EP3 – MAC422 - Sistemas Operacionais

Melhorado o memory manager

Lucas Moretto da Silva 9778602

Nesse EP modifiquei o **memory manager** do MINIX 3. Foi adicionada a chamada de sistema `memalloc()` que muda a politica de alocação de memoria entre *first-fit*, *worst-fit*, *best-fit*, *next-fit*, *random-fit*. Além disso, foi criado o utilitário `memstat` que exibe o *memory map*, e o `change_allocation_policy` para alterar a política de escalonamento da memória.

Note que é possível encontrar os códigos alterados procurando as linhas de comentario `\*EP3###...####*/` em arquivos `.c` e `.h`.

0 Informações da VDI

O minix foi instalado em `d0p0`. Caso o usuário seja deslogado ou o minix seja desligado (shutdown), favor logar com usuário root (não precisa de senha) e bootar em d0p0 (boot d0p0).

- **Teclado padrão:** br-abnt2
- **Usuário:** root
- **Senha do usuário:** não possui senha

OBS: Algumas vezes, ao ligar a VDI aparece uma mensagem *"strange, got SYN_ALARM"*. Você pode ignorar essa mensagem. Basta digitar o nome de usuário (root) e teclar ENTER, para logar normalmente.

1 Definindo a *syscall*

Como o mecanismo que faz alocação de memoria fica no **process manager**, foi neste servidor que as modificações para definir a nova *syscall* foram feitas.

Assim como no EP2, foi definida uma nova entrada no arquivo `/usr/src/servers/pm/table.c`, a entrada para `memalloc` fica na posição 66. Também foi adicionado o protótipo da nova chamada em `/usr/src/servers/pm/proto.h`. Em seguida, no arquivo `/usr/src/servers/pm/alloc.c` foi implementada a função `do_memalloc`. Tal função faz a checagem se o processo invocador tem permissões de *root*, mas sua principal ação será explicada na proxima sessão.

Para definir uma função que possa ser utilizada como uma biblioteca disponivel para o usuário, modificações foram feitas nos arquivos `/usr/src/include/minix/callnr.h` e `/usr/include/minix/callnr.h`, definindo MEMALLOC para o valor 66.

Na pasta `/usr/src/lib/posix/` adicionou-se o arquivo `_memalloc.c`, que será responsavel por fornecer a interface entre a função de usuário e a syscall.

Ao final disso, bastou realizar alguns ajustes em Makefiles e compilar as novas funções da biblioteca, assim como as bibliotecas em si.

2 Implementação da nova politica

Para deixar o uso mais simples, foram definidos novos macros em `/usr/src/include/unistd.h`, obtendo `FIRST_FIT == 0`, `WORST_FIT == 1`, `NEXT_FIT == 2`, `BEST_FIT == 3`, `RANDOM_FIT == 4`. Assim, podemos fazer a chamada `memalloc(WORST_FIT)` para mudarmos a política de alocação de memória para *worst-fit*.

O arquivo modificado para implementar essa nova política foi o `/usr/src/servers/pm/alloc.c`. Nele foi definida a variável privada `alloc_mode`. Essa representa qual a política de alocação que deve ser ativada no momento.

Nesse mesmo arquivo, temos a função `do_memalloc()`, que é responsável por aplicar a *syscall*. Primeiro, essa função faz a checagem se o *effective uid* do processo que a invocou é o *root uid*, ou seja, 0. Depois, ela pega da mensagem o modo que foi passado como argumento, e caso ele seja algum dos modos predefinidos, *seta* `alloc_mode` de acordo.

Finalmente, a função que realiza a alocação de memória em si, `alloc_mem`, foi modificada. Fizemos um condicional para diferenciar a politica que deve ser aplicada.

2.1 Worst fit

O procedimento tomado para aplicar a politica de *worst-fit* consiste em percorrer toda a lista de *holes* em busca do buraco com maior *h_len* (tamanho). Depois disso é checado se o tamanho desse buraco suporta o número de *clicks* pedido a função. Em caso positivo, o buraco é diminuido e o começo de seu endereço base é retornado. Caso negativo, tenta-se realizar o *swap* algum processo da memória para criar mais espaço.

2.2 Best fit

O procedimento tomado para aplicar a politica de *best-fit* consiste em percorrer toda a lista de *holes* em busca do buraco com que melhor se adequa ao *h_len* (tamanho) do processo. Em caso positivo, o buraco é diminuido e o começo de seu endereço base é retornado. Caso negativo, tenta-se realizar o *swap* algum processo da memória para criar mais espaço.

2.3 Random fit

Para realizar o *random-fit* primeiramente percorremos a lista de buracos e contamos a quantidade de buracos disponíveis que poderiam alocar o processo. Essa quantidade é armazenada em `randHolesAvail`.

Com tal quantidade em mãos, escolhe-se um número randomicamente entre 0 e randHolesAvail. Ele indicará qual dos buracos nós utilizaremos para o processo. Esse número é armazenado em `randPos`.

Com `randPos` em maõs, percorre-se a lista de buracos novamente, atualizando um contador `randHolesCount`. Quando `randHolesCount` for igual a `randPos`, então encontramos o buraco sorteado e o utilizamos para alocar o processo.

2.4 Next fit

Para realizar o `next-fit` realizamos a mesma lógica do `first-fit`, com a diferença de que sempre mantemos as posições dos últimos buracos utilizados nas variáveis `next_fit_head` e `next_fit_prev_ptr`. Assim, sempre que o algoritmo de alocação é executado, tentamos começar o loop de verificação de buracos por meio desses ponteiros.

3 Utilitário memstat

Para a implementação do utilitário, me inspirei no `top.c`. Assim como no programa `top`:

- Utiliza-se a *syscall* `getsysinfo` para conseguir as informações dos buracos (holes) da memoria. Ainda utilizamos uma verificação de erro ao obter as sysinfos do `SI_MEM_ALLOC` (para acessar a struct `pm_mem_info`, que possui informações sobre os buracos da memória);
- Imprimi-se o espaço disponível na memória, faz-se isso como o `top.c` faz, percorrendo os buracos nos utilizando da struct `pm_mem_info`, obtendo a base do vetor de buracos e o comprimento do mesmo. A parte *tricky* é converter esse tamanho de *clicks* para *bytes*, fiz isso realizando um *bitshift* de tamanho `CLICK_SHIFT`. Aqui, criou-se uma estrutura de dados Array (array dinâmico) para armazenar todos os tamanhos dos buracos disponíveis. Após percorrer todos os buracos e armazenar no array, calcula-se a média, desvio padrão e mediana dos tamanhos.

O programa `memstat` tem seu código fonte em `/usr/local/src/memstat.c`. Também temos o executável `/root/memstat` e pode ser executado da seguinte forma:

```
cd /root
./memstat
```

4 Utilitário change_allocation_policy

Foi criado um programa de usuário chamado `change_allocation_policy` para trocar o algoritmo de escalonamento a partir da shell. O primeiro argumento, passado para o argv do programa, é uma das strings first_fit, worst_fit, best_fit, next_fit ou random_fit. Esse programa utiliza a syscall `memalloc` definida anteriormente.

O programa `change_allocation_policy` tem seu código fonte em `/usr/local/src/change_allocation_policy.c`. Também temos o executável `/root/change_allocation_policy` e pode ser executado da seguinte forma:

```
cd /root
./change_allocation_policy first_fit
```

5 Arquivos de teste do E-Disciplinas

Os arquivos de teste auxiliares fornecidos no E-Disciplinas (forkmem.c, usemem.c, mkmemuse) foram adicionados a VDI.

Os arquivos com código do fonte C estão em:

- `/usr/local/src/forkmem.c`
- `/usr/local/src/usemem.c`
- `/usr/local/src/mkmemuse`

Os arquivos com o binário gerado pela compilação estão em:

- `/root/forkmem`
- `/root/usemem`
- `/root/mkmemuse`