EP2 – MAC422 - Sistemas Operacionais

Daniela Gonzalez Favero 10277443
Felipe Castro de Noronha 10737032

As quatro tarefas pedidas pelo enunciado foram implementadas assim:

1. A macro

Acrescentamos uma nova fila de prioridade ao Minix com a macro BATCH_Q em *proc.h*.

2/3. As chamadas de sistema

Os arquivos *_batch.c* e *_unbatch.c* em */usr/src/lib/posix/* cada uma em seu respectivo arquivo, de modo a receber uma mensagem (o próprio processo) e repassá-la para o Process Manager (através de uma chamada de sistema).
Depois, definimos as macros das chamadas de sistemas na *callnr.h* e prototipamos as funções no *proto.h*. Também alteramos o vetor de chamadas de sistemas em *table.c*.
Em *do_nice.c*, passamos a receber a exceção do BATCH_Q para possibilitar mudança de prioridade.
Em *misc.c* , estão implementadas as funções `do_batch()` e `do_unbatch()` que utiliza a `sys_nice()` para mudar a prioridade do processo passado como argumento. Quando as chamadas dão certo, elas retornam 0 e quando dão errado, retornam -1. Além disso, checamos se o pid passado é válido e se foi o processo pai que invocou essa chamada.

4. O algoritmo de escalonamento

Em *proc.c*, implementamos o novo algoritmo de escalonamento. Em `sched()`, verificamos se o processo deve ser colocado na fila de prioridade BATCH, então chamamos a função `batch()` (caso contrário, a função sched termina de ser executada, sendo que a prioridade mais baixa é dada por BATCH_Q-1).
Em `batch()`, garantimos que nenhum processo muda de fila não alterando a prioridade do mesmo. Em um laço, percorremos a fila BATCH procurando pelo menor e maior número de tiques. Se forem iguais, todos os processos têm o mesmo número de tiques, então ele roda o round robin. Se não, decidimos se o processo será colocado na frente (deve rodar) ou atrás (não deve rodar) da fila, levando em conta o número mínimo de tiques que algum processo possui.
No *proc.c* também editamos a função `balance_queues()` para garatantir que a prioridade de um processo BATCH_Q não aumente.

Arquivos modificados

Arquivo	Mudança
`/usr/src/kernel/proc.h`	Nesse arquivo definimos o novo macro para a fila `BATCH`.
`/usr/src/kernel/proc.c`	Aqui foi onde mudamos o escalonador. Basicamente fizemos um tratamento totalmente diferente para processos que estão com a prioridade definida como `BATCH`. Também alteramos o `balance_queues`.

Arquivo	Mudança
`/usr/src/kernel/system/do_nice.c`	Uma exceção foi adicionada para o caso em que a *system task* é invocada para mudar a prioridade de um processo para `BATCH_Q`
`/usr/src/servers/pm/table.c`	Adicionamos novas entradas no vetor, que representam as novas chamadas de sistama `do_batch` e `do_unbatch`.
`/usr/src/servers/pm/proto.h`	Prototipamos as novas chamadas.
`/usr/src/servers/pm/misc.c`	Fizemos aqui a definição das chamadas, ou seja, o código/maquinário das duas funções fica nesse arquivo.
`/usr/src/include/minix/callnr.h`	Definimos as macros `BATCH` e `UNBATCH`, a posição das novas chamadas no vetor.
`/usr/src/lib/posix/_batch.c`	Arquivo que vai realizar a *system call* relacionada à `BATCH`. É essa função que é chamada quando um programa de usuário invoca `batch(pid)`.
`/usr/src/lib/posix/_unbatch.c`	Arquivo que vai realizar a *system call* relacionada à `UNBATCH`. É essa função que é chamada quando um programa de usuário invoca `unatch(pid)`.

Para executar

Para construir no sistema as nossas modificações, fizemos, basicamente, douas sequencias de make:

- Compilando a biblioteca/chamadas de sistema

```
# cd /usr/src/servers
# make image && make install
# cd /usr/src/lib/posix
# make Makefile
# cd /usr/src
# make libraries
```

- Compilando o kernel e criando nova imagem de boot

```
# cd /usr/src/tools
# make hdboot && make install
```

Após isso, foi necessarios dar um *reboot* no sistema com a nova imagem.

Testando

Para testarmos nosso escalonador, criamos o arquivo *teste.c* na *home*, que possui um processo que roda em baixíssima prioridade, chamando `batch()` e `unbatch()` para esse processo. Rodamos esse arquivo em

background e executamos o comando `top`. Assim é possível ver a mudança de prioridade do processo *teste*, comprovando que o nosso escalonador funciona.