Aula de apresentação de FSO

José A. Cardoso e Cunha
DI-FCT/UNL

Este texto resume o conteúdo da aula teórica.

1 Objectivo

O objectivo da aula foi a apresentação do programa e do regime de funcionamento da disciplina.

2 Motivação para o programa

O objectivo desta disciplina é fazer uma introdução aos Sistemas de Operação (SO) dos computadores, através do estudo da interface das *chamadas ao sistema*, tal como em AC estudámos a interface da arquitectura do computador, a nível das instruções da máquina hardware.

A figura 1 localiza a matéria de FSO, em relação aos níveis da hierarquia de um sistema de computação.

Cada nível ilustrado na figura define uma interface que pode ser acedida pelos níveis que lhe estão acima:

- O nível 'arquitectura do computador' define a linguagem máquina, para a qual os compiladores produzem o código que corresponde às instruções das linguagens de mais alto nível.
- O nível 'sistema de operação' define uma interface que disponibiliza um conjunto de funções, designadas habitualmente por *chamadas ao sistema*. Estas funções podem ser invocadas pelos programas de níveis superiores da hierarquia. Por exemplo, um compilador, ao transformar o texto de um programa Pascal em código executável, no caso de instruções de leitura e de escrita de ficheiros, tais como *READ* e *WRITE*, não precisa de gerar o código máquina correspondente àquelas acções: pode limitar-se a gerar uma instrução que invoque uma 'subrotina',

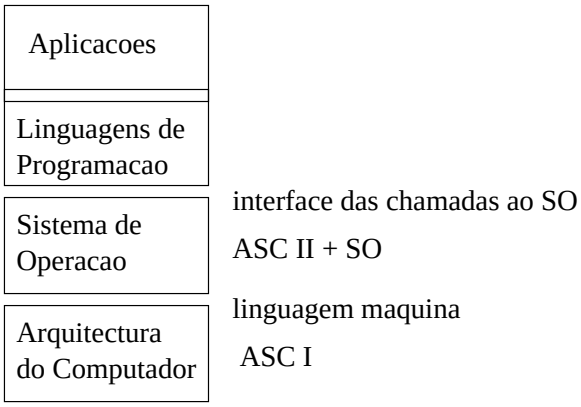

Figura 1: Níveis do Sistema de Computação

oferecida pelo sistema de operação, que realiza a acção pretendida. Estas 'subrotinas' fazem parte do código do sistema de operação.

Um Sistema de Operação é um conjunto de programas cujo objectivo é tornar mais fácil o desenvolvimento dos programas nos níveis superiores da hierarquia atrás ilustrada. Se não existisse um SO, seria necessário, para cada programa que se fizesse, incluir sempre as funções auxiliares para, por exemplo, aceder aos periféricos, para carregar o código e os dados e inicializar as zonas de pilha do programa em memória central, para gerir as transferências de páginas entre memória e disco, etc. Estas funções, correspondendo a *serviços* genéricos, isto é, que são necessários para a maioria das aplicações, passaram a fazer parte do SO, e ficam acessíveis automaticamente a qualquer programa.

Assim, o SO oferece uma interface definida por um conjunto de funções. Na inicialização de um computador, o código e as zonas de dados necessárias à execução das funções do SO, são inicializados em memória central. Estas funções, que designaremos por *chamadas ao sistema*, são acessíveis aos programas nas diversas linguagens de programação. Durante a execução, um programa pode invocar essas funções, como se fossem subrotinas, através de uma instrução máquina especial, mas que tem um efeito semelhante a $CALL$, como veremos mais adiante. A figura 2 ilustra esta interacção entre um programa numa linguagem L e o SO.

No nosso estudo dos SO, iremos identificar os diferentes tipos de chamadas ao SO oferecidas. Podemos, desde já, classificá-las em duas grandes categorias:

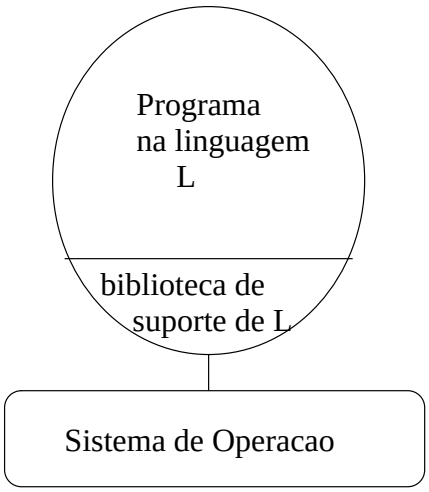

Figura 2: Interacção Linguagem–SO

1. as que se encarregam do acesso aos periféricos, efectuando as operações de entrada e saída e suportando a gestão das memórias secundárias – discos e bandas – para o arquivo da informação sob a forma de *ficheiros*; incluiremos estas funções sob a alçada do *Sistema de Ficheiros*;

2. as que se encarregam de controlar a execução dos programas, desde o seu carregamento e inicialização em memória, a supervisão da sua execução, quando por exemplo ocorrem erros, até à terminação dos programas.

Assim, numa primeira abordagem, o SO irá ser considerado como suportando dois conceitos principais: o de *Ficheiro*, associado à primeira categoria de funções, e o de *Processo*, associado à segunda categoria.

O conceito de ficheiro é mais familiar a um utilizador de computadores, pelo que deixaremos para mais adiante o seu estudo específico.

O conceito de processo é, simplesmente, uma forma que se encontrou para o SO poder gerir a execução dos programas. Considere-se um programa codificado em binário, na linguagem máquina de um dado computador. Este programa, arquivado num ficheiro, dito executável, é apenas uma lista de instruções e directivas, tendo, portanto, uma natureza estática. Quando um utilizador pretende activar a execução de um programa, desencadeia tipicamente uma sequência de passos que se resumem, de forma simplificada a seguir:

1. O utilizador indica o nome do ficheiro contendo o programa

2. O SO pesquisa o ficheiro indicado, em disco. Se não o encontra, assinala um erro. Se encontra, consulta informação de controlo que indica quantos bytes de código e de dados tem o programa, que zonas de dados devem ser inicializadas por omissão, qual o endereço da primeira instrução a executar (relativo ao início de algum segmento de código), etc.

3. Com base naquelas informações de controlo, o SO determina quantos bytes precisa de reservar, em memória central, para conter as zonas de código, de dados e de pilha, para a execução do programa. Pode, depois, carregar o programa em memória, desencadeando a sua leitura de disco (em geral, por DMA).

4. Uma vez completado o carregamento das regiões do programa em memória, poder-se-á iniciar a sua execução. Para isto, pensar-se-ia que bastaria executar uma instrução *JUMP* com destino determinado pelo endereço da primeira instrução do programa (endereço este, agora recolocado, isto é, ao qual se soma a base, correspondendo ao endereço real de memória onde o programa reside).

De facto, isto bastaria, se o SO apenas suportasse a execução dos programas, de forma sequencial: um programa, uma vez iniciado, continuaria a execução até se completar e, só depois disso, poderia o SO iniciar os passos para a activação de outros programas que o utilizador quisesse executar. Mesmo neste caso, conviria talvez fazer uma espécie de *CALL*, em vez de um *JUMP* para inciar a execução de cada programa, caso contrário, pergunta-se: como retornaria o controlo da execução ao SO, no fim da execução do programa? E, nesse caso, provavelmente a última instrução executada por um programa, deveria ser uma espécie de *RET*, que fizesse retornar ao SO, que o tinha invocado. Esta interacção SO–Programa, está ilustrada na figura 3.

Esta forma de interacção, em que o SO veria o Programa a executar como se fosse uma subrotina, não é muito adequada, por diversas razões:

1. E se o programa tiver erros, durante a sua execução? Por exemplo, se tiver um ciclo eterno, do qual nunca saia? Nesse caso, nunca retornaria ao SO e todo o sistema ficaria bloqueado.

2. E se o programa quiser ler dados de um periférico e o respectivo *buffer* de leitura estiver vazio? Todo o sistema ficará bloqueado, enquanto não houver dados para o programa ler? E além disso, durante esse tempo, o processador (CPU) não tem trabalho algum para fazer, embora possa haver outros programas que pudessem ser executados entretanto.

3. E se o programa quiser imprimir um ficheiro, também se perderá muito

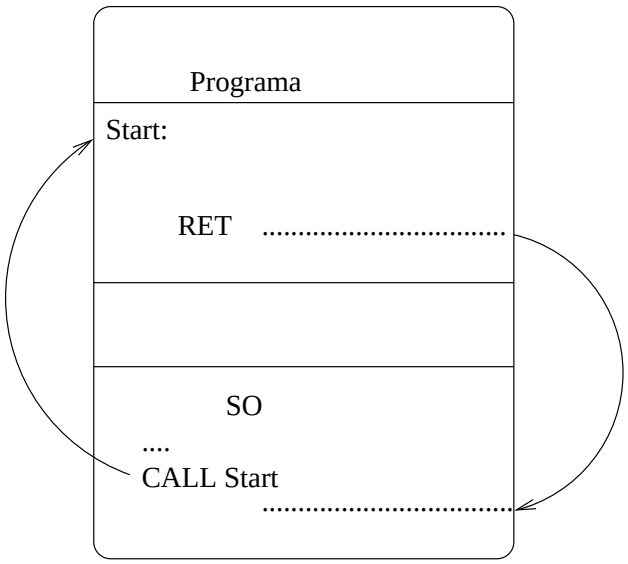

Figura 3: Interacção SO–Programa

tempo, que poderia ser aproveitado de forma mais útil, ocupando o processador na execução de outros programas.

Pelas razões apontadas, o SO tem de estar preparado para tratar as situações que, durante a execução de um programa, provocam a sua interrupção forçada, seja devido à ocorrência de um erro ou algum evento imprevisto, seja devido ao programa ter pedido operações de entrada ou saída, que demorem um tempo imprevisível a serem completadas:

- No caso de erros irrecuperáveis (eg, divisão por zero, violação de memória), o SO terá de receber o controlo da execução (isto é, um erro deve provocar o retorno da execução ao SO) e, provavelmente, terá de terminar a execução e informar o utilizador.

- No caso de eventos imprevistos, mas recuperáveis (eg, uma referência a um endereço de memória de uma página ausente de memória), o SO deve receber o controlo e tratar a situação respectiva (eg, trazer a página faltosa, de disco para memória).

- No caso de operações de entrada e saída 'demoradas', o SO deverá suspender temporariamente a execução daquele programa e, durante

esse tempo, activar um outro programa para execução (se houver algum à espera).

Para permitir o tratamento daquelas situações, o SO tem de saber, a cada momento, qual o programa que está em execução, e em que estados estão os outros programas que possam estar em situação de espera, seja da sua vez para execução, seja à espera de dados de um periférico, ou de páginas vindas de disco, etc. Assim, para cada programa cuja execução é pedida pelos utilizadores, o SO define um *processo* e regista a informação necessária em estruturas de dados em memória. Este conceito está ilustrado na figura 4.

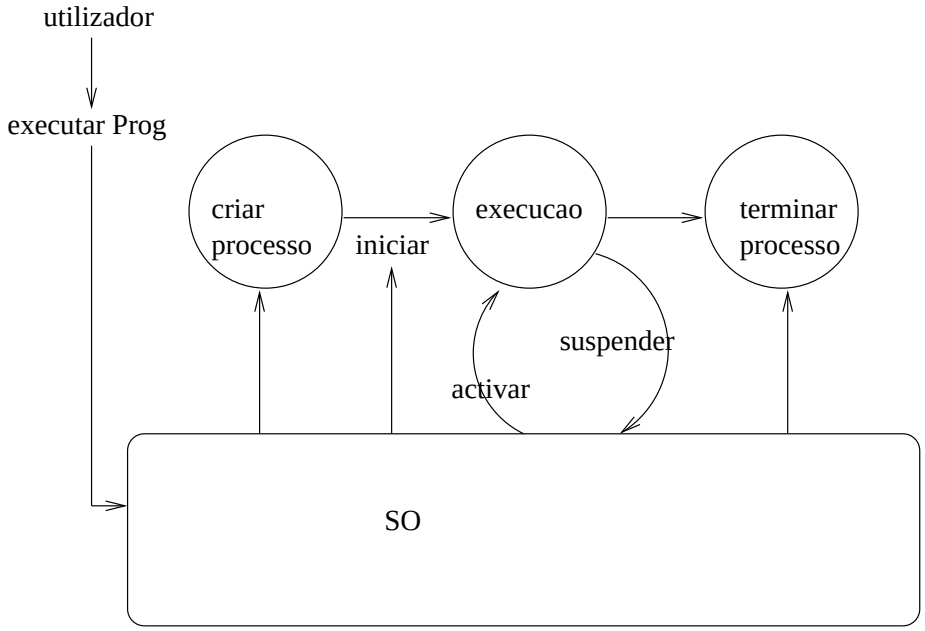

Figura 4: Vida de um Processo

Na figura vê-se que a execução de um programa pode ser temporariamente suspensa por diversas vezes, devido à ocorrência de situações como as ilustradas acima. Algumas dessas situações serão tratadas pelo SO e permitirão que, mais tarde, a execução do programa seja retomada. Outras situações, irrecuperáveis, levarão à terminação forçada da execução do programa.

Uma forma que temos para garantir que o SO possa controlar a execução dos programas, é baseada no mecanismo de *interrupções* que estudámos em ASC I. Assim:

- Todas as invocações de funções do SO (chamadas ao SO) são efectuadas através de uma instrução máquina que gera uma interrupção. Relembre a instrução *INT n* do processador 80x86, que gera uma interrupção do programa, quando é executada. Assim, após qualquer chamada ao SO, a execução salta para uma rotina de serviço de interrupções que faz parte do SO, responsável pelo seu tratamento. Este mecanismo de chamada é invisível ao programador de uma linguagem, por exemplo C, que a única coisa que 'vê' é uma chamada da função da biblioteca de suporte da linguagem C. É dentro desta biblioteca que a chamada é feita ao SO.

- Todas as situações de erro, como as descritas acima, geram uma interrupção da execução do programa, passando assim o controlo a uma rotina do SO, que decide o tratamento a efectuar. Se for caso disso, esta rotina decide retorna ao programa interrompido, de imediato (relembre a instrução do processador 80x86 *IRET*). Se o programa tiver de esperar, por exemplo, por falta de página, o SO decide activar entretanto outro programa, e suspende a execução do programa temporariamente (cujo estado foi salvaguardado devidamente, quando ocorreu a interrupção).

Estas interacções Processo – SO estão representadas na figura pelos arcos suspender/activar.

Ficamos por aqui, nesta introdução, cujo único objectivo foi tentar explicar por que motivo o conceito de processo é tão importante no estudo dos sistemas de operação dos computadores.

3 O Programa

O programa proposto tem os seguintes capítulos principais:

1. Introdução aos Sistemas de Operação (SO)
2. Conceito de Processo
3. Sistemas de Ficheiros e Programação de I/O
4. Controlo de Processos
5. Programação Concorrente
6. Chamadas ao Sistema Unix

No fim da disciplina, o aluno deve saber:

- o que é um sistema de operação
- que serviços/funções oferece
- que tipos de SO existem
- que interfaces de utilização disponibilizam
- que funções dispõem para as *chamadas ao sistema*
- como realizam os conceitos de *ficheiro* e de *processo*
- como se desenvolvem programas que recorrem directamente às chamadas ao sistema
- como se controlam os programas que desencadeiam a execução concorrente de múltiplos processos

Pode-se perguntar qual o interesse em estudar SO em profundidade. Não bastaria conhecer, como utilizador, as interfaces de utilização disponíveis através dos comandos do terminal (de texto ou de janelas e ícones gráficos)? Para um Engenheiro Informático, a resposta é 'não'.

Um Engenheiro Informático deve conhecer, não só as interfaces de utilização, como também os mecanismos e a arquitectura interna dos sistemas de computadores:

- porque pode vir a ter de modificar configurações de um dado SO, exigindo um conhecimento dos seus módulos internos;
- porque pode vir a ter de desenvolver SO especializados ou dedicados a tarefas de controlo específicas, nomeadamente em ambientes de controlo industrial, envolvendo, por exemplo, máquinas e equipamentos 'robotizados';
- porque pode vir a ter de desempenhar a tarefa de administrador e gestor de sistemas de computadores, numa dada empresa, exigindo-lhe conhecimentos sobre o hardware do computador, da infraestrura física de rede e do software do SO que a controla;
- porque pode vir a ter de desenvolver aplicações distribuídas ou de elevado desempenho, para as quais necessita de conhecer como controlar a execução de programas, ao nível do SO

Evidentemente que os conhecimentos acima mencionados não se vão adquirir todos em FSO. Veja a página da Secção de Arquitecturas de Sistemas Computacionais (http://asc.di.fct.unl.pt), uma descrição das outras disciplinas da LEI e do MEI, onde se estudam os temas desta subárea da Informática.

Em particular, FSO vai incidir sobretudo na interface das chamadas ao SO e na programação de aplicações simples, a este nível. Na disciplina de Sistemas de Operação, estuda-se a organização interna e implementação dos programas que constituem o SO.

4 Funcionamento de FSO

Nas **aulas teóricas**, discutem-se os conceitos e dão-se exemplos genéricos. Nas **aulas práticas** aplicam-se os conceitos e os exemplos estudados na teórica, desenvolvendo programas na linguagem C, no ambiente do sistema de operação Linux.

Apesar de, para cada aula teórica, haver uma ficha de resumo (como esta), é importante participar nas aulas teóricas, porque é uma oportunidade de apreender os conceitos e de os discutir. Note que nas aulas práticas não haverá exposição dos conceitos teóricos e o conhecimento destes é necessário para fazer os trabalhos práticos.

É importante participar nas práticas, porque é aí que se experimentam (e confirmam) os conceitos teóricos, mas também porque é preciso fazer os trabalhos práticos, para ter frequência e poder ir a exame.

Por cada trabalho prático, há uma ficha de enunciado, que também inclui as referências necessárias para a execução do trabalho. Tanto as fichas teóricas como práticas, de um dado trabalho, devem ser estudadas antes da primeira aula em que cada trabalho se inicia.

Toda esta informação aparece na página da disciplina (http://asc.di.fct.unl.pt/~jcc/fso).

5 Avaliação

Para ter frequência, há que fazer os 5 trabalhos práticos e entregar os relatórios nos prazos indicados.

Veja as regras na página da disciplina (http://asc.di.fct.unl.pt/~jcc/fso).

6 Bibliografia

Estes apontamentos apenas servem como referência parcial de estudo. Apontamentos adicionais vão aparecer na página da disciplina, todas as semanas, junto ao

sumário de cada aula teórica.

Há um conjunto de referências sobre a matéria específica das práticas, na página da cadeira.

O livro *Unix Systems Programming*, indicado na página da cadeira, é de consulta obrigatória. A exposição da matéria teórica segue os capítulos do livro, muito aproximadamente, no que se refere ao caso concreto do sistema Unix.

O livro *The C Programming Language*, indicado na página da cadeira, é de consulta recomendada, pois a linguagem usada nas aulas é o C (pode obviamente consultar outros livros sobre esta linguagem).