Recursos do Sistema Linux

Sistemas Embarcados

Conteúdo

1. Acesso a arquivos
2. **Processos e Sinais**
3. Threads POSIX e biblioteca *pthreads*
4. Comunicação e Sincronismo entre Processos 5. Programação usando *sockets*
6. Device Drivers

Processo

- Um dos principais conceitos dentro de um Sistema Operacional
- ***Um processo é um Programa em execução***
- **Programa**: código em disco (***passivo***)
- **Processo**: código sendo executado (***ativo***)

Processo

- O Sistema Operacional armazena as informações dos processos na ***tabela de processos.***
- Cada processo possui um **pid** (process id) - identificador único.
- Durante a execução, o processo compartilha o processador com outros processos em execução (escalonamento de processador).
- Um processo interage com outros processos através de mecanismos de comunicação.

Processo

Ambiente	- espaço de endereçamento (memória) - processo pai / filho - proprietário - arquivos abertos - sinais - estatísticas de uso
Execução	- contador de programa (***PC - program counter***) - apontador de pilha (***SP - stack pointer***) - registradores - estado (execução)

Processo

- Identificação do PID e PPID (Ex: Processo_1)

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

int main ()
{
    printf ("O identificador do processo – PID é: %d\n",
(int) getpid ());
    printf ("O identificador do processo pai – PPID é: %d
\n", (int) getppid());
    return 0;
}
```

Processo

- **UID - *User ID* / GID - *Group ID***
  - Cada processo precisa de um proprietário para que o SO possa definir as permissões de execução.
  - No UNIX isso é dado tanto pelo UID (User ID) quanto pelo GID (Group ID).

```
> ps -af

> ps -aux

> man ps
```

Tabela de Processos

- Todas as informações sobre um processo são mantidas na tabela de processos.
- A tabela de processos contem campos que dizem respeito à gerência do processo, à gerência da memória e à gerência de arquivos.
- A tabela de processos possui uma entrada por processo e os campos nela contidos variam de sistema operacional para sistema operacional.

Tabela de Processos

- Dados referentes ao **processo** na tabela de processos:
  - identificador do processo (pid), valor dos registradores, valor do contador de programa (PC), valor da palavra de estado (PSW), valor do apontador de pilha (SP), estado do processo, instante do início do processo, tempo de processador utilizado, etc.

Tabela de Processos

- Dados referentes à **memória** na tabela de processos:
  - endereço do segmento de texto, dados e pilha, estado da saída, informações sobre proteção, etc
- Dados referentes à **gerência de arquivos** na tabela de processos:
  - diretório raiz, diretório de trabalho, descritores de arquivos abertos, parâmetros de chamadas em andamento, etc

Gerenciamento de Processos (SO)

- O processo é composto tanto pelo ambiente como pela execução: Processo tradicional: ambiente + execução.
- Cada processo possui um único fluxo de controle (contador de programa) e roda de forma independente dos demais (Isolado).
- Como, em um dado instante, pode haver vários processos ativos ao mesmo tempo, o processador é chaveado entre os diversos processos.
- Por esta razão, não é possível prever o tempo de execução de um processo, pois este dependerá da carga do sistema.

Gerenciamento de Processos (SO)

P1

```
int main ()
{
    int x;
    int y;
    x = 2;
    y = 3;
    return x+y;
}
```

PC →

t1

P1

```
int main ()
{
    int x;
    int y;
    x = 2;
    y = 3;
    return x+y;
}
```

PC →

t2

P1

```
int main ()
{
    int x;
    int y;
    x = 2;
    y = 3;
    return x+y;
}
```

PC →

t3

Gerenciamento de Processos (SO)

P1

```
int main ()
{
    int x;
    int y;
    x = 2;
    y = 5;
    return x+y;
}
```

PC →

t1

P2

```
float main ()
{
    float a;
    float b;
    a = 21;
    b = 30;
    return x-y;
}
```

PC →

t1

P3

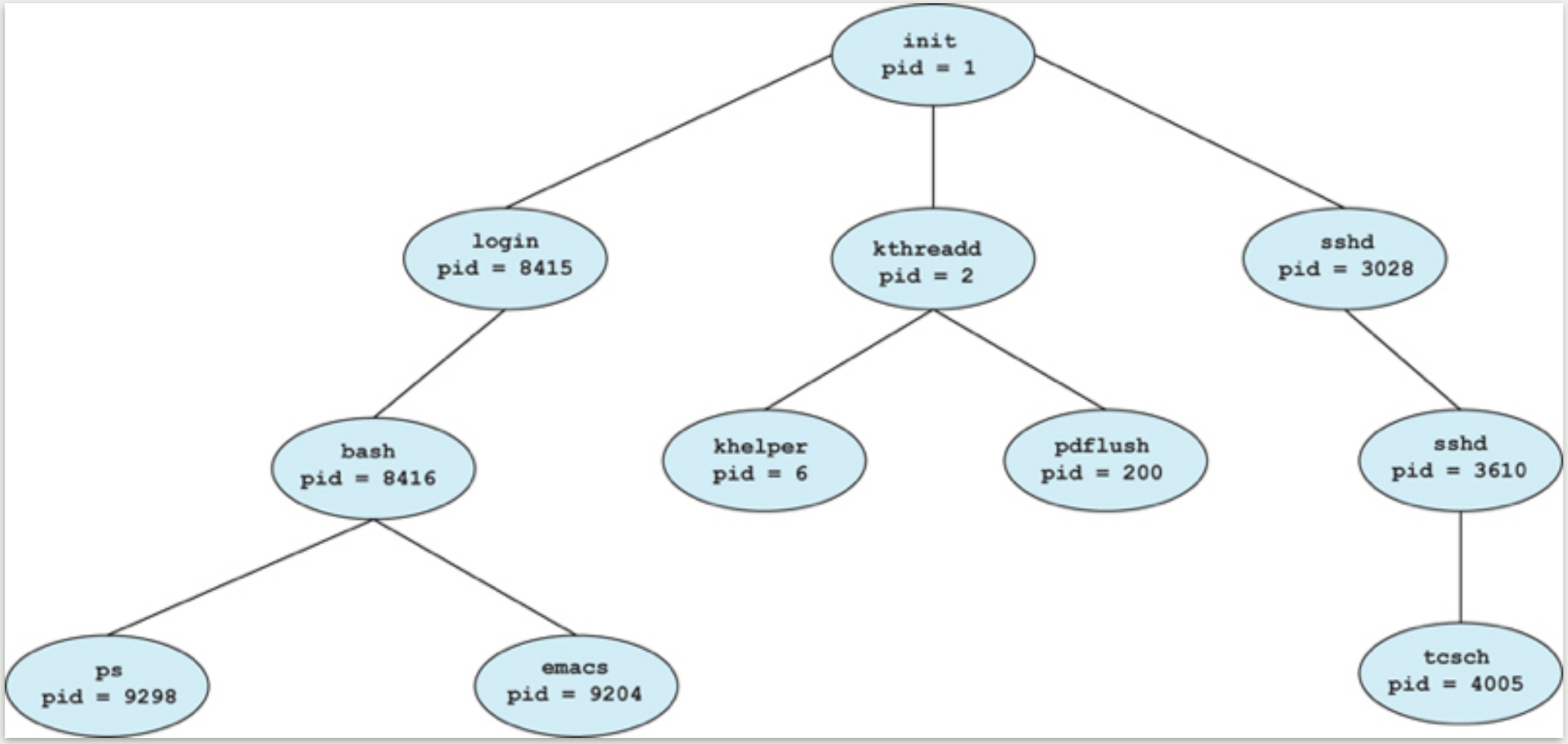
```
int main ()
{
    int m;
    int n;
    m = 5;
    n = 41;
    return x*y;
}
```

PC →

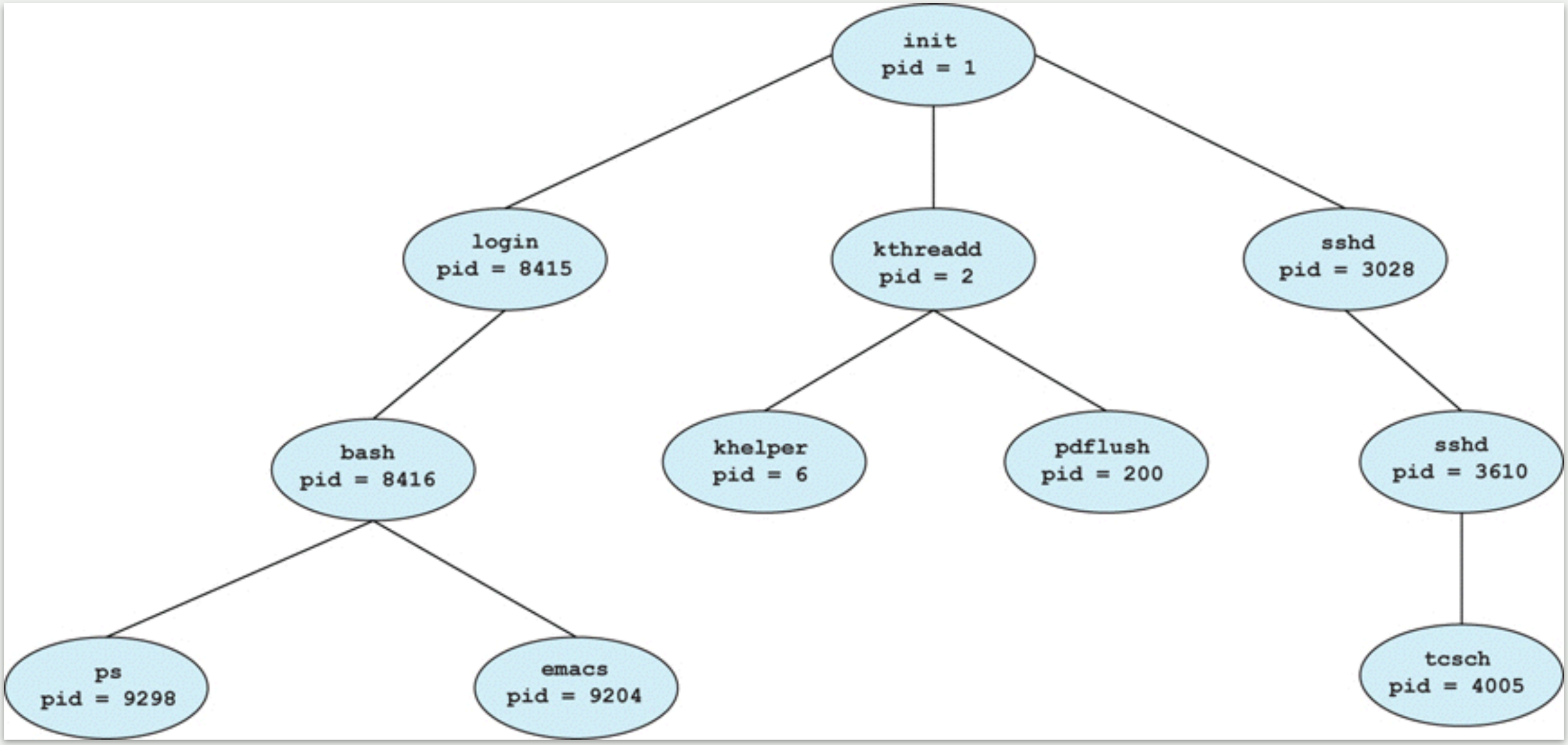
t1

Gerenciamento de Processos (SO)

- Todo sistema operacional possui mecanismos para criação de processos. Geralmente, um processo somente é criado por outro processo, o que nos leva a uma hierarquia em árvore.



Gerenciamento de Processos (SO)



Criação de Processos

- Um processo é criado a partir de:
  - um programa (criação tradicional) - MS-DOS:
    - ***create_process***(“teste.exe”);
  - outro processo (clonagem) (Unix):
    - ***system***();
    - ***fork***() e ***exec***();

Criação de Processos

- Utilizando o ***system***();
  - Função da Biblioteca stdlib.h
  - Permite executar um comando dentro de um programa, criando um sub-processo (processo filho)

Exemplo: Processo_2

```
#include <stdlib.h>
int main ()
{
    int retorna_valor;
    retorna_valor = system("ls -l /");
    return retorna_valor;
}
```

Criação de Processos

- Utilizando o ***system***();
  - A função ***system*** retorna em sua saída o status do comando no *shell*. Se o *shell* não puder ser executado, o ***system()*** retorna o valor 127; se um outro erro ocorre, a função retorna -1.
  - O uso do ***system()*** não é recomendado na maioria dos casos. É simples mas dá brechas a falhas de execução pois depende de muito do sistema.

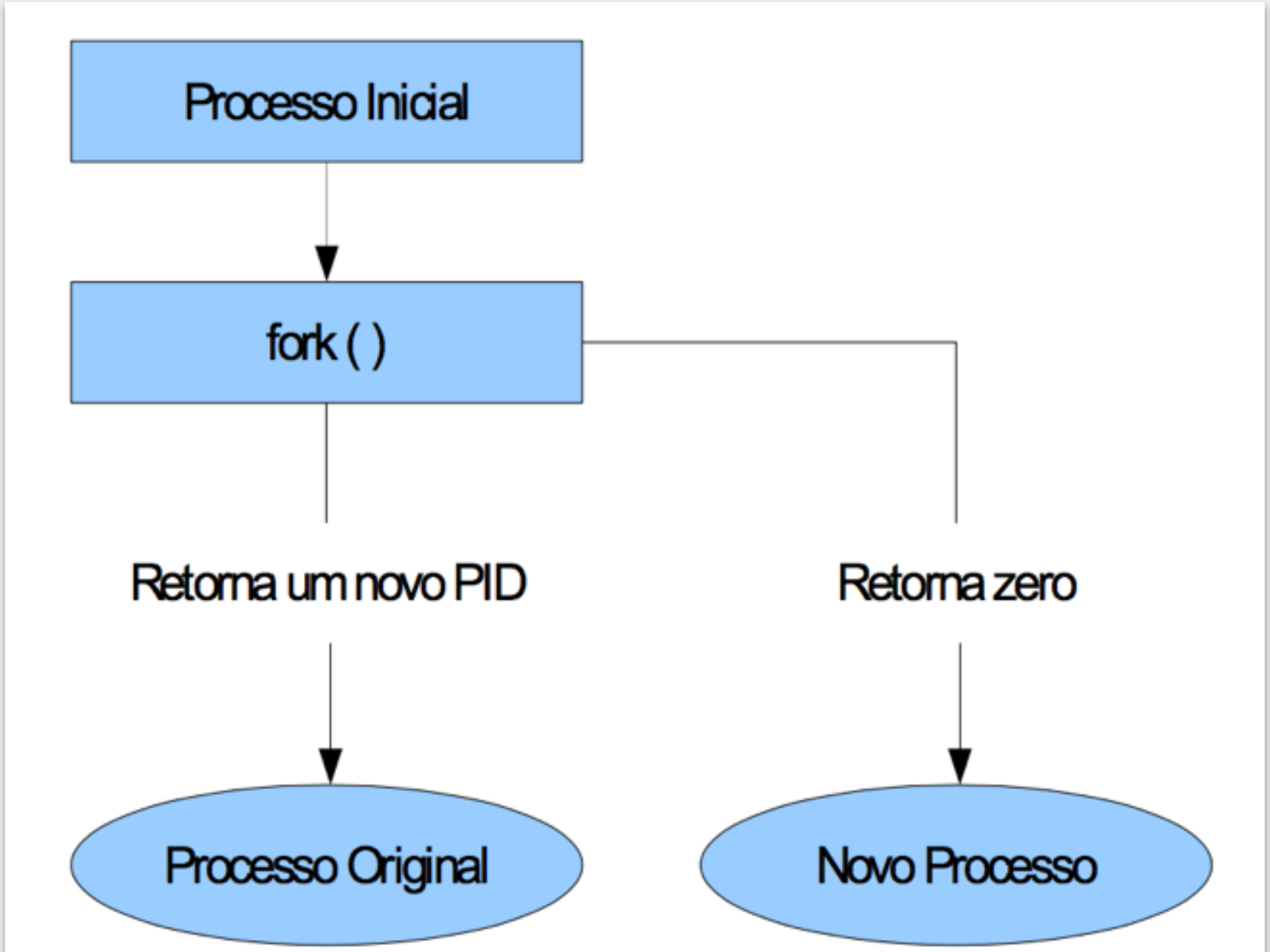
Criação de Processos

- Utilizando ***fork()*** e ***exec()***
  - No Unix não há meio para que um processo crie outro somente mandando executá-lo diretamente.
  - Não há função que crie e execute um novo processo em um único passo.
- Para isso, utiliza-se a função ***fork()*** que cria uma cópia do processo atual e em seguida a função ***exec()*** que substitui o conteúdo do novo processo por um novo programa.

Criação de Processos ***fork()***

- ***fork()*** -> uma duplicata do processo é criada - processo filho (*child process*)
- Em seguida, tanto o processo pai quanto o filho continuam a executar normalmente a partir do ***fork()*** (PC é igual)
- O processo filho tem um novo PID que pode ser verificado com o ***getpid***(). Entretanto, a função ***fork()*** retorna valores distintos. O valor de retorno no processo pai é o PID do processo filho, ou seja, retorna um novo PID. Já o valor do retorno do filho é zero.

Criação de Processos ***fork()***



Criação de Processos *fork()*

Exemplo: Processo_3

```
#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's process ID is %d\n", (int) child_pid);
    }
    else
        printf ("this is the child process, with id %d\n", (int)
getpid());
    return 0;
}
```

Criação de Processos ***exec()***

- ***exec()*** -> substitui o programa em execução de um processo por outro programa.
- Quando um programa chama a função ***exec***, o processo cessa imediatamente a execução do programa corrente e passa a executar um novo programa do início.
- ***exec()*** é na verdade uma família de funções que variam sutilmente na sua funcionalidade e também na maneira em que são chamados.
  - Funções que contém a letra '***p***' em seus nomes (***execvp*** e ***execlp***) aceitam que o nome ou procura do programa esteja no *current path*; funções que não possuem o '***p***' devem conter o caminho completo do programa a ser executado.

Criação de Processos ***exec()***

- Funções que contém a letra '***v***' em seus nomes (***execv***, ***execvp*** e ***execve***) aceitam que a lista de argumentos do novo programa seja nula. Funções que contém a letra '***l***' aceitam em sua lista de argumentos a utilização de mecanismos ***var args*** em linguagem C.
- Funções que contém a letra '***e***' em seus nomes (***execе*** e ***execle***) aceitam um argumento adicional.

Como a função ***exec*** substitui o programa em execução por um outro, ele não retorna valor algum, exceto quando um erro ocorre.

Criação de Processos ***exec()***

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

int spawn (char* program, char** arg_list) {
    pid_t child_pid;
    child_pid = fork ();
    if (child_pid != 0)
        return child_pid;
    else {
        execvp (program, arg_list);
        fprintf (stderr, "an error occurred in execvp\n");
        abort ();
    }
}
int main () {
    char* arg_list[] = {“ls”, "-l", "/", NULL};
    spawn ("ls", arg_list);
    printf ("done with main program\n");
    return 0;
}
```

Exemplo: Processo_4

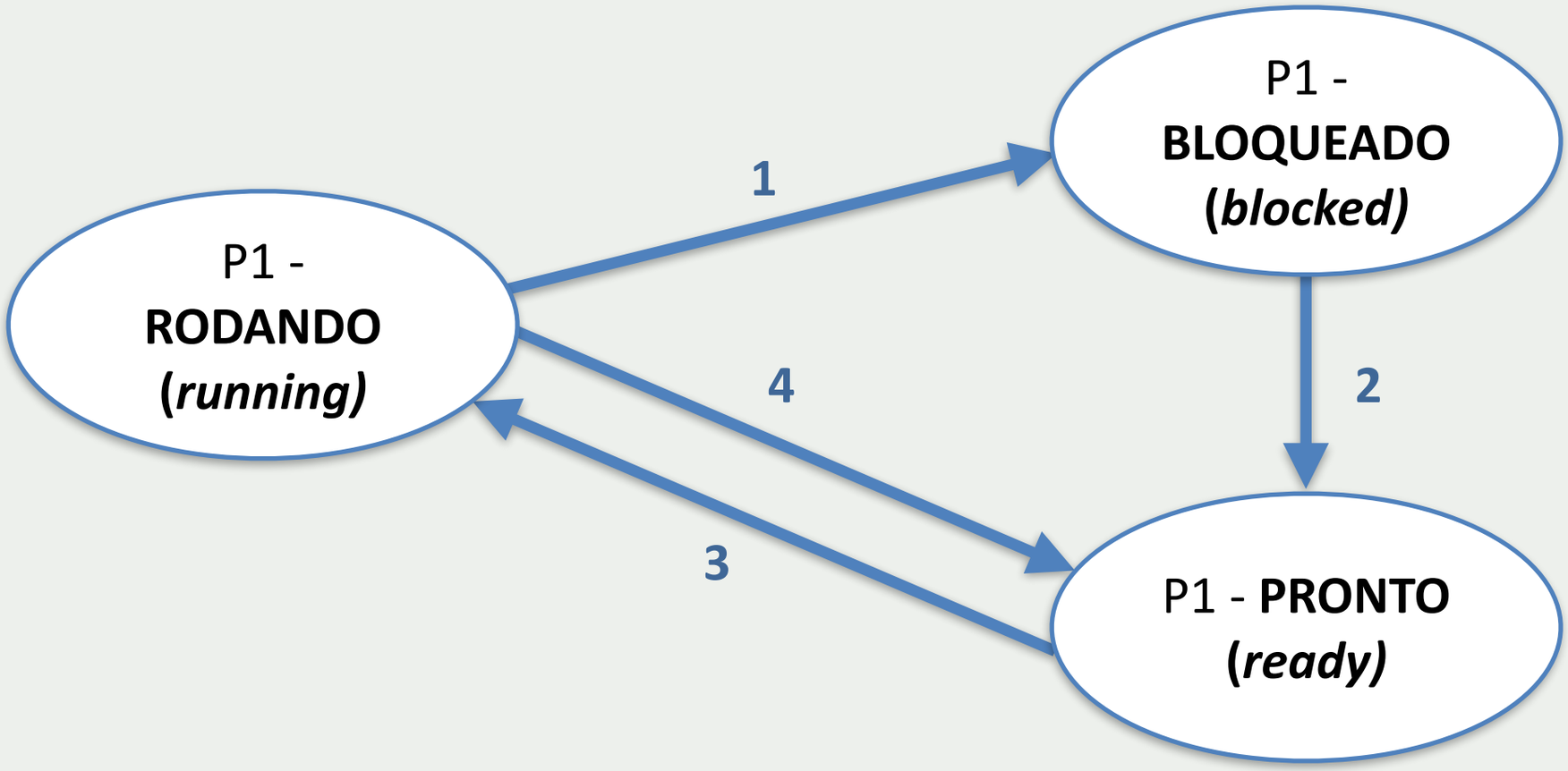
Estados de um Processos

- Apesar dos processos serem relativamente auto-suficientes, muitas vezes eles necessitam de acessar outros recursos (discos, terminais) ou mesmo se comunicar com outros processos.
- Quando um processo está ocioso esperando que um evento aconteça, nós dizemos que ele está bloqueado. Em algumas situações, o processo pode ser bloqueado a revelia pelo sistema operacional.

Estados de um Processos

- Os estados básicos de um processo são:
  - **rodando** (*running*),
  - **bloqueado** (*blocked*) e
  - **pronto** (*ready*).
- Em um sistema monoprocessado, só temos um único processo rodando a cada instante.
- Em um sistema multiprocessado (ex: *multicore*) é possível ter um número de processos igual ao número de núcleos de processamento.

Estados de um Processo



1. O processo que está rodando necessita de algum recurso de acesso não imediato e solicita seu bloqueio para aguardar um evento.
2. O evento aguardado ocorre e então o sistema operacional o coloca de volta na lista de processos pontos para execução.
3. O processo é selecionado para voltar à execução.
4. O sistema operacional decide remover o processo em execução pois sua janela de tempo em execução terminou.