HandsOn #2: Fork

Departamento de Computação/UFC – Sistemas Operacionais
José Macedo

Objetivo

- Entender a criação de processos filhos e controle de processos utilizando fork

Tudo isso está sendo executado através de um PROCESSO

Todos vocês já usaram...

Processo

- Process identifier (pid)
  - Identificador único de um processo
  - Nada mais é que um número sequencial.
  - É assim que o sistema operacional vê o processo: como um número
  - Pode ser entendido por qualquer linguagem de programação
- Parents process (ppid)
  - Todo processo rodando no seu sistema possui um pai
  - O processo pai é o processo que invocou outro determinado processo.

Qual a diferença entre User Space e Kernel?

Kernel

- É a camada que está em cima do hardware, ou seja, é o intermediário entre toda interação que acontece entre a user space e o hardware
- Essas interações incluem coisas como:
  - Ler/Escrever no diretório de arquivos
  - Enviar arquivos pela rede
  - Alocar memória
  - Tocar música através dos auto-falantes

Devido ao seu poder, nenhum programa possui acesso direto ao Kernel.

User Space

- É onde todos os seus programas são executados
- Seu programa pode fazer muita coisa sem precisar do kernel:
  - Realizar operações matemáticas
  - Realizar operações com strings
  - Controle de fluxo através de operações lógicas

Mas se você quiser fazer várias coisas legais terá que utilizar o kernel.

System call

- Qualquer comunicação entre o kernel e user space é feita através de system calls
- É a interface que conecta o kernel e a user space
- Define as interações que são permitidas entre o seu programa e o hardware

Fork

```
$ man fork
```

- É uma chamada de sistema que cria uma cópia do processo atual, duplicando código e valores de variáveis.
- Diferenças entre o processo criador (também chamado de pai) e o processo criado (chamado de filho) são:
  - o novo processo terá um PID (Process IDentification) diferente do processo criador;
  - O retorno da chamada fork() é:
    - o PID do processo-filho para o processo-pai;
    - 0 para o processo-filho;
- E, para ambos os processos, a execução continuará na instrução seguinte à chamada fork().

Desafio 1

- O que seria impresso depois do código abaixo?

```
p = fork();
if (p == 0)
   printf("FILHO\n");
else
   printf("PAI\n");
```

Desafio 2

- O que seria impresso depois do código abaixo?

```
fork();
fork();
fork();
printf("x");
```

Tutorial

- Este tutorial é baseado na seguinte fonte:
  http://www.yolinux.com/TUTORIALS/ForkExecProcesses.html

- Os fontes desta aula estão disponíveis em:

  https://github.com/josemacedo/sistemas-operacionais/tree/master/1_Lab_Fork

Prática

- http://www.yolinux.com/TUTORIALS/ForkExecProcesses.html

```
... globalVariable = 2;

main()
{
  string sIdentifier;
  int    iStackVariable = 20;

  pid_t pID = fork();
  if (pID == 0)                // child
  {
    // Code only executed by child process

    sIdentifier = "Child Process: ";
    globalVariable++;
    iStackVariable++;
  }
  else                         // parent
  {
    sIdentifier = "Parent Process:";
  }
  cout << sIdentifier;
  cout << " Global variable: " << globalVariable;
  cout << " Stack variable: "  << iStackVariable << endl;
}
```

```
$ g++ -o ForkTest ForkTest.cpp
$ ./ForkTest
```

Parent Process: Global variable: 2 Stack variable: 20
Child Process: Global variable: 3 Stack variable: 21

Vfork()

- Assim como o fork(), cria uma cópia do processo atual.
- Diferenças:
  - Não faz cópia da tabela de página do processo pai
  - A memória é compartilhada
  - A função vfork () executa o processo filho primeiro e retoma o processo pai quando o filho termina.

Prática

```
 globalVariable = 2;

main()
{
   string sIdentifier;
   int    iStackVariable = 20;

   pid_t pID = vfork();
   if (pID == 0)                // child
   {
      // Code only executed by child process

      sIdentifier = "Child Process: ";
      globalVariable++;
      iStackVariable++;
    }
    else                              // parent
    {
      sIdentifier = "Parent Process:";
    }
    cout << sIdentifier;
    cout << " Global variable: " << globalVariable;
    cout << " Stack variable: "  << iStackVariable << endl;
    exit(0);
}
```

```
$ g++ -o VForkTest VForkTest.cpp
$ ./VForkTest
```

Child Process: Global variable: 3 Stack variable: 21 Parent Process: Global variable: 3 Stack variable: 21

Compile, Execute e Explique – Fork1.cpp

```
int main()
{
int pid ;
   printf("Eu sou o pai %d e eu vou criar um filho
\n",getpid()) ;
   pid=fork() ; /* criacao do filho */
   if(pid==-1) /* erro */
   {
      perror("impossivel de criar um filho\n") ;
   }
   else if(pid==0) /* acoes do filho */
   {
      printf("\tOi, eu sou o processo %d, o filho\n",getpid()) ;
      printf("\tO dia esta otimo hoje, nao acha?\n") ;
      printf("\tBom, desse jeito vou acabar me instalando para
sempre\n");
      printf("\tOu melhor, assim espero!\n") ;
      for(;;) ; /* o filho se bloqueia num loop infinito */
   }
   else /* acoes do pai */
   {
       sleep(1) ; /* para separar bem as saidas do pai e do
filho */
       printf("As luzes comecaram a se apagar para mim,
%d\n",getpid()) ;
       printf("Minha hora chegou : adeus, %d, meu
filho\n",pid) ;
       /* e o pai morre de causas naturais */
    }
    exit(0);
}
```

O que acontece?

```
$ ps
```

Compile, Execute e Explique – Fork2.cpp

```
....
int x = 2;
int main()
{
   string id;
   int    Y  = 20;

   pid_t pID = fork();
   pID = fork();
   if (pID == 0)                // child
   {
      id  = "A";
      x++;
      Y++;
    }
    else                                // parent
    {
      id = "B";
    }

    cout << id;
    cout << " X: " << x;
    cout << " Y: "  << Y << endl;
}
```

O que acontece?

INSIGHT

*** FIM ***
Obrigado
Laboratório #1: Fork

DEPARTAMENTO DE COMPUTAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ