UNIVERSIDADE ESTADUAL DE PONTA GROSSA

SETOR DE CIÊNCIAS AGRÁRIAS E DE TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

ENGENHARIA DE COMPUTAÇÃO

MATEUS FELIPE DA SILVA JUNGES

SISTEMAS OPERACIONAIS

ATIVIDADES 08/03/2018

PONTA GROSSA

2018

1. **Considerando o código Shell_Simples.c**
   a. **Descreva o uso/funcionamento das seguintes funções**

      - strtok():

      char *strtok(char * endereçoStrOrigem, char * endereçoStrDelimitador);

      A função strktok() retorna um ponteiro para a próxima palavra apontada pelo primeiro parâmetro passado a ela, os caracteres que formam a string apontada pelo segundo parâmetro são os delimitadores que terminam a string. Quando a string acabou, ou seja, não há mais palavras, ela devolve um ponteiro nulo. No código do Shell_simples.c, é usada para obter o comando passado.

      -fork():

      A função fork() cria um processo filho.

      -execvp():

      A função execvp() fornece um vetor de ponteiros para strings não-nulas que representam a lista de argumentos para o programa executado.

      -wait():

      A função wait() suspende a execução de um processo até que um dos seus filhos termine.

      As outras funções dessa família são:

      Waitpid(), que suspende a execução até que o processo especificado mude de estado. Por padrão, essa chamada espera pelo término do processo filho, mas este comportamento pode ser modificado pelo argumento *options*,

      Waitid(), que permite controle maior sobre mudanças de estado dos processos filhos.

   b. **A função gets() apresenta um warning por estar em desuso, qual o motivo? Como resolver a questão?**

      A função gets() é usada para capturar uma string, mas, se o usuário digitar mais caracteres que o previsto, ela pega esses caracteres em excesso, o que pode causar estouro de memória. Pode ser resolvido usando o fgets().

   c. **Como funciona a passagem de parâmetros em linha de comando?**

      Os parâmetros são passados através da variável lc, que é lida usando o gets(). Em seguida, usa-se o argv[] e a função strtok() (explicada acima), para fazer a leitura de cada um dos parâmetros.

2. **Considerando o código pipe_04.c:**
   a. **Entendendo o conceito:**
      - https://ryanstutorials.net/linuxtutorial/piping.php
      - http://www.linfo.org/pipes.html
      - https://www.geeksforgeeks.org/pipe-system-call/

   b. **Descreva o uso/funcionamento das seguintes funções:**

      a) pipe():

Estabelece a comunicação entre dois processos, um de um lado e outro do outro lado do pipe. Um processo escreve o dado de um lado, e outro processo lê o dado do outro lado, ou seja, uma comunicação direta entre eles.

b) dup2():

As funções dessa família são dup(), dup2() e dup3(). Dup() retorna o escritor de arquivo de menor valor que está disponível. Dup2() faz newfd (argumento que corresponde ao novo descritor de arquivo) uma cópia de oldfd (descritor de arquivo que será duplicado).
Dup3() funciona da mesma forma que dup2(), mas pode-se especificar o flac O_CLOEXEC para o novo descritor. Se oldfd e newfd possuem o mesmo valor, a chamada falha com erro EINVAL.

c) execl():

Pode ser vista como uma lista de argumentos arg0, arg1, ..., argn, passadas para um programa em linha de comando. Também especifica o ambiente do processo executado após o ponteiro NULL da lista de parâmetros ou o ponteiropara o vetor argv com parâmetro adicional. Este parâmetro adicional é um vetor de ponteiros para strings não-nulas que deve também ser finalizado por um ponteiro NULL. As outras funções consideram o ambiente para o novo processo como sendo igual ao do processo atualmente em execução.
As funções dessa família são de diferentes grupos:

- **L**: lista de argumentos [*execl()*, *execle()* e *execlp()*]. Os argumentos que serão recebidos em *argv* são listados um a um como parâmetros da função em forma de *string*.
- **V**: vetor de argumentos [*execv()*, *execve()* e *execvp()*]. Os argumentos que serão recebidos em *argv* são passados em um vetor do tipo char* que já contém todas as *strings* previamente carregadas.
- **E**: variáveis de ambiente [*execle()* e *execve()*]. O último parâmetro destas funções é um vetor para strings (char *) que será recebido pelo novo programa no argumento *envp* contendo variáveis de ambiente pertinentes para sua execução. Para as versões sem a letra "*e*", o ambiente é adquirido a partir de uma variável externa (*extern char **environ*) já declarada na biblioteca *unistd.h*.
- **P**: utilização da variável de ambiente PATH [*execlp()* e *execvp()*]. Esta função irá buscar a nova imagem do processo nos diretórios contidos na variável PATH. Para as versões sem a letra "*p*", deverá ser passada uma *string* contendo o caminho completo para o arquivo executável.

d) wait():

Descrito no anterior.

3. **Considerando o código do programa processo_exit_wait.c:**
   a. **Descreva o uso/funcionamento das seguintes funções:**
      a) exit():A função exit da biblioteca stdlib interrompe a execução do programa e fecha todos os arquivos que o programa tenha

porventura aberto. Se o argumento da função for 0, o sistema operacional é informado de que o programa terminou com sucesso; caso contrário, o sistema operacional é informado de que o programa terminou de maneira excepcional.

b) WIFEXITED():

Retorna true se o processo filho terminou normalmente.

c) WEXITSTATUS():

Retorna o status de saída do processo filho.

4. **Considerando o código do processo_n_pipe.c:**
   a. **Descreva o uso/funcionamento das seguintes funções:**
      a. **Read():**
         A função **read** realiza a leitura de dados do arquivo para a memória. Deve-se informar de qual descritor devem ser lidos os bytes. O descritor deve estar aberto com opção **O_RDONLY** ou **O_RDWR**. Os dados serão lidos a partir da posição corrente.
      b. **Write():**
         A função **write** grava no arquivo indicado pelo descritor as informações obtidas do endereço fornecido. Os dados serão gravados a partir da posição atual do arquivo. Caso a opção **O_APPEND** tenha sido especificada na abertura, a posição atual do arquivo será antes atualizada com o valor do tamanho do arquivo. Após a gravação a posição atual do arquivo será somada da quantidade de bytes gravados no arquivo.
      c. **Getpid():**
         Retorna o PID do processo.
         Outras funções da família são:
         Getppid(), que retorna o pid od processo pai,
         Getpdig(), que retorna o id do grupo do processo especificado por pid.