Fundamentos de Sistemas Operacionais
Atividade Extra 04

Prof. Tiago Alves

Gerência de Memória

Introdução

A disciplina de Fundamentos de Sistemas Operacionais trata de diversos tópicos desses sistemas que provêem uma forma intuitiva de se utilizar as funcionalidades de computadores digitais sem que seja necessário ao usuário ou programador ter profundo conhecimento das interações entre os diferentes *hardwares* que compõem um computador.

Para construir ou adicionar funcionalidades a esses sistemas computacionais, é necessário conhecimento de linguagens de programação e ferramentas de desenvolvimento. Em nosso curso, o domínio da linguagem C é um pré-requisito para o devido acompanhamento das atividades da disciplina.

Objetivos

1) Exercitar conceitos da linguagem de programação C, especialmente aqueles referentes à programação de sistemas operacionais.
2) Exercitar princípios de gerência de memória.

Referências Teóricas

Mitchell, Mark, Jeffrey Oldham, and Alex Samuel. *Advanced linux programming*. New Riders, 2001.

Material Necessário

- Computador com sistema operacional programável
- Ferramentas de desenvolvimento GNU/Linux ou similares: compilador GCC, depurador, editor de texto.

Roteiro

1) Revisão de técnicas e ferramentas de desenvolvimento de aplicações para o sistema operacional Linux.

Colete o material acompanhante do roteiro do trabalho a partir do Moodle da disciplina e estude os princípios e técnicas de desenvolvimento de aplicações para o sistema operacional Linux.

2) Realizar as implementações solicitadas no questionário do trabalho.

Implementações e Questões para Estudo

1) Essa atividade consiste na escrita de uma versão especial das funções malloc e free que tentarão auxiliar a vida do programador por meio da conferência de erros comuns durante alocações na heap.
   - Suas versões de malloc e free serão chamadas mallocfso e freefso. É facultada a criação de uma função extra, memcheckfso, para verificação se determinado espaço de endereço alocado na heap é apropriado para uso.
   - Como programador, suas versões dependerão das versões originais de malloc e free. Além disso, suas implementações deverão primeiro conferir se a operação demandada está correta/íntegra e se faz sentido, e, somente após essa etapa, chamarão o núcleo de mallocfso e freefso para terminar o trabalho.
   - Além disso, sempre que seu programa de testes usado para validar as versões criadas necessitar alocar memória em heap, a sua aplicação de testes deverá inserir uma chamada preparatória para memcheckfso de forma a verificar se o espaço de endereçamento é válido.
   - Um módulo importante de seu programa será uma estrutura de dados que eficientemente tratará as requisições do espaço de endereçamento, o que significa buscar endereços em determinado espaço de endereços (intervalo de endereços). Essa não é uma estrutura de dados fácil de implementar eficientemente.
   - Você criará um módulo fsomalloc.c com as quatro funções descritas acima. Além disso, deverá criar um header, fsomalloc.h, com protótipos de funções de acordo com a interface a ser descrita mais à frente.
   - Importante: Você testará seu código com vários programas de testes modificando esses programas paa que usem suas novas versões em vez das versões padrão. Dessa maneira, é importante a inserção de chamadas à memcheckfso antes de qualquer requisição de memória na heap.

API de alocação com checagem

`void *mallocfso(size_t size):`

Além de alocar memória por chamada a malloc(), essa função irá registrar uma dupla (addr i, len i) para o espaço de memória requisitado à heap. Você obterá o endereço do início do espaço addr i do valor de retorno de malloc() e o valor do comprimento, len i, será obtido do parâmetro de entrada size. É importante codificar defensivamente, conferindo se size possui um valor inválido.

`void freefso(void *ptr):`

Essa função verificará primeiro se o espaço indicado por ptr foi previamente alocado no programa, então irá encaminhar o pedido a free() para que a liberação de espaço na heap seja efetivada.

Algumas condições de erro que você deverá conferir:

- liberar memória que não foi alocada por mallocfso;
- liberar memória cujo primeiro byte não é o início de uma área de memória previamente alocada.
- Liberar memória já liberada (double free)

Quando um erro for detectado, imprimir uma mensagem detalhada de erro e sair do programa com status -1.
Se todas as verificações de consistência dos parâmetros de freefso forem bem sucedidas, a função libera a área de memória e remove a dupla (addr, len) da estrutura global de controle de blocos da heap

`void *reallocfso(void *ptr, size_t size):`

Se prt for NULL, encaminhar requisição para mallocfso(). Se o size for zero e ptr não for null, encaminhar requisição para freefso(). Caso contrário, além de mudar o bloco de memória previamente alocado encaminhando a requisição a realloc(), a função deverá, antes de mais nada, conferir se há uma dupla para (addr, len) e, em seguida, deverá remover a dupla e adicionar uma nova dupla onde addr será o valor de retorno de realloc e len, o seu tamanho.

`void memcheckfso(void *ptr, size_t size):`

Essa função verificará se o espaço de endereçameto especificado pelo endereço ptr e o comprimento size estão no intervalo alocado por mallocfso() e, também, se a memória ainda não foi liberada por freefso(). Se um erro for detectado, imprimir mensagem de erro detalhada e encerrar o programa com status -1.

Exemplos:

```
char *buff = (char *)malloc (10);
char c;
struct node *np = (struct node *)malloc (sizeof(struct node));
struct node n;
int *ip = (int *)malloc (sizeof(int));
memcheck537 (&(buff[5]), 1);
c = buff[5];
memcheck537 (buff, strlen("hi mom!"));
strcpy(buff, "hi mom!");
memcheck537 (np, sizeof(struct node));
bcopy (&n, np, sizeof(struct node));
```

Instruções e Recomendações

A submissão das respostas aos problemas dos trabalhos deverá ser feita através do Moodle da disciplina.

Cada resposta a problema dessa Atividade Extra deverá ser entregue em um pacote ZIP. A dupla de alunos deverá nomear o pacote ZIP da seguinte forma: nome_sobrenome_matricula_nome_sobrenome_matricula_**ae04.**zip.

Entre os artefatos esperados, listam-se:

- códigos-fonte C das soluções dos problemas;
- documentação mínima da aplicação:
  - qual sistema operacional foi usado na construção do sistema;
  - qual ambiente de desenvolvimento foi usado;

- quais são as telas (instruções de uso)
- quais são as limitações conhecidas

Não devem ser submetidos executáveis.

Códigos-fonte C com erros de compilação serão desconsiderados (anulados).

Os trabalhos poderão ser realizados em duplas; a identificação de cópia ou plágio irá provocar anulação de todos os artefatos em recorrência.