# Documentação TP1


## Andre Lustosa Cabral de Paula Motta 2015097826

## Daniel Ishitani Melo  2015114208


# Introdução:

Nos foi apresentado o problema de recriarmos um shell básico capaz de executar comandos encadeados e realizar redirecionamentos e pipes. (Problema 1)

Além disso também deveríamos criar a nossa própria versão do comando TOP com o adicional de que o comando deveria ser capaz de receber um PID de um processo e enviar um sinal para este usando a função kill. (Problema 2)

# Implementação:

PROBLEMA 1 -

Para resolver o problema 1 utilizamos um esqueleto de bash disponibilizado na página do TP1. Ele já continha '90%' da implementação do bash, faltando para nós a implementação da execução de comandos, redirecionamento de I/O e criação de pipes concorrentes.

A execução de um comando é implementada pela função "executecmd" na linha 389 do arquivo shell.c . Essa função essencialmente realiza um fork e no processo filho chama a execução do comando passado na entrada.

A redireção de I/O é implementada pela função "redirectIO" na linha 420 do arquivo shell.c . Essa função essencialmente realiza um fork e redireciona a stdout ou a stdin de acordo com o comando ">" ou "<" e chama o runcmd que é

a função que executa seja redireção exec ou pipe novamente para o comando que exista.

Por fim a criação de um Pipe é implementada na linha 97 do arquivo shell.c. Essa Função é a mistura das duas acima. Criando um fork para um processo filho executar mas redirecionando a saída do primeiro programa na entrada do segundo. Chamando a função runcmd retroativamente para o lado esquerdo e direito do comando.

PROBLEMA 2 -

Para resolver o problema dois tivemos que usar o paralelismo e a biblioteca de pthreads além de outras três libraries que se mostraram essenciais para o correto funcionamento do programa dirent.h(usada para navegar pelo diretório /proc/) , pwd.h(usada para obter um username a partir do user id) e sys/stat.h ( usada para obter dados necessários para a impressão correta.

Definimos também uma estrutura de dados chamada Node, que continha todas as informações pertinentes a cada processo para que possamos imprimir o processo.

A função top fica em loop infinito. Começa por imprimir o cabeçalho do TOP e depois aloca um Node com as informações do primeiro processo obtido pela função readdir da dirent.h em seguida imprime os dados pertinentes do processo da free no Node e recomeça. Ela faz isso por default para os 20 primeiros processos mas isto pode ser modificado pelo if na linha 109 do arquivo meutop.c o número 20 pode ser modificado para o número de processos que quer ler.

Após imprimir os 20 primeiros processos a função chama sleep por um segundo. Limpa o I/O Stream padrão do computador e faz tudo de novo.

A função pidSig também fica em loop infinito, aguardando a entrada de um PID e o comando respectivo para a função kill. Quando recebe os dados ela executa a função kill com o parâmetro passado.

Para que ambas as funções rodem ao mesmo tempo criamos um thread para a

função TOP e a função pidSig roda indefinitivamente na main. O programa não fecha sozinho, mas podemos chamar kill para o próprio programa.

# Conclusão:

O trabalho prático  foi extremamente proveitoso para entender o funcionamento das chamadas de sistema e geração de processos de um Sistema Operacional. E  foi importante para reaver o ritmo de programação e recordar a utilização de paralelismo.

A maioria das funções pedidas foram bem simples de serem implementadas e específicas para o desenvolvimento do mesmo. O trabalho então foi concluído com sucesso e segundo as especificações não possuindo erros.