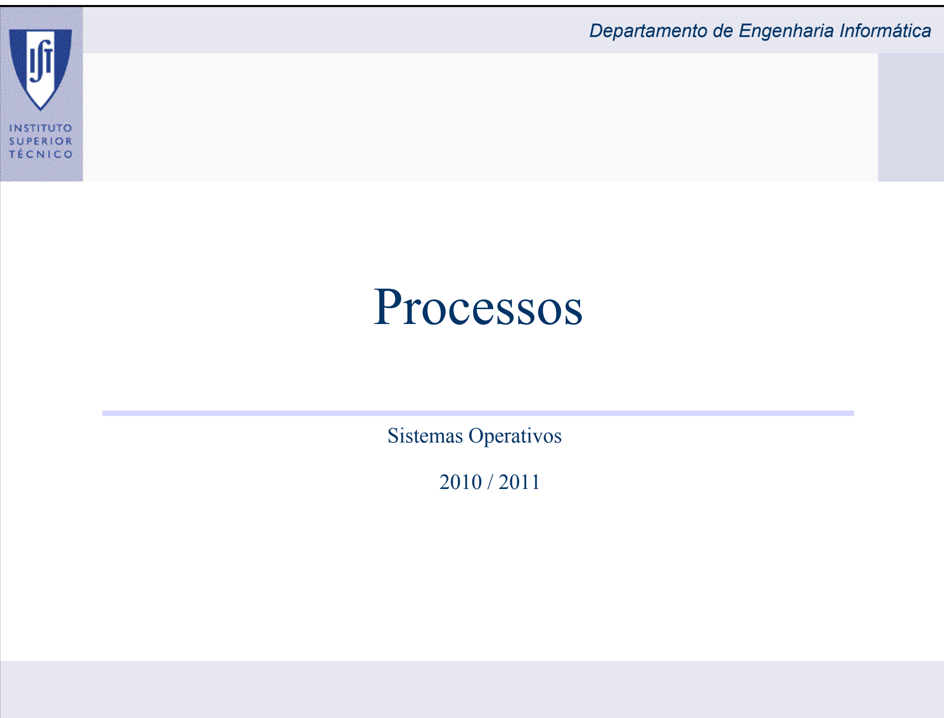
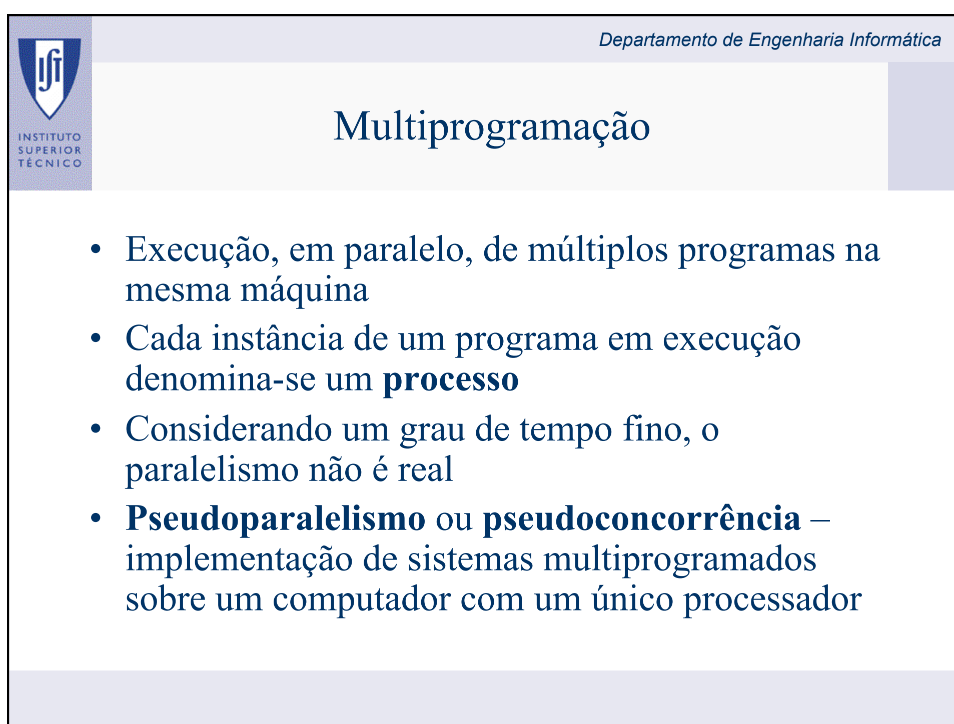
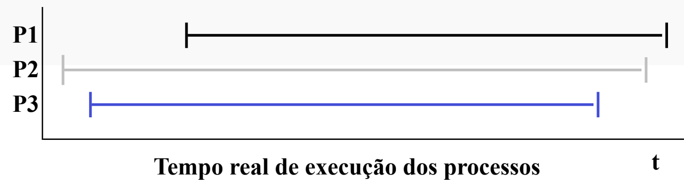
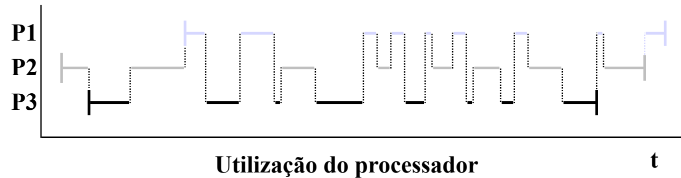

Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Processos
Sistemas Operativos
2010 / 2011


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Multiprogramação
Execução, em paralelo, de múltiplos programas na mesma máquina
Cada instância de um programa em execução denomina-se um processo
Considerando um grau de tempo fino, o paralelismo não é real
Pseudoparalelismo ou pseudoconcorrência – implementação de sistemas multiprogramados sobre um computador com um único processador

Pseudoconcorrência





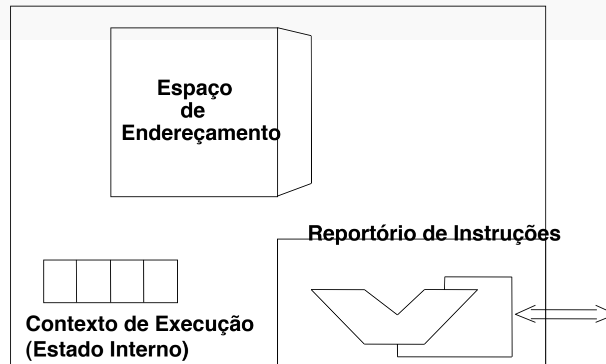
- Analogia com a Física
  - As leis da física quântica determinam a evolução de um sistema mas na maioria dos fenómenos as leis fisicas macroscópicas representam adequadamente o seu comportamento
- Analogia com a vida de um estudante do IST?

Departamento de Engenharia Informática

Processos vs. Programas

- Programa = Fich. executável (sem actividade)
- Um processo é um objecto do sistema operativo que suporta a execução dos programas
- Um processo pode, durante a sua vida, executar diversos programas
- Um programa ou partes de um programa podem ser partilhados por diversos processos (ex.: biblioteca partilhadas as DLL no Windows)

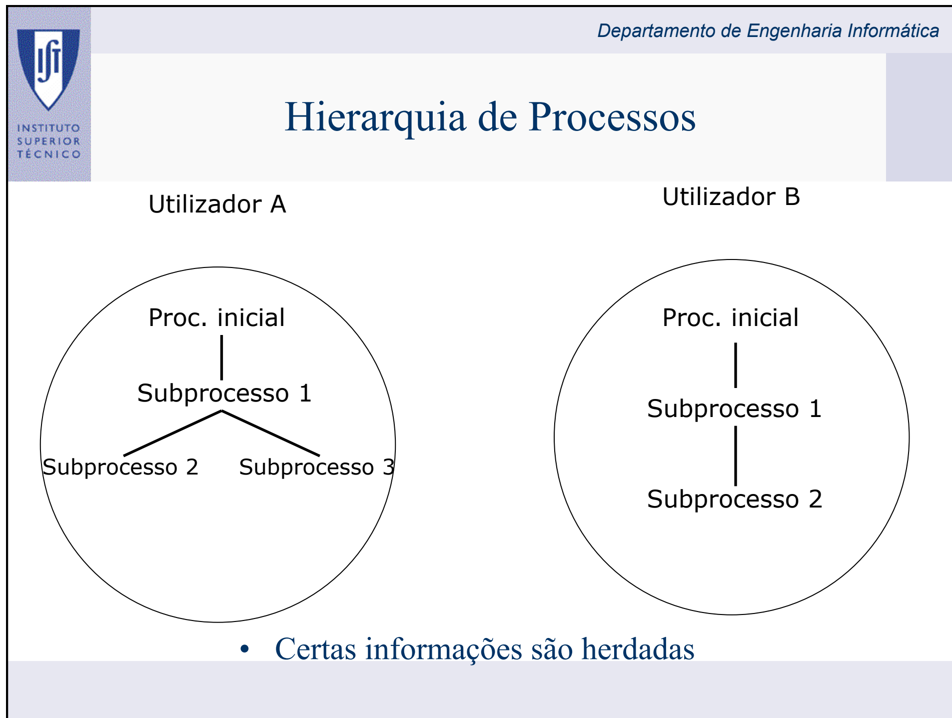
Processo Como Uma Máquina Virtual

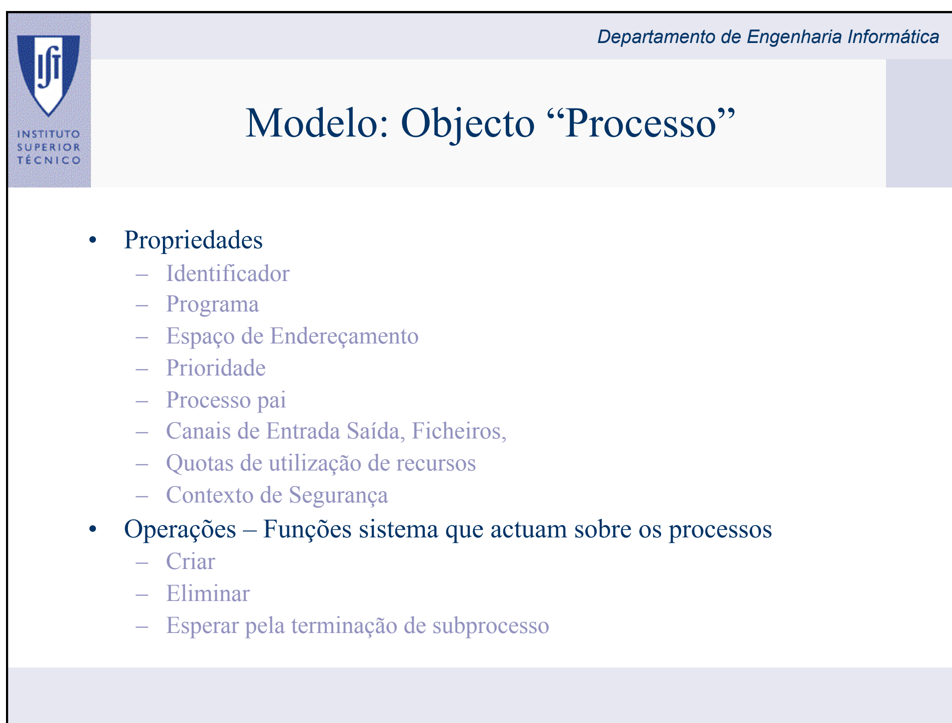


- Elementos principais da máquina virtual que o SO disponibiliza aos processos

Processo Como Uma Máquina Virtual

- Tal como um processador um processo tem:
  - **Espaço de endereçamento (virtual):**
    - Conjunto de posições de memória acessíveis
    - Código, dados, e pilha
    - Dimensão variável
  - **Reportório de instruções:**
    - As instruções do processador executáveis em modo utilizador
    - As funções do sistema operativo
  - **Contexto de execução (estado interno):**
    - Valor dos registos do processador
    - Toda a informação necessária para retomar a execução do processo
    - Memorizado quando o processo é retirado de execução

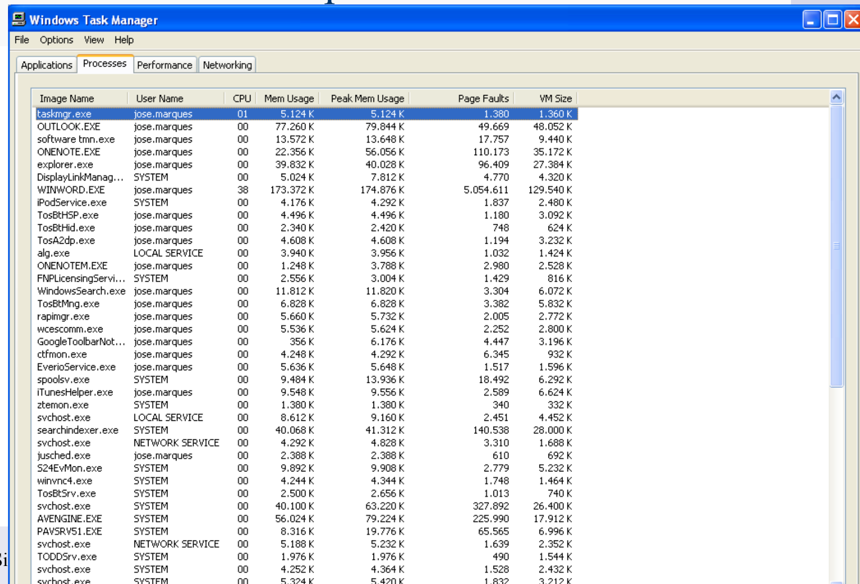

Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Hierarquia de Processos
Utilizador A
Utilizador B
Proc. inicial
Subprocesso 1
Subprocesso 2
Subprocesso 3
Proc. inicial
Subprocesso 1
Subprocesso 2
• Certas informações são herdadas


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Modelo: Objecto "Processo"
• Propriedades
– Identificador
– Programa
– Espaço de Endereçamento
– Prioridade
– Processo pai
– Canais de Entrada Saída, Ficheiros,
– Quotas de utilização de recursos
– Contexto de Segurança
• Operações – Funções sistema que actuam sobre os processos
– Criar
– Eliminar
– Esperar pela terminação de subprocesso

Exemplo: Unix

```
ps -ef | more
   UID    PID  PPID  C    STIME TTY    TIME CMD
  root      0     0  0   Sep 18 ?      0:17 sched
  root      1     0  0   Sep 18 ?      0:54 /etc/init -
  root      2     0  0   Sep 18 ?      0:00 pageout
  root      3     0  0   Sep 18 ?      6:15 fsflush
  root    418     1  0   Sep 18 ?      0:00 /usr/lib/saf/sac -t 300
daemon    156     1  0   Sep 18 ?      0:00 /usr/lib/nfs/statd
```


Exemplo: Windows



Windows Task Manager

File Options View Help

Applications | Processes | Performance | Networking

Image Name	User Name	CPU	Mem Usage	Peak Mem Usage	Page Faults	VM Size
taskmgr.exe	jose.marques	01	5.124 K	5.124 K	1.380	1.360 K
OUTLOOK.EXE	jose.marques	00	77.260 K	79.844 K	49.669	48.052 K
software tmn.exe	jose.marques	00	13.572 K	13.648 K	17.757	9.440 K
ONENOTE.EXE	jose.marques	00	22.356 K	56.056 K	110.173	35.172 K
explorer.exe	jose.marques	00	39.832 K	40.028 K	96.409	27.384 K
DisplayLinkManag...	SYSTEM	00	5.024 K	7.812 K	4.770	4.320 K
WINWORD.EXE	jose.marques	38	173.372 K	174.876 K	5.054.611	129.540 K
iPodService.exe	SYSTEM	00	4.176 K	4.292 K	1.837	2.480 K
TosBtHSP.exe	jose.marques	00	4.496 K	4.496 K	1.180	3.092 K
TosBtHid.exe	jose.marques	00	2.340 K	2.420 K	748	624 K
TosA2dp.exe	jose.marques	00	4.608 K	4.608 K	1.194	3.232 K
alg.exe	LOCAL SERVICE	00	3.940 K	3.956 K	1.032	1.424 K
ONENOTEM.EXE	jose.marques	00	1.248 K	3.788 K	2.980	2.528 K
FNPLicensingServi...	SYSTEM	00	2.556 K	3.004 K	1.429	816 K
WindowsSearch.exe	jose.marques	00	11.812 K	11.820 K	3.304	6.072 K
TosBtMng.exe	jose.marques	00	6.828 K	6.828 K	3.382	5.832 K
rapimgr.exe	jose.marques	00	5.660 K	5.732 K	2.005	2.772 K
wcescomm.exe	jose.marques	00	5.536 K	5.624 K	2.252	2.800 K
GoogleToolbarNot...	jose.marques	00	356 K	6.176 K	4.447	3.196 K
ctfmon.exe	jose.marques	00	4.248 K	4.292 K	6.345	932 K
EverioService.exe	jose.marques	00	5.636 K	5.648 K	1.517	1.596 K
spoolsv.exe	SYSTEM	00	9.484 K	13.936 K	18.492	6.292 K
iTunesHelper.exe	jose.marques	00	9.548 K	9.556 K	2.589	6.624 K
ztemon.exe	SYSTEM	00	1.380 K	1.380 K	340	332 K
svchost.exe	LOCAL SERVICE	00	8.612 K	9.160 K	2.451	4.452 K
searchindexer.exe	SYSTEM	00	40.068 K	41.312 K	140.538	28.000 K
svchost.exe	NETWORK SERVICE	00	4.292 K	4.828 K	3.310	1.688 K
jusched.exe	jose.marques	00	2.388 K	2.388 K	610	692 K
S24EvMon.exe	SYSTEM	00	9.892 K	9.908 K	2.779	5.232 K
winvnc4.exe	SYSTEM	00	4.244 K	4.344 K	1.748	1.464 K
TosBtSrv.exe	SYSTEM	00	2.500 K	2.656 K	1.013	740 K
svchost.exe	SYSTEM	00	40.100 K	63.220 K	327.892	26.400 K
AVENGINE.EXE	SYSTEM	00	56.024 K	79.224 K	225.990	17.912 K
PAVSRV51.EXE	SYSTEM	00	8.316 K	19.776 K	65.565	6.996 K
svchost.exe	NETWORK SERVICE	00	5.188 K	5.232 K	1.639	2.352 K
TODDSrv.exe	SYSTEM	00	1.976 K	1.976 K	490	1.544 K
svchost.exe	SYSTEM	00	4.252 K	4.364 K	1.528	2.432 K
svchost.exe	SYSTEM	00	5.324 K	5.420 K	1.832	3.212 K

Si
1.

Criação de um processo

```
IdProcesso = CriarProc (Código, Prioridade,... )
```

A função tem frequentemente diversos parâmetros: a prioridade, canais de entrada/saída, ...

Quando a criação tem sucesso o sistema atribuí um identificador interno (PID) ao processo que é retornado na função

Na criação de um processo tem de ficar definido qual é o programa que o processo vai executar. Normalmente é especificado um ficheiro contendo um programa executável.

Eliminação de processos

- Eliminação do processo quando o seu programa termina, libertando todos os recursos e estruturas de dados detidas pelo processo

```
Sair ([Estado])
```

- Eliminação de outro processo

```
EliminarProc ( IdProcesso )
```

O processo cujo identificador é passado como parâmetro é eliminado. O núcleo do SO valida se o processo que invoca esta função tem privilégios para a poder executar

Terminação do Processo Filho

- Em numerosas situações o processo pai pode querer bloquear-se esperando a terminação de um processo filho

```
Estado = EsperarTerminacao (Idprocesso)
```

O processo pai pode esperar por um processo específico ou genericamente por qualquer processo

Modelo de Segurança

- Um processo em execução tem de estar associado a um Utilizador (entidade que pode ser responsabilizada pelos seus actos)
- Os utilizadores são representados no sistema por um código que os identifica (User IDentifier – UID)
- Para facilitar a partilha o utilizador pode pertencer a um grupo ou grupos de utilizadores (identificador por um GID)

Controlo dos Direitos de Acesso

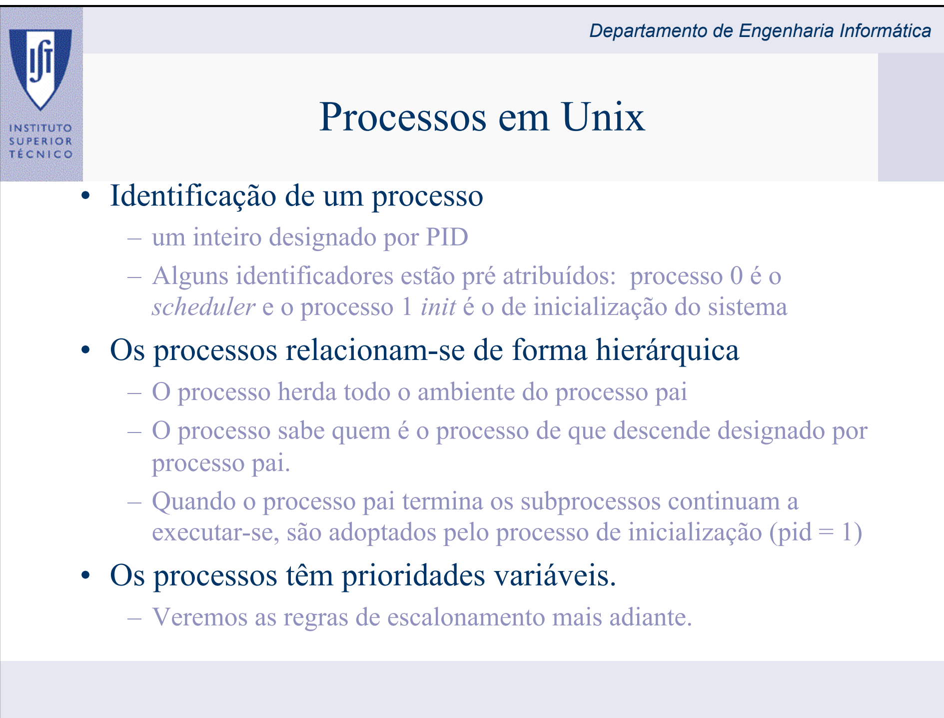
- Autorização - operação que valida os direitos do utilizador sobre um recurso antes deste poder executar uma operação sobre ele.
- A autorização baseia-se conceptualmente numa Matriz de Direitos de Acesso

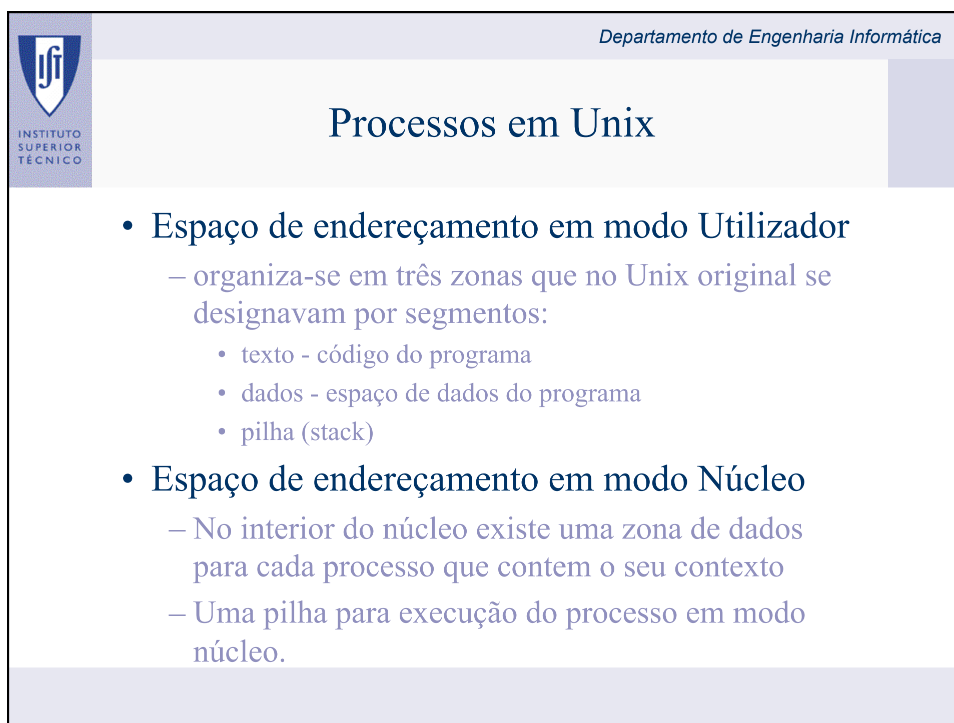
		Objectos	
Utilizadores	**1**	**2**	**3**
1	Ler	-	Escrever
2	-	Ler/ Escrever	-
3	-	-	Ler

- Para um dado objecto a coluna da matriz define a Lista de Direitos de Acesso (ACL)
- Para um dado utilizador a linha respectiva define todos os seus direitos normalmente designados por Capacidade

Unix – Processos

(Sob o ponto de vista do utilizador)


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Processos em Unix
• Identificação de um processo
– um inteiro designado por PID
– Alguns identificadores estão pré atribuídos: processo 0 é o scheduler e o processo 1 init é o de inicialização do sistema
• Os processos relacionam-se de forma hierárquica
– O processo herda todo o ambiente do processo pai
– O processo sabe quem é o processo de que descende designado por processo pai.
– Quando o processo pai termina os subprocessos continuam a executar-se, são adoptados pelo processo de inicialização (pid = 1)
• Os processos têm prioridades variáveis.
– Veremos as regras de escalonamento mais adiante.


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Processos em Unix
• Espaço de endereçamento em modo Utilizador
– organiza-se em três zonas que no Unix original se designavam por segmentos:
• texto - código do programa
• dados - espaço de dados do programa
• pilha (stack)
• Espaço de endereçamento em modo Núcleo
– No interior do núcleo existe uma zona de dados para cada processo que contem o seu contexto
– Uma pilha para execução do processo em modo núcleo.

INSTITUTO SUPERIOR TÉCNICO

Processos em Unix

- Cada processo também tem associado um contexto de execução acessível em modo utilizador e que contém diversas variáveis úteis para os programas utilitários ou para as aplicações.
- Exemplo:

```
HOME=/usr/jam
SHELL=/bin/csh
USER=jamarques
PATH=/usr/jam/bin/:/usr/local/bin:/bin
```

- Este contexto é herdado do processo pai e pode ser modificado livremente porque reside no espaço utilizador.
- Nos programas em C é acessível através do parâmetro do main ou de uma variável externa:

```
main (arc, arv, envp)
extern char **environ
```

Departamento de Engenharia Informática

INSTITUTO SUPERIOR TÉCNICO

Criação de um Processo

```
id = fork()
```

A função não tem parâmetros, em particular o ficheiro a executar. A imagem do novo processo é uma cópia da do criador.

O contexto do processo pai é copiado para o filho

A função retorna o PID do processo.

Este parâmetro assume valores diferentes consoante o processo em que se efectua o retorno:

- ao processo pai é devolvido o "pid" do filho
- ao processo filho é devolvido 0
- -1 em caso de erro

Retorno de uma função com valores diferentes → não existente na programação sequencial

Exemplo de fork

```
main() {
    int pid;

    pid = fork();
    if (pid == 0) {
        /* código do processo filho */
    } else {
        /* código do processo pai */
    }

    /* instruções seguintes */
}
```


Terminação do Processo

- Termina o processo, liberta todos os recursos detidos pelo processo, ex.: os ficheiros abertos
- Assinala ao processo pai a terminação

```
void exit (int status)
```

Status é um parâmetro que permite passar ao processo pai o estado em que o processo terminou.

Normalmente um valor negativo indica um erro

Terminação do Processo

- Em Unix existe uma função para o processo pai se sincronizar com a terminação de um processo filho
- Bloqueia o processo pai até que um dos filhos termine

```
int wait (int *status)
```

Retorna o pid do processo terminado. O processo pai pode ter vários filhos sendo desbloqueado quando um terminar

Devolve o estado de terminação do processo filho que foi atribuído no parâmetro da função exit

Exemplo de Sincronização entre o Processo Pai e o Processo Filho

```
main () {
    int pid, estado;

    pid = fork ();
    if (pid == 0) {
        /* algoritmo do processo filho  */
        exit(0);
    } else {
        /* o processo pai bloqueia-se à espera da
           terminação do processo filho  */
        pid = wait (&estado);
    }
}
```

Execução de um Programa

- O **fork** apenas permite lançar processo com o mesmo código → problemas?
- A função **exec** permite substituir a imagem do processo onde é invocada pela contida num ficheiro executável.
- Não há retorno numa chamada com sucesso.
- Parâmetros: valores que são passados para os parâmetros de entrada na função main do código a executar.
- Os ficheiros mantêm-se abertos.

Execução de um Programa

```
int execl(char* ficheiro, arg0, argl,…, argn,0)

int execv(char* ficheiro, *argv [])
```

Caminho de acesso ao ficheiro executável

Argumentos para o novo programa. Podem ser passado como apontadores individuais ou como um array de apontadores. Estas parâmetros são passados para a função main do novo programa e acessíveis através do argv

Exemplo de Exec

```
main ()
{
    int pid;

    pid = fork ();
    if (pid == 0) {
        execl ("/bin/who", "who", 0);
        /* controlo deveria ser transferido para o novo
             programa  */
        printf ("Erro no execl\n");
        exit (-1);
    } else {
        /* algoritmo do proc. pai */
    }
}
```

Por convenção o arg0 é o nome do programa

Shell

- O shell constitui um bom exemplo da utilização de fork e exec (*esqueleto muito simplificado*)

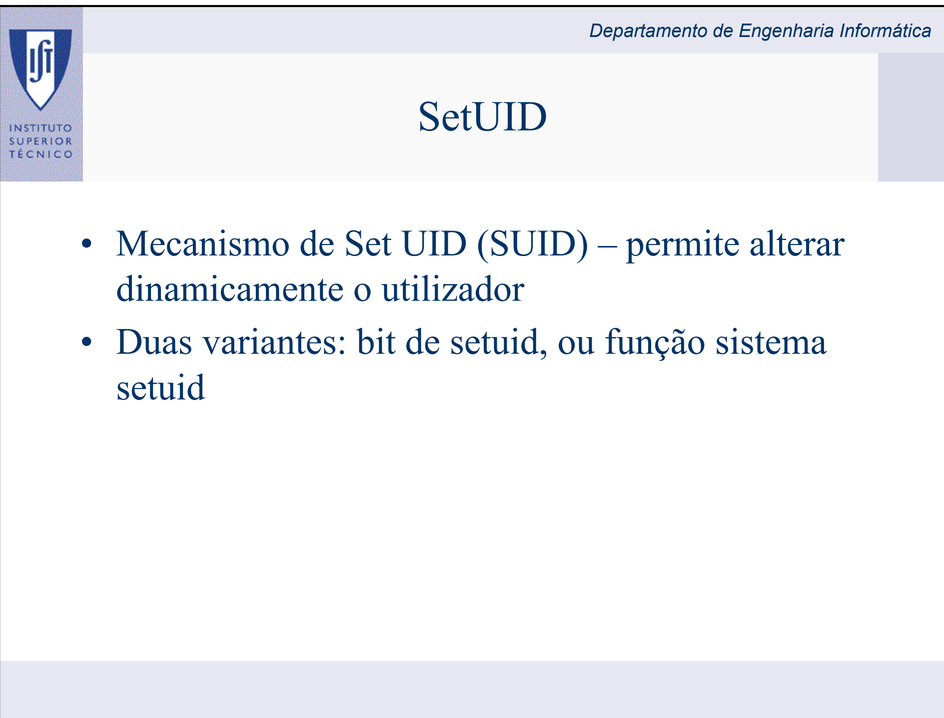
```
while (TRUE){
    prompt();
    read_command (command, params);

    pid = fork ();
    if (pid < 0) {
        printf ("Unable to fork"):
         continue;
    }
    if (pid !=0) {
        wait(&status)
    } else{
        execv (command, params):
    }
}
```

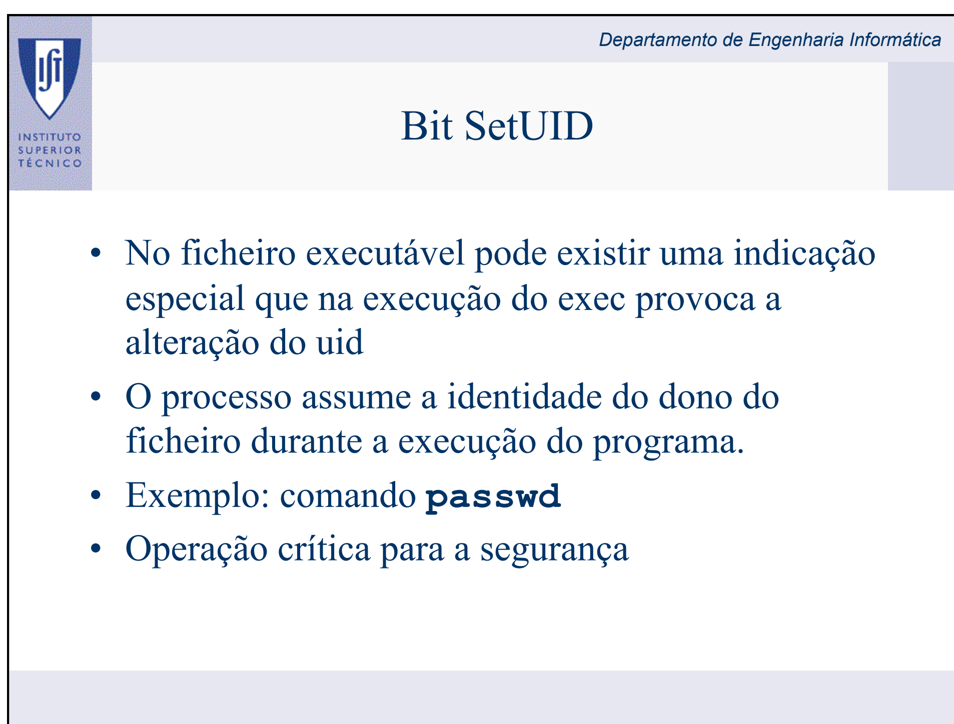
Autenticação

- Um processo tem associados dois identificadores que são atribuídos quando o utilizador efectua o login (se autentica) perante o sistema:
  - o número de utilizador UID - *user identification*
  - o número de grupo GID - *group identification*
- Os UID e GID são obtidos do ficheiro **`/etc/passwd`** no momento do *login*
- O UID e o GID são herdados pelos processos filhos
- *superuser* é um UID especial – zero. Normalmente está associado ao utilizador root (privilegiado).

Protecção no Acesso aos Recursos

- A protecção dos recursos em Unix é uma versão simplificada do modelo de Listas de Controlo de Acesso (ACL)
- Para um recurso (ficheiro, socket, etc.) a protecção é definida em três categorias:
  - Dono (*owner*): utilizador que normalmente criou o recurso
  - Grupo (*group*): conjunto de utilizadores com afinidades de trabalho que justificam direitos semelhantes
  - Restantes utilizadores (*world*)


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
SetUID
• Mecanismo de Set UID (SUID) – permite alterar dinamicamente o utilizador
• Duas variantes: bit de setuid, ou função sistema setuid


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Bit SetUID
• No ficheiro executável pode existir uma indicação especial que na execução do exec provoca a alteração do uid
• O processo assume a identidade do dono do ficheiro durante a execução do programa.
• Exemplo: comando passwd
• Operação crítica para a segurança

Funções Sistema de identificação

- *Real UID e GID* – UID e GID originais do processo
- *Effective UID e GID* – usado para verificar permissões de acesso e que pode ter sido modificado pelo setuid

`getpid()` **- devolve a identificação do processo**

`getuid()`, `getgid()`

devolvem a identificação real do utilizador

`geteuid()`, `getegid()`

devolvem a identificação efectiva do utilizador

`setuid(uid)`, `setgid(gid)`

altera a identificação efectiva do utilizador para uid e gid

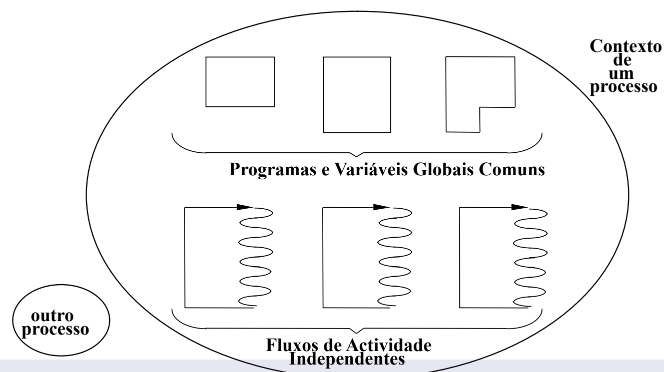
só pode ser invocada por processos com privilégio de superutilizador

Tarefas (Threads)

Múltiplos fluxos de execução no mesmo processo

Tarefas

- Mecanismo simples para criar fluxos de execução independentes, partilhando um contexto comum



Tarefas vs. Processos

- Porque não usar processos?
  - Processos obrigam ao isolamento (espaços de endereçamentos disjuntos) → dificuldade em partilhar dados (mas não impossível… exemplos?)
  - Eficiência na criação e comutação

Tarefas: Exemplos de Utilização

- Servidor (e.g., web)
- Aplicação cliente de correio electrónico
- Quais as tarefas em cada caso?

Modelos Multitarefa no Modelo Computacional

- Operações sobre as Tarefas

```
IdTarefa = CriarTarefa(procedimento);
```

A tarefa começa a executar o procedimento dado como parâmetro e que faz parte do programa previamente carregado em memória

```
EliminarTarefa (IdTarefa);
EsperaTarefa (IdTarefa)
```

Bloqueia a tarefa à espera da terminação de outra tarefa ou da tarefa referenciada no parâmetro Idtarefa

Interface POSIX

```
err = pthread_create (&tid, attr, function, arg)
```

- &tid: **Apontador para o identificador da tarefa**
- attr: **Utilizado para definir atributos da tarefa como a prioridade**
- function: **Função a executar**
- arg: **Parâmetros para a função**

```
pthread_exit(void *value_ptr)
```

```
int pthread_join(pthread_t thread, void **value_ptr)
```

- se a tarefa alvo não terminou a tarefa continua, senão bloqueia-se

INSTITUTO SUPERIOR TÉCNICO
Departamento de Engenharia Informática

Exemplo (sequencial)

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];
int nsomas;

void *soma_linha (int *linha) {
  int c, soma=0;
  int *b = linha;
  for (c = 0; c < TAMANHO - 1; c++) {
    soma += b[c];
    nsomas++;
  }

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

```
int main (void) {
  int i,j;

  for (i=0; i<N; i++){
    for (j=0; j< TAMANHO - 1; j++)
      buffer[i] [j] =rand()%10;
  }

 for (i=0; i< N; i++)
   soma_linha(buffer[i]);


 imprimeResultados(buffer);

exit(0);
}
```

Exemplo (paralelo)

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];
int nsomas;

void *soma_linha (int *linha) {
  int c, soma=0;
  int *b = linha;
  for (c = 0; c < TAMANHO - 1; c++) {
    soma += b[c];
    nsomas++;
  }

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```

Departamento de Engenharia Informática

Exemplo (paralelo)


```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define N 5
#define TAMANHO 10

int buffer [N] [TAMANHO];
int nsomas;

void *soma_linha (int *linha) {
  int c, soma=0;
  int *b = linha;
  for (c = 0; c < TAMANHO - 1; c++) {
    soma += b[c];
    nsomas++;
  }

  b[c]=soma; /* soma->ult.col.*/
  return NULL;
}
```


```
int main (void) {
  int i,j;
  pthread_t tid[N];

  /* inicializa buffer … */

 for (i=0; i< N; i++){
   if(pthread_create (&tid[i], 0,soma_linha,
                     (void *) buffer[i])== 0) {
      printf ("Criada a tarefa %d\n", tid[i]);
   }
   else {
     printf("Erro na criação da tarefa\n");
     exit(1);
   }
 }

 for (i=0; i<N; i++){
   pthread_join (tid[i], NULL);
 }
 printf ("Terminaram todas as threads\n");

 imprimeResultados(buffer);

 exit(0);
}
```

Programação num ambiente multitarefa

- As tarefas partilham o mesmo espaço de endereçamento e portanto têm acesso às mesmas variáveis globais.
- A modificação e teste das variáveis globais tem de ser efectuada com precauções especiais para evitar erros de sincronização.
- Veremos no cap. 4 a forma de resolver estes problema com objectos de sincronização.

Alternativas de Implementação

- Tarefas-núcleo
- Tarefas-utilizador (pseudotarefas)
  - Projecto da cadeira

Pseudotarefas (Tarefas-Utilizador)

- As tarefas implementadas numa biblioteca de funções no espaço de endereçamento do utilizador.
- Ideia proveniente das linguagens de programação.
- Núcleo apenas "vê" um processo.
- Processo guarda lista de tarefas, respectivo contexto

Pseudotarefas (Tarefas-Utilizador)

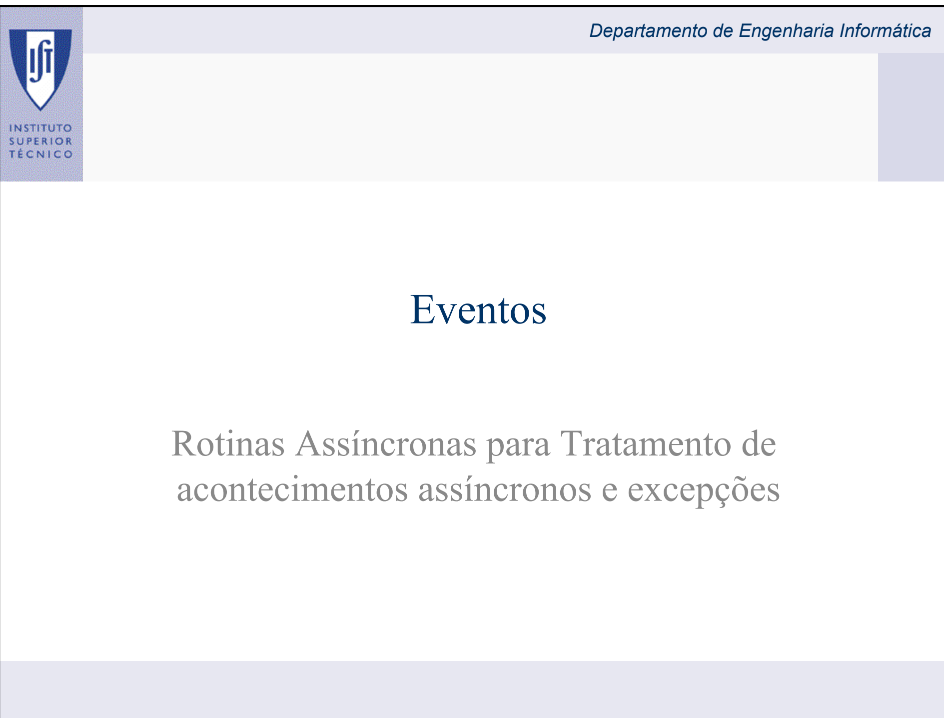
- A comutação entre tarefas explícita → função **`thread_yield`**
  - Pode ser contornado usando interrupções ("preempção")
- Problema: e se uma tarefa faz chamada bloqueante?
- Solução?

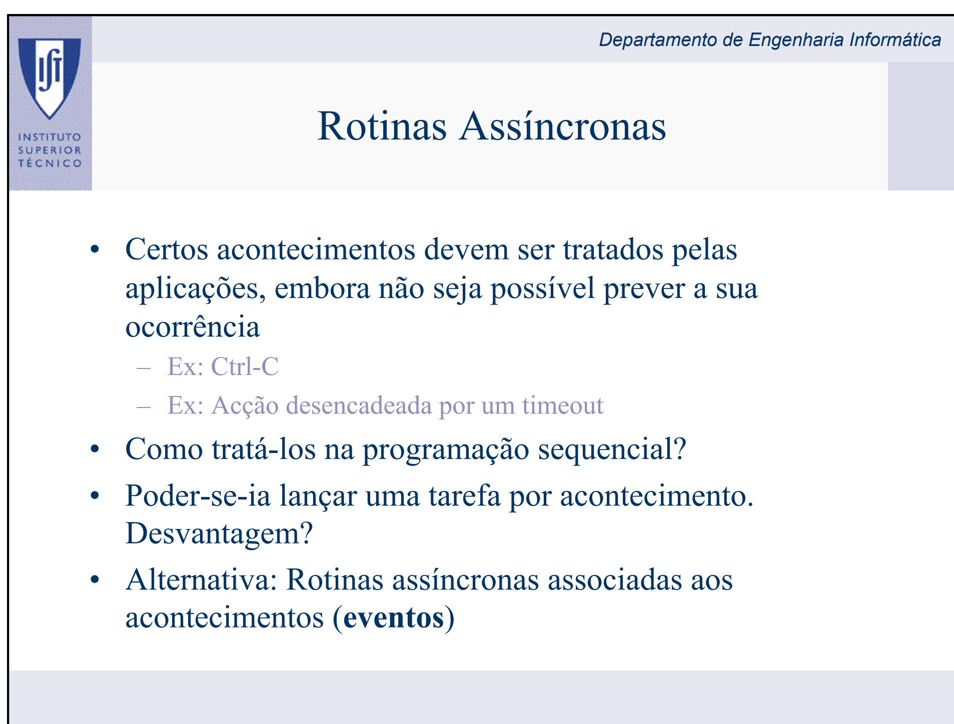
Tarefas-Núcleo (ou Tarefas Reais)

- Implementadas no núcleo do SO
  - Mais comuns
- Lista de tarefas e respectivo contexto são mantidos pelo núcleo

Comparação Tarefas Utilizador e Núcleo

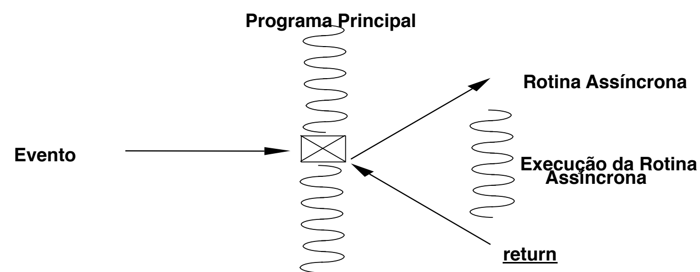
1. Capacidade de utilização em diferentes SOs?
2. Velocidade de criação e comutação? (vs. processos?)
3. Tirar partido de execução paralela em multiprocessadores?
4. Aproveitamento do CPU quando uma tarefa bloqueia (ex: ler do disco)?





Page 25

Modelo de Eventos



- Semelhante a outro conceito...

Rotinas Assíncronas

```
RotinaAssincrona (Evento,Procedimento)
```

Tem de existir uma tabela com os eventos que o sistema pode tratar

Identificação do procedimento a executar assincronamente quando se manifesta o evento.

Signals – Acontecimentos Assíncronos em Unix

Signal	Causa
SIGALRM	O relógio expirou
SIGFPE	Divisão por zero
SIGINT	O utilizador carregou na tecla para interromper (normalmente o CNTL-C
SIGQUIT	O utilizador quer terminar o processo e provoca
SIGKILL	Signal para terminar o processo. Não pode ser tratado
SIGPIPE	O processo escreveu para um pipe que não tem receptores
SIGSEGV	Acesso a uma posição de memória inválida
SIGTERM	O utilizador pretende terminar ordeiramente o proc
SIGUSR1	Definido pelo utilizador
SIGUSR1	Definido pelo utilizador

Excepção

Interacção com o terminal

Desencadeado por interrupção HW

Explicitamente desencadeado por outro processo

- Definidos em **`signal.h`**

Tratamento dos Signals

3 Possibilidades:

- Terminar o processo.
- Ignorar signal.
  - Alguns signals como o SIGKILL não podem ser ignorados. Porquê?
- Correr rotina de tratamento (handler)
  - Associamos rotina de tratamento a signal pela função sistema **`signal`**

Cada signal tem um tratamento por omissão, que pode ser terminar ou ignorar

Chamada Sistema "Signal"

```
void (*signal (int sig, void (*func)(int))) (int);
```

- void (*signal: A função retorna um ponteiro para função anteriormente associada ao signal
- int sig: Identificador do signal para o qual se pretende definir um handler
- void (*func)(int): Ponteiro para a função ou macro especificando:
  - SIG_DFL – acção por omissão
  - SIG_IGN – ignorar o signal
- (int): Parâmetro para a função de tratamento

Departamento de Engenharia Informática

Exemplo do tratamento de um Signal

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

apanhaCTRLC () {
   char ch;
   printf ("Quer de facto terminar a execucao?\n");
   ch = getchar();
   if (ch == 's') exit(0);
   else {
      printf ("Entao vamos continuar\n");
      signal (SIGINT, apanhaCTRLC);
   }
}

main () {
   signal (SIGINT, apanhaCTRLC);
   printf("Associou uma rotina ao signal SIGINT\n");
   for (;;)
     sleep (10);
}
```

Chamada Sistema Kill

- Envia um signal ao processo
- Nome enganador. Porquê?

```
kill (pid, sig);
```

Identificador do processo

Se o pid for zero é enviado a todos os processos do grupo

Está restrito ao superuser o envio de *signals* para processos de outro user

Identificador do signal

Outras funções associadas aos signals

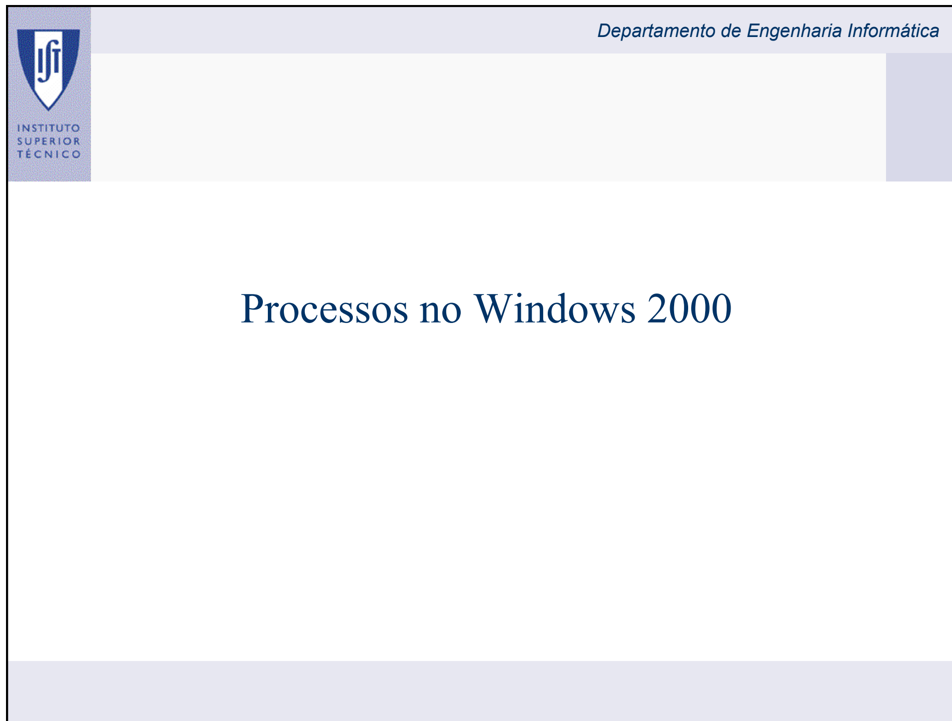
- `unsigned alarm (unsigned int segundos);`
  - o *signal* SIGALRM é enviado para o processo depois de decorrerem o número de segundos especificados. Se o argumento for zero, o envio é cancelado.
- `pause();`
  - aguarda a chegada de um *signal*
- `unsigned sleep (unsigned int segundos);`
  - A função faz um alarm e bloqueia-se à espera do signal

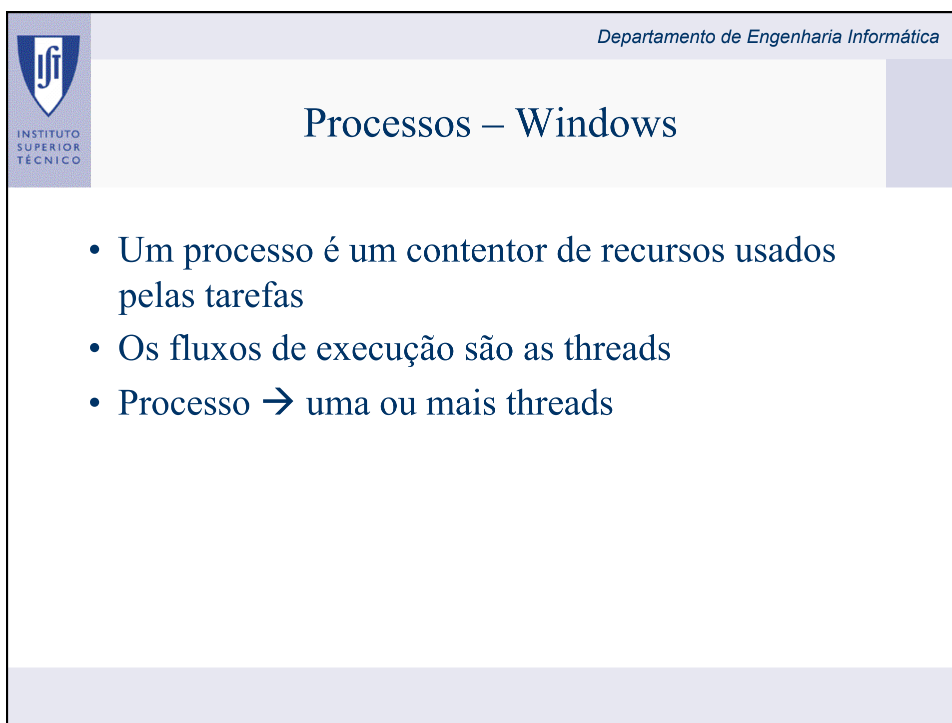
Versões Iniciais - Unix V e Unix BSD

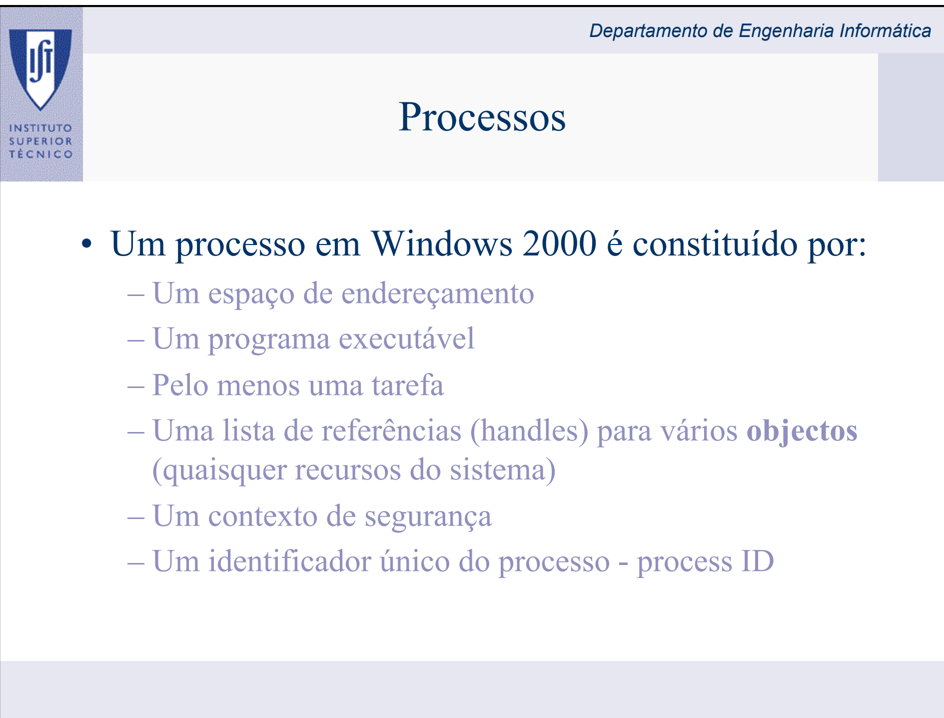
- System V:
  - A associação de uma rotina a um *signal* é apenas efectiva para uma activação
  - Depois de receber o *signal*, o tratamento passa a ser novamente o por omissão (necessário associar de novo)
  - Entre o lançamento de rotina de tratamento e a nova associação → tratamento por omissão
  - Solução: restabelecer a associação na primeira linha da rotina de tratamento
  - Pode gerar problemas se houver recepção sucessiva de *signals*
- BSD:
  - a recepção de um novo *signal* é inibida durante a execução da rotina de tratamento

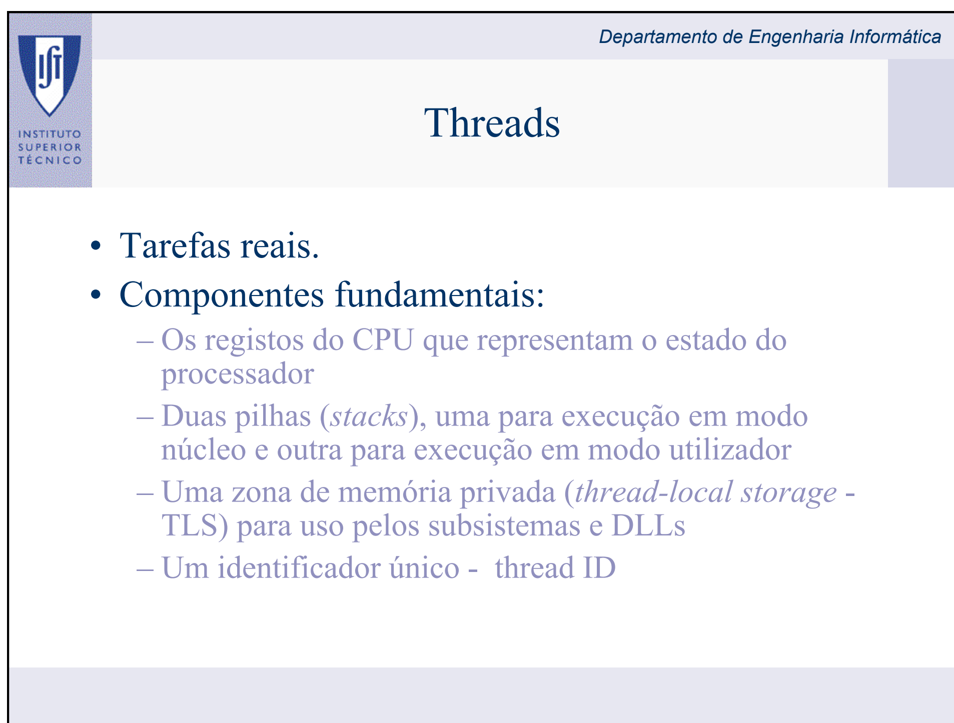
Função System

```
#include <signal.h>

int system(char *argv[]) {
   int pid, status;
   void (*del) (), (*quit) (); /* variaveis (ponteiros para f.) */

   del = signal (SIGINT, SIG_IGN);
   quit = signal (SIGQUIT, SIG_IGN);
   switch (pid = fork ()) {
      case 0 : signal (SIGINT, del);
               signal (SIGQUIT, quit);
               execl ("/bin/sh", "sh", "-c", argv, 0);
               exit (-1);
      case -1: /* ... */
      default: while (wait(&status) != pid);
   }
   signal (SIGINT, del);
   signal (SIGQUIT, quit);
}
```


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Processos no Windows 2000


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Processos – Windows
• Um processo é um contentor de recursos usados pelas tarefas
• Os fluxos de execução são as threads
• Processo → uma ou mais threads


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Processos
• Um processo em Windows 2000 é constituído por:
– Um espaço de endereçamento
– Um programa executável
– Pelo menos uma tarefa
– Uma lista de referências (handles) para vários **objectos** (quaisquer recursos do sistema)
– Um contexto de segurança
– Um identificador único do processo - process ID


Departamento de Engenharia Informática
INSTITUTO SUPERIOR TÉCNICO
Threads
• Tarefas reais.
• Componentes fundamentais:
– Os registos do CPU que representam o estado do processador
– Duas pilhas (*stacks*), uma para execução em modo núcleo e outra para execução em modo utilizador
– Uma zona de memória privada (*thread-local storage* - TLS) para uso pelos subsistemas e DLLs
– Um identificador único - thread ID

Fibers

- Pseudotarefas geridas no espaço de endereçamento do utilizador.
- Uma thread pode ter múltiplas fibers.
- Fibers não são vistas pelo núcleo
- As fibers são criadas e comutadas explicitamente com chamadas à biblioteca Win32 mas que não produzem chamadas ao sistema.

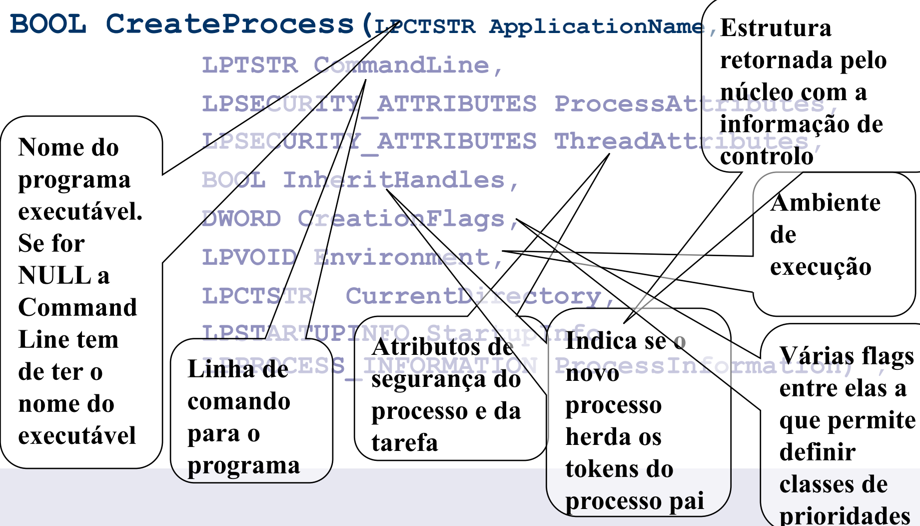
Jobs

- Grupo de processos
  - Permite gestão uniforme (e.g., terminar em conjunto)
- Um processo só pode ser associado a um job e em principio todos os seus descendentes pertencem ao mesmo job

Segurança

- O contexto de segurança de um processo ou de uma thread é um objecto designado *Access Token*
- Um *Access Token* regista os utilizadores, grupos, máquinas, e domínios que estão associados ao processo.
- Sempre que é acedido um objecto no sistema o *executive* valida o token contra uma ACL
- Acesso concedido se não exisitir nenhuma recusa, e existir pelo menos uma permissão num dos utilizadores, grupos, etc.

Criação de processos Win32



Criação de processos Win32

- Diferenças vs. fork+exec:
  - No Windows não se cria automaticamente uma relação pai-filho. Embora o processo pai fique com um handle para o filho.
  - Um processo tem associado uma thread (main thread).
  - Na criação do processo pode definir-se a classe de prioridade a que as threads do processo ficam associadas.
  - A criação com sucesso retorna um valor diferente de zero.

Criação de processos Win32

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
STARTUPINFO startInfo;
PROCESS_INFORMATION processInfo;
…
strcpy(lpCommandLine,
  "C:\\WINNT\\SYSTEM32\\NOTEPAD.EXE temp.txt");
ZeroMemory(&startInfo, sizeof(startInfo));
startInfo.cb = sizeof(startInfo);
If(!CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
  HIGH_PRIORITY_CLASS  CREATE_NEW_CONSOLE,
  NULL, NULL, &startInfo, &processInfo))  {
      fprintf(stderr, "CreateProcess failed on error %d\n",
      GetLastError());
      ExitProcess(1);
};
```

Eliminação de Processos

- Existem três formas para terminar um processo
  - Chamada à função ExitProcess que autotermina o processo
  - Chamada à função TerminateProcess que permite a um processo com o requerido privilégio terminar outro processo
  - Terminando todas as threads de um processo

Departamento de Engenharia Informática

Eliminação de Processos

```
void ExitProcess (UINT uExitCode);
```

• Informa todas as DLLs que o processo termina.
• Fecha todos os handles do processo
• Termina todas as threads

Código de terminação

```
bool TerminateProcess (Handle Processo, UINT uExitCode);
```

Criação Thread

```
HANDLE CreateThread (
        LPSECURITY_ATTRIBUTES ThreadAttr,
        DWORD stakSz,
        LPTHREAD_START_ROUTINE StrtAddr,
        LPVOID Parm,
        DWORD CreatFlgs,
        LPDWORD ThreadId);
```

Handle para a thread ou NULL se falhou

Atributos de segurança

Tamanho do stack

Endereço da função inicial

Um parâmetro que pode ser passado à thread

Esperar Pela Terminação de Subprocesso

- `WaitForSingleObject(handle, timeout)`
- Função genérica de espera sobre um objecto (entidade do sistema operativo)