# Administração de processos em Minix
# Relatório do Projeto 3

Diogo T. Sant'Anna, *RA 169966,* Luciano G. Zago, *RA 182835,*
e Maria A. Paulino, *RA 183465*

**Resumo**—Este projeto teve por objetivo promover a familiarização com alguns modos de escalonamento estudados e alterar o sistema de escalonamento de processos do Minix. Para tal, foi requerido projetar e implementar uma solução de software que seja capaz de administrar os processos de um sistema operacional e agendá-los segundo uma política baseada em parentesco. Os processos pais devem se executar em *Round Robin* e seus filhos em *First Come First Serve*. O projeto exigiu conhecimentos sobre como o Minix executa a distribuição de processos ao escalonador e sobre os métodos de escalonamento utilizados. Por fim, foi possível desenvolver uma solução capaz de executar todas as funções exigidas no enunciado deste projeto.

## 1 Introdução

Neste projeto desenvolveu-se um administrador para coordenar a atuação de dois escalonadores, também implementados pelo grupo. Os últimos possuem políticas e aplicações distintas

- *Round Robin*(RR): O algoritmo atribui frações de tempo (*quantum*) para cada processo, em partes iguais, de forma circular (o que elimina o problema de inanição). O processo é interrompido (preempção) quando se esgota o *quantum*[1]. Após isso, se executa o próximo processo da fila circular. Este deve ser atribuído à processos que têm *shell* como pai.
- *First Come First Serve*(FCFS): Cria uma fila de execução de processos por ordem de chegada. O processo que chegou antes é executado por completo (sem preempção), após o término de sua execução, o mesmo acontece com o próximo da fila. Como se trata de uma fila, todos os processos tendem a ser executados (evita a inanição), a não ser que ocorra algum erro ou loop infinito, pois como não existe preempção, não pode interromper o processo para executar o próximo[2]. Este deve ser atribuído à processos que não têm como pai *shell*.

## 2 Desenvolvimento

Sabendo que todos os processos de usuários são encaminhados para o servidor SCHED do Minix, implementamos os escalonadores e a administração deles no próprio SCHED e parte no PM.

O arquivo `schedule.c` do servidor SCHED implementa as funções `do_start_scheduling` e `do_stop_scheduling`, que são chamadas pelo Minix solicitando que o SCHED escalone e pare de escalonar um processo de usuário, respectivamente. Foram essas duas principais funções que usamos para implementar nossos escalonadores.

Na função `do_start_scheduling` nós identificamos o processo como pai ou filho e administramos seu escalonamento da seguinte forma: se o novo processo for um filho de INIT ou de RS (representando um processo de sistema), nós o tomamos como um processo pai e o escalonamos de acordo com a política do *Round*

*Robin*; caso contrário, nós o tomamos como um processo filho e o escalonamos segundo a política do *First Come First Serve*(FCFS).

### 2.1 Implementação dos Escalonadores

Atualmente o algoritmo *Round Robin* já está implementado no *scheduler* do Minix, porém com multi-prioridades[3]. Portanto, iremos utilizar o algoritmo já implementado no server *"sched"*, apenas alternando a distribuição de prioridade para que todos os processos tenham a mesma prioridade.

Para implementar a política de escalonamento FCFS, no próprio *do_start_scheduling* nós verificamos se não há nenhum processo filho já escalonado, nesse caso imediatamente escalonamos o novo processo e indicamos que um processo filho já está escalonado. Caso contrário, nós apenas o adicionamos numa fila simples para que esse processo seja escalonado em alguma chamada do *do_stop_scheduling*.

No *do_stop_scheduling*, portanto, nós também verificamos se o processo finalizado é um processo filho. Em caso afirmativo, verificamos se há algum processo filho na fila esperando para ser executado e, em caso afirmativo, nós pegamos o primeiro elemento da fila e o escalonamos.

### 2.2 Administrador de Escalonadores

Segundo o enunciado deste projeto, o administrador deve selecionar a política usada para cada processo de acordo com o descrito anteriormente. Por praticidade e simplicidade, decidimos fazer com que os filhos de INIT sejam tomados como os pais, escalonados sob a política Round-Robin, e os demais processos processos sejam tomados como filhos, escalonados sob a política FCFS. Um problema que surgiu com essa decisão vem do fato de que muito dos filhos de INIT tem outros processos filhos. Caso que gera inanição ao passo que os filhos imediatos de INIT podem esperar o término da execução de seus filhos para prosseguir a sua própria execução, mas a política de escalonamento FCFS impõe que outros processos possam vir a rodar somente quando o processo em execução seja finalizado, então esses filhos nunca serão executados. Para viabilizar a utilização do FCFS da maneira como foi proposta, precisamos criar algumas exceções para que o sistema operacional pelo menos completasse o *boot*. Essas exceções foram tratadas verificando os novos processos e alterando o parentesco dos que causam problemas no processo de inicialização do sistema, fazendo com que o SCHED os interprete como filhos do INIT. Foi preciso modificar o servidor PM para implementar essas exceções, uma vez que somente nesse servidor nós temos acesso ao nome dos processos executados. A principal exceção a ser tratada foi o processo *sh*, que corresponde ao *shell* do minix e é filho direto do INIT, o que faria com que todos os filhos de *sh* fossem escalonados com FCFS, impedindo que quaisquer filhos do *shell* tenham filhos. Portanto, alteramos o parentesco de todos os processos filhos do *shell* indicando que são filhos direto de INIT, para que então sejam escalonados com Round Robin. Sob a mesma justificativa, redirecionamos os filhos dos processos `sshd`, `inetd`, `syslogd` e `dhcpcd`(por serem processos *daemons* do sistema) como sendo filhos direto de INIT. Quanto ao seu desenvolvimento, o arquivo `schedule.c` do servidor PM realiza mudanças no parentesco de determinados processos, para facilitar a verificação posterior no servidor SCHED. A vantagem do servidor PM é conhecer o nome dos processos que serão executados. Dessa forma, os processos de nome `sshd`, `inetd`, `syslogd` e `dhcpcd`, por serem processos *daemons* do sistema, são tratados de forma especial mesmo sendo filhos de *shell* e, portanto, são colocador no escalonador RR, através da mudança do pai para `INIT_PROC_NR`.

### 2.3 Testes e Validação

Foram realizados 2 testes principais com o objetivo de verificar a utilização dos escalonadores desenvolvidos. O primeiro tem por objetivo observar o *Round Robin*, e através de um script chama dois processos paralelamente, passando como parâmetro um identificador para cada um desses que por sua vez imprimem seu identificador. O primeiro processo chamado tem propositalmente tempo total de execução muito

superior ao segundo, porém devido à política de escalonamento, o menor deve terminar antes mostrando assim que o propósito do RR foi atendido.

Já o segundo, explicita o funcionamento do FCFS. Consiste em um script chamando um único processo, que por sua vez realiza duas vezes o comando `fork()`, criando assim dois filhos e cada um deles, assim como no teste descrito anteriormente, imprime o seu identificador. Tendo em vista a política do escalonador, o primeiro a ser criado deve ser executado completamente antes do início da execução do segundo, apesar de possuir tempo total de execução superior.

## 3 CONCLUSÃO

O projeto foi concluído implementando os modos de escalonamento exigidas pelo enunciado. Os processos pais se executam em *Round Robin* e seus filhos em FCFS. Essas tarefas foram realizadas através da edição do server *sched* e do PM. Foi possível conhecer o gerenciador de escalonamento do Minix e os métodos de escalonamento utilizados. Configuramos também o CI para rodar os testes criados pelo grupo. Por fim, foi possível desenvolver uma solução capaz de executar todas as funções exigidas no enunciado deste projeto.

## REFERÊNCIAS

[1] https://pt.wikipedia.org/wiki/Round-robin
[2] https://pt.wikipedia.org/wiki/FIFO_(escalonamento)
[3] http://wiki.minix3.org/doku.php?id=developersguide:userspacescheduling
[4] http://condor.depaul.edu/glancast/443class/docs/lecOct12.html#slide18
[5] http://www.minix3.org/docs/scheduling/report.pdf