Departamento de Ciência da Computação
Métodos de Programação 2/2015

Projeto de Disciplina

Datas de entrega:

Definição dos grupos: até 16/11/15 23:55

Fase 1 : 23/11/15 23:55

Final : 8/12/15 ate as 23:55

Rede de Distribuição

Descrição :

Faça um programa em C ou C++

O programa deve simular uma rede de distribuição de recursos.

A simulação é composta de:

a) Geradores: são unidades que apenas geram o recurso. Geram uma quantidade x de recuso por segundo.

b) Cidades: são unidades que apenas recebem o recurso. Elas consomem uma quantidade y de recurso por segundo. Mesmo em caso de falhas na interconexão a cidade deve ser servida com pelo menos 30% do valor de y. Se não for, isto deve ser registrado.

c) Interconexão: transporta o recurso dos geradores para as cidades. Tem uma capacidade de transporte de z recursos por segundo. A interconexão pode falhar com uma chance de x% em cada segundo. Quando isto acontece, ela demora y segundos para ser consertada. A interconexão tem um custo pelo conserto.

d) Adaptadores : eles podem juntar mais de uma linha de interconexão e deles podem sair uma ou mais linhas de interconexão. Ex. se um adaptador junta duas interconexões que transportam x recursos por segundo e y recursos por segundos. Se elas se juntam, a saída pode ser no máximo x+y recursos por segundos, não podendo ser maior que a capacidade de transmissão da interconexão de saída.

Eles podem também dividir uma linha de transmissão em n outras. Uma entrada de interconexão com capacidade de x recursos por segundo. Ela pode dividir entre duas conexões. A divisão deve ser proporcional à capacidade de transmissão das interconexões de saída. Ex. entra um interconexão com 9 rec/seg (recursos por segundo). A saída ‘a’ tem capacidade de 20 rec/seg e a saida ‘b’ tem capacidade de 10 rec/seg. A divisão será 6 rec/seg para ‘a’ e 3 rec/seg para ‘b’.

Requisitos:

O programa deve ter uma interface que permita carregar os arquivos com as informações. Deve mostrar as informações em forma gráfica. Deve ser possível rodar a simulação por um tempo especificado. Deve ser possível salvar o resultado da simulação. Não é necessário ser possível editar.

O arquivo a ser lido deve ter o seguinte formato:

Todos os números são inteiros com exceção de chance_falha em interconexão que é um float.

O arquivo é dividido em um elemento por linha. Os dados são separados por um ou mais espaços:

Linha com elemento Cidade:

C nome_cidade pos_x pos_y recurso_necessario

C indica que é uma cidade, pos_x e pos_y são posições em quilômetros que também corresponde as posições onde a cidade será mostrada na interface.

recurso_necessario diz quanto de recurso a cidade precisa por segundo

Ex:

C Brasília 15 47 143

Linha com elemento Gerador:

G nome_gerador pos_x pos_y recurso_produzido custo_gerador

G indica que é um gerador pos_x e pos_y são posições em quilômetros que também corresponde as posições onde o gerador será mostrado na interface.

recurso_ produzido diz quanto de recurso o gerador produz por segundo

custo_gerador diz o custo de geração do recurso por segundo

Ex:

G Usina 10 42 100 1000

Linha com elemento Interconexão:

I nome_interconexao pos_inic_x pos_inic_y pos_final_x pos_final_y capacidade_max chance_falha tempo_conserto custo_do_conserto

I indica que é uma interconexão com posições iniciais e finais em quilômetros que também corresponde onde a interconexão será mostrada na interface.

Ex:

I inter_1 2 2 5 5 100 0.1 15 20

A interconexão inter_1 liga o ponto (2,2) ao ponto (5,5) com capacidade máxima de 100 recursos por segundo. Pode falhar com chance de 10% (0.1) em cada segundo e demora 15 segundos para ser consertada em caso de falha e custa 20 para ser consertada.

Os números aleatórios devem ser gerados baseado no seguinte código:

```
float num;

float chace_falha = 0.01;

srand(1); //so é executado uma vez na simulacao

num = ((float)rand())/RAND_MAX;
```

if ((chace_falha > 0) && (chace_falha >=num)) cout << "\n FALHA!!";

A simulação será da seguinte forma:

Em cada segundo, se percorre os elementos de interconexão na ordem em que foram lidos no arquivo de forma que o primeiro número aleatório gerado será usado para calcular a chance de ocorrência de falha para a primeira interconexão lida no arquivo. O segundo número para segunda interconexão e assim por diante até o final da simulação.

Isto é importante pois todas as simulações devem gerar os mesmos números aleatórios e gerar os mesmos resultados.

Linha com elemento Adaptadores:

A nome_adaptador pos_x pos_y

A indica que é um adaptador na pos_x e pos_y são posições em quilômetros que também corresponde as posições onde o adaptador será mostrado na interface.

Ex:

A adapta_1 2 3

O adaptador fica na posição (2,3). Todas as interconexões que terminam neste ponto são as entradas para o adaptador. Todas as interconexões que saem deste ponto são saída do adaptador.

Após ler todas as linhas o programa deve avisar de todos os problemas de consistência dos elementos. Ex. cidades e geradores desconectadas, interconexões que não distribuem recursos. Adaptadores que tem apenas entradas ou apenas saídas, etc.

A interface deve permitir gerar um arquivo de saída com o seguinte formato:

Tempo total da simulação: 20 segundos

Custo total na simulação: 160000

Total de geradores: 5

Energia total gerada: 15000

Total de cidades : 10

Energia total gasta pelas cidades: 10000

Tamanho total das interconexões: 100 km

Número de falhas nas interconexões: 3

Número de cidades que ficaram com menos recurso que o necessário: 3

Tempo que ficaram sem recurso: 60 segundos

Número de cidades que ficaram com menos de 30% dos recursos : 1

Tempo que ficaram com menos de 30% de recurso: 20 segundos

O tempo sem recurso será a soma dos tempos que as cidades ficaram sem recurso.

A interface gráfica deve mostrar:

Cidades: posição, se está atendida com sua capacidade máxima, capacidade mínima, ou se está abaixo disto

Interconexão: posição, se está funcionando ou não. Deve dar uma idéia de quanto da capacidade está sendo carregada.

Geradores: posição

Deve mostrar : recurso total que vai para as cidades

Qual o tamanho total das interconexões.

Gera estatísticas de quantas falhas aconteceram, quantas cidades foram afetadas e de que forma foram afetadas.

1) Devem ser aplicado neste trabalho todos os conceitos vistos nos trabalhos anteriores.

a) Modularização (makefile, .h e .c),

b) Testes utilizando o Gtest. Como estes testes mostram que o programa segue a especificação.

c) Assertivas de entrada e de saída. Estas assertivas devem ser colocadas no código através do comando **assert** ou **ifs**. Comentários no código também devem especificar quais são estas assertivas

d) Para cada uma das funções adicionar comentários indicando qual são as interfaces explícita, implícita com a devida descrição, quais são os requisitos e as hipóteses de cada função. Além disto, as funções devem ter finalização adequada liberando memória, fechando arquivos, etc.

e) Faça a modelagem física das estruturas de dados. Use cabeças de estrutura de dados.

f) Assertivas de entrada e de saída como parte da especificação (comentários antes das funções).

g) Assertivas como comentários de argumentação.

h) Assertivas estruturais: elas definem a validade de uma coletânea de dados, ou estruturas de dados, e dos estados associados a estes dados

i) Coloque nos comentários antes das funções quais são as assertivas do **contrato na especificação.** Diga o que deve ser esperado da função cliente em relação à entrada e o que deve ser garantido pela função servidora na saída.

j) Utilize iteradores, programação genérica e ponteiros para função.

2) Instrumente o código usando o gcov. Usando o gcov. (http://gcc.gnu.org/onlinedocs/gcc/Gcov.html). O makefile deve ser modificado de forma incluir as flags -ftest-coverage -fprofile-arcs. Depois de rodar o executável rode gcov nomearquivo e deverá ser gerado um arquivo .gcov com anotação.

O gcov é utilizado para saber qual percentual do código é coberto pelos testes. Neste caso os teste devem cobrir pelo menos 80% do código por módulo.

Faça a análise estática do programa utilizando o *cppcheck*, corrigindo os erros apontados pela ferramenta

3) Sugestão, utilizar o Valgrind (http://valgrind.org/) . O Valgrind e muito útil para encontrar problemas relacionados a memória. O Valgrind **não** será utilizado na correção dos trabalhos mas pode tornar o desenvolvimento de depuração bem mais rápida.

4) Interface gráfica utilizará a **biblioteca ncurses.**
Ela pode ser instalada no terminal shell do Linux usando o comando

> sudo apt-get install libncurses5-dev

Uma vez instalada, a biblioteca ncurses ela pode ser compilada utilizando o GCC e a diretiva –lncurses.
Dependendo da instalação ncurses pode ser incluída por <ncurses/ncurses.h> ou <ncurses.h>.

Existem vários tutoriais disponíveis como :
http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html

Módulos

O programa deve ser dividido em módulos. Os módulos implementados nos laboratórios podem ser utilizados aqui.

Observações:

Utilize os princípios de modularidade na criação dos módulos, definidos seus módulos de definição, implementação e organizando suas compilações e ligações (**links**) de forma adequada via um único **makefile**.

Na criação dos módulos, lembre-se de usar as diretivas de controle para evitar inclusões múltiplas e identificando também o módulo servidor.

Controle de qualidade das funcionalidades

Depois de pronto, o aluno deve criar um Makefile. Deve-se criar um *módulo controlador de teste* (disciplinado) usando o Gtest para testar se as principais funcionalidades e restrições dos módulos, atendendo aos critérios da **Interface para linha de Comando**. O teste disciplinado deve seguir os seguintes passos:

1) Antes de testar: produzir um roteiro de teste

    a. definir o contexto (cenário) necessário e selecionar a massa de teste contendo a seqüência de ações e valores de teste com os respectivos resultados esperados e que foi criada segundo um critério de teste.

2) Antes de iniciar o teste: estabelecer o cenário do teste

3) Criar um módulo controlador de teste, usando a ferramenta Gtest para testar as principais funcionalidades de cada módulo.

4) Ao testar: produzir um laudo em que todas as discrepâncias encontradas são registradas. Esse laudo pode ser uma saída da execução do Gtest. Somente termine o teste antes de completar o roteiro, caso observe que não vale mais a pena continuar executando o roteiro, uma vez que o contexto para o resto está danificado.

5) Após a correção: repetir o teste a partir de **2** até o roteiro passar sem encontrar falhas.

Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro terá de ser documento usando DoxyGen. Comentários que vão ficar na documentação devem ser do estilo Javadoc. A documentação deverá ser feita conforme visto em aula

Os padrões de codificação e documentação devem estar conforme capítulo 4 e apêndice 4 e 5 do material bibliográfico da disciplina.

Parte 2. Escrita

· Um arquivo **LEIAME.TXT** contendo a explicação de como utilizar o(s) programa(s).
· Tantos arquivos **RELATORIO-nome.TXT** quantos forem os membros do grupo. O tema **nome** deve identificar o membro do grupo ao qual se refere o relatório. Estes arquivos devem conter uma tabela de registro de trabalho organizada como a seguir:

Data | Horas Trabalhadas | Tipo Tarefa | Descrição da Tarefa Realizada

Na descrição da tarefa redija uma explicação breve sobre o que o componente do grupo fez. Esta descrição deve estar de acordo com o Tipo Tarefa. Cada Tipo Tarefa identifica uma natureza de atividade que deverá ser discriminada explicitamente, mesmo que, durante uma mesma sessão de trabalho tenham sido realizadas diversas tarefas. Os tipos de tarefa são:

- estudar aulas e laboratórios relacionados
- especificar os módulos
- especificar as funções
- revisar especificações
- projetar
- revisar projetos
- codificar módulo
- Rodar os CPPCheck e retirar warnings
- revisar código do módulo
- redigir casos de teste
- revisar casos de teste
- realizar os testes
- instrumentar via gcov
- documentar com Doxygen

Observações:

- Dica: Preencha esta tabela de atividades ao longo do processo. NÃO DEIXE PARA ÚLTIMA HORA, POIS VOCÊ NÃO SE LEMBRARÁ DO QUE FEZ TAL DIA, TAL HORA. Com relatórios similares a esse você aprende a planejar o seu trabalho.

O trabalho pode ser feito em grupo de no máximo 3.

O programa deve ser feito para Linux, utilizado o GCC e GDB na mesma versão do Linf

Deve ser enviado um único arquivo compactado (.zip) contendo todos os arquivos necessários (.c .h makefile, estrutura de diretório, informação de como utilizar, etc).

Apenas um integrante do grupo deve enviar o trabalho. O nome do trabalho deve ser algo como:
MP_Jose_12345_Joao_54321_Maria_12345.zip
onde são os primeiros nomes e matriculas dos integrantes do grupo.

Cópias de trabalho terão nota zero.

Excelente trabalho!

Deve ser enviada pelo ead.unb.br até :

Fase 1 : 23/11/15 23:55

Consiste de um programa capaz de ler um arquivo de entrada e de gerar o relatório parcial baseado na entrada.

Final: 08/12/15 ate as 23:55

Consiste da Fase 1 e de todo o resto do trabalho