Universidade Federal de Uberlândia
Engenharia de Controle e Automação / Engenharia Mecatrônica
Sistemas Embarcados II / Sistemas Digitais para Mecatrônica
Prof. Éder Alves de Moura
Semana 03 – Linux como Ambiente de Programação

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:

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
        printf("The process ID is %d\n", (int) getpid());
        printf("The parent process ID is %d\n", (int) getpid());
}
```

```
fischmannn@fischmannn-VirtualBox:~/semana3$ gcc -o get-pid get-pid.c
fischmannn@fischmannn-VirtualBox:~/semana3$ ./get-pid
The process ID is 1704
The parent process ID is 1704
```

Podemos ver os processos que estão abertos no momento:

```
fischmannn@fischmannn-VirtualBox:~/semana3$ ps
    PID TTY          TIME CMD
   1430 pts/0    00:00:00 bash
   1749 pts/0    00:00:00 ps
```

Ou, para mais detalhes:

```
fischmannn@fischmannn-VirtualBox:~/semana3$ ps -e -o pid,ppid,command
    PID    PPID COMMAND
      1       0 /sbin/init splash
      2       0 [kthreadd]
      3       2 [rcu_gp]
      4       2 [rcu_par_gp]
      6       2 [kworker/0:0H-events_highpri]
      7       2 [kworker/0:1-events]
      9       2 [mm_percpu_wq]
     10       2 [rcu_tasks_rude_]
     11       2 [rcu_tasks_trace]
     12       2 [ksoftirqd/0]
     13       2 [rcu_sched]
     14       2 [migration/0]
     15       2 [idle_inject/0]
     16       2 [cpuhp/0]
     17       2 [cpuhp/1]
     18       2 [idle_inject/1]
     19       2 [migration/1]
     20       2 [ksoftirqd/1]
     22       2 [kworker/1:0H-events_highpri]
     23       2 [cpuhp/2]
     24       2 [idle_inject/2]
     25       2 [migration/2]
     26       2 [ksoftirqd/2]
     28       2 [kworker/2:0H-events_highpri]
     29       2 [cpuhp/3]
     30       2 [idle_inject/3]
     31       2 [migration/3]
     32       2 [ksoftirqd/3]
     34       2 [kworker/3:0H-events_highpri]
     35       2 [kdevtmpfs]
     36       2 [netns]
     37       2 [inet_frag_wq]
     38       2 [kauditd]
     39       2 [khungtaskd]
     40       2 [oom_reaper]
     41       2 [writeback]
     42       2 [kcompactd0]
     43       2 [ksmd]
     44       2 [khugepaged]
     91       2 [kintegrityd]
     92       2 [kblockd]
     93       2 [blkcg_punt_bio]
     94       2 [tpm_dev_wq]
     95       2 [ata_sff]
     96       2 [md]
     97       2 [edac-poller]
     98       2 [devfreq_wq]
     99       2 [watchdogd]
    100       2 [kworker/3:1-events]
```

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.

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        pid_t child_pid;

        printf("the main program process ID is %d\n", (int) getpid());

        child_pid = fork();
        if (child_pid != 0){
                printf("this is the parent process with id %d\n", (int) getpid());
                printf("the child process ID is %d\n", (int) child_pid);
        }
        else{
                printf("this is the child process, with id %d\n", (int) getpid());
        }
        return 0;
}
```

```
fischmannn@fischmannn-VirtualBox:~/semana3$ gcc -o fork fork.c
fischmannn@fischmannn-VirtualBox:~/semana3$ ./fork
the main program process ID is 1922
this is the parent process with id 1922
the child process ID is 1923
this is the child process, with id 1923
```

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.

5. (Visando o Projeto final 1) Implemente o jogo Snake, disponível no vídeo:

https://www.youtube.com/watch?v=H4TXHI9BRCQ

Cf. snake.py