Insper

POSIX Lab

Sistemas Hardware-Software

Entrega: **12/06**

A segunda parte do curso apresenta (e exercita) conceitos importantes de sistemas operacionais, focando em sistemas compatíveis com o padrão POSIX. Neste lab iremos melhorar um framework de testes usando as chamadas de sistemas vistas em aula para deixá-lo mais robusto a erros e mais rápido.

Parte 0 - testes unitários em *C*

Nesta seção vamos revisar a implementação do framework de testes que já usamos na atividade 3. O arquivo *mintest/macros.h* contém *macros* usada dentro das funções de teste para dar prints e checar se uma condição que deveria ser verdade

O arquivo *mintest/runner.h* contém uma função `main` que chama cada um dos testes e mostra seus resultados. Um arquivo de testes segue o modelo abaixo.

```c
#include "mintest/macros.h"

int test1() {
    test_printf("Hello! %d %f\n", 3, 3.14);
    return 0;
}

int test2() {
    test_assert(1 == 0, "This always fails!");
    return 0;
}

int test3() {
    test_printf("<-- Name of the function before the printf!\n");
    test_assert(1 == 1, "This always succeeds");
    return 0;
}

test_list = { TEST(test1), TEST(test2), TEST(test3) };

#include "mintest/runner.h"
```

Basta compilar o arquivo acima, rodar e você deve obter a seguinte saída.

```
Running 3 tests:
====================

test1: Hello! 3 3.140000
test1: [PASS]
test2: [FAIL] This always fails! in example.c:9
test3: <-- Name of the function before the printf!
test3: [PASS]


====================
2/3 tests passed
```

Esta implementação, apesar de funcional, tem diversos problemas:

1. falhas em um teste podem afetam a execução de outros (um teste pode, acidentalmente, apontar para dados de outros testes)
2. um teste que dê *Segmentation Fault* impede a execução dos testes seguintes
3. um teste que entre em loop infinito impede a execução dos testes seguintes

Seria interessante, também, que nosso programa mostrasse uma mensagem de confirmação quando o usuário pressiona `Ctrl+C` para terminar o programa e que fosse possível rodar os testes em paralelo para que eles terminassem mais rápido.

Parte 1 - requisitos de implementação

Este laboratório exercitará os seguintes conceitos vistos em sala de aula:

1. Criação e gerenciamento de processos e sinais;
2. Tratamento de arquivos e redirecionamento de saída;
3. Compilação de programas e *C* avançado;

Vamos melhorar o código disponibilizado implementando as seguintes funcionalidades.

1. Cada teste deverá rodar em um processo separado.

Isto melhorará significativamente tanto a velocidade da execução dos testes quanto seu isolamento: a falha de um teste não implicará em nada na execução de outro. Um teste comunica se houve passou ou não via seu valor de retorno. Se o teste retornar `0` então ele passou. Qualquer valor diferente disto implica que houve um erro.

Isto também possibilita que os testes sejam executados em paralelo, diminuindo consideravelmente o tempo de execução.

2. A saída padrão de cada teste deverá ser redirecionada de modo a não bagunçar o terminal.

Com diversos testes rodando ao mesmo tempo é possível que a saída padrão mostre seus resultados todos misturados. Cada processo deverá ter sua saída redirecionada para um arquivo temporário (ou armazenado em memória) e a saída de cada teste deve ser mostrada apenas após o seu fim.

Da mesma maneira, a saída de erros dos processos filhos também deverá ser redirecionada.

3. Deverá ser implementado um tempo limite para os testes rodarem. Se um teste passar deste limite deverá ser cancelado e uma mensagem de erro deve ser apresentada.

Este tempo limite pode ser implementado tanto no pai como nos filhos. Pense na forma mais inteligente e sucinta de fazê-lo.

4. Seu programa deverá mostrar uma mensagem de confirmação de saída

Implementação

Você precisará modificar os arquivos dentro da pasta *mintest* para implementar os itens acima. Não deverão ser necessárias mudanças no *teste_exemplo.c*.

Importante: em diversos momentos será necessário decidir se uma funcionalidade é implementada no processo original (que cria todos os processos filhos) ou nos filhos (que efetivamente rodam as funções de teste).

Parte 2 - Entrega

Você deverá entregar:

1. seus arquivos modificados *macros.h* e *runner.h*
2. um ou mais arquivos contendo testes escritos por você mesmo. Faça pelo menos um teste de cada tipo abaixo
    1. cause erro e termine com falha de segmentação
    2. cause erro e termine com divisão por zero
    3. fique em loop infinito

4. faça muito trabalho, mas eventualmente acabe (sem usar `sleep`)
5. tenha *assert*s que falham e passem no mesmo teste
6. tenha testes que façam muitos prints
7. tenha testes que sejam rápidos e testes que sejam lentos (pode usar `sleep` para simular).

3. um arquivo *README.txt* explicando sua implementação.

Sua implementação deverá ser compatível com a original, mas oferecer as extensões e melhorias descritas neste enunciado.

Conceitos

A rubrica adotada neste projeto é incremental. Para obter um conceito é necessário realizar todas as tarefas de todos os conceitos anteriores.

Conceito I

Não entregou ou entregou o exemplo disponibilizado sem modificações

Conceito D

Implementou algum item, mas não todos, do conceito *C*. Os arquivos de testes entregues não contém testes para todos os casos descritos.

Conceito C

- Cada teste roda em um processo separado;
- Processos filhos rodam os testes **em paralelo**.
- Processo pai obtem o resultado do processo filho usando `wait`.

Conceito B

- Saída dos testes é redirecionada e é impressa por inteiro ao final do teste.
- É criado um tempo de limite de execução para cada teste usando a chamada `alarm`.
- O processo pai avisa quando o processo filho for terminado com erro usando uma mensagem explicativa com o nome do erro.

Conceitos B+, A, A+

A partir do conceito **B** cada uma das seguintes características adiciona um ponto na nota:

- Processo original responde a Ctrl+C com uma mensagem de confirmação de saída
- A biblioteca possui uma API para que cada teste possa modificar seu tempo limite. Ela só pode ser chamada uma vez por função e deverá retornar um erro caso seja chamada mais de uma vez.
- O relatório de erros imprime as funções na ordem de declaração, não na ordem que foram finalizadas, e mostra o tempo que cada teste demorou para rodar.

A partir do conceito **C** cada uma das seguintes funcionalidades adiciona 0.25 na nota final:

- ao passar "-html" como argumento seu programa retorna um relatório em HTML **bem formatado**
- os status dos testes são impressos em cores.
- (adicional ao de cima) se a saída de errros do `main` for redirecionada para um arquivo imprima tudo sem cores.