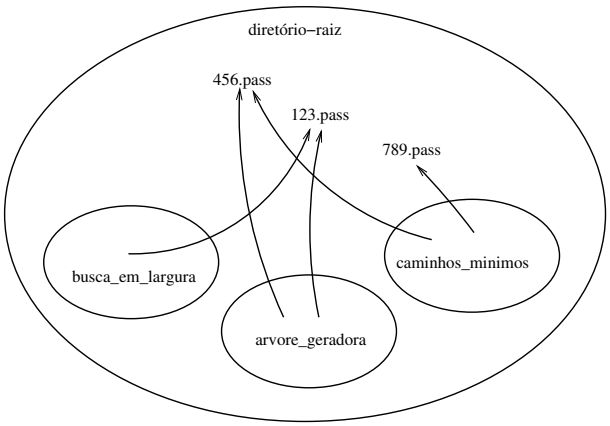

Figura 1: Exemplo de trabalhos e arquivos de grupos.

1.2.2 Enunciado

O sistema pode exibir o enunciado de cada trabalho. O enunciado pode ser um arquivo html dentro do diretório do trabalho ou pode ser uma url. Se for um arquivo html, o sistema exibe o arquivo. Se for uma url, o sistema exibe um link para ele. O sistema também exibe informações sobre o trabalho, como datas, linguagens, número e nomes de arquivos e outras.

1.2.3 Download de casos-de-teste

Se houver um arquivo chamado `casos-de-teste.tgz` no diretório do trabalho então o sistema exibe um link para download desse arquivo na página do enunciado.

1.2.4 Download de código-fonte fornecido juntamente com o enunciado

Se houver um arquivo chamado `include.tgz` no diretório do trabalho então o sistema exibe um link para download desse arquivo na página do enunciado.

1.3 Offline

Se o diretório-raiz tiver um arquivo chamado `offline` então o sistema exibe uma mensagem de manutenção e fecha todas as seções abertas.

2 Envio, compilação e execução

Quando um usuário envia um trabalho, o sistema verifica o identificador, a senha, o número máximo de submissões, as datas-limite, a linguagem, a

quantidade, os nomes e os sufixos dos nomes dos arquivos. Se o usuário com identificador `id` submete o trabalho de programação `T` então depois das verificações os arquivos enviados são gravados no diretório `raiz/T/id/`.

Quando o trabalho é apenas para recepção de arquivos PDF então o sistema grava os arquivos no diretório do usuário e termina, sem emissão de uma pontuação. Caso contrário, o programa é compilado (exceto Python e Octave) e executado para cada um dos casos-de-teste que estiverem no diretório do trabalho. Depois de executados, os resultados dos casos-de-teste são verificados e o usuário recebe uma pontuação de 0 a 100%, descontada da multa por atraso se houver. A pontuação pode ser proporcional ao número de casos-de-teste bem sucedidos ou pode ser "0 ou 100". Se não houver casos-de-teste, o sistema recebe o programa e compila e a pontuação registrará apenas "recebido" ou "recebido +d" para um atraso de d dias.

Se houver um diretório `include` então todos os arquivos dele serão copiados para o mesmo diretório em que os fontes estarão durante a compilação.

Se houver um diretório `extra-files` então todos os arquivos e subdiretórios dele serão copiados para o mesmo diretório em que o executável estará durante a execução dos casos-de-teste.

O `include` e o `extra-files` não devem conter arquivos com os sufixos `.in` e `.out`, que são entradas e saídas dos casos-de-teste.

Se o usuário fizer uma re-submissão do trabalho `T` e se a opção de backup estiver ativada, os arquivos pré-existentes serão movidos para o diretório `raiz/T/backup/id.tentativa.data/`, onde `tentativa` é o número daquele envio e `data` é a data em que o programa substituído havia sido enviado. Se a opção de backup estiver desativada então os arquivos pré-existentes serão removidos.

2.1 Verificação e casos-de-teste

A forma dos casos-de-teste depende do tipo de verificação que será realizada. Há duas formas de verificação de acerto na execução de um caso-de-teste: por comparação com uma saída esperada e por programa verificador. A seleção do tipo de correção é feita no arquivo de configuração do trabalho.

Verificação por comparação Se a verificação for por comparação com uma saída esperada, cada caso-de-teste `z` é composto por dois arquivos:

- `z.in`, com a entrada para o programa e
- `z.out`, com a saída esperada do programa quando alimentado com a entrada `z.in`.

O resultado da execução de cada caso-de-teste é comparado com o esperado usando `diff`. Para cada arquivo `z.out`, o sistema pode escrever um arquivo `z.out.lc`, usado para melhorar o desempenho das comparações com

diff. Os arquivos `lc` são gerados de novo se forem apagados ou se o arquivo `out` correspondente for atualizado.

O resultado da execução de um caso-de-teste pode ser:

- `bem-sucedido`: o resultado produzido pelo programa está idêntico ao esperado.
- `saída com formatação incorreta`: o resultado produzido pelo programa difere do esperado por espaços, fins-de-linha ou caixa de caracteres. O caso-de-teste é considerado mal-sucedido.
- `saída incorreta`: o resultado produzido pelo programa difere do esperado. O caso-de-teste é considerado mal-sucedido.
- `limite de tempo ou memória excedido`.
- `violação de memória`.
- `erro de ponto flutuante`.
- `erro de execução (n)`: erro de execução com exit status n diferente de 9, 11 ou 8.

Embora o nome do arquivo de um caso-de-teste possa ser uma string arbitrária, a tela do sistema exibe apenas números de 1 até n que correspondem à ordem lexicográfica dos nomes dos arquivos. Uma boa prática de organização tem sido dar nomes da forma `dd-string.{in,out}`, porque permitem uma associação direta entre o nome exibido na tela e o nome do arquivo. Por exemplo, `01-grafo-vazio.in`, `02-k5.in`, `03-k3-3.in` etc.

Verificação por programa A verificação através de programa pode ser usada quando a resposta correta para uma entrada não for única.

Se a verificação for feita por um programa, então cada caso-de-teste deverá ser composto apenas de um arquivo de entrada `z.in`. O resultado da execução será igual ao produzido na verificação por comparação e dependerá do valor retornado pelo programa verificador, como discutido na Seção 4.2.

2.2 Log de submissões

O sistema grava um log de submissões chamado `sqtpm.log` no diretório-raiz. Cada linha do log refere-se a uma submissão ou tentativa de acesso.

3 Linguagens e compiladores

As linguagens de programação com as quais os sqtpm já foi testado são C com gcc, C++ com g++, Fortran com gfortran, Pascal com gpc, Java com JDK Oracle, Python com o interpretador default do Linux e GNU Octave. O acoplamento entre os compiladores/interpretadores e o sqtpm

é moderadamente forte, de forma que a troca de algum compilador pode exigir modificar o sqtpm.

As opções globais para os compiladores são definidas no `sqtpm.cfg`. Elas podem ser redefinidas para cada trabalho, como está descrito na Seção 4.2. Não haverá problema em não ter um compilador instalado se a linguagem não for usada.

Programas em C, C++ e Java podem ser formados por um ou mais arquivos. Java exige que a função main esteja na classe Main, então deve haver uma arquivo `Main.java`. Python pode ser formado por um ou mais arquivos, se houver apenas um ele pode ter qualquer nome, se houver mais de um deverá haver um arquivo chamado `main.py`. Octave está limitada a um único arquivo. Fortran e Pascal estão limitadas a receber apenas um arquivo-fonte (as versões mais antigas do sqtpm tinham essa limitação e não fiz os ajustes necessários para removê-las.) Essas limitações estão cravadas no código do sqtpm.

Os nomes dos arquivos-fonte devem ser cadeias de letras, dígitos, pontos, hífens e sublinhados com os seguintes sufixos, de acordo com a linguagem:

- C: .c e .h
- C++: .cpp e .h
- Pascal: .pas
- Fortran: .f ou .F
- Python3: .py
- Java: .java
- Octave: .m

Há ainda a "linguagem" PDF, que apenas permite a recepção de arquivos desse tipo. Para esses arquivos o sufixo deve ser `.pdf`.

4 Configuração

Há um arquivo para configuração do sistema, cada trabalho deve ter seu próprio arquivo de configuração e cada trabalho pode ter arquivos de configuração por grupo de usuários.

Tais arquivos contêm linhas da forma `diretriz=valor`. O caractere `#` faz com que o conteúdo de uma linha seja ignorado a partir de sua primeira ocorrência.

4.1 Configuração do sistema

O arquivo `raiz/sqtpm.cfg` define valores-padrão de diretrizes para todos os trabalhos e diretrizes de configuração do sistema.

Um modelo de arquivo `sqtpm.cfg` aparece a seguir. A seção 4.2 inclui uma descrição das diretrizes.

```
languages = C Fortran PDF C++ Pascal Python3 Java Octave
tries = 20
backup = on
grading = proportional
penalty = 100
keep-open = 3
files = 1,3
cputime = 1
virtmem = 32768
stkmem = 8192

make = /usr/bin/make
diff = /usr/bin/diff
indent = /usr/bin/indent
tar = /bin/tar

gcc = /usr/bin/gcc
gcc-args = -Wall -O3
g++ = /usr/bin/g++
g++-args = -Wall -O3
gfortran = /usr/bin/gfortran
gfortran-args = -Wall
gpc = /usr/local/bin/gpc -Wa,--32 -B /usr/lib32
gpc-args = -Wall -O2
jdk = /opt/jdk/bin
#javac-args =

moss-id = 1234567890
moss-m = 100000
```

As primeiras 10 definem valores padrão que afetam a recepção dos trabalhos e execução dos casos-de-teste.

O segundo grupo tem diretrizes que especificam o path para programas externos usados pelo sqtpm.

No terceiro grupo estão as diretrizes que especificam o path para os compiladores (no caso de Java, o path para o diretório dos binários do JDK) e os parâmetros para eles. Observe que não há um parâmetro para Python e Octave. Octave não permite verificação da sintaxe sem executar e Python até permite mas é chato demais para valer a pena; portanto os intepretadores são usados apenas pelo wrapper de execução (Seção s:seguranca). Além de Octave e Python, também há definições dos interpretadores para Java no wrapper de execução.

As duas últimas diretrizes afetam o funcionamento do Moss e são descritas na Seção 6.

As diretrizes que certamente podem ser omitidas são as de compiladores que não são usados, do Moss e do indent. A omissão das demais não foi testada, nem será.

O sqtpm usa ulimit para limitar o uso de recursos pelo programa. Parece ser muito difícil limitar memória para Java e Python usando ulimit, então essas linguagens estão com limitação apenas de tempo nos meus executores (veja as seções 4.2 e 7).

Diretrizes que aparecem na configuração do sistema podem ser redefinidas na configuração de cada trabalho e trabalho por grupo, mas as duas últimas não terão efeito se forem redefinidas para um trabalho.

4.2 Configuração de trabalhos

Cada diretório de trabalho deve conter um arquivo chamado `config`. Cada diretório de trabalho pode conter arquivos com nome com prefixo `config-` seguido de um sufixo para um grupo (prefixo de arquivo `.pass`), que permite refinar as configurações para cada grupo que pode submeter aquele trabalho. Por exemplo, na configuração ilustrada na Figura 1, no diretório `arvore_geradora` deve haver o arquivo `config` e pode haver os arquivos `config-123` ou `config-456`.

Como regra geral, as definições que existirem nos arquivos mais especializados substitirão as que porventura existam nos mais gerais. As diretrizes definidas nos arquivos mais especializados afetarão apenas o trabalho ou grupo de um trabalho para o qual forem definidas. Além disso,

- Para usuários do tipo aluno todos os arquivos serão incluídos.
- Para usuários do tipo monitor arquivos de configuração por grupo não serão incluídos.
- Para usuários do tipo professor arquivos de configuração por grupo não serão incluídos e as linguagens serão as definidas em `sqtpm.cfg`.

4.3 Diretrizes

As diretrizes válidas são as seguintes.

- `languages = lista` Define uma lista de linguagens de programação que serão aceitas. Os itens da lista devem ser separados por espaços. Os valores válidos são: `C Fortran PDF C++ Pascal Python3 Java Octave`. Caixas altas e baixa são importantes e estritas.
- `startup = aaaa/mm/dd hh:mm:ss` Define a data e o horário para começar a exibir o enunciado e passar a aceitar submissões de um programa. Se não for incluída, o trabalho passa a ser exibido para usuários que fizerem login depois que os links para arquivos de usuários forem criados. O formato deve ser seguido estritamente.
- `deadline = aaaa/mm/dd hh:mm:ss` Define a data e o horário limites para a submissão do trabalho. Se não for incluída, o trabalho não deixa de ser aceito. O formato deve ser seguido estritamente.
- `keep-open = n` Especifica o número de dias depois do deadline durante os quais um trabalho encerrado continuará sendo aceito mas sem substituir a última submissão feita dentro do prazo. O número

máximo de submissões é acrescido de 10 tentativas e esse limite aumentado é verificado. Se o backup estiver ativo, o sistema salva os envios. Não funciona para PDF.

- `penalty = n` Define a multa percentual sobre os pontos obtidos no trabalho, cobrada por dia de atraso passado da data limite para submissão. Deve ser um número inteiro no intervalo $]0, 100]$. A multa passa a ser aplicada já no primeiro segundo após a data limite. O sistema continuará aceitando o trabalho enquanto o usuário puder ter pontuação maior que zero.
- `grading = {total,proportional}` Define a forma de calcular a pontuação para o trabalho. `total` significa pontuação 100% se o programa for bem sucedido em todos os casos-de-teste ou pontuação 0% caso contrário. `proportional` significa proporcional ao número de casos-de-teste bem sucedidos.
- `description = string` Define o enunciado do trabalho. Há duas alternativas. A primeira é o nome de um arquivo html no diretório do trabalho, que será ecoado no browser. A outra é uma url iniciada por `http` para a qual o sqtpm exibirá um link. Exemplos são:

  ```
  description = http://143.107.183.131/problemas/t1.html
  description = enunciado.html
  ```

- `backup = {on,off}` Define se o sistema manterá ou não cópias das submissões anteriores.
- `cputime = n` Define o tempo de CPU máximo para executar cada caso-de-teste, em segundos (número inteiro). Esse valor é usado com `ulimit -t`.
- `virtmem = n` Define o tamanho máximo de memória virtual para executar cada caso-de-teste, em kilobytes (número inteiro). Esse valor é usado com `ulimit -v`.
- `stkmem = n` Define o tamanho máximo de pilha para executar cada caso-de-teste, em kilobytes (número inteiro). Esse valor é usado com `ulimit -s`.
- `showcases = lista` Define uma lista de nomes de casos-de-teste para os quais o sistema mostrará a entrada, a saída esperada e a saída produzida pelo programa enviado. Se a correção for feita por programa verificador, então não haverá a exibição da saída esperada. Por exemplo:

  ```
  showcases = 01-grafo-vazio 03-k3-3
  ```

- `tries = n` Define o número máximo de vezes que um trabalho pode ser enviado.

- `files = n,m` Se esta diretriz for definida com valores inteiros `n` e `m` tais que $0 < n \leq m$ então o sistema exige que o número de arquivos fonte esteja entre `n` e `m`.
- `filenames = lista` Define uma lista de nomes separados por espaços. Se for definida então um subconjunto dos arquivos submetidos deve ter arquivos com nomes iguais aos da lista. Se a cadeia {`uid`} fizer parte de algum nome, ela será substituída pelo identificador do usuário. Se a cadeia {`assign`} fizer parte de algum nome, então ela será substituída pelo nome do trabalho. Pode ser combinada com a diretriz `files`.
- `verifier = comando` Define um comando para a execução de um programa verificador. Se esta diretriz não estiver presente, o sistema fará a verificação dos casos-de-teste por comparação da saída.

  O programa verificador deve aceitar como parâmetros dois nomes de arquivos: a entrada de um caso-de-teste e a saída produzida pelo programa (ambos com path absoluto), nesta ordem. O programa verificador deve retornar 0 se a saída for correta, 1 se for incorreta, 2 se houver erro de formatação e > 2 se houver um erro de execução do próprio verificador. O programa verificador não será interrompido pelo sqtpm e portanto deve ser o mais robusto possível para que não surjam zumbis.

  Se o comando começar com o caractere `@`, então o caractere `@` será substituído pelo path do diretório do trabalho. Por exemplo, se um trabalho `T` possui como verificador um executável chamado `verif`, que está no próprio diretório do trabalho, a diretriz será:

  ```
  verifier = @verif
  ```

  Outros exemplos são:

  ```
  verifier = /tmp/verif
  verifier = /usr/bin/perl @verif.pl
  verifier = /usr/bin/perl /tmp/verif.pl
  ```

- ```
  gcc-args = string
  g++-args = string
  gpc-args = string
  gfortran-args = string
  javac-args = string
  ```

  Definem parâmetros adicionais para os compiladores. Não devem incluir o parâmetro `-o`.
  ```

4.4 Mudança do nome de trabalhos

Se o diretório de um trabalho for renomeado enquanto ele estiver aberto, haverá inconsistência entre arquivos de tabelas de acertos (sufixo `.grades`) e o nome do diretório do trabalho. Se houver essa inconsistência, o usuário será bloqueado ao tentar acessar um relatório de submissão a partir da tabela de acertos. Para evitar essa situação, os arquivos `.grades` devem ser removidos sempre que um trabalho for renomeado. Observe que as submissões feitas antes da mudança de nome ainda terão o nome antigo nos relatórios. Situação semelhante pode ocorrer se o diretório da submissão de um usuário for removido e os arquivos `.grades` ficarem inconsistentes.

Quando um usuário é bloqueado, a sessão do usuário é terminada, a linha do usuário no arquivo `.pass` é comentada e uma linha é adiciona ao log.

4.5 Múltiplas instâncias no mesmo servidor

Não deve ser um problema ter mais de uma instância do sqtpm no mesmo servidor.

5 Relatórios e tabelas de acertos

O relatório de uma submissão pode ser visto pelo próprio aluno e pelos professores que tiverem acesso ao trabalho. O relatório de submissão inclui os arquivos enviados. Se o arquivo enviado for um programa então a sintaxe será destacada pelo google-code-prettifier. Programas C também serão endentados usando indent. Arquivos PDF ficam disponíveis para download.

Professores podem ver as submissões de todos os usuários dos trabalhos que fizerem parte tabuladas por arquivo de usuários. O sistema gera uma tabela para cada trabalho individualmente e uma para todos os trabalhos juntos. Um link em cada elemento da tabela permite a exibição do relatório da submissão. A ordem de exibição nas tabelas é alfabética. Juntamente com a tabela para um trabalho, o sqtpm vai mostrar histogramas de submissões se a opção de backup estiver ativada.

6 Moss

Os programas podem ser enviados pelo sqtpm para o Moss. O sqtpm exibe um link para comparação, que será refeita a cada 14 dias (tempo que o Moss mantém os resultados on-line) ou sempre que houver uma nova submissão.

No arquivo de configuração, `moss-id` deve conter um número de registro para um usuário do Moss e `moss-m` é o valor do parâmetro `-m`.

7 Integridade e execução dos casos-de-teste

Para evitar que alguém explore os diretórios dos trabalhos diretamente e veja os programas enviados por outras pessoas e os arquivos de configuração do sqtpm, o servidor http deve ser configurado para bloquear listagem de diretórios e bloquear acesso a qualquer arquivo que não seja `.cgi`, `.js` e `.css`. Essa restrição deve ser aplicada ao diretório-raiz e a todos os subdiretórios dele. Também é interessante usar alguma ferramenta para bloquear ataques ao servidor.

Executar código alienígena é uma brecha de segurança enorme. O sqtpm executa cada caso-de-teste através de um wrapper chamado `sqtpm-etc.sh` que recebe como parâmetros
o identificador do usuário, o nome do trabalho, a linguagem, o nome do programa, o tempo máximo de CPU, o tamanho da memória virtual e o tamanho da pilha.
O nome do programa é sempre `elf` para linguagens compiladas. Para Java, o jar se chama `elf.jar`. Para Octave o nome do programa é o nome do arquivo `.m` submetido pelo usuário. Para Python, é `elf.tar` se houver mais de um arquivo fonte ou o nome do arquivo `.py` caso contrério. Os interpretadores de Python, Java e Octave também são usados pelo wrapper.

O wrapper deve executar cada caso-de-teste aplicando as restrições de memória e tempo de CPU. Ele também deve fazer com que o conteúdo do diretório `extra-files` esteja no mesmo lugar onde o executável estiver. Para cada arquivo `z.in` esse wrapper deve escrever os seguintes arquivos no diretório `raiz/trabalho/_uid_tmp_/`:

- `z.run.out`: stdout da execução do caso-de-teste.
- `z.run.err`: stderr da execução do caso-de-teste.
- `z.run.st`: exit status da execução do caso-de-teste.

O wrapper mais simples e insano executa os casos-de-teste no próprio servidor. Esse é o `sqtpm-etc-localhost.sh`, que criei apenas para testar. Soluções melhores usam uma máquina virtual ou um container.

Criei o par `sqtpm-etc-vbox-shared.sh` e `vbox-etc-shared.sh` para usar com uma máquina virtual no VirtualBox com diretório compartilhado. Testei também um par de scripts para VirtualBox sem diretório compartilhado e foram bem mais lentos. Mais recentemente criei o par `sqtpm-etc-docker.sh` e `docker-etc.sh`.

Esses wrappers escrevem em um arquivo `sqtpm-etc.log` quando falham.

Sempre que houver uma tentativa de enviar uma requisição http com parâmetros incorretos, por exemplo para um trabalho para o qual o usuário não tem permissão, a sessão do usuário será terminada e ele será bloqueado. O arquivo `.pass` que contém o usuário será alterado e o sistema vai adicionar uma linha ao log.

No segundo semestre de 2013 houve um número muito grande de submissões por alguns alunos (na casa das centenas), o que é um tipo de ataque. Então implementei um limite do número de submissões. Como opção considerei também uma espera à medida que o número de submissões aumenta, mas gostei mais do limite.

8 Arquivos e permissões

Os arquivos e diretórios que compõem o sqtpm estão exemplificados a seguir. Diretórios de trabalhos devem ter modo 2770, como o do trabalho `trabalho1` exemplificado na figura.

```
-rwxr-x---   1 gpt  www-data  sqtpm.cgi
-rwxr-x---   1 gpt  www-data  sqtpm-pwd.cgi
-rw-r-----   1 gpt  www-data  sqtpm.pm
-rw-r-----   1 gpt  www-data  sqtpm.cfg
-rw-r-----   1 gpt  www-data  sqtpm.css
-rw-r-----   1 gpt  www-data  sqtpm.js
-rw-rw----   1 gpt  www-data  sqtpm.log

-rw-r-----   1 gpt  www-data  bula.html
-rw-r-----   1 gpt  www-data  home.html
-rw-r-----   1 gpt  www-data  envio.html
-rw-r-----   1 gpt  www-data  saida.html
-rw-r-----   1 gpt  www-data  icon.png
-rw-r-----   1 gpt  www-data  keep-calm-and-0-no-sqtpm-200.png

drwxr-s---   2 gpt  www-data  google-code-prettify
-rw-r-----   1 gpt  www-data  moss-sqtpm

lrwxrwxrwx   1 gpt  www-data  sqtpm-etc.sh -> sqtpm-etc-docker.sh
-rwxr-x---   1 gpt  www-data  sqtpm-etc-docker.sh

drwxrws---   2 gpt  www-data  trabalho1
drwxrws---   2 gpt  www-data  trabalho2
```

Ainda há os seguintes arquivos de apoio (diretório Utils):

- `fix-perms.sh`: Script para ajustar as permissões dos arquivos.

  Se for executado com o nome do trabalho como parâmetro, ajusta apenas as permissões dos arquivos daquele trabalho. Esse tipo de execução deveria ser feito depois que o diretório de um trabalho for criado e os arquivos forem colocados nele, mas antes da criação dos links dos grupos.

  Se for executado sem parâmetros ajusta as permissões de todos arquivos do sistema e de todos os trabalhos. Essa execução só precisa ser feita uma vez, depois de instalar o sistema e antes de acessá-lo pela primeira vez.
- `sqtpm-clean-sessions.sh`: Script que remove os arquivos de sessões expiradas.

- `sqtpm-etc-docker.sh` e `docker-etc.sh`: Par de programas para execução de casos-de-teste em um container Docker.