Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II
2º Semestre de 2014

Trabalho Prático 1
Alocação de Memória

Valor: 15 pontos
Prazo: 09/10/2014

O objetivo do trabalho é implementar um mecanismo de alocação dinâmica de memória semelhante ao oferecido pelo tradicional comando `malloc`, disponível na biblioteca padrão das linguagens C e C++. Para fazer uma distinção clara entre as funções deste trabalho e as da biblioteca padrão, vamos denominar a nova função de alocação como `valloc`.

Em vez de alocar memória dinâmica a partir do `heap`, o espaço que será gerenciado pela função `valloc` será formado com base em um vetor global, dimensionado estaticamente. A partir disso, a função `valloc` deverá alocar e liberar blocos de memória do vetor, respeitando a lógica de funcionamento semelhante ao `malloc` e ainda dentro dos seguintes princípios:

- Retornar um apontador para o bloco de memória reservado;
- Gerenciar a reserva de blocos de memória, de modo a garantir que nenhuma chamada subsequente a `valloc` ofereça espaço previamente alocado, a menos que o espaço tenha sido liberado;
- Gerenciar a liberação de blocos de memória e sua reoferta para nova alocação;
- Executar de forma eficiente, de modo a causar o menor impacto possível sobre o desempenho dos programas.

Naturalmente, sendo construída para funcionar baseada em um vetor global estático, a função valloc não compartilhará o espaço de memória com outros processos em execução na máquina, como é o caso de malloc. A função valloc terá a seguinte assinatura:

```
void *valloc(size_t size);
```

Quando um bloco de memória tem sua alocação solicitada, a função `valloc` deve determinar se existe algum segmento livre do vetor com a capacidade necessária. Se não houver, a função retornará NULL. Se houver, o bloco é reservado, e seu endereço inicial é retornado. A alocação deve considerar blocos de 1 byte, não havendo alinhamento de endereços nem paginação de blocos.

Outras funções semelhantes às tradicionais serão necessárias (consultar a documentação de C para ver o comportamento esperado). Criar as funções e inseri-las no header para uso público:

```
void vfree(void *p);
void *vcalloc(size_t nitems, size_t size);
void *vrealloc(void *p, size_t size);
```

O vetor global terá tamanho máximo igual a 1.024.576 bytes, podendo ser instanciado da seguinte maneira:

```
#define MAX_MEM 1024576
extern unsigned char MEM[MAX_MEM];
```

Tipos Abstratos de Dados

A principal preocupação na construção de algoritmos e estruturas de dados para o `valloc` deve ser com a fragmentação do espaço disponível no vetor. Para resolver isso eficientemente, defina um TAD que represente os blocos de memória alocados e organize-os em uma lista, de modo a gerenciar a ocupação e a disponibilidade de espaço. Lembre-se, ao modularizar seu código, que é necessário (1) encontrar um bloco livre que possa conter o espaço solicitado, (2) quando um bloco é liberado, o(s) bloco(s) livre(s) adjacente(s) deverá(ão) ser agrupado(s), formando um bloco maior, e (3) na realocação de espaço, pode ser necessário mover o conteúdo anterior para uma nova posição.

Pré-configuração e Status

Para pré-configurar e verificar o status da memória, três funções adicionais deverão ser criadas:

```
void inicializa_gerencia_memoria(void) // aloca e inicializa TAD auxiliar
void finaliza_gerencia_memoria(void) // libera espaço de memória da TAD auxiliar
void imprime_status_memoria(void)
```

A função `imprime_status_memoria` deverá ser a única função responsável pela escrita na saída padrão em seu código, seguindo o formato:

```
Status agora:
Pos: 0, Size: 20, Status: USED
Pos: 20, Size: 1024556, Status: FREE
```

Onde, na segunda linha, `POS` indica a posição do inicio do primeiro bloco de memória, que foi alocado com tamanho 20 e está atualmente sendo utilizada. Já a terceira linha indica que a partir da posição 20 existe um bloco contiguo de tamanho 1.024.556 livre.

Testes e avaliação

Deverá ser produzido obrigatoriamente um arquivo de `header` com o nome `valloc.h` contendo a assinatura das funções indicadas, de modo que este possa ser integrado ao programa principal, como pode ser visto no arquivo `main.c` fornecido.

As funções terão seu funcionamento avaliado pela execução do conjunto de instruções interpretadas pela função `main` (ver anexo), que por sua vez fará a chamada das funções fornecidas por sua implementação. Dessa forma uma série de alocações, realocações e liberações de memória serão realizadas, bem como o acompanhamento do status da memória. A sintaxe e o comportamento das novas funções deverão ser idênticos aos de `malloc`, `free`, `calloc` e `realloc`.

A saída será comparada com a de uma implementação padrão dentro do *Prático*. Cada teste do *Prático* é composto por uma sequência de instruções, que tem seu formato e sintaxe descrita no anexo. Sua saída será correspondente a uma série de chamadas da função `imprime_status_memoria`, mencionada anteriormente. Será fornecido um exemplo de entrada e saída no *Moodle*.

Comentários Gerais:

1. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
2. Clareza, indentação e comentários no programa também serão avaliados.
3. O trabalho é individual.
4. A submissão será feita pelo Prático (`aeds.dcc.ufmg.br`)
5. O Prático desconsidera espaços, quebras de linha e tabulações a mais de sua saída, portanto não é necessário alinhar de forma exata estes itens à saída padrão fornecida.
6. Trabalhos copiados, comprados, doados, etc. serão penalizados conforme anunciado.
7. Penalização por atraso: (2^d – 1) pontos, onde d é o número de dias de atraso.

Referências

[1] **A *malloc* tutorial**: documento que explica o funcionamento completo da função *malloc* e mostra sua implementação em C, para o caso "real" (memória dinâmica).
*www.inf.udec.cl/~leo/**Malloc_tutorial**.pdf*

Anexo I – Sintaxe interpretador

O interpretador utiliza as seguintes instruções:

```
VAR <numero de variáveis>
```

Indica ao interpretador o número de variáveis que devem ser alocadas pelo programa. Esse comando é obrigatório para a execução do programa.

```
VALLOC <tamanho do bloco> V<identificador da variável>
VCALLOC <tamanho do bloco> V<identificador da variável>
VREALLOC <tamanho do bloco> V<identificador da variável>
```

Solicita ao interpretador chamadas respectivas funções: `valloc`, `vcalloc`, `vrealloc`. Onde <*tamanho de bloco*> representa o tamanho do bloco a ser armazenado, e <*identificador*> o identificador da variável onde deve ser armazenada a primeira posição do bloco, retornada pela função.

```
VFREE V<identificador>
```

Solicita a chamada da função `vfree` a variável com <*identificador*> fornecido na instrução.

```
PRINT
```

Solicita a chamada da função `imprime_status_memoria`.

Anexo II - Execução do programa:

./tp1 <arquivo entrada>

Anexo III - Exemplo Entrada

```
VAR 2
PRINT
VALLOC 10 V0
PRINT
VCALLOC 10 V1
PRINT
VREALLOC 20 V0
PRINT
VFREE V0
VFREE V1
PRINT
```

Anexo IV – Exemplo Saída

```
VAR 2
Status agora:
Pos: 0, Size: 1024576, Status: FREE

VALLOC 10 V0
Status agora:
Pos: 0, Size: 10, Status: USED
Pos: 10, Size: 1024566, Status: FREE

VCALLOC 10 V1
Status agora:
Pos: 0, Size: 10, Status: USED
Pos: 10, Size: 10, Status: USED
Pos: 20, Size: 1024556, Status: FREE

VREALLOC 20 V0
Status agora:
Pos: 0, Size: 10, Status: FREE
Pos: 10, Size: 10, Status: USED
Pos: 20, Size: 20, Status: USED
Pos: 40, Size: 1024536, Status: FREE

VFREE 0

VFREE 1
Status agora:
Pos: 0, Size: 1024576, Status: FREE
```