Insper

11 - Alocação Dinâmica de Memória

Sistemas Hardware-Software - 2019/1

Igor Montagner

Parte 1 - Aquecimento

Exercício 1: Abra o arquivo *ex1.c* em um editor, compile usando as flags da disciplina (`-Og -Wall -std=c99`) e responda:

a) Analisando seu código-fonte, o que este programa faz?

b) Na execução deste programa, existe alguma possibilidade da alocação dinâmica com `malloc` falhar? Caso sim, indique as situações onde isto poderia acontecer.

c) O que seria necessário alterar no código para checar se `malloc` falha ou não? Efetue estas alterações. Para testar, edite o código para que sejam solicitados espaços de memória de tamanho muito grande (acima da quantidade de memória disponível).

Exercício 2: Abra o arquivo *ex2.c* em um editor, compile usando as flags da disciplina (`-Og -Wall -std=c99`) e responda:

a) Execute o programa recém compilado. Ocorreu algum problema nesta etapa?

b) Faça uma análise visual do código-fonte. Você consegue identificar algum problema?

Parte 2 - Valgrind

Para poder identificar mais facilmente problemas relativos a memória, iremos utilizar uma ferramenta chamada Valgrind.

O Valgrind é um detector de má gestão de memória, ele mostra vazamentos de memória, erros de desalocação que nem sempre podem ser facilmente identificados pelo programador apenas com a análise visual do código.

Instale o Valgrind com os seguintes comandos:

```
sudo apt-get update
```

```
sudo apt-get install valgrind
```

Para que os problemas encontrados pelo Valgrind sejam mais facilmente identificados, iremos passar a compilar utilizando a flag `-g`.

Exercício 3: Abra o arquivo *ex3.c* em um editor, compile usando as flags da disciplina (`-Og -g -Wall -std=c99`) e responda:

a) Faça uma análise visual do código-fonte. Você consegue encontrar algum problema em relação ao uso da memória?

b) Rode o Valgrind com `valgrind --leak-check=yes ./ex3`. Quais foram os problemas encontrados e em quais linhas do código?

c) Efetue a correção dos problemas e rode novamente o Valgrind para conferir.

Exercício 4: Abra o arquivo *ex4.c* em um editor, compile usando as flags da disciplina (`-Og -g -Wall -std=c99`) e responda:

a) Faça uma análise visual do código-fonte. Você consegue encontrar algum problema em relação ao uso da memória?

b) Rode o Valgrind com `valgrind --leak-check=yes ./ex4`. Quais foram os problemas encontrados e em quais linhas do código?

c) Efetue a correção dos problemas e rode novamente o Valgrind para conferir.

Exercício 5: Abra o arquivo *ex5.c* em um editor, compile usando as flags da disciplina (`-Og -g -Wall -std=c99`) e responda:

a) Faça uma análise visual do código-fonte. Você consegue encontrar algum problema em relação ao uso da memória?

b) Rode o Valgrind com `valgrind --leak-check=yes ./ex5`. Quais foram os problemas encontrados e em quais linhas do código?

c) Efetue a correção dos problemas e rode novamente o Valgrind para conferir.

Exercício 6: Abra o arquivo *ex6.c* em um editor, compile usando as flags da disciplina (`-Og -g -Wall -std=c99`) e responda:

Ao analisar o código-fonte deste exercício, podemos perceber que foi definida uma função `sorteio()`. Este função utiliza a geração de números aleatórios para simular milhões de sorteios de um dado de seis faces. Observe que os sorteios são armazenados em um vetor alocado dinamicamente. Os valores sorteados não são utilizados para nenhuma aplicação prática, o que foi feito de propósito apenas para deixar o código mais limpo! Na função `main()`, são realizadas repetidas chamadas da função `sorteio()` até que o usuário digite `0` para sair.

a) Você consegue encontrar algum problema ao analisar visualmente o código da função `sorteio()`?

b) Neste item, iremos rodar o programa e acompanhar a execução pelo monitor de processos `htop`. Caso não tenha, instale com `sudo apt-get install htop`. Preferencialmente, abra duas janelas do terminal e organize-as lado a lado (utilize a tecla WINDOWS + SETA para DIREITA ou ESQUERDA). Execute o htop na janela da esquerda e o programa deste exercício na janela da direita.

No htop, filtre pelo nome do program do exercício (func) para que fique mais fácil acompanhar.

Agora, no programa do exercício, digite valores diferentes de zero várias vezes (para que a função `sorteio()` seja chamada várias vezes) e acompanhe o que acontece com o uso de memória (colunas VIRT, RES e MEM%).

Você consegue explicar o motivo deste comportamento?

Como você faria para corrigir o problema?

Efetue a análise do programa com o Valgrind.

Substitua a chamada da função malloc por `calloc`. Rode novamente no Valgrind.

Parte 3 - Programar do zero!

Nos exercícios da Parte 2, você recebeu códigos prontos e precisou fazer correções relativas ao acesso em memória. Nesta parte, você terá que construir todo o código que resolve um problema.

Exercício 7: Crie um programa para ler o preço de fechamento do dia de uma determinada ação para n dias. Na entrada, o primeiro valor será um inteiro n referente a quantidade de dias, seguido por n valores reais referentes ao preço de fechamento da ação em cada um dos n dias. Você precisará alocar um vetor de tamanho n para armazenar as leituras. Após, calcule algumas estatísticas (média, máximo, mínimo, desvio padrão, etc.) e exiba estas informações.

Compile, efetue alguns testes e confira pelo Valgrind se existe algum bug de memória.

Exercício 8: Refaça o programa anterior. Agora, considere que a quantidade de dias não será mais informada e o programa irá parar quando ler qualquer valor negativo. Inicialmente, efutue a alocação de um vetor de cinco posições. Cada vez que o vetor estiver cheio e for necessária uma nova posição, aloque mais cinco utilizando a função `realloc`. Ao final, exiba os valores na ordem inversa a da entrada.

Parte 4 - Problemas com strings.

Nesta parte, iremos praticar alocação dinâmica e strings.

Exercício 9: Abra o arquivo *ex8strcopy.c* e implemente a função *mystrcpy*. Esta função recebe uma string, e devolve uma cópia desta em memória, alocando apenas o espaço realmente necessário.

a) Efetue alguns testes no terminal e confira se está ok.

b) Confira com o Valgrind se a sua implementação produz algum erro em relação aos acessos de memória.

Exercício 10: Abra o arquivo *ex9strcat.c* e implemente a função *mystrcat*. Esta função recebe duas string, e devolve uma terceira que é a concatenação das duas primeiras, alocando apenas o espaço realmente necessário.

a) Efetue alguns testes no terminal e confira se está ok.

b) Confira com o Valgrind se a sua implementação produz algum erro em relação aos acessos de memória.

Exercício 11: Você percebeu que, no código base dos dois exercícios anteriores a memória alocada dinamicamente não foi devolvida ao sistema? Nestes casos, explique qual parte do código deve ser responsável pela liberação e por que?

Parte 5 - Alocação de array n-dimensional

Podemos também utilizar alocação dinâmica para requisitar espaços em memória para armazenar arrays 2D, 3D, 4D... etc.

Exercício 12: Neste exercício, será necessário implementar algumas funções que trabalham com ponteiros para ponteiros.

a) Pesquise sobre alocação de matrizes em C.

b) Abra o arquivo *ex11mat.c* e implemente as funções para alocar, exibir e somar uma matriz 2D em C. A opção escolhida para alocar matrizes neste caso foi o uso de ponteiros para ponteiros.

c) Caso tenha utilizado a sintaxe de matrizes na implementação do exercício anterior, crie um novo utilizando apenas a sintaxe de ponteiros.

d) Confira com o Valgrind se a sua implementação produz algum erro em relação aos acessos de memória.