



Gestor de Processos Núcleo Unix, Linux, Windows

Sistemas Operativos

2012 / 2013

Sistemas Operativos – DEI - IST



Unix e Linux

Gestor de Processos

Sistemas Operativos – DEI - IST



Contexto dos Processos

- Em Unix encontrava-se dividido em duas estruturas:
 - A estrutura `proc` – sempre mantida em memória para suportar o escalonamento e o funcionamento dos signals
 - A estrutura `u` - `user` – só era necessária quando o processo se estivesse a executar → transferida para disco se houvesse falta de memória
- As estruturas `proc` eram organizadas num vector cuja dimensão ditava o número máximo de processos que o sistema poderia ter.

Sistemas Operativos – DEI - IST



Unix - Contexto Núcleo de um Processo

- | | |
|--|--|
| <ul style="list-style-type: none">• estrutura <code>proc</code>:<ul style="list-style-type: none">– <code>p_stat</code> – estado do processo– <code>p_pri</code> – prioridade– <code>p_sig</code> – sinais enviados ao processo– <code>p_time</code> – tempo que está em memória– <code>p_cpu</code> – tempo de utilização– <code>p_pid</code> – identificador do processo– <code>p_ppid</code> – identificador do processo pai• parte do contexto do processo necessária para efectuar as operações de escalonamento | <ul style="list-style-type: none">• estrutura <code>u</code>:<ul style="list-style-type: none">– registos do processador– pilha do núcleo– códigos de protecção (UID, GID)– referência ao directório corrente e por omissão– tabela de ficheiros abertos– apontador para a estrutura <code>proc</code>– parâmetros da função sistema em execução |
|--|--|

Sistemas Operativos – DEI - IST



Linux: task_struct

```
struct task_struct {
    volatile long state;
    unsigned long flags;
    int sigpending;
    mm_segment_t addr_limit;
    struct exec_domain *exec_domain;
    volatile long need_resched;
    unsigned long ptrace;
    int lock_depth;
    unsigned int cpu;
    int prio, static_prio;
    struct list_head run_list;
    prio_array_t *array;
    unsigned long sleep_avg;
    unsigned long last_run;
    unsigned long policy;
    unsigned long cpus_allowed;
    unsigned int time_slice, first_time_slice;
    atomic_t usage;
    struct list_head tasks;
    struct list_head ptrace_children;
    struct list_head ptrace_list;
    struct mm_struct *mm, *active_mm;
    struct linux_binfmt *binfmt;
    int exit_code, exit_signal;
    int pdeath_signal;
    unsigned long personality;
    int did_exec;
    unsigned task_dumpable:1;
    pid_t pid;
    pid_t pgrp;
    pid_t tty_old_pgrp;
    pid_t session;
    pid_t tgid;
    int leader;
    struct task_struct *real_parent;
    struct task_struct *parent;
    struct list_head children;
    struct list_head siblings;
    struct task_struct *group_leader;
    struct pid link_pids[PIDTYPE_MAX];
    wait_queue_head_t wait_chldexit;
    struct completion *vfork_done;
    int *set_child_tid;
    int *clear_child_tid;
    unsigned long rt_priority;

    unsigned long it_real_value, it_prof_value, it_virt_value;
    unsigned long it_real_incr, it_prof_incr, it_virt_incr;
    struct timer_list real_timer;
    struct tms times;
    struct tms group_times;
    unsigned long start_time;
    long per_cpu_utime[NR_CPUS], per_cpu_stime[NR_CPUS];
    unsigned long min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
    int swappable:1;
    uid_t uid, euid, suid, fsuid;
    gid_t gid, egid, sgid, fsgid;
    int ngroups;
    gid_t groups[NR_GROUPS];
    kernel_cap_t cap_effective, cap_inheritable, cap_permitted;
    int keep_capabilities:1;
    struct user_struct *user;
    struct rlimit rlim[RLIM_NLIMITS];
    unsigned short used_math;
    char comm[16];
    int link_count, total_link_count;
    struct tty_struct *tty;
    unsigned int locks;
    struct sem_undo *semundo;
    struct sem_queue *semlsoping;
    struct thread_struct thread;
    struct fs_struct *fs;
    struct files_struct *files;
    struct namespace *namespace;
    struct signal_struct *signal;
    struct sighand_struct *sighand;
    sigset_t blocked, real_blocked;
    struct sigpending pending;
    unsigned long sas_ss_sp;
    size_t sas_ss_size;
    int (*notifier)(void *priv);
    void *notifier_data;
    sigset_t *notifier_mask;
    void *uxx_info;
    void (*tux_exit)(void);
    u32 parent_exec_id;
    u32 self_exec_id;
    spinlock_t alloc_lock;
    spinlock_t switch_lock;
    void *journal_info;
    unsigned long ptrace_message;
    siginfo_t *last_siginfo;
};
```

Sistemas Operativos – DE1 – IST

Porque motivo deixou
de estar separada
em duas estruturas?



Modo Utilizador/Modo Núcleo

- Processo em modo utilizador:
 - executa o programa que está no seu segmento de código
- Muda para modo sistema:
 - sempre que uma exceção ou interrupção é desencadeada
 - exceção ou interrupção que pode ser provocada pelo utilizador ou pelo hardware
- A mudança de modo corresponde a:
 - mudança para o modo de protecção mais privilegiado do processador
 - mudança do espaço de endereçamento do processo utilizador para o espaço de endereçamento do núcleo
 - mudança da pilha utilizador para a pilha núcleo do processo
- A pilha núcleo:
 - É usada a partir do instante em que o processo muda de modo utilizador para modo núcleo
 - está vazia quando o processo se executa em modo utilizador

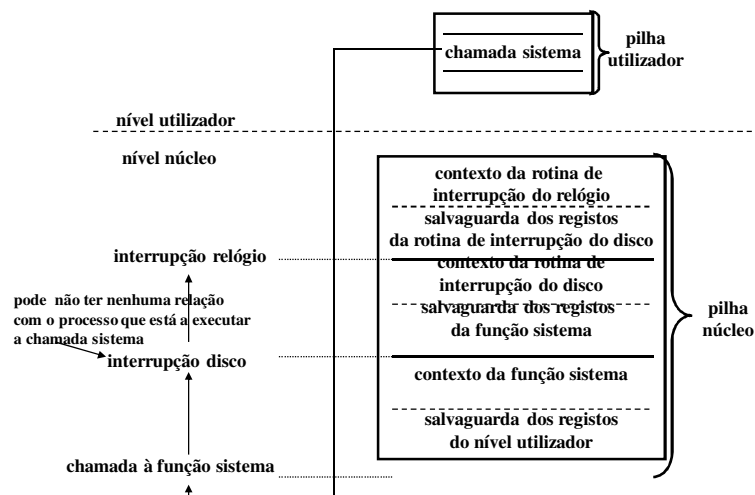
Porque são necessárias duas pilhas?

Sistemas Operativos – DE1 – IST

Pilha Utilizador/Pilha Núcleo

- Porque são necessárias duas pilhas?
 - A pilha em modo utilizador é a base da computação dos programas
 - A pilha em modo núcleo tem de ser diferente para garantir a estanquicidade de informação entre a actividade das funções do núcleo e do utilizador
 - A pilha em modo núcleo está vazia quando o processo passa para modo utilizador e contém o contexto de invocação das funções do núcleo quando está em modo núcleo.
 - Como o processo se pode bloquear no núcleo tem de ser uma por processo para permitir guardar de forma independente o contexto
 - As interrupções do hardware também tem de ser servidas numa pilha
 - é uma decisão de desenho ter uma pilha separada ou utilizar como “hospedeiro” a pilha núcleo do processo corrente.

Utilização da Pilha Núcleo



Escalonamento

- Prioridades em modo utilizador
 - O escalonamento é preemptivo
 - As prioridades em modo utilizador são calculadas dinamicamente em função do tempo de processador utilizado.
 - Quando o processo se bloqueia no núcleo é-lhe atribuída uma prioridade em modo núcleo
- Prioridades em modo núcleo
 - O processo em modo núcleo não é comutado
 - As prioridades em modo núcleo são definidas com base no acontecimento que o processo está a tratar e são fixas
 - As prioridades núcleo são sempre superiores às prioridades utilizador.
 - Quando passa de modo núcleo para modo utilizador o sistema recalcula a prioridade do processo e atribui-lhe uma prioridade em modo utilizador

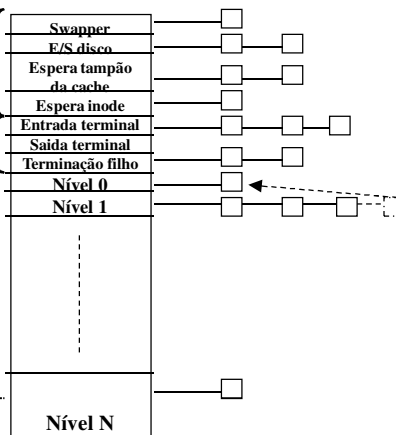
Unix: Prioridades

- Depende do tipo de recursos que o processo detém quando é bloqueado
- Definida quando o processo está bloqueado → Usada ao desbloquear

prioridades em modo núcleo

não podem receber signals

podem receber signals



- Actualizadas periodicamente
- Depende do tempo de processador já usado

prioridades em modo utilizador



Unix: Prioridades em Modo Utilizador

- ⇒ Corre processo mais prioritário durante quantum (100 ms)
- ⇒ Round-robin entre os mais prioritários
- ⇒ Ao fim de 1 segundo (50 “ticks” de 20 ms), recalcula prioridades:
- ⇒ Para cada processo:
 - ⇒ $\text{TempoProcessador} = \text{TempoProcessador} / 2$
 - ⇒ $\text{Prioridade} = \text{TempoProcessador} / 2 + \text{PrioridadeBase}$
- ⇒ Em cada tick, rotina de tratamento da interrupção incrementa p_cpu

Tempo (s)	Processo A		Processo B		Processo C	
	Prior.	T. Cpu	Prior.	T. Cpu	Prior.	T. Cpu
0	50	0	50	0	50	0
		50		-		-
1	62	25	50	0	50	0
		-		50		-
2	56	12	62	25	50	0
		-		-		50
3	53	6	56	12	62	25
		56		-		-
4	64	28	53	6	56	12

Sistemas Operativos – DEI - IST



Linux: Escalonamento

- Problema com algoritmo de escalonamento do Unix?
 - má escalabilidade com o número de processos a correr no sistema.
 - todos os segundos é necessário efectuar o cálculo das prioridades de todos os processos,
 - e este cálculo poder demorar algum tempo se for grande o número de processos em execução na máquina.
- Linux: Tempo dividido em épocas
 - Época termina quando todos os processos executáveis usaram o seu quantum (caso o pretendam)
- Processos diferentes têm durações do quantum diferentes
 - $\text{quantum} = \text{quantum_base} + \frac{\text{quantum_por_utilizar_epoca_anterior}}{2}$
 - Pode ser reduzido com chamadas sistema

Sistemas Operativos – DEI - IST



Linux: Prioridades

- Prioridade de um processo:
 - Linux: prioridade mais alta → mais prioritário
 - $Prio = prio_base + quantum_por_usar_nesta_época - nice$
- Processo mais prioritário é escolhido em primeiro lugar
- Melhora a escalabilidade. Porquê?
 - as durações de cada quantum são recalculadas uma vez por época, e
 - a duração da época aumenta à medida que aumenta o número de processos em execução simultânea
 - Versões iniciais ainda tinham problemas de escalabilidade, porque mantinham processos em lista única
 - Versão actual usa duas runQueues, cada uma uma multi-lista

Sistemas Operativos – DEI - IST



Linux: Escalonamento “Real-Time”

- Também é possível definir prioridades estáticas superiores às dinâmicas (modo utilizador) – classe “real-time”
- Necessárias permissões
- Não é um sistema de tempo-real
- A partir do 2.6.26 existe um escalonador virtual que faz escalonamento de escalonadores:
 - o primeiro dos quais é um escalonador de “Tempo Real”.

Sistemas Operativos – DEI - IST



Escalonamento: Chamadas Sistema

- `nice (int val)`
 - Decrementa a prioridade “val” unidades
 - Apenas superutilizador pode invocar com “val” negativo
- `int getpriority (int which, int id)`
 - Retorna prioridade do processo (ou grupo de procs.)
- `setpriority (int which, int id, int prio)`
 - Altera prioridade do processo (ou grupo de procs.)

Sistemas Operativos – DE1 - IST

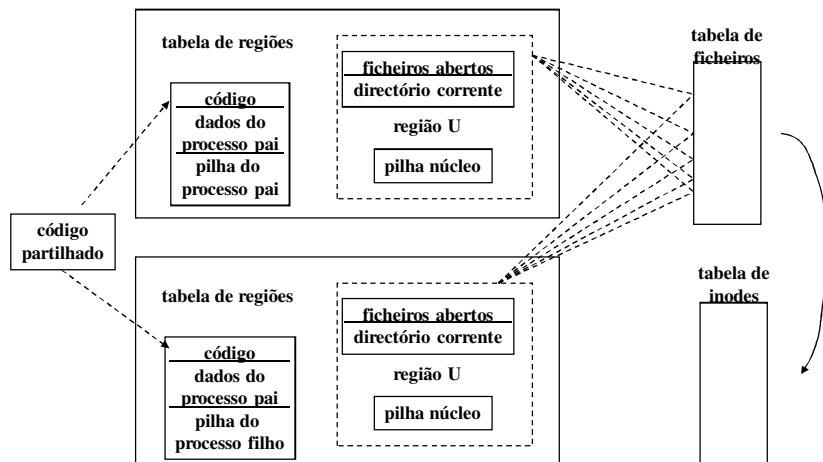


Criação de um Processo

- Reservar uma entrada na tabela `proc` (Unix) e verificar se o utilizador não excedeu o número máximo de subprocessos
- Atribuir um valor ao `pid`, normalmente um mero incremento de um inteiro mantido pelo núcleo
- Copiar a imagem do processo pai:
 - Dado que a região de texto é partilhada, apenas é incrementado o contador do número de utilizadores que acedem a essa região
 - As restantes regiões são copiadas (algumas incrementalmente)
 - retornar o valor do `pid` do novo processo para o processo pai, zero para o processo filho (coloca os valores apropriados nas respectivas pilhas)

Sistemas Operativos – DE1 - IST

Criação de um Processo (cont.)



Sistemas Operativos – DEI - IST

Terminação de um Processo

- Função exit:
 - fechar todos os ficheiros
 - libertar directório corrente
 - libertar regiões de memória
 - actualizar ficheiro com registo da utilização do processador, memória e I/O
 - enviar signal death of child ao processo pai (por omissão é ignorado)
 - registo proc / task mantém-se no estado zombie:
 - permite ao processo pai encontrar informação sobre o filho quando executa wait
- Função wait:
 - procura filho zombie
 - pid do filho e estado do exit são retornados através do wait
 - liberta a estrutura proc do filho
 - se não há filho zombie, pai fica bloqueado

Sistemas Operativos – DEI - IST



Execução de um Programa

- A função `exec` executa um novo programa no âmbito de um processo já existente:
 - Verifica se o ficheiro existe e é executável
 - Copia argumentos da chamada a `exec` da pilha do utilizador para o núcleo (pois o contexto utilizador irá ser destruído)
 - Liberta as regiões de dados e pilha ocupadas pelo processo e eventualmente a região de texto (se mais nenhum processo a estiver a usar)
 - Reserva novas regiões de memória
 - Carrega o ficheiro de código executável
 - Copia os argumentos da pilha núcleo para a pilha utilizador
- O processo fica no estado executável
- O contexto núcleo mantém-se inalterado:
 - identificação
 - ficheiros abertos

Sistemas Operativos – DEI - IST



Sincronização no Núcleo

- Vimos mecanismos que permitem às actividades a nível utilizador partilhar recursos de forma totalmente consistente
- Problema é genérico:
 - também se aplica a qualquer situação em que duas actividades concorrentes dentro do núcleo partilham recursos (aplica-se à maioria das tabelas e variáveis internas)
- Coloca-se quando:
 - Multiprocessadores
 - Uniprocessadores - interrupções

Sistemas Operativos – DEI - IST



Sincronização no Núcleo Linux

- os programadores do núcleo do SO têm ao seu dispor um conjunto de funções de sincronização para programar as secções críticas.
 - São variantes das Fechar_hard e Abrir_hard vistas anteriormente
 - mecanismo do tipo *spinlock* (ciclo de espera activa até que o trinco abra), e desactivam as interrupções

Porquê ambas ?

Multiprocessadores !

Sistemas Operativos – DEI - IST



Sincronização Interna: Wait Queues

- Objecto de sincronização do núcleo (e.g.: usado para bloquear processo num semáforo ou à espera do disco)
- Duas primitivas: `sleep_on` e `wake_up`
- `sleep_on`: bloqueia sempre o processo
- `wake_up`: desbloqueia todos os processos
 - Alguns podem ter de voltar a bloquear-se

Sistemas Operativos – DEI - IST



Wait Queues: API Linux

- `init_waitqueue_head (&my_queue);`
- `sleep_on (&my_queue);`
- `wake_up (&my_queue);`
- `interruptible_sleep_on (&my_queue);`
- `wake_up_interruptible (&my_queue);`
- Estas funções são internas ao núcleo e não podem ser chamadas por código utilizador.

Sistemas Operativos – DEI - IST



Signals

- Envio de um signal:
 - O sistema operativo coloca a 1 o bit correspondente ao signal, este bit encontra-se no contexto do processo a quem o signal se destina.
 - Não é guardado o número de vezes que um signal é enviado
- Tratamento do signal
 - Unix verifica se há signals:
 - quando o processo passa de modo núcleo para modo utilizador
 - entra ou sai do estado bloqueado
 - Linux verifica quando processo comuta para estado EmExecução.
 - No descritor do processo encontra-se o endereço da rotina de tratamento de cada signal
 - A pilha de modo utilizador é alterada para executar a função de tratamento do signal.
 - A função de tratamento executa-se no contexto do processo que recebe o signal como se fosse uma rotina normal

Sistemas Operativos – DEI - IST



Gestor de Processos do Windows 2000

Sistemas Operativos – DEI - IST



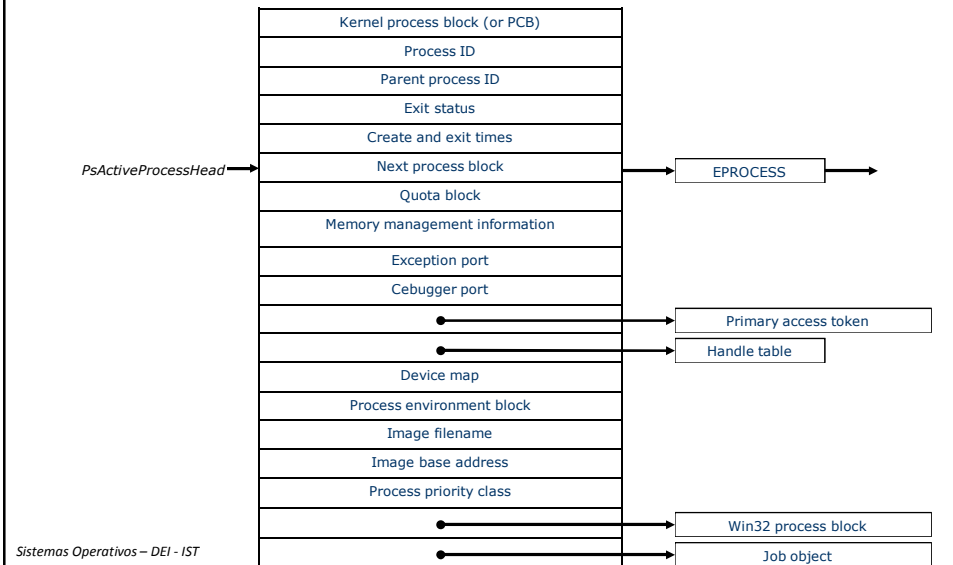
Objectos e Referências

- O sistema está estruturado internamente com base em objectos (e.g., processos, tarefas, ficheiros, trincos).
- Permitem:
 - Interface uniforme para acesso e partilha dos recursos do SO
 - Centralização das funções de segurança, autorização
 - Sistema simples de recolha automática dos objectos não necessários:
 - Gerir as referências para saber quando um objecto pode ser libertado porque ninguém o usa. (GC)

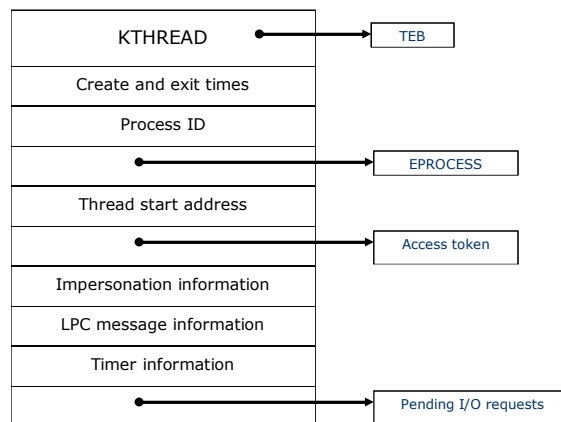
Sistemas Operativos – DEI - IST



Objecto Processo



Objecto Thread



Sistemas Operativos – DEI - IST

Prioridades

Dividem-se em dois grupos distintos

- Prioridades fixas para processos soft real time
- Prioridades variáveis para os processos interactivos

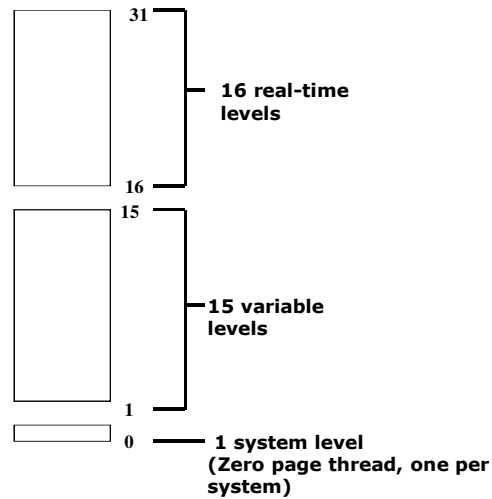
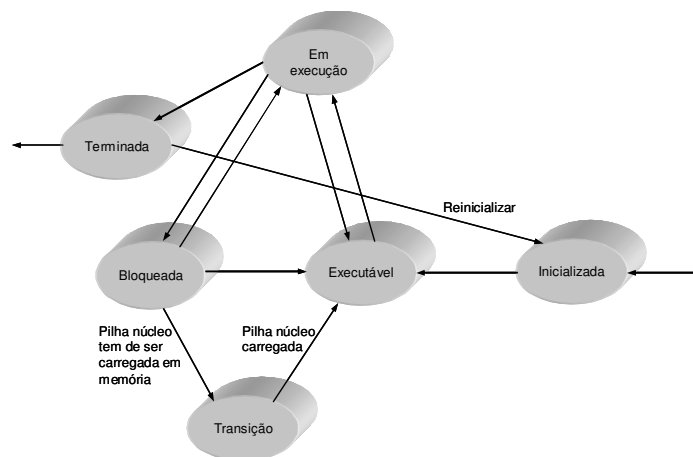
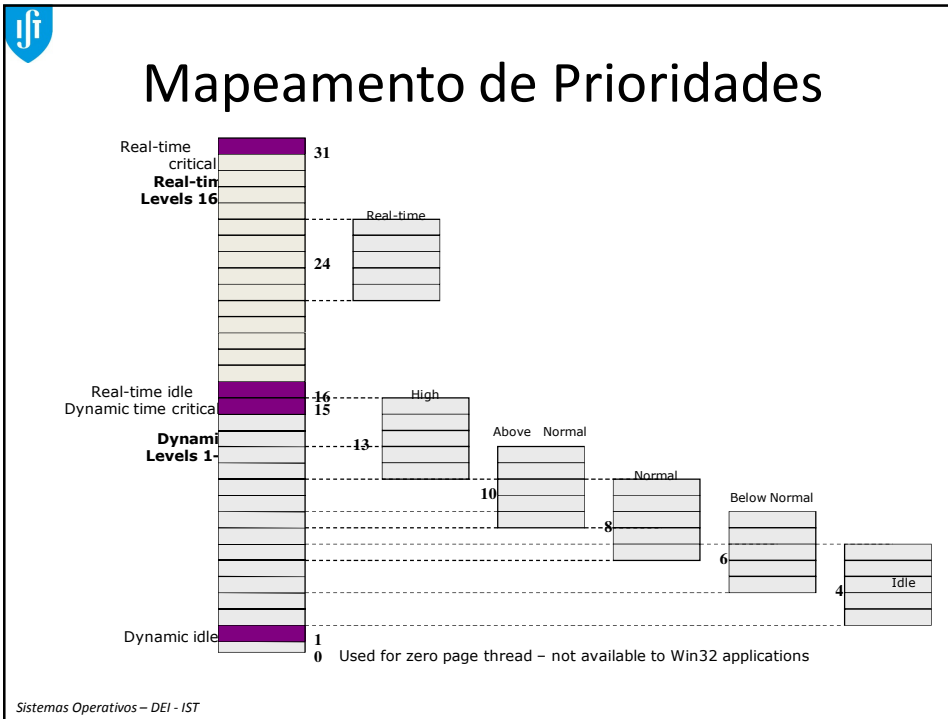


Diagrama de Estado das Tarefas em Windows





Valores de Aumento da Prioridade

Device	Boost
Disk, CD-ROM, parallel, video	1
Network, mailslot, named pipe, serial	2
Keyboard, mouse	6
Sound	8

Sistemas Operativos – DEI - IST