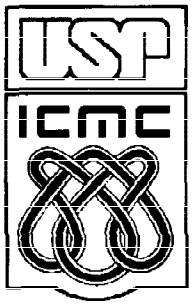
UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

SQTPM: SISTEMA PARA RECEPÇÃO E CORREÇÃO DE TRABALHOS DE PROGRAMAÇÃO

GUILHERME P. TELLES

Nº 299

RELATÓRIOS TÉCNICOS



São Carlos – SP
Jun./2007

sqtpm: sistema para recepção e correção de trabalhos de programação

Guilherme P. Telles

31 de maio de 2007

O sqtpm é um sistema implementado em Perl para recepção via web, compilação e teste de programas. Foi criado com a finalidade de ser usado como corretor automático de trabalhos em cursos de programação. Seu nome é um acrônimo de "sistema que trabalha por mim".

O sqtpm teve sua inspiração inicial no sistema MC102 [1] construído por 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 foi tirada do MC102. Também havia conhecimento acerca da existência de outros sistemas do gênero, como o Panda [2] e o Susy [3], mas achei melhor fazer do zero ao invés de usar ou adaptar outros sistemas. Desta forma eu teria o prazer de programar e poderia construir um sistema simples em uma linguagem que conheço bem e que eu pudesse modificar sem dificuldades.

As principais funcionalidades são a possibilidade de gerenciar múltiplos trabalhos simultaneamente, restrição do acesso através de senhas, gerenciamento através de arquivos e links, sem o uso de SGBD e a possibilidade de corrigir trabalhos por casos-de-teste ou por programa verificador. Além disso, espera-se criar listas de usuários muito facilmente a partir dos dados fornecidos por sistemas acadêmicos e visualizar relatórios que possam ser transportados facilmente para planilhas de notas.

Este documento descreve o sqtpm. Serviu para amadurecer as idéias antes de programar e tem servido agora para ajudar a manter, instalar e configurar o sistema. Toda a discussão supõe GNU/Linux e Apache. O sistema vem sendo desenvolvido desde 2004 e a versão descrita neste relatório é a v4. Esta versão não deve sofrer alterações substanciais, pois acredito que já tem a funcionalidade desejada.

O sqtpm é software livre que adere à GNU GPL.

Conteúdo

1 Organização do sistema

O sistema é composto por programas e por uma estrutura de diretórios onde os programas, arquivos de configuração, arquivos para correção e os trabalhos submetidos são armazenados. Os usuários são mantidos em arquivos texto e as permissões para submissões são definidas criando links simbólicos. Nas próximas seções cada um desses ítens é discutido em mais detalhes.

1.1 Programas

O sqtpm é composto pelos seguintes arquivos:

- `sqtpm.pl`: gera o formulário de submissão, compila e executa os programas submetidos para um conjunto de casos-de-teste e mostra os relatórios de submissões.
- `sqtpm-pass.pl`: gera o formulário para cadastramento e alteração de senhas e atualiza os arquivos de senhas.
- `sqtpm.pm`: biblioteca de funções.

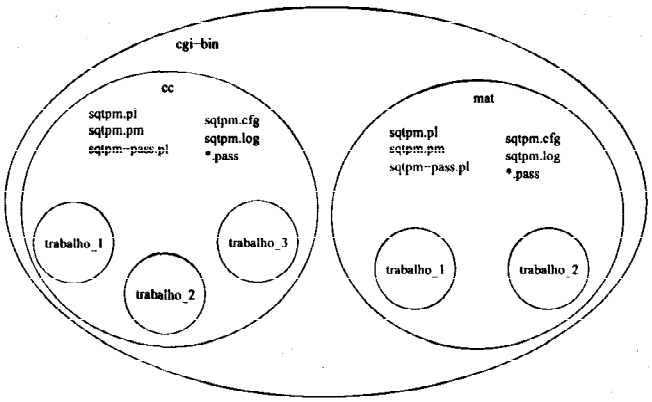


Figura 1: Servidor http com dois diretórios-raiz sqtpm.

1.2 Diretórios

O sistema funciona em um diretório em algum ponto da hierarquia `cgi-bin`, que chamaremos **diretório-raiz** deste ponto em diante, que contém

- os programas e os módulos Perl,
- o arquivo de configuração do sistema, `sqtpm.cfg`,
- O arquivo `sqtpm.log`, criado pelo sistema,
- os arquivos de usuários, com extensão `.pass`, e
- um subdiretório para cada trabalho que o sistema receberá.

Em um mesmo servidor http pode haver mais de um diretório-raiz, bastando que existam os arquivos necessários. Tais diretórios-raiz devem estar em algum ponto da hierarquia `cgi-bin`. Na Figura 1 mostramos um exemplo de estrutura para o servidor, com dois diretórios-raiz: um para o curso cc e outro para o curso mat, criados diretamente sob o diretório `cgi-bin`. Neste caso, o sistema seria acessado pelas urls:

```
http://servidor/cgi-bin/cc/sqtpm.pl
http://servidor/cgi-bin/mat/sqtpm.pl
```

Cada trabalho fica em um subdiretório do raiz. O sistema considera que todo subdiretório do raiz que não tenha nome da forma `.*` ou `_*` contém um trabalho e seus arquivos. Os nomes de diretórios de trabalhos não devem ter espaços.

No diretório de um trabalho ficam

- links para arquivos de usuários,
- os casos-de-teste do trabalho e/ou o programa verificador,
- os programas submetidos, os relatórios de submissão e os arquivos de comentários de correção, e
- o arquivo de configuração do trabalho `config`.

1.3 Usuários e senhas

No diretório-raiz deve haver um ou mais arquivos de usuários, que devem ter a extensão `.pass`. O formato do conteúdo de um arquivo de usuários é:

```
identificador:senha
identificador:senha
...
identificador:senha
```

Identificadores de usuários devem ter a forma [a-zA-Z0-9]+. O identificador será usado pelo usuário para enviar trabalhos, visualizar relatórios e cadastrar ou alterar senhas.

Um arquivo de usuários pode ser criado sem as senhas, o que permite que o próprio usuário cadastre sua senha usando o programa sqtpm-pass.pl sem intervenção. Nesse caso as linhas podem ter ou não o símbolo de dois pontos após o identificador. A idéia é que listas de números de matrícula de alunos sejam extraídas de sistemas acadêmicos e formatadas usando `cut`, `sed` ou outros programas. Assim espera-se que seja fácil produzir os arquivos de usuários.

Usuários que têm um asterisco precedendo seu identificador no arquivo de senhas são usuários privilegiados com permissão para ver relatórios com informações sobre todas as submissões e para submeter trabalhos sem autorização, em linguagem diferente das permitidas e fora do prazo.

Idealmente os usuários devem ter como identificador uma chave que seja única na união de todos os arquivos de senhas, mas um mesmo identificador pode estar em mais de um arquivo. O sistema gerencia essa redundância, atualizando todos os arquivos quando o usuário cadastrar ou trocar sua senha. O único problema previsto é no caso em que as senhas são diferentes, o que pode fazer com que o usuário não consiga logar-se.

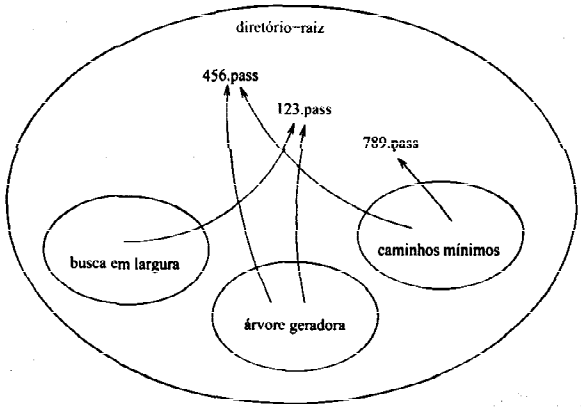


Figura 2: Links para arquivos de usuários.

1.4 Permissão para submissão

Um usuário pode submeter um certo trabalho x se seu identificador estiver em algum arquivo de usuários dentro do subdiretório x. Os arquivos de usuários que estão em diretórios de trabalhos devem ser links simbólicos para arquivos de usuários no diretório-raiz do sistema. Essa estrutura permite administrar as autorizações através da criação e remoção de links simbólicos.

Por exemplo, suponha que há três turmas de alunos 183, 456 e 789, que submeterão trabalhos. Suponha que os trabalhos para o semestre são os seguintes:

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 podem ser construídos da forma como aparecem na Figura 2.

2 Submissão, compilação e execução

Quando um usuário envia um trabalho, o sistema verifica o identificador, a senha, a linguagem aceita para o trabalho e as datas-limite de submissão, de acordo com o que estiver definido no arquivo de configuração do trabalho. Usuários privilegiados podem submeter qualquer trabalho, em qualquer linguagem e fora do prazo.

Suponhamos que um usuário com identificador `id` submete o trabalho de programação `x`. Depois das verificações de identificador, senha, linguagem e prazo, o programa submetido é gravado com o nome `id.ext` no diretório `x`, onde `ext` é `c`, `cpp`, `pas` ou `f`.

Se o usuário está fazendo uma re-submissão e se a opção backup estiver ativa, o arquivo pré-existente é movido para o diretório `x/backup` com nome `id-data.ext`, onde data é a data em que o programa substituído havia sido enviado. Se o arquivo movido tem mais de 5kb, então ele é compactado com gzip.

Depois de gravado, o programa é compilado e executado para cada um dos casos-de-teste que estiverem no respectivo diretório. Por questões de segurança, os casos-de-teste são executados em uma porção isolada do sistema de arquivos, chamada de jaula. O sistema executa os casos-de-teste em um diretório na jaula, que tem nome formado pela concatenação do identificador do usuário, da data e do número do processo que ele está executando. Desta forma espera-se que o isolamento entre as execuções de casos-de-teste seja suficiente para evitar conflitos não-intencionais de nomes de arquivo.

Depois de executados, os resultados dos casos-de-teste são verificados e o usuário recebe como resposta um percentual de acerto de 0 a 100%, proporcional ao número de casos-de-teste bem sucedidos. O resultado da execução é gravado em um arquivo chamado `id.rep`. Este arquivo é lido pelo `sqtpm.pl` para produzir relatórios.

Se não houver casos-de-teste, o sistema apenas recebe o programa submetido, compila e grava no diretório.

2.1 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 é composto por dois arquivos:

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

onde `z` é um nome de arquivo sem espaços.

O resultado da execução de cada caso-de-teste é comparado com o esperado usando `diff`. O resultado pode ser:

- **`bem-sucedido`**: o resultado produzido pelo programa está idêntico ao esperado.
- **`saída com formato incorreto`**: o resultado produzido pelo programa difere por espaços, fins-de-linha ou caixa de caracteres. O caso-de-teste é considerado mal-sucedido.
- **`saída diferente da esperada`**: o resultado produzido pelo programa difere do esperado. O caso-de-teste é considerado mal-sucedido.

Verificação por programa Se a verificação for feita por um programa, então cada caso-de-teste deve ser composto apenas de um arquivo de entrada `z.in`, onde `z` é um nome de arquivo sem espaços. A verificação através de programa é útil quando houver mais de uma resposta correta para uma entrada.

O resultado da execução pode ser o mesmo descrito para a verificação por comparação, em função do valor retornado pelo programa verificador. Veja também a seção 3.2.

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 e tem os seguintes campos:

```
data ip usuário trabalho acerto
```

2.3 Comentários de correção

Para cada usuário que enviou um trabalho pode-se criar um arquivo com o nome `id.cor` no subdiretório do trabalho. O conteúdo dos arquivos `.cor` são mostrados juntamente com o relatório de submissões do usuário. O arquivo pode incluir comandos html. A intenção é que comentários sobre a correção dos trabalhos sejam visualizadas através do sqtpm, evitando o envio de emails ou outras alternativas de comunicação.

3 Configuração

Há um arquivo para configuração do sistema e cada trabalho pode ter seu próprio arquivo de configurações. Diretrizes que aparecem tipicamente na configuração do sistema podem aparecer na configuração de trabalhos, por exemplo, para mudar a forma como um trabalho em particular é compilado. Diretrizes que aparecem tipicamente na configuração de trabalhos podem aparecer na configuração do sistema para que sejam aplicadas a todos os trabalhos.

3.1 Configuração do sistema

O arquivo `sqtpm.cfg` no diretório-raiz define valores-padrão de diretivas de execução dos trabalhos, o path para a jaula e as linhas de comando para a execução dos compiladores e demais programas externos usados pelo sqtpm. Contém linhas é da forma `diretriz=valor`. Um exemplo completo de arquivo aparece abaixo.

```
backup=off
penalty=100
maxmem=32768
maxcputime=1
languages=c c++ pascal fortan77

jail=/mnt/jail

pascal_compiler=/usr/local/bin/gpc (source) -o (exec)
c_compiler=/usr/bin/gcc -Wall -lm (source) -o (exec)
c++_compiler=/usr/bin/g++ -Wall -lm (source) -o (exec)
fortran77_compiler=/usr/bin/g77 -Wall -lm (source) -o (exec)

cp=/bin/cp
diff=/usr/bin/diff
find=/usr/bin/find
gzip=/usr/bin/gzip
bash=/bin/bash
sudo=/usr/bin/sudo
chroot=/usr/sbin/chroot
```

Nas diretrizes dos compiladores, as cadeias `(source)` e `(exec)` são substituídas pelo sqtpm pelo nome do arquivo fonte e do arquivo executável que deverá ser gerado, respectivamente.

3.2 Configuração de trabalhos

Cada diretório de trabalho pode conter, opcionalmente, um arquivo chamado `config` que contém linhas com a forma `diretriz=valor`. As diretrizes afetam apenas o trabalho para o qual foram definidas. As diretrizes válidas são listadas abaixo.

- **`languages=lista`** Define uma lista de linguagens de programação que serão aceitas para aquele trabalho. Os ítens da lista são separados por espaços. As linguagens aceitas atualmente são: `c`, `pascal`, `c++` e `fortran77`. Exemplos são:

  ```
  languages=c
  languages=pascal c c++
  ```

- **`deadline=aaaa/mm/dd hh:mm:ss`** Define a data e o horário limite para a submissão do trabalho. Se não for incluída, o trabalho não deixa de ser aceito.

- `wakeup=aaaa/mm/dd hh:mm:ss` Define a data e o horário para começar a aceitar submissões de um programa. Se não for incluída, o trabalho passa a ser aceito logo depois que os links para arquivos de usuários forem criados.

- `penalty=[0,100]` Define a multa percentual por dia de atraso depois da data limite (número inteiro). 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 acerto maior que zero.

- **`description=string`** Há duas alternativas. A primeira é uma url iniciada por **`http`** onde a descrição do trabalho pode ser encontrada, para a qual o browser é redirecionado. A outra é um nome de um arquivo html no diretório do trabalho, que será ecoado no browser. Exemplos são:

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

- **`backup={on,off}`** Define se o sistema manterá cópias de todos os programas submetidos.

- `copy=lista` Esta diretiva permite definir uma lista de arquivos que serão copiados para o mesmo diretório em que o executável estará durante a execução dos casos-de-teste. Esta opção é útil quando o trabalho exige a leitura de algum arquivo, como por exemplo, um compactador. Os ítens da lista são separados por espaços.

- `maxcputime=n` Define o tempo de CPU máximo para executar cada caso-de-teste, em segundos (número inteiro).

- **`maxmem=n`** Define a memória máxima para executar cada caso-de-teste, em kilobytes (número inteiro).

- **`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, um que é a entrada de um caso-de-teste e outro com a saída produzida pelo programa submetido para a entrada (ambos com path absoluto), nesta ordem. O programa verificador deve retornar 0 se a saída é válida para a entrada, 1 se é inválida, 2 se houver erro de formato, ou > 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 **x** possui como verificador um executável chamado **`verif`**, que está no próprio diretório do trabalho, a diretriz pode ser simplesmente

  `verifier=#verif`

  Se o verificador está em um outro diretório, pode ser

  `verifier=/tmp/verif`

  Ainda, se o verificador é um programa Perl em um outro diretório,

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

- **`showcases=lista`** Define uma lista de casos-de-teste (nomes de arquivo sem a extensão **`.in`**) para os quais o sistema mostrará a entrada, a saída esperada e a saída produzida pelo programa testado.

- `receiveonly={on,off}` Se esta diretiva for definida com valor `on`, o sistema simplesmente recebe um arquivo e renomeia com o id do usuário, preservando a extensão. Nenhuma outra ação é realizada. O backup não é feito nessa situação. A linguagem selecionada pelo usuário é irrelevante. O arquivo de submissão registra o nome do arquivo recebido. O campo acerto no log de submissões é substituído pela extensão do arquivo recebido.

4 Relatórios

Quando a opção de relatório é selecionada no formulário de submissão, o sistema mostra relatórios de notas e comentários de correção para os usuários. Usuários privilegiados verão as notas de todos os usuários tabuladas por arquivo de usuários.

Adicionalmente o sqtpm produz um placar de acertos nos moldes do sistema Boca [4] usado na maratona SBC/ACM. O início da contagem deve ficar no arquivo `sqtpm-score-start` com o formato `aaaa/mm/dd hh:mm:ss`. Se não houver tal arquivo o placar não é exibido. A execução deve ser feita passando um parâmetro: `sqtpm.pl?score=1`. Não pede senha.

5 Segurança

A solução escolhida para amenizar os problemas de segurança causados pela execução de código alienígena foi a criação de uma jaula (do inglês jail) [5].

Espera-se que tal medida resolva a maior parte dos problemas potenciais, tais como a alteração de arquivos de configuração do sistema, dos programas do próprio sqtpm e dos trabalhos depositados pelos usuários. Se a jaula estiver em outro volume, isso também vai evitar que algum programa alienígena encha o disco. Combinadas à jaula, há restrições para limitar o uso de memória, tempo de CPU e a prioridade dos programas alienígenas.

Alternativamente o sqtpm pode funcionar sem uma jaula. Para isso basta deixar de definir a variável `jail` no arquivo `sqtpm.cfg`.

Jaula e sqtpm A jaula é um diretório que contém uma fração das bibliotecas, executáveis e arquivos de configuração do GNU/Linux. A idéia é que um usuário não-privilegiado fique confinado à jaula para a execução dos programas enviados ao sqtpm.

A localização da jaula é definida pela variável `jail` no arquivo `sqtpm.cfg`. Durante sua operação, depois de receber um programa e compilá-lo, o sqtpm

cria um subdiretório `dir` em `jail/home/sqtpm` e copia o executável e os casos-de-teste para este subdiretório. Depois executa

```
sudo chroot jail /bin/su - sqtpm id dir maxcputime maxmem
```

O `chroot` define a jaula como nova raiz do sistema de arquivos para um processo. Como apenas o super-usuário pode executar o `chroot`, essa operação é feita através do `sudo`, para evitar a necessidade de um wrapper. O `su - sqtpm` vai executar um shell como usuário `sqtpm`, que está configurado em `jail/etc` da seguinte forma:

```
sqtpm:x:501:501:sqtpm:/home/sqtpm:/bin/execCT
```

Ao "logar-se", o usuário `sqtpm` executa apenas o script `execCT`. Esse script executa o programa que está no diretório `dir` com cada um dos casos-de-teste, respeitando a restrição de memória `maxmem` e de tempo de CPU `maxcputime`, e termina. Os resultados da execução são gravados em arquivos no mesmo diretório.

Assim, todos os casos-de-teste serão executados como `sqtpm` e restritos à *hierarquia enraizada em* `jail`. *O usuário* `sqtpm` *não precisa existir fora da* jaula.

Criação da jaula Para criar a jaula, o script `make_chroot_jail.sh` [6] foi executado com o comando

```
bash make_chroot_jail.sh sqtpm /mnt/jail/bin/bash /mnt/jail
```

Em seguida o script `execCT` foi acrescentado a `jail/bin` e o arquivo `jail/etc/passwd` foi editado, substituindo `bash` por `execCT`. As seguintes linhas foram acrescentadas a `jail/etc/security/limits.conf` para limitar os recursos:

```
*               -       stack           32768
*               -       priority        20
```

A linha de configuração do `sudo` em `/etc/sudoers` é

```
apache ALL=NOPASSWD:/usr/sbin/chroot
```

6 Resumo da configuração

Todos os arquivos e diretórios mencionados abaixo devem ser do grupo que executa o servidor http.

Para criar um trabalho é necessário:

1. Criar um diretório sob o diretório-raiz, com modo `drwxrws---`.
2. Criar os arquivos dos casos-de-teste com modo `-rw-r-----`.
3. Se necessário, criar o arquivo `config`, com modo `-rw-r-----`.
4. Criar links simbólicos (`ln -s`) para os arquivos de usuários desejados.

Para instalar o sqtpm é necessário:

1. Opcionalmente, criar uma jaula no sistema como descrito na Seção 5.
2. Criar um diretório sob `cgi-bin` com modo `drwxrws---`.
3. Copiar os arquivos `sqtpm.pl` e `sqtpm-pass.pl` com modo `-rwxr-x---`. Copiar `sqtpm.pm` e `sqtpm.cfg` com modo `-rw-r-----`.
4. Editar o arquivo `sqtpm.cfg` adequadamente.
5. Criar os arquivos de usuários com modo `-rw-rw----`.

Referências

[1] Z. Dias. MC102. Comunicação pessoal, 2004.

[2] IME-USP. Panda. http://panda.ime.usp.br, 2007.

[3] T. Kowaltowski. Susy. http://www.ic.unicamp.br/~susy/, 2007.

[4] C.P. de Campos. BOCA. http://incubadora.fapesp.br/sites/boca/, 2007.

[5] S. Friedl. Go directly to jail. *Linux Magazine*, December 15th, 2002.

[6] W. Fuschlberger. make_chroot_jail. http://www.fuschlberger.net, 2007.