[MAC0211] Laboratório de Programação I
Aula 9
Sistemas Operacionais e Bibliotecas Estáticas e Dinâmicas

Alair Pereira do Lago

DCC–IME–USP

24 de março de 2015

Softwares

Podem ser divididos grosseiramente em dois grupos:

- **Programas de sistema** – gerenciam o funcionamento do computador
- **Programas aplicativos** – executam o trabalho que o usuário quer que seja feito

O **sistema operacional** é o principal programa de sistema de um computador.

Principais funções de um sistema operacional

- **Máquina estendida** ou **máquina virtual** – fornece ao usuário interfaces que facilitam a interação com o hardware (e.g., serviços que os programas podem acessar por meio de *chamadas ao sistema*)

- **Gerenciador de recursos** – gerencia todas as "peças" que constituem um computador moderno (processadores, memórias, interfaces de rede, dispositivos de entrada e saída, etc.), se preocupando em prover uma alocação ordenada e controlada dos recursos para os vários programas que os disputam

Motivações históricas para o desenvolvimento dos SOs

- Primeira fase – hardware muito mais caro que a mão de obra especializada
  Objetivo: maximizar a taxa de utilização do hardware

- Segunda fase (inversão) – hardware mais barato, mão de obra mais cara
  Objetivo: tornar o computador mais fácil de ser usado pelas pessoas

Evolução dos sistemas de computadores

1. Computadores a válvulas – **sistemas sem SO** algum
2. Computadores a transistores – **sistemas em lote** (*batch*)
3. Circuitos integrados – **sistemas multiprogramados**, **sistemas de tempo compartilhado**
4. Computadores pessoais – **sistemas gráficos**, **sistemas de redes**, **sistemas distribuídos**

Multiprogramação × multiprocessamento

- **Multiprogramação** (ou **multitarefa**) – diversos programas distintos executando em um mesmo processador
- **Multiprocessamento** – diversos processadores (em um mesmo sistema computacional) executando programas distintos ou cooperando na execução de um mesmo programa

Componentes centrais de um sistema operacional

- **Escalonador de processos** – determina quando e por quanto um processo é executado em um processador
- **Gerenciador de memória** – determina quando e como a memória é alocada aos processos e o que fazer quando a memória principal estiver cheia
- **Gerenciador de E/S** – atende às solicitações de entrada/saída de/para dispositivos de hardware
- **Gerenciador de comunicação interprocessos (IPC)** – permite que os processos se comuniquem uns com os outros
- **Gerenciador de sistemas de arquivos** – organiza coleções nomeadas de dados em dispositivos de armazenamento e fornece uma interface para acessar os dados nesses dispositivos

Modo privilegiado ou supervisor (núcleo) × modo usuário

A maioria das UCPs possui dois modos de funcionamento, controlados por 1 bit de um dos registradores de uso específico:

Modo supervisor

Nesse modo, a UCP pode executar qualquer instrução do seu conjunto de instruções e usar qualquer atributo do seu hardware.

⇒ O sistema operacional é executado em modo supervisor, por isso tem acesso a todo o hardware.

Modo usuário

Esse modo permite a execução de apenas um subconjunto das instruções e um subconjunto dos atributos de hardware. De forma geral, todas as instruções que envolvem entrada/saída e proteção de memória são inacessíveis nesse modo.

⇒ Os programas aplicativos são executados em modo usuário.

Arquiteturas de sistemas operacionais

- **Monolítica** – cada componente do SO é contido no núcleo e pode comunicar-se diretamente com qualquer outro (por meio de chamadas à função) [arquitetura mais antiga e mais comum]
- **Em camadas** – agrupa em camadas os componentes que realizam tarefas similares
- **Micronúcleo** – fornece apenas um número pequeno de serviços (para manter o núcleo pequeno e escalável)
- **Cliente-servidor** – implementa muitas das funcionalidades do SO como processos de usuários (divididos entre processos clientes e processos servidores). O núcleo se encarrega da comunicação entre clientes e servidores

Biblioteca: interface e implementação

Biblioteca

- É uma coleção de implementações de comportamentos que possui uma interface bem definida, por meio da qual um comportamento é invocado.

Interface

- Expressa **o quê** a biblioteca faz.

Implementação

- Expressa **como** a biblioteca faz o que está definido em sua interface.

Bibliotecas

Características

- Promovem o reuso de "comportamentos" (= código)
- São organizadas de forma que possam ser usadas por programas diferentes (e independentes)
- Os comportamentos implementados nelas podem ser integrados (= **ligados**) aos do programa invocador em diferentes momentos do ciclo de vida do programa:
  - na geração do executável do programa **ou**
  - na invocação da execução do programa **ou**
  - durante a execução do programa

Bibliotecas estáticas e dinâmicas

Bibliotecas estáticas

- São as que bibliotecas cujo código é acessado durante a construção (*build*) do executável do programa invocador

Bibliotecas dinâmicas

- São as que bibliotecas que são integradas (= ligadas) a um outro programa depois que a execução do mesmo já tiver sido invocada. A ligação das bibliotecas dinâmicas pode ocorrer de duas maneiras:
  - **na inicialização**: quando o programa é carregado para execução, todas as bibliotecas dinâmicas que ele referencia também são carregadas com ele
  - **carga preguiçosa** (mais comum): a biblioteca só é carregada quando ela for necessária para a continuidade da execução do programa

Bibliotecas dinâmicas

Vantagens

- Compartilhamento entre os vários programas que as usam (tanto na memória quanto no disco)
- Menor uso de memória na máquina como um todo (imagine o que aconteceria se a libc fosse estática...)
- Novas versões (com melhorias ou correções) das bibliotecas são aproveitadas mesmo pelos programas mais antigos

Desvantagens

- A carga de programas com bibliotecas dinâmicas é mais lenta
- *DLL hell* – complicações frenquente relacionadas ao uso de bibliotecas dinâmicas nas versões mais antigas do Windows

Bibliotecas estáticas

Vantagens

- Não há o risco de não se localizar uma biblioteca em tempo de execução (já que o código das funções da biblioteca que um programa utiliza são adicionados ao seu executável final)
- A versão das bibliotecas é fixa, portanto, não há o perigo de incompatibilidade de versões
- Mudanças no comportamento das bibliotecas novas não afetam a corretude dos programas antigos
- A carga de programas com bibliotecas estáticas é mais rápida

Desvantagens

- Tamanho do arquivo executável fica maior
- O código da biblioteca é adicionados ao código executável mesmo que em tempo de execução ele não seja usado

Bibliotecas estáticas e dinâmicas no Linux

Como são nomeadas

- Bibliotecas estáticas: extensão `.a`
- Bibliotecas dinâmicas: extensão `.so`

Convenções

- Nomes de bibliotecas geralmente possuem o prefixo `lib` (exemplos: `libteste.a` ou `libteste.so`).

As bibliotecas de C usam essa convenção (exemplos: `libc.so` e `libm.so`).

Bibliotecas estáticas e dinâmicas no Linux

Convenções

- Nomes de bibliotecas geralmente possuem o prefixo `lib` (ex.: `libteste.a` ou `libteste.so`).

No GCC, na ligação, a referência à biblioteca (na linha de comando) por meio do parâmetro `-l` não conterá o sufixo e nem a extensão do nome da biblioteca:

```
$ gcc meu_prog.c -lm -lpthread
```

Bibliotecas referenciadas no exemplo (para inclusão na ligação):

- biblioteca matemática – `/usr/lib/libm.so`
- biblioteca para o uso de *threads* – `/usr/lib/libpthread.so`

Criação de uma biblioteca estática

- Passo 1: gerar os códigos-objeto dos fontes que farão parte da biblioteca estática
- Passo 2: executar o comando `ar` para "compactar" os objetos em um único arquivo com extensão `.a`

Exemplo

```
$ gcc -c calc_media.c -o calc_media.o
$ gcc -c calc_dp.c -o calc_dp.o
$ ar -rcv libestat.a calc_media.o calc_dp.o
```

- Para listar os objetos existentes dentro de uma biblioteca estática, use `ar -t`. Exemplo:

```
$ ar -t libestat.a
```

Ligação de uma biblioteca estática

Exemplo 1: biblioteca criada por um usuário

```
$ gcc -o meu_prog meu_prog.c libestat.a
```

ou, para indicar a localização da biblioteca, usar a opção `-L`:

```
$ gcc -o meu_prog meu_prog.c -L[caminho] libestat.a
```

Exemplo 2: biblioteca padrão de C

Para forçar a ligação com a versão estática da biblioteca, usar opção `-static`:

```
$ gcc -o meu_prog meu_prog.c -static -lm -lc
```

Criação de uma biblioteca dinâmica

- Passo 1: gerar os códigos-objeto dos fontes que farão parte da biblioteca dinâmica usando a opção `-fPIC` (para gerar um código "independente de posição", ou seja, que funciona corretamente não importa em qual posição da memória ele for colocado)
- Passo 2: usar o opção `-shared` do GCC para gerar o arquivo `.so`

Exemplo

```
$ gcc -c -fPIC calc_media.c -o calc_media.o
$ gcc -c -fPIC calc_dp.c -o calc_dp.o
$ gcc -o libestat.so -shared calc_media.o calc_dp.o
$ mv libestat.so /home/user/libs/
```

Ligação de uma biblioteca dinâmica

Exemplo

```
$ gcc -o meu_prog meu_prog.c -L/home/user/libs -lestat
```

Observações:

- com a opção `-lestat`, o GCC busca a biblioteca de nome `libestat.so` (se fosse `-static -lestat`, procuraria `libestat.a`)
- a biblioteca não será incluída no executável `meu_prog`; ela será ligada dinamicamente ao programa na execução
- o comando `ldd` mostra as bibliotecas dinâmicas que serão carregadas para um executável. Ex.:

  ```
  $ ldd meu_prog
  ```

Executando um programa que usa a biblioteca dinâmica

Para que um executável encontre as bibliotecas necessárias para a ligação em tempo de execução, é preciso indicar ao ligador onde as bibliotecas devem ser procuradas. Isso pode ser feito de diferentes maneiras:

- Editando a variável de ambiente `LD_LIBRARY_PATH` (para uso temporário)

```
$ export LD_LIBRARY_PATH=/home/user/libs:$LD_LIBRARY_PATH
```

- Criando um arquivo `/etc/ld.so.conf.d/mylibs.conf` e incluindo nele o caminho. Você pode fazer isso com o comando:

```
$ echo "/home/user/libs" |
                    sudo tee -a /etc/ld.so.conf.d/mylibs.conf
$ sudo ldconfig
```

Ligação de bibliotecas no Linux

Variáveis de ambiente

- `LD_LIBRARY_PATH` – indica ao ligador onde ele deve pegar as bibliotecas dinâmicas
- `LD_PRELOAD` – lista de bibliotecas a serem carregadas antes de todas as outras
- `LD_BIND_NOW` (*on* ou *off*) – quando habilitado, faz as bibliotecas dinâmicas serem carregadas de uma vez no início da execução de um programa (caso contrário, o padrão é carga sob demanda, i.e., *lazy binding*)

Bibliografia e materiais recomendados

- Capítulo 1 do livro *Operating Systems: Design and Implementation*, de Tanembaum e Woodhull [disponível na biblioteca do IME]
- Uma introdução "analítica" aos sistemas operacionais, feita pelo prof. Martin C. Rinard, do MIT:
  `http://people.csail.mit.edu/rinard/osnotes/h1.html`
- Bom tutorial sobre a criação e uso de bibliotecas estáticas e dinâmicas no Linux:
  `http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html`
- Mais informações sobre bibliotecas no GCC (na verdade, se trata de um livro de introdução ao GCC):
  `http://www.network-theory.co.uk/gcc`
- Mais detalhes sobre o "Inferno das DLLs":
  `http://en.wikipedia.org/wiki/DLL_Hell`
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

- Interpretador de comandos (*Shell*)