Processos em Unix

Conteúdo:

- Definição de Processos em Unix
- Estrutura, tipo e escalonamento de processos em Unix
- Processos, Pai, Filho e Zumbi.

O que é um Processo em Unix?

- É um espaço de endereçamento único, com uma única linha de controle.
- É um ambiente de execução que consiste de:
  - -01 segmento de instruções
  - -02 segmentos de dados (dados e pilha)
- É essencialmente um programação em execução.

Identificador de Processo:

- Process Identifier : PID
  - Número único para cada processo.
  - Número inteiro entre 2 - 32.0000

Tabela de Processo:

- Estrutura de dados descrevendo todos os processos que estão correntemente carregados.
- comando ps

```
?> ps
?> PID   TTY  STAT   TIME  COMMAND

?> man ps

?> ps -ax
?> ps -aux
?> ps -auf
```

- processo init (número 1) é o primeiro processo a ser criado:
  - É o ancestral de todos os outros.

Disparando um Processo:

```
#include <stdlib.h>
int system (const char *string);  // string: comando shell
```

>>> Execute o programa sistema.cpp

```
// Programa sistema.cpp
#include <stdlib.h>
```

```
#include <stdio.h>

int main()
{
  printf("Programa que executa o comando ps \n");
  system("ps -ax");
  printf("Fim de programa\n");
  exit(0);
}
```

>>> Mude o programa sistema.cpp para que ele execute o seguinte comando:
ps -ax &
>>> Escreva um programa que execeute o comando:
echo Meu login eh: $LOGNAME

Duplicando um Processo
- Primitiva fork()

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

pid_t  fork(void);  // cria um processo filho
                                    // pid_t é um inteiro
```

Valor de retorno: 0 -> para processo filho
PID do Filho -> para o processo Pai
-1 -> em caso de erro

Pai e Filho:
- compartilham o mesmo código
- utilizam cópias dos dados
- compartilham os descritores

Estrutura Geral de Programa

```
...
pid_t new_pid;

new_pid = fork();
...
switch (new_pid)
 {
```

```
  case -1: // erro
  ...
  break;
  case 0 :  // processo filho
  ...
  break;
  default:  // processo pai
  ...
  break;
 }
...
}
```

Execute e analise os seguintes programas:
>>> Programa fork1.cpp
>>> Programa fork2.ccp
>>> Programa fork3.ccp

Identificando Pai, Filho, Grupo
  - funções para identificação de processos

```
#include <unistd.h>
pid_t getpid();                    // retorna o PID do processo
pid_t getppid();                   // retorna o PID do processo Pai
int setpgid(pid_t pid, pid_t pgid);    // seta o valor do ID do grupo
                                   // (pgid) para o processo identificado
                                                                  // pid
pid_ getpgid( pid_t pid);          // retorna o ID do grupo
```

Valor de Retorno: 0 -> sucesso na execução
                  -1 -> erro na execução

>>> Execute o seguinte programa

```
// Programa quem_sou.cpp
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
```

```
  printf( "Eu sou o processo %d de pai %d e de grupo %d \n",
          getpid(), getppid(), getpgid(getpid()) );
  printf("Programa que executa o comando ps \n");
  system("ps -ax");
  printf("Fim de programa\n");
  exit(0);
}
```

– Execute o programa e depois o comando ps

- Escreva um programa que tenha dois processos. O Pai escreve 03 vezes a mensagem: "Eu sou o Processo Pai e meu PID é .... e o PID de meu Filho é ....
O Filho, por sua vez, escreve 5 vezes a mensagem: "Eu sou o Processo Filho, com PID ..., e meu PAI tem PID ...."

– Escreva um programa com Avô, Pai e Filho.
– Escreva um programa com um Pai e 02 Filhos.

Esperando por um Processo

- Primitiva wait(): o Pai só morre após a morte de um Filho
  – o Pai suspende sua execução e só retorna quando recebe um sinal vindo de um de seus filho indicando a morte deste.
  – E se o Processo Filho já estiver morto?

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t  wait(int  *stat_loc);  // o Pai espera pela morte ou parada um Filho específico

pid_do_processo_morto  wait (tipo_de_morte);
```

Execute e analise os seguinte programas:
>>> Programa wait1.cpp
>>> Programa wait2.cpp

Processo Zumbis

- Quando o Filho morre antes do Pai, ou quando o Pai não está esperando (wait), ele se torna um Processo Zumbi.

- Mude o programa fork1.cpp
  - faça o Filho terminar antes do Pai.
  - execute o comando ps antes do Pai terminar e tente matar o Filho com o comando kill <pid>.
  - execute o comando ps após a morte do Pai.

>>> Execute e analise o programa zumbi.cpp

- Escreva um programa familia.cpp que faça o seguinte:
  - O Pai cria 03 Filhos (zezinho, huguinho e luizinho);
  - O Pai tem os filhos, respectivamente com 20, 25 e 30 anos.
  - O Pai diz quantos filhos tem e o PID de caso um;
  - Uma maldição de família faz com que todos os filhos homens morram ao completarem 60 anos.
  - Todo ano, após a nascimento do primeiro Filho, é dito a situação de todos os membros da família (idade, vivo ou morto).

- Comando kill
> kill <pid> <=== matará o processo cujo identificador é pid
>kill -9 <pid> <== uma forma radical de matar um processo.

- Função exit( ) - Quando um processo executa esta função, todos os seus processos filhos são herdados pelo processo init (pid 1).
  - exit(0) – término com sucesso
  - exit(-1 || 1) – término com erro