# EP3 – MAC422 - Sistemas Operacionais

## Melhorado o memory manager

Daniela Gonzalez Favero 10277443
Felipe Castro de Noronha 10737032

Nesse EP modificamos os **memory manager** do MINIX 3. Adicionamos a chamada de sistema `memalloc()` que muda a politica de alocação de memoria entre *first-fit* e *worst-fit*. Além disso, implementamos o utilitário `memmap` que exibe o *memory map*.
Note que é possível encontrar os códigos que alteramos procurando as linhas de comentario /*EP3###... ####*/ em arquivos `.c` e `.h`.

# 1 Definindo a *syscall*

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

Assim como no EP2, definimos uma nova entrada no arquivo `/usr/src/servers/pm/table.c`, a entrada para `memalloc` fica na posição 66. Adicionamos também 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, modificamos os 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/` adicionamos o arquivo `_memalloc.c`, que sera 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

Primeiramente, para deixar as coisas mais claras, definimos duas novas macros em `/usr/src/include/stdlib.h`, obtendo `WORST_FIT == 1` e `FIRST_FIT == 0`. 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 definimos a variável privada `alloc_mode`, ela representa qual a política de alocação que queremos 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. 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 *swappar* algum processo da memória para criar mais espaço.

# 3 Utilitário

Para a implementação do utilitário, nos inspiramos muito em outro utilitário, o `top.c`. Assim como no programa `top`, criamos um arquivo com o código fonte do mesmo em `/usr/src/commands/simple` chamado `memmap.c`. O código do utilitário consiste em:

- Utilizar a *sycall* `getsysinfo` para conseguir as informações dos processos que estão ocupando a memória e os buracos (holes) da mesma. 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) e do `SI_PROC_TAB` (para acessar a struct `mproc` que contém as informações dos processos, como *pid*, *start* e *end*);
- Imprimir os pids, início e final (na memória) dos processos que estão ocupando memória na máquina. Percorrendo por todos os *NR_PROCS* processos, obtemos do i-ésimo `mproc[i]` seu pid (acesso direto na struct), seu início e final (delimitados pelo vetor mp_seg[], o segmento de memória, que possui 3 posições; portanto obtemos `mp_seg[0]` e `mp_seg[2]`);
- Imprimir o espaço disponível na memória, fazemos 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*, fizemos isso realizando um *bitshift* de tamanho `CLICK_SHIFT`. Somamos e comprimento de todos os buracos e obtemos a memória disponível em Bytes.

Após a implementação do código, foi necessário alterar o Makefile do `/usr/src/commands/simple`, fizemos analogamente aos outros arquivos do simple: o Makefile trata da compilação e instalação do nosso utilitário (busque por `memmap` no Makefile para ver as alterações).

Demos `$ make` em `/usr/src/commands/simple` e `$ make install` em `/usr/src/commands`, é possível verificar que o binário foi criado em `usr/bin`.

# 4 Teste

Há um arquivo `teste.c` em `/home`. Compile-o e execute para ver a saída do memory map conforme modificamos a políticas de alocação de memória, além de uma checagem se o memalloc funciona.

Também é possível testar o memory map chamando `$ memmap` no terminal.