Continuação: Syncronização Compilação, Ligação e Carga

Marcelo Johann

Na aula anterior…

Sincronização: Produtor e Consumidor

Plano da aula

1. Semáforos POSIX
   Pergunta: como se faz com processos?
   Jantar dos Filósofos e Deadlock
   Código Thread-safe

2. **Introdução à Gerência de Memória**
   Endereçamento lógico vs. físico
   Ciclo de compilação

Semáforos POSIX

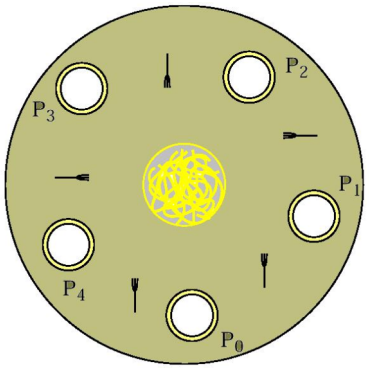
- Named semaphores
- Kernel-persistent
- Need open/close, and init

- testes

Como se faz com processos?

- Memória compartilhada
- Semáforos POSIX…

Problema dos filósofos



```
#define N 5
#define LEFT(i) (i+N-1)%N
#define RIGHT(i) (i+1)%N
#define THINKING 0
#define HUNGRY 1
#define EATING 2
int state[N];
sema_t mutex; // = 1
sema_t Sem[N]; // = 0

void philosopher(int i) {
  while(TRUE){
    think();
    take_forks(i);
    eat();
    put_forks(i);
    }
  }
void take_forks(int i) {
  sema_wait(&mutex);
  state[i] = HUNGRY;
  test(i);
  sema_post(&mutex);
  sema_wait(&Sem[i]);
  }
```

```
void put_forks(int i)
  {
  sema_wait(&mutex);
  state[i] = THINKING;
  test(LEFT);
  test(RIGHT);
  sema_post(&mutex);
  }

void test(int i)
  {
  if ( state[i] == HUNGRY &&
    state[LEFT(i)]!=EATING &&
    state[RIGHT(i)]!=EATING )
    {
    state[i] = EATING;
    sema_post(&Sem[i]);
    }
  }
```


Código Thread-Safe

- Distinção entre threads e funções
- Programas multi-thread
- Funções e bibliotecas reentrantes

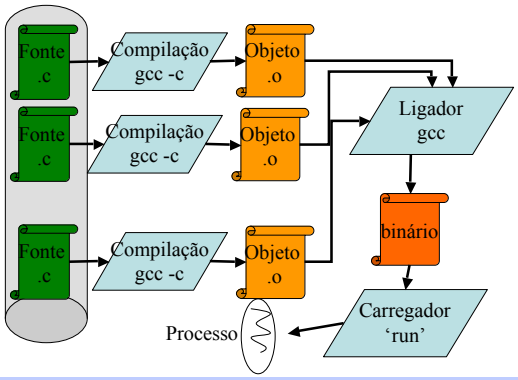
You need to be thread-safe!

- Kernel reentrante

O compilador

- O **compilador** traduz um programa escrito numa linguagem de alto-nível (e.g. C) em linguagem assembly e/ou de máquina.
  - Possivelmente se distingue a **montagem** da **compilação**. A montagem passa do assembly para o código de máquina.
  - Antes da compilação, pode haver pre-compilação
    - Processamento de macros (#define)
  - Com o gcc : gcc –c fonte.c
    - Obtenção de "fonte.o"
    - Cada arquivo fonte compilado produz seu próprio arquivo .o.
    - Os arquivos resultantes .o se chamam "arquivos objetos".
- A **ligação** (amarração) permite juntar os arquivos objetos para se obter o executável.
  - Efetua a amarração

Ciclo de compilação/execução



Ligador: objetos e amarração (link edition)

- O arquivo objeto contém código de máquina gerado a partir de um arquivo fonte.
- No caso geral, um programa é constituído de vários módulos, cada qual em um arquivo distinto.
  - Facilidade de desenvolvimento, manutenção, teste…
  - Necessidade de acessar variáveis externas (globais);
  - Necessidade de acessar funções externas.
- Cada arquivo objeto contém os símbolos locais a seu arquivo fonte, mas há referências a **símbolos externos**.
  - Veja a instrução 'nm' no Linux.
  - O **compilador** gera uma tabela de símbolos mais uma tabela de referências cruzadas.
- O **ligador** resolve os símbolos externos (referências cruzadas)
  - Amarração estática: à compilação;
  - Amarração dinâmica: durante a execução.

Algoritmo de ligação

- O ligador copia todos os objetos para o executável final.
  - Cria uma tabela de definição de símbolos
- Durante a cópia, ele verifica a tabela de referências cruzadas de cada objeto
  - Em uma segunda leitura da tabela de definição, verifica se todas as referências foram obtidas na tabela de def. de símbolos.

Endereços absolutos e relativos

- Uma vez compilado e amarrado, o código executável é uma seqüência de Bytes.
  - Ele faz acessos a endereços de memória.
  - Código gerado a partir do end. 0 da memória (lógica).
  - Exemplo: [Cf. Livro Carissimi]

Carga de um executável na memória

- O **carregador** copia o binário do disco para a memória
  - Coloca-o em um processo (alocar memória).
  - Prepara-o para execução.
- Problema: na hora de efetuar a compilação, não se sabe qual será o endereço inicial na memória aonde será carregado o executável!
  - Será preciso deslocar os endereços relativos.
  - Vai acontecer, por exemplo, no caso de ligação de vários módulos!
- O compilador marca os endereços relativos.
- Cabe ao carregador corrigir os mesmos.
  - Relocação.
  - Por exemplo, se soma o endereço inicial do segmento de código.

Dois tipos de carregadores

☞Em alguns casos, não há nada para fazer
  - Só têm endereços absolutos.
  - Desde a compilação se soube o endereço do início do programa, na memória.
  - Fala-se de **carregador absoluto**.
  - (usado em computadores antigos e/ou sistemas embutidos).

☞**Carregador relocador**:
  - Corrige apenas o endereço inicial.
  - Altera todos os valores relativos em função do endereço inicial.
  - Exige informação extra no arquivo executável, além da imagem do binário (mapa de relocação)
    - Bitmap.

Carga (e ligação) **dinâmica** de procedimentos

- Extensão do carregador relocável: um procedimento (compilado) somente é carregado na memória quando é chamado.
- Ligação (amarração) dinâmica:
  - Bibliotecas fatoram procedimentos corriqueiros pre-compilados
    - Ex.: software matemático, linguagens de programação, biliotecas de threads…
  - Elas podem ser ligadas estaticamente ao programa.
    - Desperdiça espaço no disco e na RAM pois tem uma cópia por programa as usando.
    - Quando muda a biblioteca, é preciso recompilar o programa.
  - Elas podem ser ligadas dinamicamente, durante a execução do programa.
    - Compartilhamento de bibliotecas.
    - DLL

Próxima Aula

Kernel monolítico
e micro kernel