MAC0422 - EP2

Daniel Martinez - 10297709
Pedro Paulo Bambace - 10297668

Arquivos criados/modificados

Visualização da tabela de processos

- `minix_os/usr/src/servers/is/dmp_kernel.c`
- `minix_os/usr/src/servers/is/dmp.c`
- `minix_os/usr/src/servers/is/proto.h`
- `minix_os/usr/include/minix/callnr.h`
- `minix_os/usr/src/include/minix/callnr.h`

Chamada de sistema para alteração de prioridade

- `minix_os/usr/include/sys/resource.h`
- `minix_os/usr/src/lib/posix/priority.c`
- `minix_os/usr/src/servers/pm/misc.c`
- `minix_os/usr/src/servers/pm/proto.h`
- `minix_os/usr/src/servers/pm/table.c`
- `minix_os/usr/src/lib/syslib/sys_nice.c`
- `minix_os/usr/src/include/minix/syslib.h`
- `minix_os/usr/include/minix/syslib.h`
- `minix_os/usr/include/minix/com.h`
- `minix_os/usr/src/include/minix/com.h`
- `minix_os/usr/src/kernel/system.c`
- `minix_os/usr/src/kernel/system.h`
- `minix_os/usr/src/kernel/kernel.h`
- `minix_os/usr/src/kernel/system/Makefile`

- `minix_os/usr/src/kernel/system/.depend`
- `minix_os/usr/src/kernel/config.h`
- `minix_os/usr/src/kernel/system/do_priority.c`

Outros

- `minix_os/usr/src/b` - Arquivo para automatizar o build
- `minix_os/root/teste.c` - Arquivo que testa a chamada implementada

Detalhes de implementação

Visualização da tabela de processos

Nessa parte do EP, pegamos a tabela de processos através da chamada de sistema `sys_getproctab()` e a ordenamos em ordem decrescente de prioridade (na verdade, ordenamos em ordem crescente de p_priority, que quanto menor é, maior a prioridade do processo). Definimos que quando o usuário pressiona a tecla F4, imprimimos para cada processo dessa tabela as seguintes informações:

- nr do processo
- Nome do processo
- Prioridade de execução
- PID do processo
- Tempo de CPU
- Tempo de Sistema
- Endereço do ponteiro da pilha

Para conseguir o PID do processo, foi implementada uma chamada de biblioteca `getpidfromnr()` que retorna o PID do processo dado seu nr. Ela executa a chamada de sistema `CHPRIORITY` do *Process Manager* como modo 1, que relaciona as tabelas de processos `mproc` e `proc` e retorna o PID do processo especificado.

Se houver mais processos que os que cabem na tela, eles serão exibidos na próxima vez que o usuário pressionar F4.

Chamada de sistema para alteração de prioridade

Implementamos a chamada de biblioteca `chpriority()` como especificado no enunciado, que executa a mesma chamada de sistema `CHPRIORITY`, mas como modo 0. Esta verifica se a prioridade passada está entre `MAX_USER_Q` e

`MIN_USER_Q`, que são as prioridades permitidas para um processo de usuário e também se o processo cujo PID foi passado é filho do processo que a chamou através do atributo `mp_parent` do processo. Em seguida, ela executa a chamada de *kernel* `sys_priority` para a alteração da prioridade. Essa chamada de *kernel* retira o processo da fila de execução, altera sua prioridade e prioridade máxima, e o coloca na nova fila.

Para retornar números negativos nem a utilização de `errno`, utilizamos a função `_taskcall()` no lugar da `_syscall()`.

Arquivo de teste

Há um arquivo de teste em `/root/teste.c` que testa a chamada de sistema implementada nesse EP para três casos:

- Processo não-filho, onde o resultado esperado é -2
- Prioridade inválida, onde o resultado esperado é -1
- Prioridade válida para um processo filho, onde o resultado esperado é a mudança de prioridade de tal processo, retornando seu valor.

Nesse último caso, utilizamos a chamada de sistema `fork()` para a criação de um processo filho, que fica somente espera 100 segundos e termina, enquanto o processo pai faz a chamada de sistema implementada, também esperando 100 segundos depois disso para que seja possível verificar se o valor realmente foi alterado na tabela de processos (pressionando F4).

Arquivo de build

É um simples script que automatiza o processo de build. Ele executa o `make world` para compilar todas as partes do Minix, move a imagem de *kernel* gerada para `/boot/image/image_big` e se tudo der certo, reinicia o sistema automaticamente.