Sistemas Operacionais: Processos

Prof. Dr. Rafael Lopes Gomes
Universidade Estadual do Ceará (UECE)

Agenda

- Processos
- Threads
- Comunicação entre processos
- Escalonamento de Processos

Processos

- Definição: Uma abstração de um programa em execução;
- Suporte a operações (pseudo) concorrentes;
  - Mesmo quando há uma CPU disponível;
  - CPU virtual;
- A CPU muda de um processo para o outro rapidamente, executando cada um periodicamente (nano/milisegundos);
  - Pseudoparalelismo (ilusão de paralelismo).

Modelo de Processo

- Processo é uma instância de um programa em execução, incluindo:
  - Valores atuais do contador;
  - Registradores;
  - Variáveis.
- Cada processo tem sua própria CPU virtual;
- CPU real troca constantemente de processo em processo (difícil prever taxa de computação);
- Multiprogramação: mecanismo de trocas rápidas.

Modelo de Processo

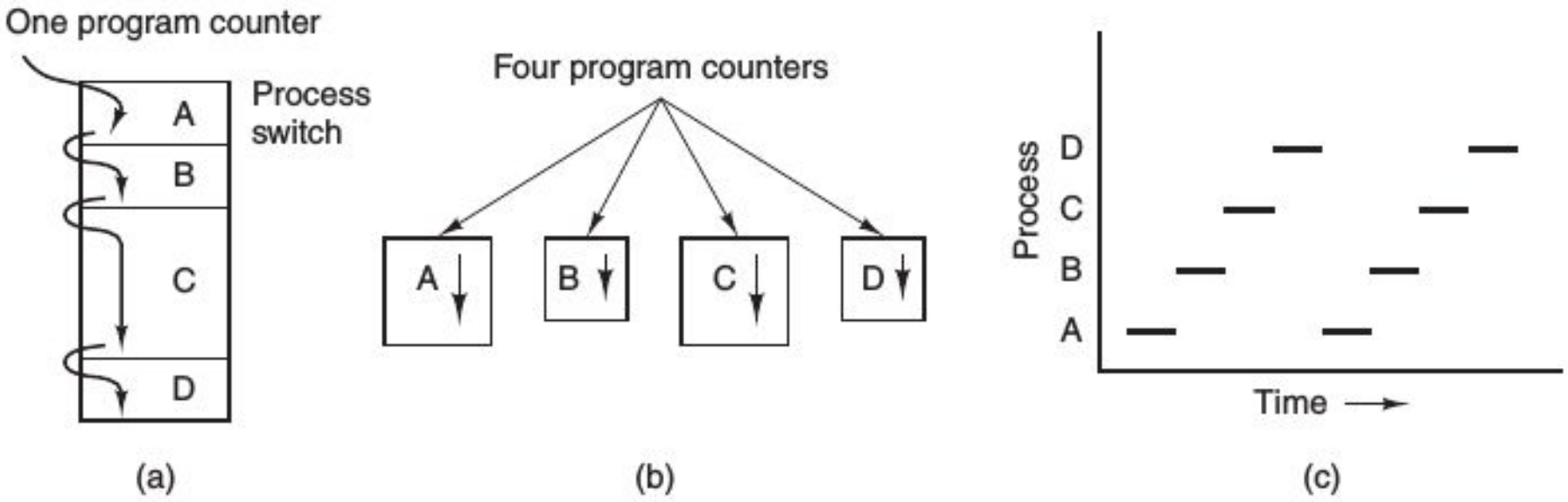


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Modelo de Processo

- Processo tem um programa, uma entrada, uma saída e um estado.
- Um único processador pode ser compartilhado entre vários processos
  - Algoritmo de escalonamento: determina quando parar o trabalho em um processo e servir outro.
- Um programa é algo que pode ser armazenado em disco sem fazer nada (texto/código).

Criação de Processos

- Processos são criados quando:
  - Inicialização do sistema
  - Execução de uma chamada de sistema de criação de processo por um processo em execução;
  - Solicitação de um usuário para criar um novo processo;
  - Início de uma tarefa em lote.

Criação de Processos

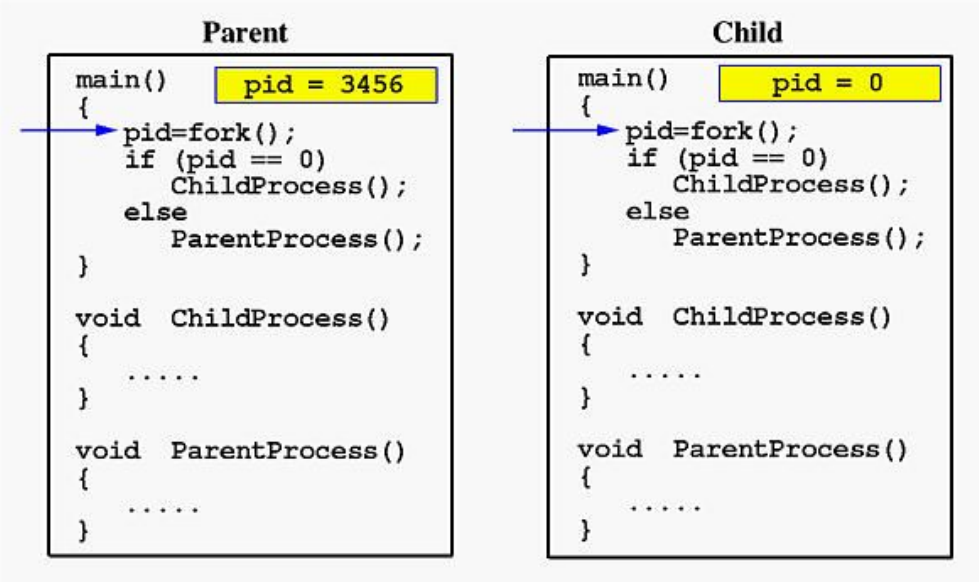
- Modos de operação de processos
  - 1º Plano: processos que interagem com os usuários e realizam trabalho para eles
  - 2º Plano (daemons): não estão associados com os usuários em particular, realizam funções específicas.

Criação de Processos

- Em todos os casos, um novo processo é criado por outro já existente executando uma chamada de sistema de criação de processo;
- Essa chamada de sistema diz ao SO para criar um novo processo e indica (direta ou indiretamente) qual programa executar nele.

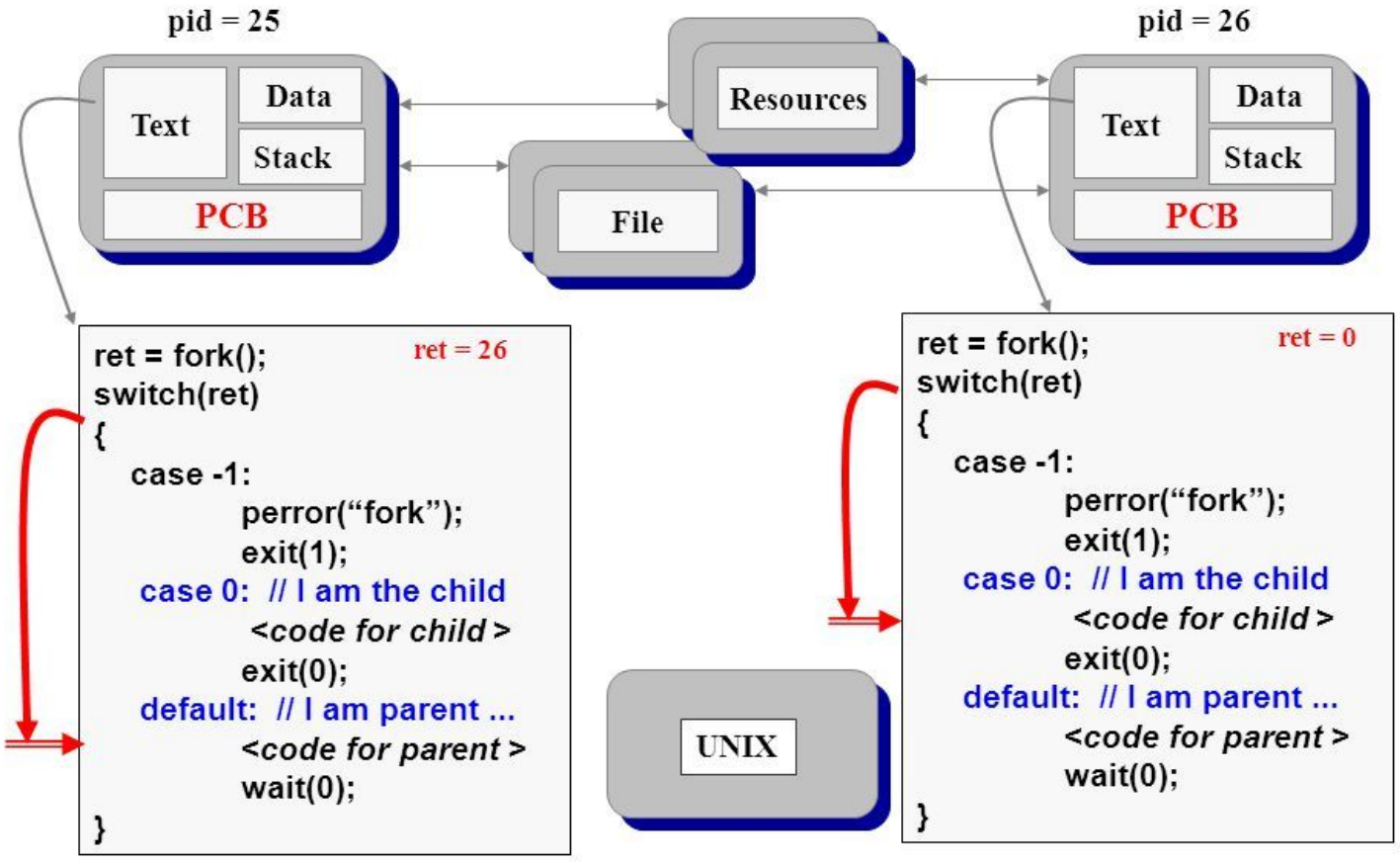
Criação de Processos

- Em UNIX a chamada ***fork*** cria um novo processo
  - Cria um clone exato do processo que chamou;
  - Dois processos (pai e filho) têm a mesma imagem de memória, variáveis de ambiente e arquivos abertos;
  - Pode ***execve*** para a imagem de memória e executar um novo programa.



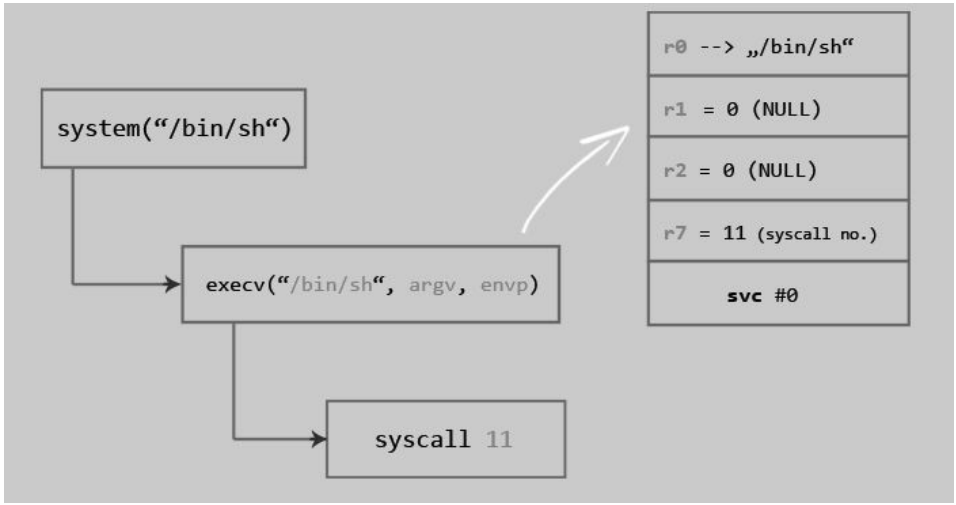
Criação de Processos

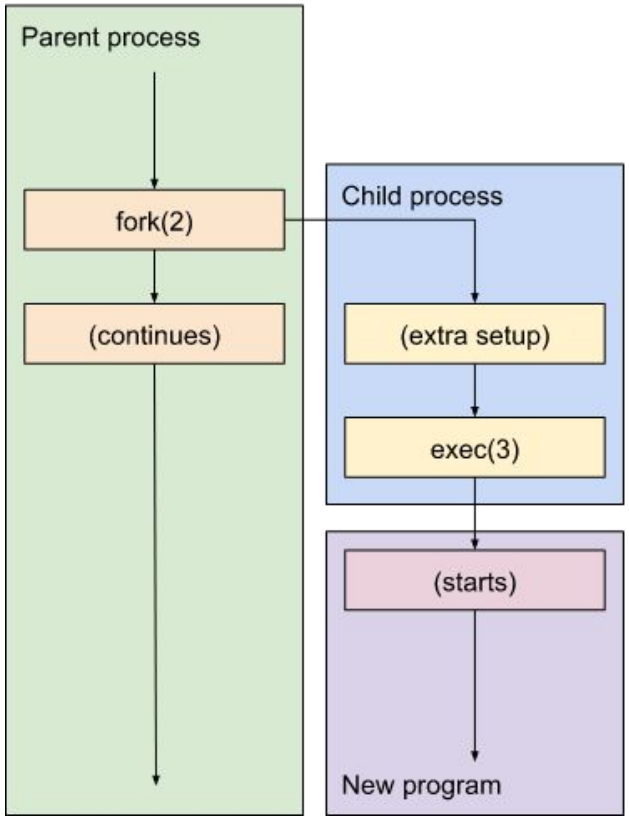
- Chamada ***fork***



Criação de Processos

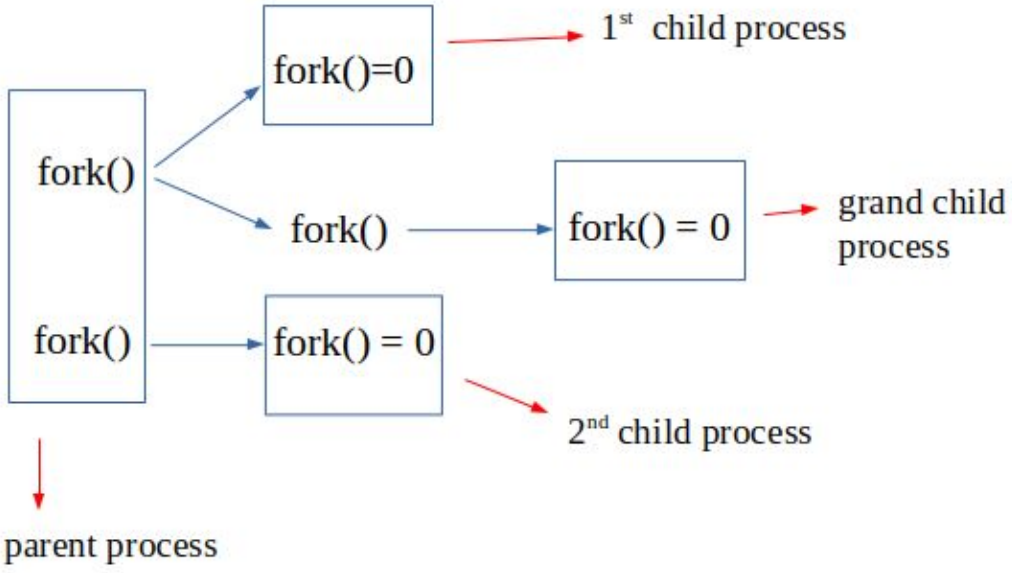
- Chamada ***execve***





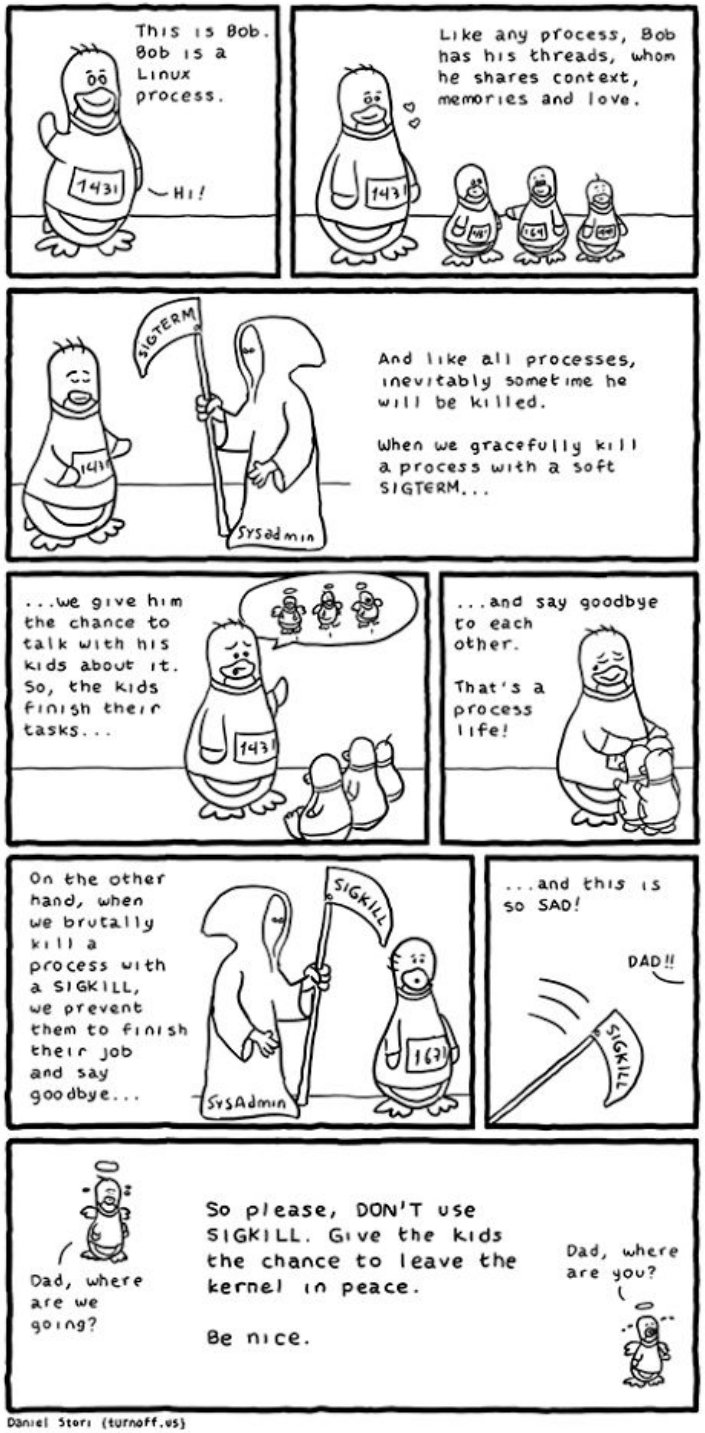
Criação de Processos

- Após um processo ser criado, pai e filho têm os seus próprios espaços de endereço distintos
  - Se um dos dois processos muda uma palavra no seu espaço de endereço, a mudança não é visível para o outro processo.



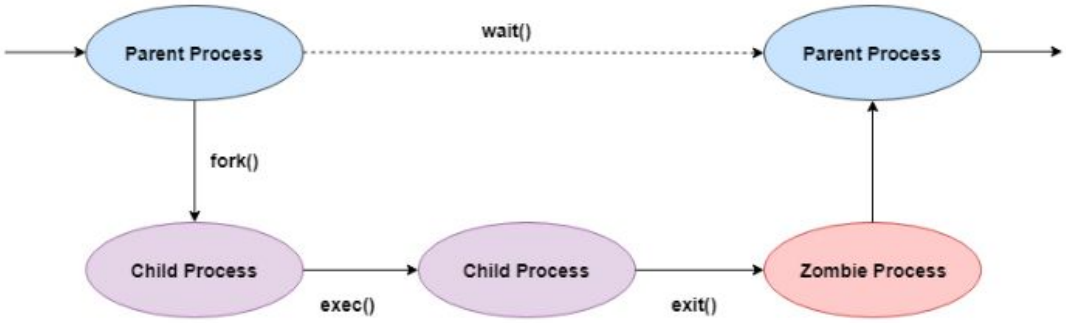
Término de Processos

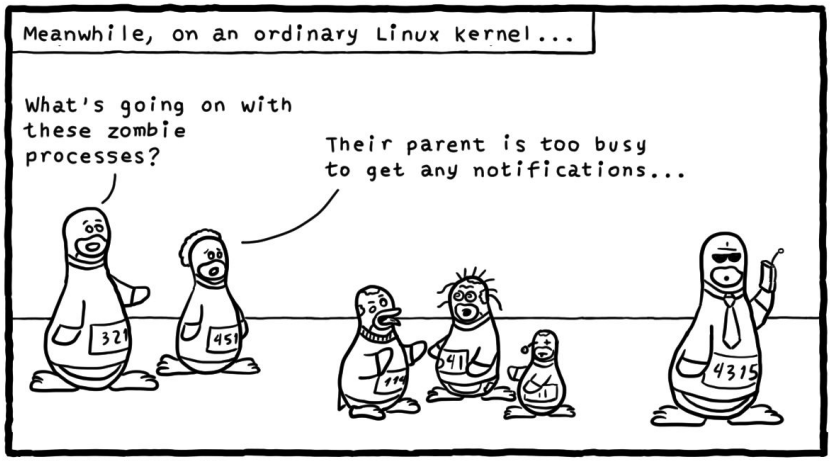
- As seguintes condições podem ocorrer:
  - Saída Normal (Voluntário)
  - Erro Fatal (Involuntário)
  - Saída por Erro (Voluntário)
  - Morto por outro processo (Involuntário)
    - Kill no UNIX



Término de Processos

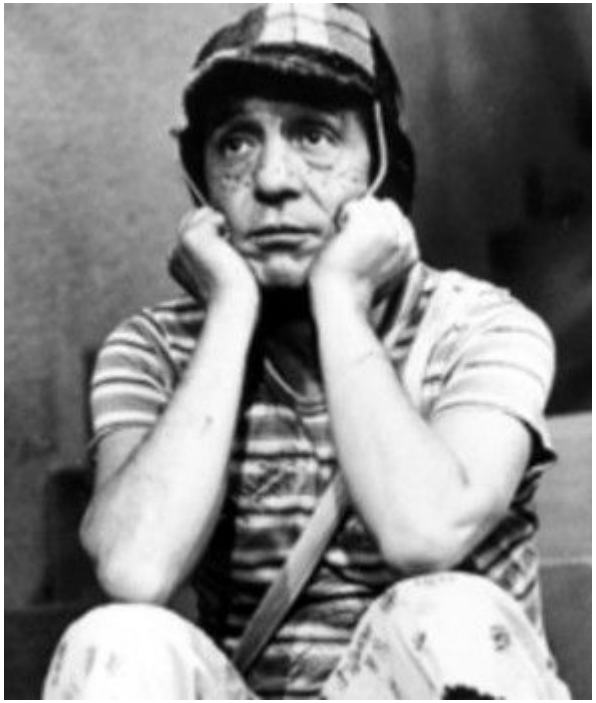
- Processo Zumbi (Zombie/Defunct)
  - Processo que completou sua execução (exit) mas ainda possui uma entrada na tabela de processos (consome recursos).
  - Ocorre quando o processo Pai não sabe de seu término.
    - Recursos não são liberados.
  - Encontrando Zumbis:
    - top ou ps aux | egrep "Z|defunct"





Término de Processos

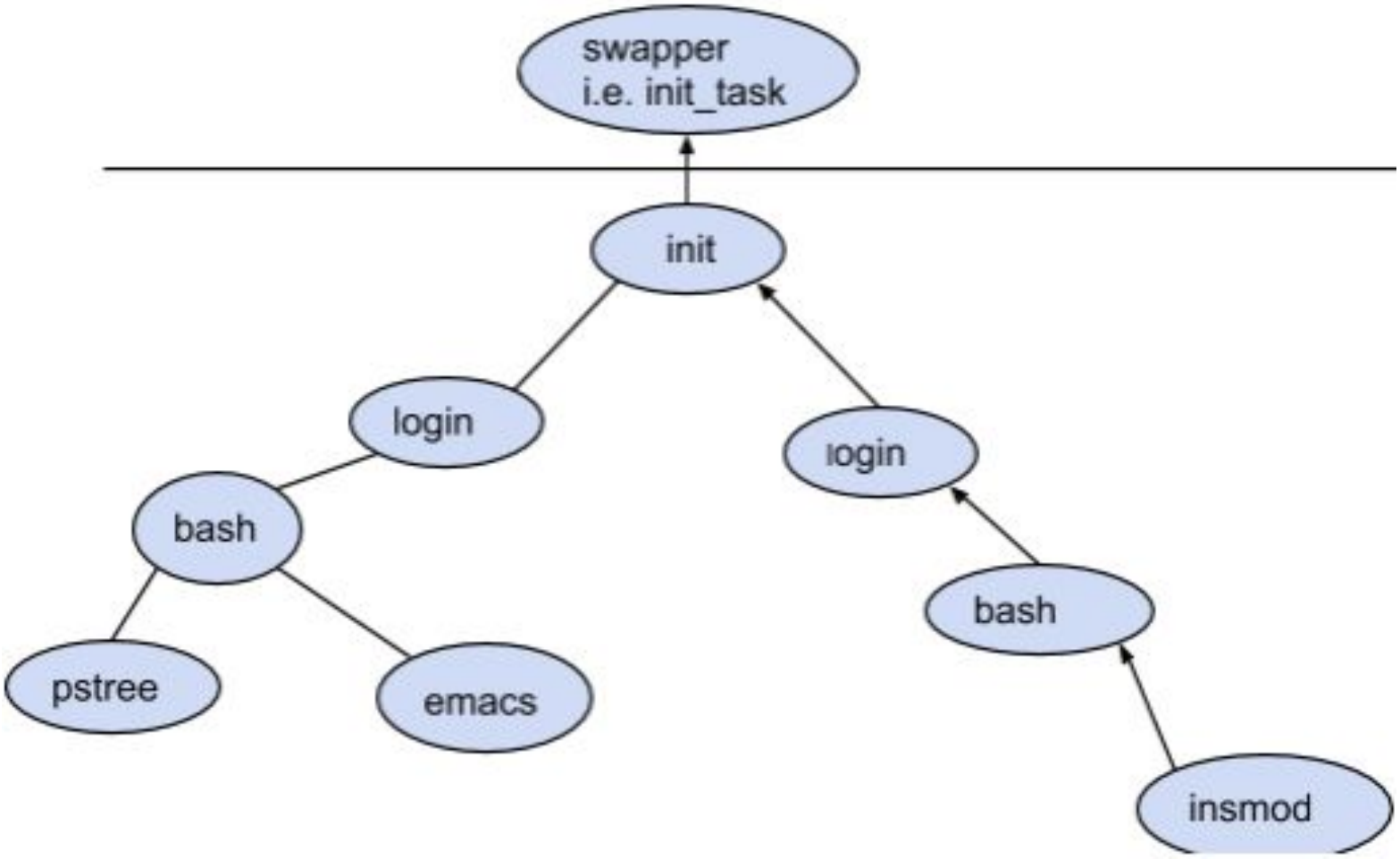
- Processo Órfã (Orphan)
  - Processo pai completou sua execução (exit ou kill) mas o processo filho ainda está executando.
  - Não faz a limpeza necessária e abandona os processos filhos.
  - Em alguns SOs o processo base (init no linux) “adota” os processos.
  - Encontrar processos órfãos não é tão simples:
    - Examinar todos os filhos do processo init e verificar se eles são legítimos ou não



Hierarquia de Processos

- Um processo filho pode criar outros processos, gerando uma hierarquia de processos.
- Exemplo prático: inicialização do UNIX
  - Um processo especial (***init***) está presente na imagem de inicialização;
  - Após ele bifurca em novos processos para criar os terminais;
  - O terminal espera alguém se conectar, e posteriormente um shell é inicializado.
  - A partir do shell pode-se iniciar novos processos.
    - pstree no linux

Hierarquia de Processos

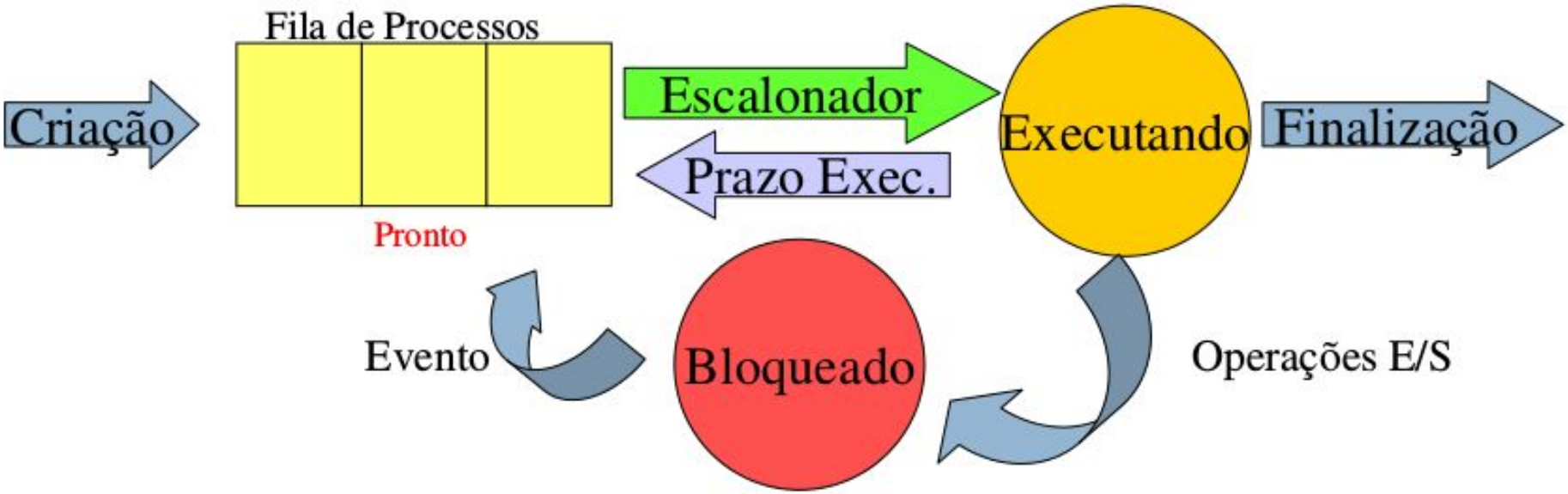


Estado de Processos

- Um processo pode estar em um desses três estados:
  - Em execução: realmente usando a CPU;
  - Pronto: executável, mas temporariamente parado (outro processo executando);
  - Bloqueado: incapaz de executar até que algum evento externo aconteça.

1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

Estado de Processos



Implementação de Processos

- Tabela de processos: uma entrada para cada processo (blocos de controle de processo - PBC)
- Cada PBC contêm informações necessárias para um processo ser colocado **em execução** quando **pronto** ou **bloqueado**
  - Estado do processo
  - Contador do programa
  - Ponteiro de pilha
  - Alocação de memória
  - Arquivos abertos
  - Contabilidade
  - Escalonamento

Exemplo: fork1.c

```
/* Exemplo de uso de fork. */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main() {
  pid_t r_fork;

  printf("Processo pai. PID do avô = %d\n", getppid());
  printf("Processo pai. PID = %d\n", getpid());

  if ((r_fork = fork()) != 0)
     printf("Processo pai. PID do filho = %d\n", r_fork);
  else {
    sleep(1);
    printf("Processo filho. PID = %d\n", getpid());
  }
  return 0;
}
```

Exemplo: fork2.c

```
/* * Exemplo de uso de fork. Quantos processos serão criados? */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {

  fork();
  fork();
  fork();

  printf("PID = %d\n", getpid());

  return 1;
}
```

Exemplo: fork3.c

```
/* * Exemplo de hierarquia de processos        A
 *                                             / \
 *                                            B   C          */
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main() {
  if (fork())
    if (fork())
      printf("Processo A\n");
    else
      printf("Processo C\n");
  else
    printf("Processo B\n");

  return 0;
}
```

Exemplo: wait1.c

```
/* * Exemplo de uso de fork e wait. */
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
int main() {
  pid_t pid;
  printf("Processo pai. PID = %d\n", getpid());
  if ((pid = fork()) != 0) {
    printf("Processo pai. PID do filho = %d\n", pid);
    wait(NULL);
    printf("Meu filho morreu... :-(\n");
  }
  else {
    sleep(2); /* Filho demora um pouco para executar. */
    printf("Processo filho. PID = %d\n", getpid());
  }
  return 0;
}
```

Exemplo: wait2.c

```
int main() {
  int i;
  if (fork())  /* Processo A */
    if (fork()) {
      for (i = 0; i < 2; i++)
      wait(NULL); /* Espera a morte de dois filhos */
      printf("Processo A\n");
    }
    else
      printf("Processo C\n");
  else  /* Processo B */
    if (fork())
      if (fork())
      if (fork()) {
        for (i = 0; i < 3; i++)
          wait(NULL); /* Espera a morte de tres filhos
      */
        printf("Processo B\n");
      }
      else
        printf("Processo F\n");
      else
      printf("Processo E\n");
    else
      printf("Processo D\n");
  return 0;
}
```

Exemplo: execv1.c

```
/* * Exemplo de uso de execve. execve1 /bin/ls */
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main(int argc, char **argv, char **envp) {
  if (fork() == 0) {  /* Processo filho */
    execve(argv[1], &argv[1], envp);
    fprintf(stderr,"Nao conseguiu executar execve.\n");
    return 2;
  }
  else {                  /* Processo pai   */
    wait(NULL);
    printf ("Filho terminou sua execucao.\n");
  }
  return 0;
}
```

Exemplo: zombie.c

```
/*
 * Exemplo de zumbi
 */
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main() {
        if(fork() == 0){
                //Filho
                printf("Terminando Filho PID %d \n", getpid());
                exit(0);
        } else {
                printf("Executando Pai PID %d \n", getpid());
                while(1); //loop infinito
        }
        return 0;
}
```

Exemplo: orphan.c

```
/*
 * Exemplo de orfao
 */
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>

int main() {
        if(fork() == 0){
                //Filho
                printf("Terminando Filho PID %d \n", getpid());
                while(1);
        } else {
                printf("Executando Pai PID %d \n", getpid());
                while(1); //loop infinito
                exit(0);
        }
        return 0;
}
```

FIM