Programação de Sistemas

Processos

Definição de processo (1)

[Def] Programa: sequência de instruções, que manipula valores de localizações (variáveis, parâmetros e retorno de funções).

Nota: o programa pode ser visto como entidade passiva.

[Def] Processador: dispositivo com capacidade de executar as instruções (ex: Intel Pentium, AMD Athlon64)

[Def] Processo: programa em execução, designado por *instância* do programa.

Nota: O processo é visto como entidade activa. Os arquitectos do Multics foram primeiros a adoptar o conceito de processo, mas não existe acordo sobre a definição exacta.

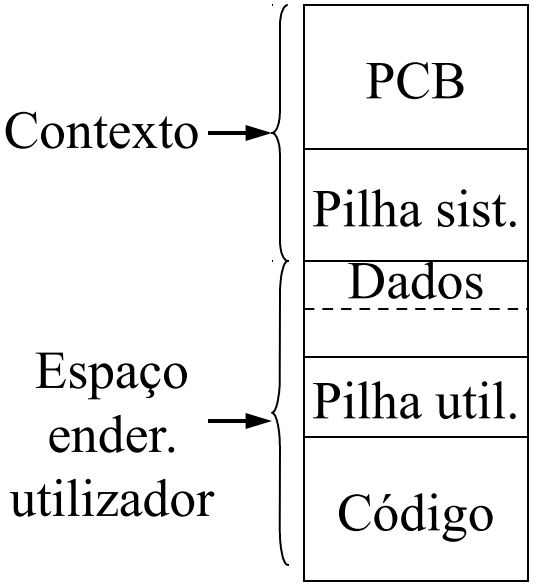
Definição de processo (2)

[Def] PID: número que identifica univocamente o processo.

- PID é do tipo `pid_t` (tipicamente int).
- Por compatibilidade, o valor máximo é 32_768 (`short`), mas pode ser alterado em `/proc/sys/kernetl/pid_max`

- Cada processo é constituído por 2 partes, designados globalmente por **imagem**.

A. Espaço de endereçamento de utilizador
B. Contexto.



Definição de processo (3)

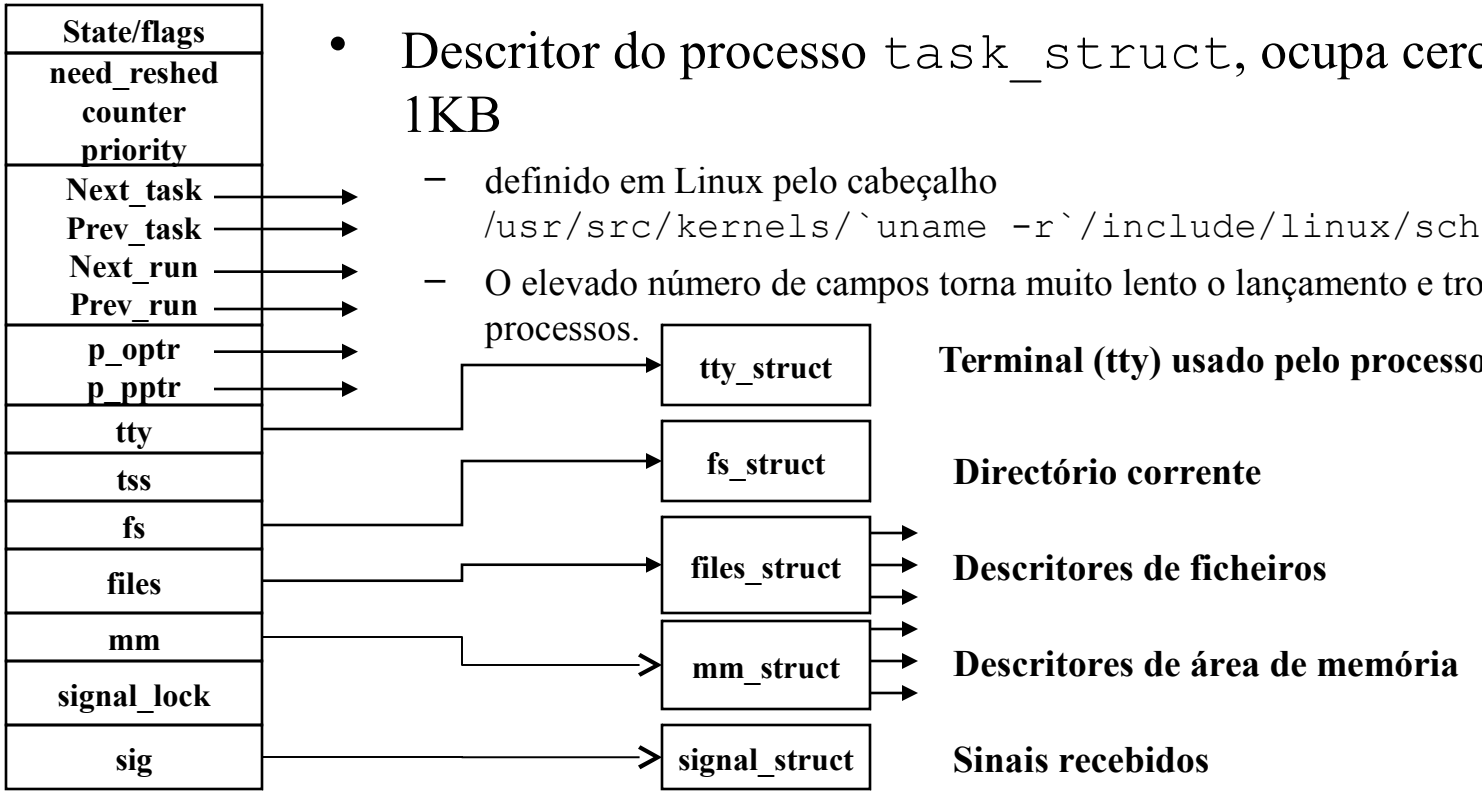
A. Espaço de endereçamento do utilizador
   Esta zona é composta por duas partes:
   1. **Código** do utilizador.
      Nota: No Linux, esta parte é designada por *text segment*
   2. **Dados** de utilizador
      - Globais.
        Nota: No Linux, este espaço é composto por duas partes: *initialized* e *uninitialized* (*bss*) *data segments*
      - Pilha de utilizador (para chamadas a funções e dados locais).

B. Contexto
   1. **Pilha de núcleo**, para chamadas de sistema.

Definição de processo (4)

2. **PCB** "Process Control Block": estrutura de controlo do processo, que no Linux é designada por *system data segment e* contém:
   - Identificação do processo: PID, processo pai, utilizador dono do processo, terminal associado
   - Estado do processador:
     - Contador de programa (PC-"program counter")
     - Registos do processador (acumulador,…)
   - Elementos de controlo de execução da imagem:
     - Estado do processo
     - Prioridade
     - Limites de memória RAM reservados pelo SO para o processo para
       - » Variáveis globais
       - » Pilha de execução, que contém endereços de retorno, variáveis locais a funções e variáveis temporárias (por exemplo, necessárias a cálculo de expressões)
     - Lista dos eventos que o processo está à espera.
     - Lista dos descritores de ficheiros abertos

Definição de processo (5)



- Descritor do processo `task_struct`, ocupa cerca 1KB
  - definido em Linux pelo cabeçalho `/usr/src/kernels/`uname -r`/include/linux/sched.h`
  - O elevado número de campos torna muito lento o lançamento e troca de processos.

Nota: Dimensões obtidas através do comando `size`

```
[rgc@asterix ~]$ size /usr/bin/gcc
   text    data     bss     dec     hex filename
 196215    4124       0  200339   30e93 /usr/bin/gcc
```

Definição de processo (7)

- Os limites de memória RAM são verificados nas instruções do programa de acesso à memória, por forma a impedir interferência entre processos.
  Nota: MSDOs e versões iniciais do Windows não preveniam acessos indevidos à memória, constituindo uma das causas para a má fama.
- Os processos podem ser distribuídos pelo(s) processadore(s) do computador de vários modos:
  - Lotes: um processo é executado de início até ao fim, um de cada vez.
  - **Multiprocessamento**: o despacho ("scheduler") atribui um intervalo de tempo a cada processo.
  - Paralelismo: cada processo é distribuído por um processador (o computador possui vários).
  - Distribuição: cada processo é distribuído por um computador.

Modo abordado em PSis

Ambiente de processo (1)

[Def] Ambiente: família de variáveis usadas no interpretador de comandos ("shell").
Ex: PATH lista os directórios de procura de comandos e de ficheiros.

A. Obtenção do ambiente em C, por chamada de sistema
POSIX:
```
#include <stdlib.h>
char *getenv(const char *);
```

- As variáveis de ambiente são sempre manipuladas pelo interpretador de comandos ("shell"), não pelo núcleo.
- Ex: char *path=getenv("PATH");

Ambiente de processo (2)

B. Obtenção do ambiente no interpretador de comandos
   - Comando `env` lista todas as variáveis do ambiente
   - Para obter valor de uma variável do ambiente, executar comando `echo $var`
     Exemplo: `echo $SHELL` identifica o interpretador de comandos.

```
[rgc@asterix ~]$ echo $SHELL
/bin/bash
```

•. Por omissão, todos os processos filho herdam o ambiente do processo pai.

Ambiente de processo (3)

- POSIX define várias variáveis de ambiente

[Def] Locale: convenções de formatação usadas numa zona geográfica para unidades temporais monetárias e numéricas.

No RedHat, as convenções são definidas no ficheiro `/etc/environment`

LANG=en_US.UTF-8

TZ=GMT

Variável	Significado
COLUMNS	Largura preferencial do terminal
HOME	Directório entrada do utilizador
LANG	Locale por omissão
LC_ALL	Sobreposição de nome do locale
LINES	Num. de linhas preferenciais do terminal
LOGNAME	Login associado ao processo
PATH	Directórios de pesquisa dos ficheiros
PWD	Caminho absoluto do directório corrente
SHELL	Caminho do interpretador de comandos
TERM	Tipo de terminal para saída
TMPDIR	Caminho do directório para fich temporários
TZ	Fuso horário

Figura 7-7, Advanced Programming UNIX Environment

Concorrência vs. Paralelismo (1)

- Na programação de sistemas, a simultaneidade pode ser expressa por dois conceitos próximos, mas distintos.
  - Paralelismo: simultaneidade em processadores distintos.
  - Concorrência: simultaneidade entre processos (residentes num único processador, ou em processadores distintos).

[Def] Computação paralela: execução do mesmo processo em vários processadores distintos.

Concorrência vs. Paralelismo (2)

[Def] Computação concorrente: execução de vários processos que interagem entre si.

- Os processos podem ser executados
  - num único processador,
  - por multiprocessadores ligados por
    - Proximidade: no mesmo "chip" (ex: Pentium Dual Core), bus-sistema designado por aglomerado-"cluster"1 ou em ligação rápida (ex: Ethernet Gigabit)
    - Rede, designado por sistemas distribuído.
- A comunicação entre os processos pode ser efectuada por
  - Memória partilhada
  - Mensagens

1 O computador mais potente é o "cluster" Blue Gene/L da IBM atinge 280 TFLOPS, com 64K processadores Power PC instalado nos laboratórios Livermore,CA/USA

Identificação de processos

- Um processo pode aceder a vários PID, por chamada de sistema

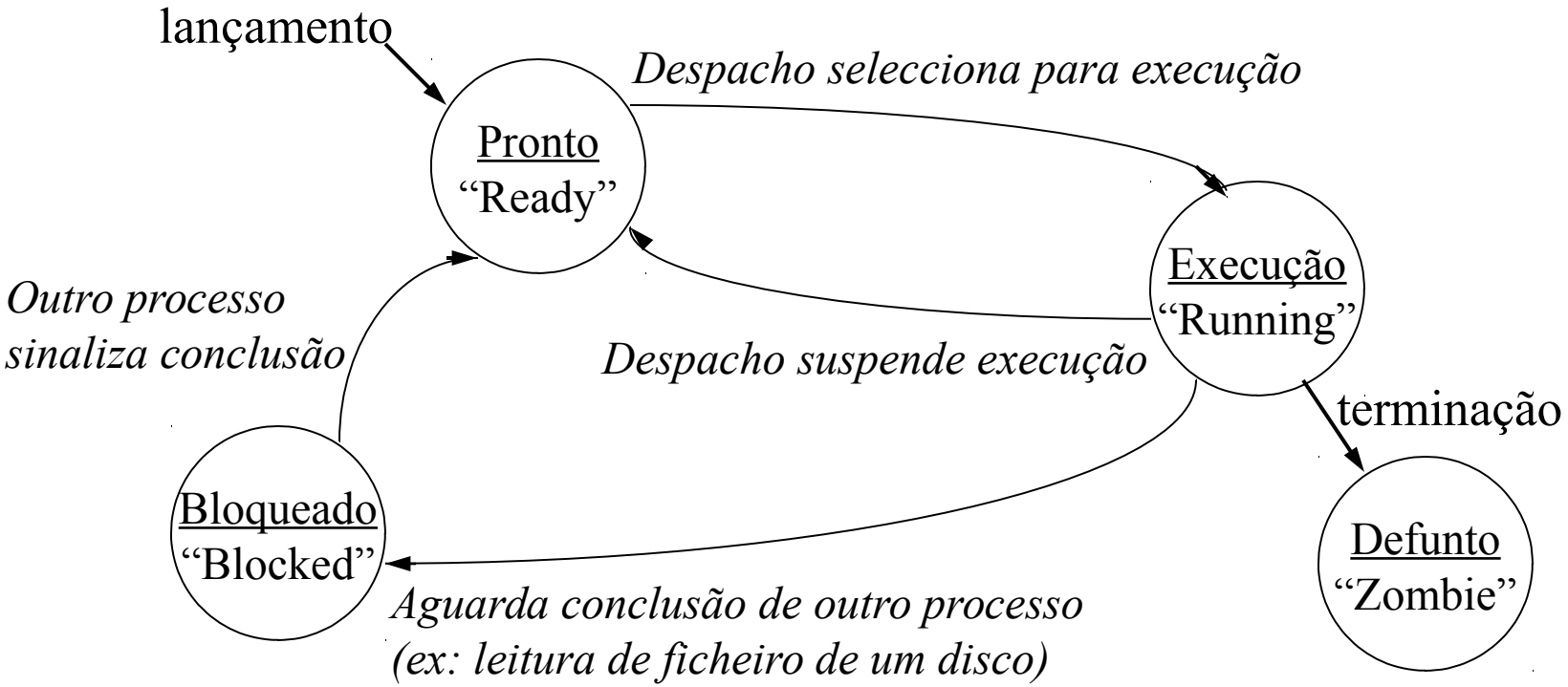
POSIX:

```
#include <unistd.h>
pid_t getpid();   /* próprio PID */
pid_t getppid();  /* PID do pai */
```

Atencão! Erros no projecto podem levar à criação de processos sem os destruir, sobrecarregando o computador. Verifique periodicamente a listagem dos seus processos correntes e termine os processos inúteis.

Estados de um processo (1)

- Um processo passa por diversos estados durante o seu tempo de vida.



Estados de um processo (2)

- **Execução**: as instruções estão a ser executadas
- **Pronto**: o processo está à espera que lhe seja atribuído o processador (vários processos podem se encontrar neste estado).
- **Bloqueado**: o processo está à espera que algum evento ocorra (recepção de sinal ou terminação de alguma operação de I/O).
- **Defunto**: o processo terminou, mas ainda não foi retirado das tabelas do SO.

Nota: Em cada instante, apenas um processo pode estar no estado de *execução*. Vários processos podem encontrar-se em modo de *prontidão* ou de *bloqueio*.

Lançamento de um processo (1)

- No "boot" do Linux é lançado o primeiro processo `init`, de PID=1.
  - *init* é responsável pela inicialização do Linux.
  - Na inicialização, o init executa o ficheiro `/etc/rc.d` de lançamento dos serviços.
  - Por cada entrada de utilizador, *init* lança uma cópia que executa sucessivamente programas até o interpretador de comando ("shell").
- Cada processo é lançado por outro, formando uma árvore de processos.
  A raíz da árvore de processos é `init`.

Lançamento de um processo (2)

- O numero máximo de processos abertos por um utilizador estabelecido no ficheiro `/etc/security/limits.conf` na forma <domain> <type> <item> <value>
  - <domain> é utilizador (* para todos)
  - <type> é `hard` (limite absoluto) ou `soft` (inicial, que pode ser incrementado pelo utilizadores até ao limite hard
  - <item> é o recurso (nproc para processos)

Lançamento de um processo (3)

- No lançamento do processo, o SO pode seguir várias estratégias na relação entre o pai e o filho.
  - Modos de execução
    - Pai e filho executam concorrencialmente, de forma independente: opção adoptada pelo Linux
    - Pai fica à espera que filho termine
  - Ocupação de memória
    - Filho duplica espaço do pai: opção adoptada pelo Linux (mas não pode aceder ao espaço de dados do processo pai)
    - Filho carrega novo programa
  - Partilha de recursos
    - Pai e filhos partilham todos os recursos
    - Filhos partilham subconjunto de recursos: opção adoptada pelo Linux (ficheiros abertos, objectos para comunicação intrer-processos)
    - Pais e filhos não partilham recursos

Lançamento de um processo (4)

A. Lançamento de processo, por chamada de sistema

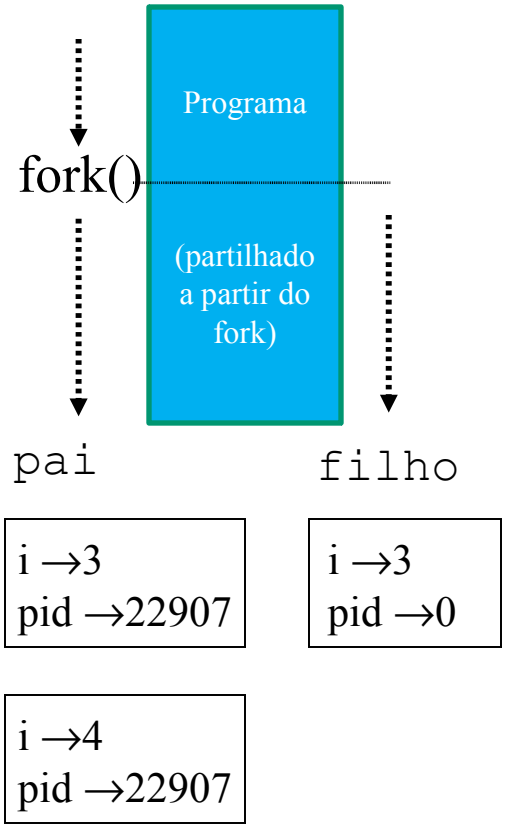
POSIX:
```
#include <unistd.h>
pid_t fork();
```

- É lançado um novo processo filho, com cópia de todas as variáveis.
  Nota: depois de executado o `fork`, cada processo altera a sua versão da variável (a outra versão da variável apenas pode ser alterada pelo outro processo).
- `fork` é uma função que devolve um valor POR DUAS VEZES!
  - No processo filho, o valor devolvido é 0.
  - No processo pai, o valor devolvido é o PID do processo filho.
  - Todas as restantes variáveis do processo filho ficam com o mesmo valor do processo pai (mas que podem ser alteradas a partir daqui de forma independente).

Lançamento de um processo (5)

Tipicamente, um programa tem o seguinte formato

```
pid_t pid;
int i=3;
…
pid = fork();
if (pid==0) {
   /* processo filho */
   … }
else {
   /* processo pai */


   i++;
   … }
```



Lançamento de um processo (6)

Exemplo de lançamento de processo

```
#include <sys/types.h>
#include <stdio.h>
main() {
   int x = 1;
   pid_t pid = fork();
   if (pid == 0) {
        printf("Filho inicia com x = %d\n",x);
        printf("Filho tem x = %d\n", ++x); }
   else {
       printf("Pai continua com x = %d\n",x);
       printf("Pai tem x = %d\n", --x); }
   printf("Sai do processo %d, x=%d\n",getpid(),x);
}
```

```
[rgc@asterix Processos]$ testaFork
Pai continua com x = 1
Filho inicia com x = 1
Filho tem x = 2
Sai do processo 21513, x=2
Pai tem x = 0
Sai do processo 21512, x=0
```

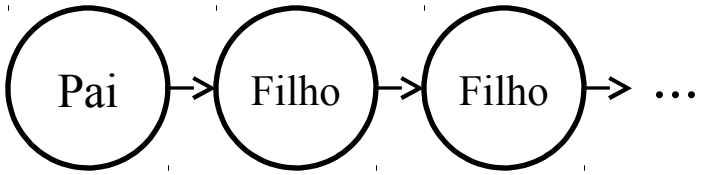
Resultado obtido (nota: os PID podem ser distintos e a ordem de execução dos processos alterada!)

Lançamento de um processo (7)

- Para lançamento de vários processos podem ser adoptadas duas estratégias:

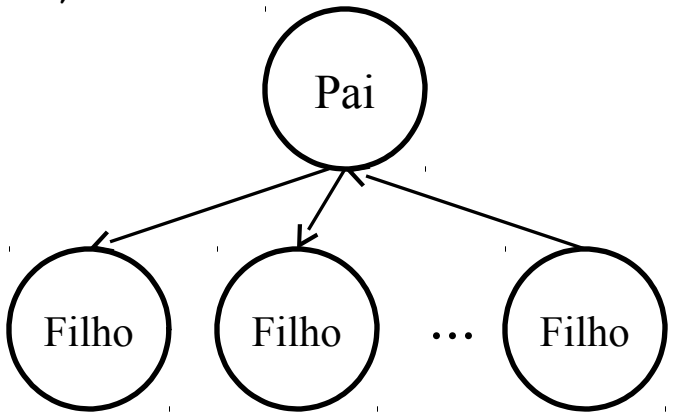
Cadeia

```
pid_t childpid = 0;
for (i=1;i<n;i++)
  if (childpid = fork())
  break;
```



Ventoinha

```
pid_t childpid = 0;
for (i=1;i<n;i++)
  if ((childpid = fork())<=0)
  break;
```



Lançamento de um processo (8)

B. Lançamento de processo, pela biblioteca normalizada do C:

```
#include <stdlib.h>
int system(const char *cmdstring);
```

A função `system` executa em sequência os seguintes serviços de sistema:

1. Lançamento de novo processo por `fork`
2. Substituição do processo por `exec`
3. O processo pai espera terminação do processo descendente por `wait`

Lançamento de um processo (8)

Substituição de programa C:

```
#include <unistd.h>

int execve(const char *filename, char *const argv[],
                   char *const envp[]);
```

A função `execve` substitui a imagem de um processo

Novo código, novo ambiente de execução:

`man 2 execve`

`Man 3 exec`

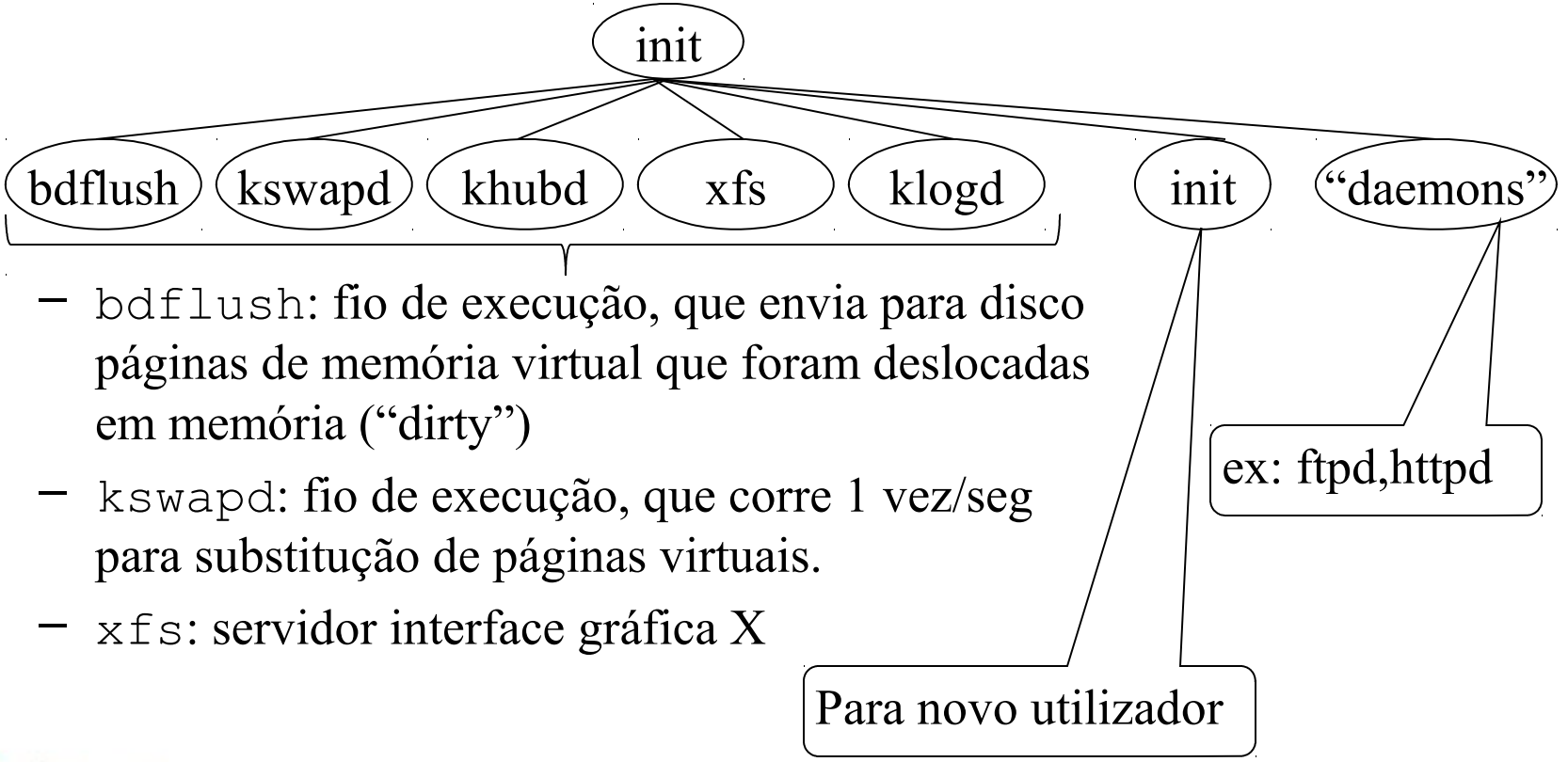
Lançamento de um processo (9)

C. Lançamento de processo no interpretador de comandos
   Basta indicar o ficheiro de execução!
   - O interpretador de comandos procura o programa na lista dos directórios indicados na variável de ambiente `PATH`.
   - O processo interpretador de comandos fica à espera que o programa termine.
   - Se o utilizador não quiser ter de esperar pela terminação do novo processo, inserir `&` no fim da linha de comando e o processo corre em paralelo ("background"). O interpretador de comandos indica o PID do processo e avisa terminação do novo processo com a mensagem `Done`.

```
asterix.ist.utl.pt> gcc test.c &
[1] 1932
asterix.ist.utl.pt>
[1] Done        gcc test.c
```

Lançamento de um processo (10)

- O Linux lança vários processos no “boot”



  - `bdflush`: fio de execução, que envia para disco páginas de memória virtual que foram deslocadas em memória (“dirty”)
  - `kswapd`: fio de execução, que corre 1 vez/seg para substituição de páginas virtuais.
  - `xfs`: servidor interface gráfica X

Eliminação de um processo (1)

- A eliminação de um processo não afecta os processos filho, porque estes são entidades independentes.
- Quando um processo filho termina, o Linux mantém tabelas desse processo até o processo pai terminar. Durante este período, o processo filho é designado por "zombie".

  Porquê processos zombie?

  - O processo pai pode ficar à espera que o processo filho termine.
  - A instrução de terminação indica o código do resultado do processo filho por forma que o pai aceda ao código do resultado.
  - Enquanto o pai não terminar, o processo filho tem de continuar registado após terminação, para que o pai aceda ao resultado do processo filho.

Eliminação de um processo (2)

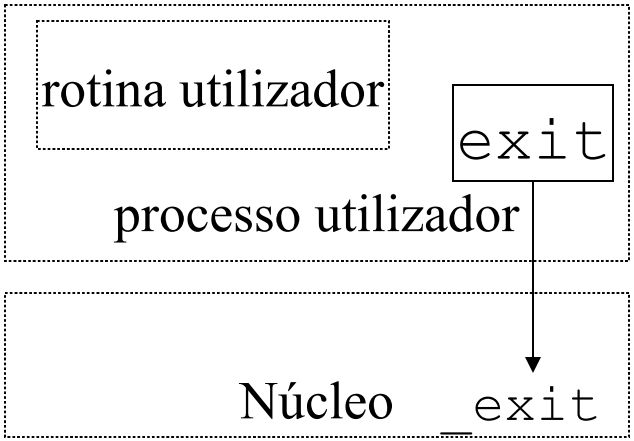
A. Eliminação em C, por chamada de sistema

POSIX: `#include <unistd.h>`

- Na terminação com sucesso, usar parâmetro 0
- Na função `int main(int argc, char *argv[])` pode-se terminar pela instrução `return n;`

Eliminação, pela biblioteca normalizada do C:

```
#include <stdlib.h>
void exit(int status);
```



Eliminação de um processo (3)

Um processo elimina outro pela chamada de sistema

POSIX: `#include <signal.h>`

```
int kill(pid_t, int);
```

- Se o 1º parâmetro for 0, o sinal é enviado a todos os processos do grupo

B. Eliminação no interpretador de comandos
Executar comando `kill [-s signal] PID`

- Se for indicado para PID o valor 0, são eliminados todos os processos no grupo do processo corrente
- Se o PID for negativo, são eliminados todos os processos do grupo.

Nota: nunca terminar processo `init`, porque tal provoca o "shutdown" do sistema operativo!

Eliminação de um processo (4)

Exemplo de processo Zombie

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

int main() {
   if (fork() == 0) {
      printf("Fim filho,PID=%d\n",
              getpid());
      exit(0); }
   else {
      printf("Corre pai,PID=%d\n",
              getpid());
      while (1)
        ; /* Ciclo infinito */
    }
}
```

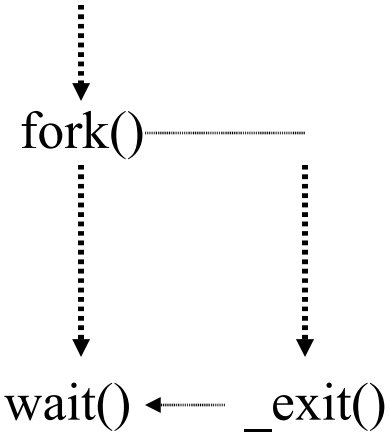
```
[rgc@asterix Processos]$ testaZombie &
[1] 6160
[rgc@asterix Processos]$ Fim filho, PID = 6161
Corre pai, PID = 6160

[rgc@asterix Processos]$ ps
  PID TTY          TIME CMD
 5951 pts/2    00:00:00 bash
 6160 pts/2    00:00:15 testaZombie
 6161 pts/2    00:00:00 testaZombie <defunct>
 6162 pts/2    00:00:00 ps
[rgc@asterix Processos]$ kill 6160
[rgc@asterix Processos]$
[1]+  Terminated              testaZombie
[rgc@asterix Processos]$ ps
  PID TTY          TIME CMD
 5951 pts/2    00:00:00 bash
 6163 pts/2    00:00:00 ps
[rgc@asterix Processos]$
```

Eliminação de um processo (5)

- O processo corrente pode ser suspenso na linha de comandos pelo teclado
  - `CTRL-C` para terminação
  - `CTRL-Z` para suspensão. O utilizador tem as seguintes opções de comando
    - `bg`: processo suspenso retoma em “background”
    - `fg`: processo suspenso retorma execução em ”forground”
    - `kill %`: o processo termina

Espera por um processo (1)

- Um processo pode lançar um processo, para executar um trabalho específico, e ficar à espera que este termine.



- Para que o processo pai não fique a ocupar inutilmente tempo de CPU, um processo à espera fica bloqueado até que o processo filho termine.

Espera por um processo (2)

A. Espera em C, por chamada de sistema

POSIX: `#include <sys/wait.h>`

```
pid_t waitpid(pid_t,int *,int);
```

- Processo fica à espera que um processo filho específico termine.
- 1º parâmetro: PID do processo a esperar (-1 qualquer)
- 3º parâmetro: opções indicadas em bits
  - WNOHANG: chamada não suspende
  - WUNTRACED: processo filho termina ou suspende

POSIX: `#include <sys/wait.h>`

```
pid_t wait(int *);
```

- Processo fica à espera que um qualquer processo filho termine (PID indicado no retorno).
- O parâmetro identifica o endereço da localização onde é colocado o estado do retorno (como o processo terminou e o código) .

Nota: `wait(&status)` equivalente a `waitpid(-1, &status, 0)`

Espera por um processo (3)

- O POSIX define 3 pares de macros: a primeira macro testa o método de terminação e a segunda macro recolhe o código.
  - `WIFEXITED(int)`: testa se terminou normalmente.
    Em caso positivo, o valor de `exit` é dado pelos 8 bits de menor prioridade por `WEXITSTATUS(int)`.
  - `WIFSIGNALED(int)`: testa se processo terminou com sinal.
    Em caso positivo o sinal é dado por `WTERMSIG(int)`.
  - `WIFSTOPPED(int)`: testa se processo parado com sinal.
    Em caso positivo o sinal é dado por `WSTOPSIG(int)`.

B. Espera no interpretador de comandos
   Executar comando `wait [n]`
   - `n` é o PID do processo a esperar.

1º EXERCÍCIO TEORICO-PRATICO

- Crie um programa `pratica num`, que entrega a um processo o cálculo do produto de uma tabela.
- Execute os seguintes passos:
  A. Verifique o número de parâmetros ser correcto. `num` tem de ser, pelo menos, igual a 2.
  B. Lê do utilizador `num` inteiros para uma tabela.
  C. Um subprocesso calcula o produto dos elementos da tabela, escreve o valor no terminal e sai com um dos seguintes valores:
     `1` - resultado positivo
     `2` – resultado nulo
     `3` – resultado negativo
  D. O processo principal espera pela conclusão do cálculo imprimindo o valor de saída do processo de cálculo.

Substituição de imagem (1)

- Os processos pai e filho partilham o mesmo código, embora cada um execute apenas a sua parte.
  - Alterar uma parte obriga a compilar também a outra.
  - A partilha impede a separação do desenvolvimento de software.
- A famíla de funções de sistema `exec` permitem substituir a imagem (ou programa) corrente, por outra com o mesmo PCB do anterior processo.
  - Existem duas famílias de funções exec, indicadas na primeira letra do sufixo, que determinam a forma de passagem dos argumentos: “l” (lista) e “v” (vector).
  - Os argumentos são recebidos pelo novo programa a partir de argv[0].
  - O novo programa recebe do anterior o ambiente e os descritores de ficheiros.
- Não há retorno de `exec`

Substituição de imagem (2)

- Funções execl:
  - `int execl(const char *path,const char *arg0,...);`
    O programa path indicado em absoluto.
  - `int execlp(const char *path,const char *arg0,...);`
    O progama path é procurado nos directórios listados na variável de ambiente $PATH
  - `int execle(const char *path, const char *arg0,..., char *envp[]);`
    O ambiente do processo é substituído pelo parâmetro envp.

  Nota: a lista de argumentos deve terminar por NULL.
- Funções execv: os argumentos argi são passados por uma tabela
  - `int execv(const char *path, const char *argv[]);`

Substituição de imagem (3)

```
/* programa testaExec.c */
#include <stdio.h>
#include <sys/types.h>
int main() {
   pid_t pid = fork();
   if (pid == 0) { /* Child */
      printf("Substituo programa\n");
      execl("/home/ec-ps/public_html/Exemplos/Processos/alvo",
            "alvo", "par1", "par2", NULL); }
   else { /* Parent */
      printf("Lancei filho PID = %d\n", pid);
      printf("Termina pai, PID = %d\n", getpid());
      exit(0); }
}
```

Substituição de imagem (4)

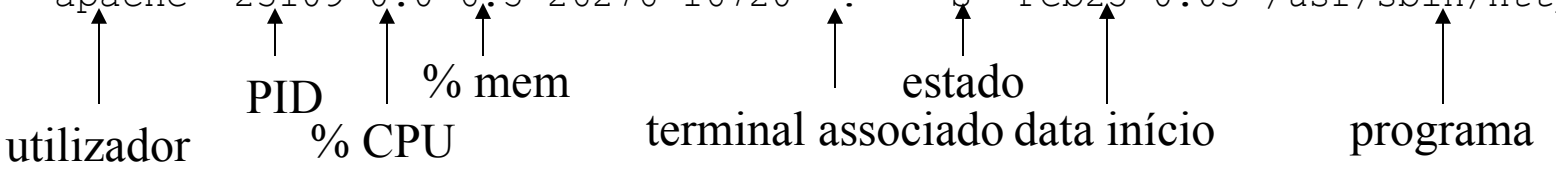
```
/* programa alvo.c */
int main(int argc,char *argv[]) {
  int i;
  printf("**********\n");
  printf("Sou programa [%s]\n",argv[0]);
  for(i=1;i<argc;i++)
     printf("argv[%d]=%s%s",i,argv[i],i==(argc-1) ? "\n" : ", ");
  printf("Termina PID = %d\n", getpid());
  printf("**********\n");
  exit(0); }
```

```
[rgc@asterix Processos]$ testaExec
Substituo programa
Lancei filho PID = 20822
Termina pai, PID = 20821
[rgc@asterix Processos]$ **********
Sou programa [alvo]
argv[1]=par1, argv[2]=par2
Termina PID = 20822
**********
[rgc@asterix Processos]$
```

Consulta de processos (1)

- Os processos correntes no Linux são listados pelo comando `ps -aux`

```
asterix.ist.utl.pt>ps -aux
rgc      386   0.0 0.6  5224  1544 pts/1 Rs 11:47 0:00 -tcsh
apache  23109 0.0 6.5 26276 16720  ?    S  Feb25 0:03 /usr/sbin/httpd
```



Estado	Estado
D	Suspenso sem hipótese interrupção (I/O,…)
R	Execução, ou na fila de prontidão
S	Suspenso à espera de evento
T	Parado
Z	Defunto, à espera de ser terminado pelo processo pai

Consulta de processos (2)

- Tabela dos 10 maiores processos listada pelo comando `top`

```
asterix.ist.utl.pt> top
top - 22:31:04 up  8:40,  2 users,  load average: 0.00, 0.02, 0.00
Tasks:  88 total,   1 running,  87 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3% us,  0.3% sy,  0.0% ni, 99.3% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:    255700k total,   231436k used,    24264k free,    34628k buffers
Swap:   524280k total,        0k used,   524280k free,   138748k cached
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3443 root      15   0 59920 8708 2516 S  0.7  3.4   1:44.10 X
 5343 rgc       16   0  1956  956  756 R  0.3  0.4   0:00.04 top
    1 root      16   0  1692  552  476 S  0.0  0.2   0:01.42 init
    2 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    3 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 events/0
    4 root      11  -5     0    0    0 S  0.0  0.0   0:00.02 khelper
    5 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kthread
    7 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   65 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
```

nice

prioridade

cod+dados

Memória total (cod+dados+bibliotecas partilhadas+swap)

Troca de contexto (1)

- Um processo em execução pode ser trocado por outro em diversas situações:
  - Interrupt externo
    - Temporizador (tempo esgotou-se)
    - I/O (ocorreu um evento)
  - TRAP interno
    - Operação inválida (ex, divisão por zero)
    - Tentativa de acesso a parte de programa, que não se encontra residente na memória central (“Pagefault”)
  - Chamada de sistema

Troca de contexto (2)

- Quando o CPU é entregue a outro processo, o SO tem executar as seguintes acções:
  1. Salvaguardar o estado do processo que vai perder o CPU.
  2. Carregar o estado do processo que vai ganhar o CPU (que foi salvaguardado anteriormente).
- A troca de contexto não corresponde a actividade útil ao utilizador ("overhead"), pelo que deve ser minimizado
  - número de trocas de contexto
  - duração das trocas de contexto

Troca de contexto (3)

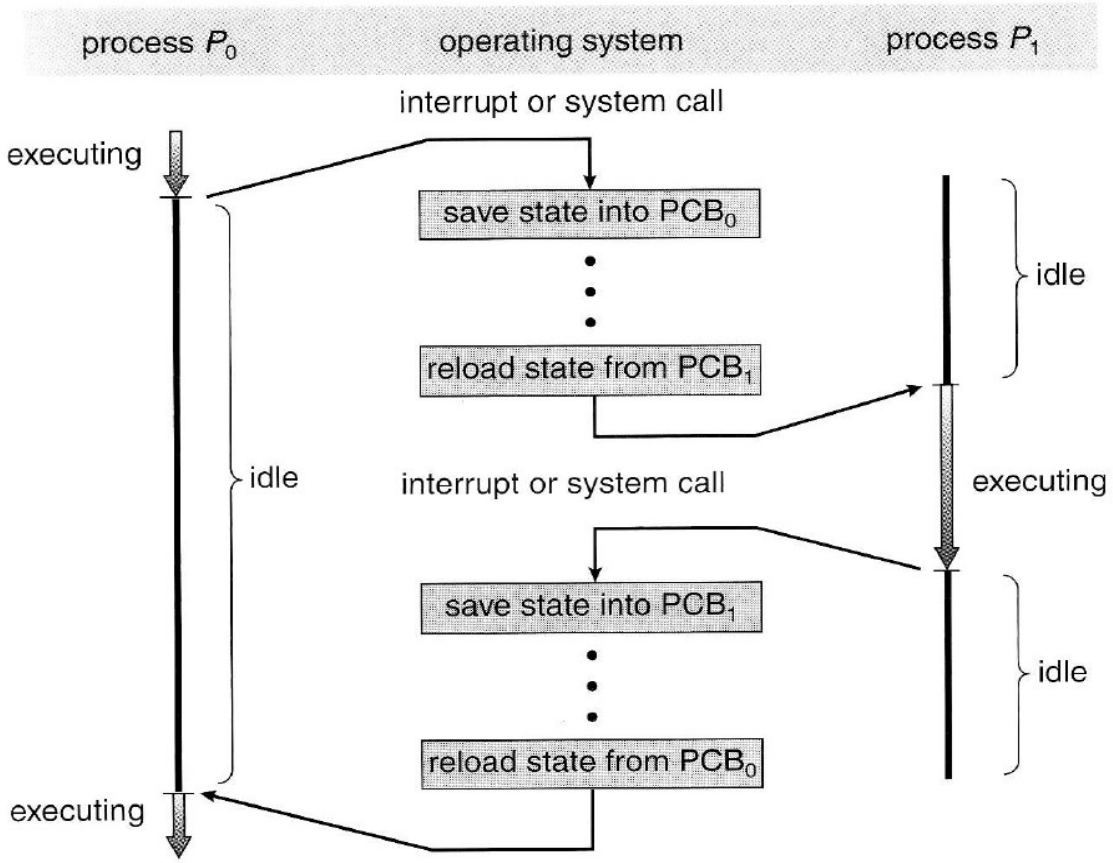


Figura 3-8, A. Silberschatz et al., Operating Systems Concepts with Java

Troca de contexto (4)

- O despacho gere os processos através de uma lista de prioridade (para os processos prontos a executar) e listas ligadas (para os processos bloqueados).

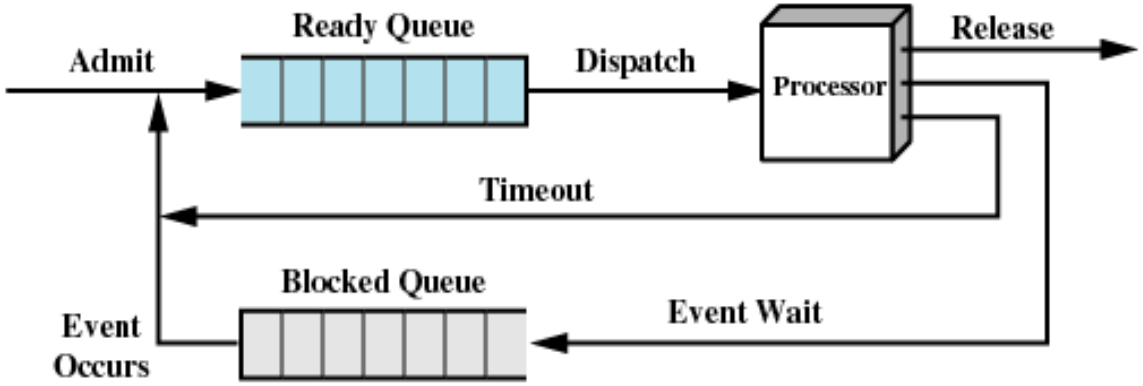


Figura 3-8, William Stallings, Operating Systems: Internals and Design Principles

Escalonamento Linux (1)

- Vários algoritmos de escalonamento foram criados para seleccionar o processo da lista de prontidão para execução:
  - Sequencial (“FCFS”) : CPU entregue pela ordem de chegada.
  - Prioridade: CPU entregue ao processo mais prioritário
  - SJF : CPU entregue ao processo de menor duração
  - Circular (“Round Robin”): a cada processo é entregue uma fatia de tempo, designado “quantum”-tipicamente 100 ms.

Escalonamento Linux (2)

- Aos processos de utilizador são atribuídos uma prioridade base e um quantum.
  - Prioridades variam entre 0 (máxima) e 139 (mínima).
    - Processo `swapper` recebe prioridade 0.
    - Processos de utilizador recebem prioridade base 50.
  - Processo com prioridade superior recebe um quantum maior.
- Os processos são escalonados de acordo com a prioridade. Quando o processo esgota o CPU (ou ficar bloqueado), ele tem de esperar até que todos os outros ocupem o CPU – aproximação circular, que evita a carência.

Escalonamento Linux (3)

- A prioridade varia no Linux em diversas formas:
  1. A prioridade varia 5 pontos, dependendo da orientação (orientado a E/S sobe, orientado ao processamento desce).
  2. Prioridade dos processos recalculada, em cada 4 tiquetaques do relógio (frequência determinada por HZ-definido em /usr/include/asm/param.h, varia entre 100 até ao kernel 2.5, passando por 512 no RedHat 8, até 1000 nas versões mais recentes do núcleo), segundo a fórmula
     *prioridade = Base+CPU_usage/4 + 2*nice*
     - *CPU_usage* é incrementado em cada tiquetaque do relógio.
     - *nice* pode ser incrementado para baixar prioridade em processos pouco importantes.
       POSIX:XSI `nice(int);`
     - No Shell usar
       `nice valor comando args`
       Ex: `nice 10 prog`

- Os processos do núcleo são não preemptivos.

Lançamento de daemons no RH (1)

- Para além dos processo de utilizador, o Linux possui processos especiais dedicados a serviços ou tarefas de gestão.

[Def] Daemon: processo lançado no "boot" do Linux para execução de tarefas essenciais ao SO (ex: despacho de processos) ou servidores (ex: servidor de correio).

- Não possui terminais associados (mensagens enviadas para ficheiros de registo, normalmente em `/var/log`)
- O dono é o `root`.
- O primeiro `daemon` lançado é o `inetd`, com PID=1.
  Nota: fazer o `kill` do `inetd` faz "crashar" o Linux!

Lançamento de daemons no RH (2)

- O lançamento de um "daemon" é feito pelo administrador do sistema pelo comando

  `/sbin/service httpd start` ← Outras opções:

  `stop`
  `status`
  `restart`

- Na opção status é indicado o estado e os PIDs do grupo de processos.

```
[rgc@asterix public_html]$ /sbin/service httpd status
httpd (pid 15258 15257 15256 12183 12131 10089 5826 5824 5821 5818 5817 5814 5812 5811 5491
    2015) is running...
[rgc@asterix public_html]$
```

Lançamento de daemons no RH (3)

- Para que o lançamento de um "daemon" seja efectuado automaticamente pelo Linux no arranque, o administrador deve executar o comando
  `/sbin/chkconfig httpd on`
- Seleccionado (retirado) o lançamento no arranque, o identificador do "script" passa a ter o prefixo `S` (`K`)

Outras opções:

`off`

`reset`

```
[rgc@asterix Processos]$ ls /etc/rc.d/rc5.d/
K01dnsmasq     K50netconsole   K91capi     S26udev-post     K01smartd  K50snmpd
K92ip6tables   S28autofs       K02NetworkManager  K50snmptrapd     K92iptables     S35qemu
K02NetworkManagerDispatcher    K50vsftpd   S00microcode_ctl S44acpid   K05innd
K69rpcsvcgssd  S05kudzu        S55sshd     K05saslauthd     K73ldap    S06cpuspeed
S65dovecot    K10dc_server    K73winbind   S09isdn          S78spamassassin  K10psacct
K73ypbind     S10network      S80sendmail        K12dc_client     K74lm_sensors    S11auditd
S85httpd      K15gpm          K74nscd      S12restorecond   S88nasd    K-1bluetooth
K74ntpd       S12rsyslog      S90ConsoleKit      K20jetty         K76openvpn S13irqbalance
S90crond      K20nfs          K84btseed    S13rpcbind       S90smolt   K20tomcat5
K84bttrack    S13setroubleshoot            S95atd           K24irda    K85racoon
S14nfslock    S96avahi-daemon K25squid     K87multipathd    S15mdmonitor     S97libvirtd
K35backuppc         K87named        S18rpcidmapd       S97yum-updatesd  K35nmb
K88wpa_supplicant             S19rpcgssd   S98cups          K35smb     K89dund
S-1bluetooth   S98haldaemon    K36lisa     K89netplugd      S22messagebus    S99anacron
K36mysqld     K89pand         S25netfs     S99firstboot     K36postgresql    K89rdisc
S25pcscd      S99local
[rgc@asterix Processos]$
```