Processos – Criação e comunicação

Mário Olímpio de Menezes

Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática

Mackenzie 2013

Identificação de Processos

Process ID

O Unix/Linux identifica os processos por um valor inteiro único chamado *ID do processo*. Cada processo tem também um *ID do processo pai*, que é inicialmente o ID do processo que o criou. Se este *processo pai* termina, o processo é adotado por um processo do sistema de modo que o ID do processo pai sempre identifica um processo válido.
As funções `getpid` e `getppid` retornam o ID do processo e o ID do processo pai, respectivamente.

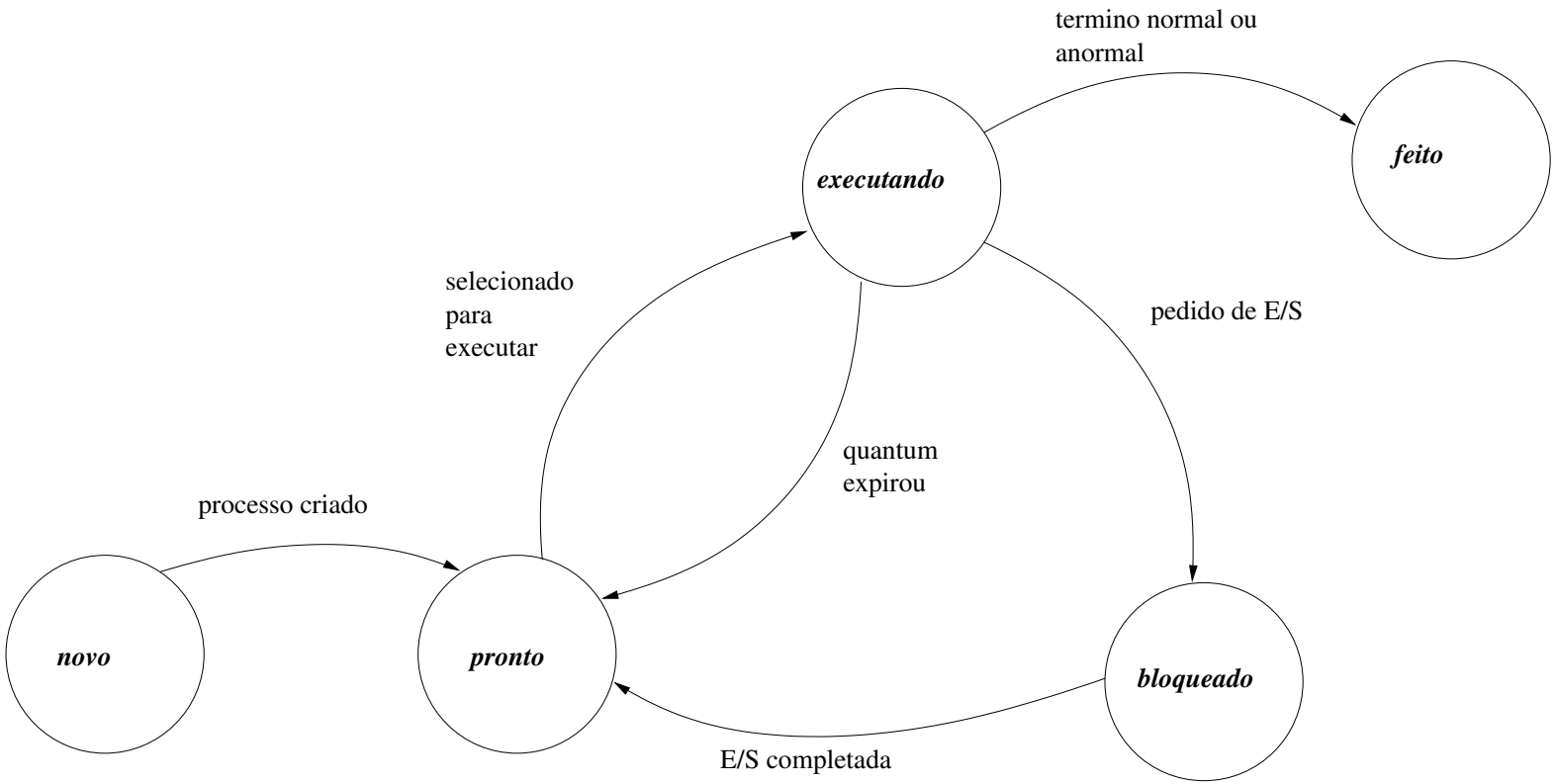
```
#include <stdio.h>
#include <unistd.h>

int main (void) {
   printf("I am process %ld\n", (long)getpid());
   printf("My parent is %ld\n", (long)getppid());
   return 0;
}
```

Estados de um Processo

Estado	Significado
novo	sendo criado
executando	instruções estão sendo processadas
bloqueado	esperando por um evento tal como E/S
pronto	esperando para ser colocado em um processador
feito	terminado

Estados de um Processo



Examinando o estado dos processos

Podemos examinar os processos em execução e visualizar seus estados utilizando o comando `ps -a`
Executando `ps -la` temos uma saída mais completa com várias informações adicionais sobre os processos.

```
$ ps -la
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000  9675  3931  0  76   0 -  1208 -      tty1     00:00:00 bash
0 R  1000 11468  9525  0  76   0 -   849 -      pts/0    00:00:00 ps
```

Campos reportados pelo comando **ps**

cabeçalho	opção	significado
F	-l	flags (octal e aditivo) associados com o processo
S	-l	estado do processo
UID	-f,-l	ID de usuário do proprietário do processo
PID	(todos)	ID do processo
PPID	-f,-l	ID do processo pai
C	-f,-l	utilização de processador usada para escalonamento
PRI	-l	prioridade do processo
NI	-l	valor nice
ADDR	-l	endereço de memória do processo
SZ	-l	tamanho em blocos da imagem do processo
WCHAN	-l	evento no qual o processo está esperando
TTY	(todos)	terminal que está controlando
TIME	(todos)	tempo de execução cumulativo
CMD	(todos)	nome do comando (argumentos com a opção -f)

Tabela: Campos reportados para as várias opções do comando ps na Extensão POSIX:XSI.

Criação de Processos

Um processo pode ser criado chamando-se `fork`
O processo chamador se torna o *pai* e o processo criado é chamado de *filho*

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

int main(void) {
   int x;

   x = 0;
   fork();
   x = 1;
   printf("I am process %ld and my x is %d\n", (long)getpid(), x);
   return 0;
}
```

fork - pai e filho

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

int main(void) {
   pid_t childpid;

   childpid = fork();
   if (childpid == -1) {
      perror("Failed to fork");
      return 1;
   }
   if (childpid == 0)                                  /* child code */
      printf("I am child %ld\n", (long)getpid());
   else                                               /* parent code */
      printf("I am parent %ld\n", (long)getpid());
   return 0;
}
```

Pegando o Process ID

O que acontece quando o seguinte programa executa?

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

int main(void) {
   pid_t childpid;
   pid_t mypid;

   mypid = getpid();
   childpid = fork();
   if (childpid == -1) {
      perror("Failed to fork");
      return 1;
   }
   if (childpid == 0)                                    /* child code */
      printf("I am child %ld, ID = %ld\n", (long)getpid(), (long)mypid);
   else                                                 /* parent code */
      printf("I am parent %ld, ID = %ld\n", (long)getpid(), (long)mypid);
   return 0;
}
```

Criando uma cadeia de processos

Abaixo está o código de um programa que cria uma cadeia de n processos, onde n é um argumento de linha de comando

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

int main (int argc, char *argv[]) {
   pid_t childpid = 0;
   int i, n;
   if (argc != 2){   /* check for valid number of command-line arguments */
      fprintf(stderr, "Usage: %s processes\n", argv[0]);
      return 1;
   }
   n = atoi(argv[1]);
   for (i = 1; i < n; i++)
      if (childpid = fork())
         break;

   fprintf(stderr, "i:%d  process ID:%ld  parent ID:%ld  child ID:%ld\n",
           i, (long)getpid(), (long)getppid(), (long)childpid);
   return 0;
}
```

Exercícios

Alguns exercícios com processos

1. Rode o programa anterior para valores grandes de `n`. As mensagens sempre estarão ordenadas pelo valor de `i`?
2. O que acontece se o programa anterior escreve-se as mensagens para `sys.stdout`, usando `print`, ao invés de para `sys.stderr`?