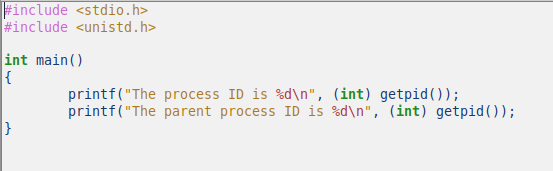


3. Faça um resumo de todas as seções do Capítulo 3, do livroAdvanced Linux Programming, e implemente os exemplos disponibilizados.

**Processos:**

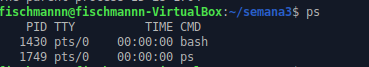
Um processo é qualquer conjunto de operação efetuado pela máquina que usa capacidade de cálculo do processador. Pode ser para o funcionamento da máquina, ou do shell de um programa qualquer.

Para ter o ID de um processo, podemos usar o código seguinte:

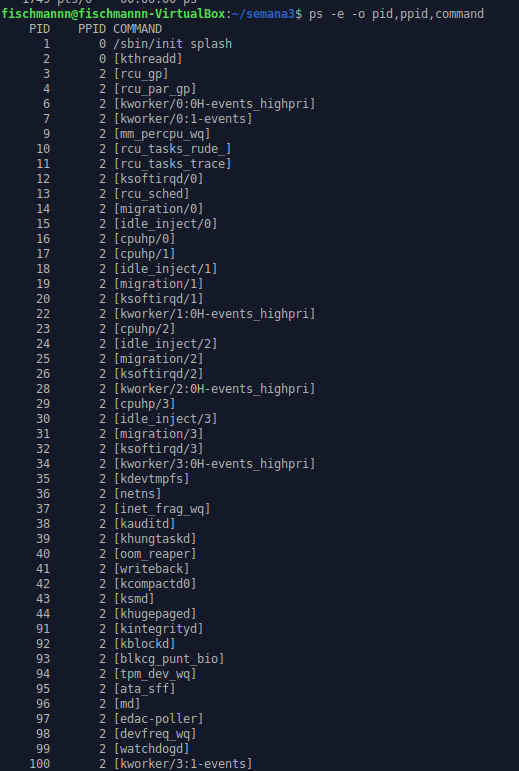




Podemos ver os processos que estão abertos no momento:



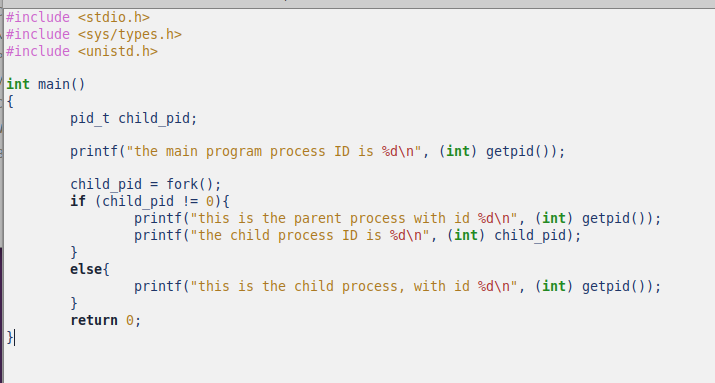
Ou, para mais detalhes:

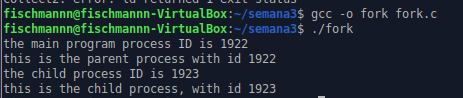


O processo pode ser matado com o comando kill, fechado usando o comando exit, ou podemos esperar que ele termine com o comando wait.

Para **criar um processo,** podemos criar um código em C como visto em cima, e executa-lo depois de compilar ele com o comando gcc.

Um processo pai pode criar um processo filho usando uma função chamada **‘fork’.** O processo filho é um duplicata do processo pai mas retorna outros valores.





A **função exec** permite trocar um processo em andamento com outro programa.

Se a função tem um ‘p’ no nome (execvp, execlp), ela aceita o noe de um programa e procura o nome dele no endereço das exceções. Se não ter o ‘p’ no nome, é preciso dar o endereço completo do programa a ser executado.

Se a função tem um ‘v’ no nome (execv, execvp, execve), ela aceita a lista de argumentos do novo programa como um array com termino nulo. Se ela ter u ‘l’ no nome (execl, execlp, execle), ela aceita a lista de argumentos usando o mecanismo varargs da linguagem C.

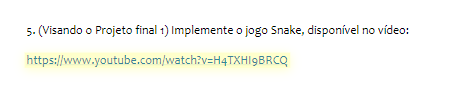
Se a função tem um ‘e’ no nome ( execve, execle), ela aceita mais um array de variáveis como argumento num formato especifico.

É comum usar a função fork antes da função exec para poder executar o processo pai e o processo filho juntos.

O Linux **permite gerenciar a prioridade dos processos** com o comando ‘nice’.

Os processos recebem **sinais** que podem servir a comunicar um erro no sistema. Os sinais são assíncrono, ou seja, não estão na fila de processos. Mas eles não devem ser usados para operações de I/O ou chamada de bibliotecas.

Se um processo filho termina sem o processo pai usar **a função wait**, o processo pai não recebe o estado de saída do processo filho. O processo filho se converte em **processo zombie**, ou seja, ele terminou mas não foi limpado pelo processo pai. É possível limpar os processos filhos usando a função wait quando o processo filho é chamado, mas também é possível usar comunicação interprocesso para notificar o processo pai quando o processo filho termina. O Linux já faz isso automaticamente.



Cf. snake.py