Clase #23 de 27
Stack & Heap

Oct 18, Jueves

Agenda para esta clase

- Call Stack
- Stack Frames & Variables Automáticas
- Heap: new & delete
- Disposición de Memoria

Call Stack

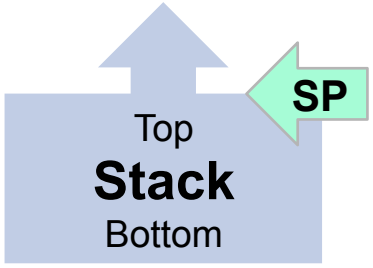
Pila de Invocaciones a Funciones

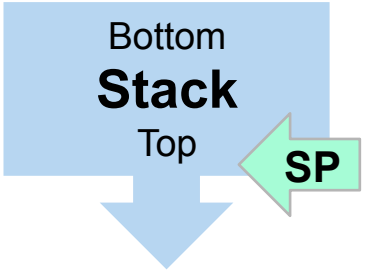
Call Stack (Pila de Invocaciones), a.k.a. *El Stack*

- Un **proceso** es un **programa en ejecución**
- El **call stack** tiene las funciones en **ejecución**
- Cada **proceso** tiene su **call stack**
  - En un sistemas con procesos que permiten más de un **thread** (**hilo de ejecución**), **cada thread** posee su call stack
- Justo después de comenzar el proceso y justo antes de tarminar, el call stack está **vacío**
- Cuando una función invoca a otra se realiza un **Push**
- Cuando una función retorna se realiza un **Pop**
- En el **top** (cima) está la función **activa**
- La función en un instante determinado en el **top no invocó** en ese instante a otra función
- La función en el top fue i**nvocada por la función debajo**, y así sucesivamente
- En la **base** está **main**
- El **Sistema Operativo** invoca a **main**

- Ejemplo de call stack dónde **main** invocó a **foo** que invocó a **bar**
  - 3 bar
  - 2 foo
  - 1 main
  - 0 Sistema Operativo
- Ejemplo: Call stack de un juego que está **moviendo un personaje**
  - 8 DibujarCírculo
  - 7 DibujarPupila
  - 6 DibujarOjo
  - 5 DibujarCabeza
  - 4 DibujarPersonaje
  - 3 MoverPersonaje
  - 2 Jugar
  - 1 main
  - 0 Sistema Operativo

- La **base** del call stack es **constante**
- El registro **SP** de la CPU (Stack Pointer) apunta al **top** del call stack.
- **SP** se incrementa o decrementa con los **Push** y **Pop**
- La plataforma de ejecución determina detalles de implementación como:
  - **Ubicación** de la **base** del stack
  - Si el stack **crece** hacia **abajo** o hacia **arriba**





Ejercicio #1 de Call Stack:

Basado en hoja 6 de Memory Management in C

- Dado el call stack:
  - 3 bar
  - 2 foo
  - 1 main
- ¿Cuál función llamó a cuál?
- Dibuje un digrafo que represente el call stack y las relación "invocó-a"
- ¿Cuántas funciones hay en ejecución?
- ¿Cuál es la invocación activa?
- Codifique un programa que se corresponda a este call stack:
  - ¿Es correcta la sigueinte resolución?

```
void foo(int);

int bar();

int main(){
  foo(bar());
}

void foo(int i){}

int bar(){return 42;}
```

Ejercicio #2 de Call Stack:

- Dibuje la **secuencia de estados del call stack** para el programa ejemplo "*Saludos*" del texto Funciones:
  - Cada instante presenta el call stack en un estado determinado
  - Tanto en el primer instante como en último instante el call stack está vacío

Ejercicio #3 de Call Stack:

- Ejecute "*Mostrar Adición de Dos Números Solicitados*" del texto Funciones y deje que el proceso se bloquee esperando input, para ese instante:
  1. Dibuje el call stack
  2. Busque en su sistema Operativo una herramienta que le permita inspeccionar el call stack y campare con su dibujo. Ayuda: *ActivityMonitor* y *Task Manager*
  3. Utilice un IDE con *debugger* para inspecionar el call stack, compare con su dibujo.

Ejercicio #4 de Call Stack:

- Implemente la función *factorial* de forma recursiva y escriba un programa que la pruebe. Busque una forma para pausar el programa en el caso base y, para ese instante:
  1. Dibuje el call stack
  2. Busque en su sistema Operativo una herramienta que le permita inspeccionar el call stack y campare con su dibujo. Ayuda: *ActivityMonitor* y *Task Manager*
  3. Utilice un IDE con *debugger* para inspecionar el call stack, compare con su dibujo.

Stack Frames & Variables Automáticas

Stack Frame (Activation Record)

Cuadro de Pila (Registro de Activación)

- Los **Stack Frames ó Activation Records** son los elementos del **Call Stack** (Pila de Invocaciones)
- Cada **stack frame** se corresponde a la **invocación** de una función que está en **ejecución**
- El **contenido es dependiente de la plataforma**, pero en *general*, un stack frame mantiene:
  - **Variables Automáticas**, que en *general* son las **locales**
  - Dirección de **retorno** para continuar la ejecución, es decir el anterior **IP** (Instruction Pointer)
  - **Argumentos** enviados por la función llamante
- Cada invocación a función realiza un **Push** de un nuevo **Stack Frame** (**Activation Record**) al call stack
- Cada retorno de función realiza un **Pop** del call stack
- En assembler tanto **las invocaciones y los Push** como **los retornos y los Pop** se implementan con más de una instrucción

Variables Automáticas

- ¿Qué **acción** se debe hacer **programar** en una función para **reservar espacio en memoria para variables** que se usan en esa función?
- ¿Y para **liberar** es espacio?
- ¿**Automática** implica **local**?
- ¿**Local** implica **automática**?

```
// Ejemplo
void foo(int a){    // Automatic
  int b = 2;         // Automatic
  static int c = 3; // Static
}

// ¿Correcto? ¿Por qué?
int* pa(){
  int a = 1;
  return &a;
}

// ¿Correcto? ¿Por qué?
int* pp(int a){
  int b = 2;
  return &a;
}

// ¿Correcto? ¿Por qué?
int* ps(){
  static int s = 3;
  return &s;
}

// ¿Correcto? ¿Por qué?
int* p(int* p){
  return &p;
}
```

Ejercicio #5 de Call Stack: Saludos

Basado en hoja 6 de Memory Management in C

- Dibuje la **secuenica de estados del call stack** para el programa ejemplo "*Mostrar Adición de Dos Números Solicitados*" del texto Funciones:
  - Cada instante presenta el call stack en un estado determinado
  - Tanto en el primer instante como en último instante el call stack está vacío
  - Los elementos del call stack deben ser **stack frames** con, por lo menos, esta información:
    - **Nombre** de la función invocada
    - **Argumentos** de la invovación y **variables** automáticas
    - **Número de línea** del código fuente de la función llamante que indica dónde continuar la ejecución al finalizar la función llamada.

Empujando los Límites del Stack

```
int main(){
    void Chrome();
    Chrome();
}

void Chrome(){
    int a[42*42*42*42];
    Chrome();
}
```

Reserva Explícita de Memoria & Heap

Operadores new & delete

Reserva, Analogía con Restaurant u Hotel

- Llamado para hacer una reserva
  - ¿Qué se pide? ¿Nombre? ¿Cantidad?
  - ¿Qué se registra?
  - ¿Qué se retorna?
  - Operador new
- Llamado para cancelar reserva
  - Operador delete
- Diferencias con Stack y las Variables Automáticas
  - Reserva
  - Nombre de los objetos
  - Duración de los objetos
  - Liberación
  - Tiempo y Espacio para Reserva y Liberación
- ¿Qué aplicación tiene?
- Garbage Collector

```
// Un int
int* p = new int;
…
delete p;

// Arreglo de ints
unsigned n;
cin >> n;
int* pa = new int[n];
…
delete p[];
```

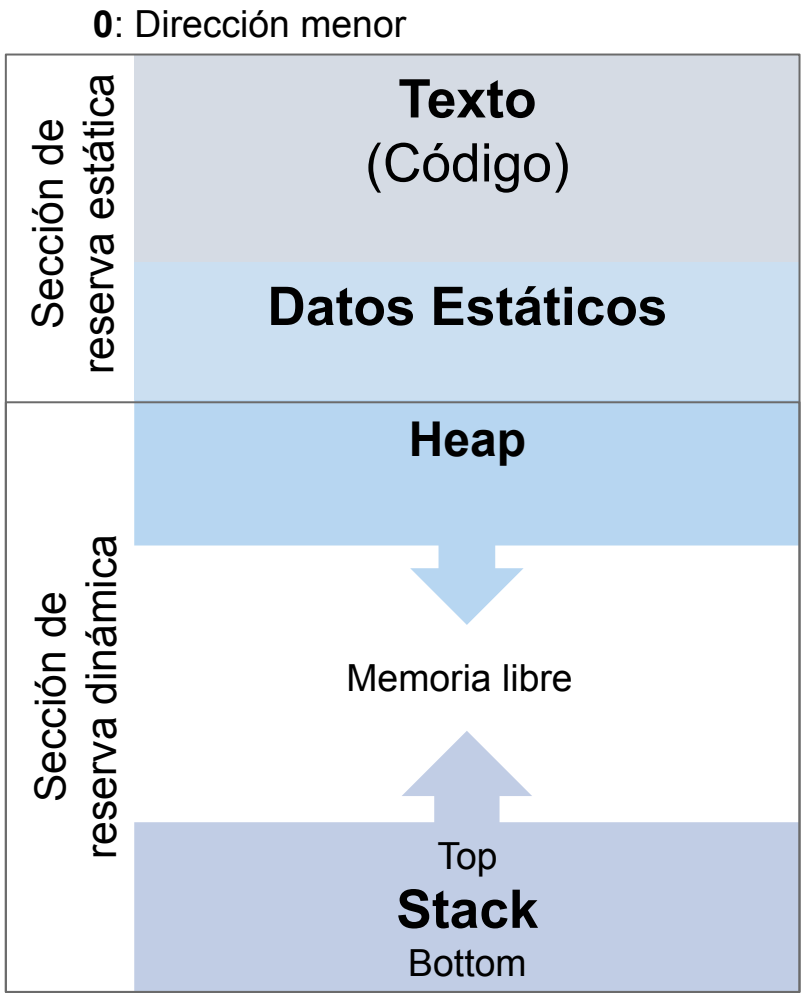
Empujando los Límites del Heap

```
int main(){
    void Chrome();
    Chrome();
}

void Chrome(){
    for(;;) new int[42*42*42*42];
}
```

Layout de la Memoria

Layout (Disposición) de la Memoria




```
void f();
int* g(int);

int a = 0;              // Static

int main(){
  int b = 1;            // Stack (automatic)

  f();

  int* p = &b;          // Stack
  *p = 3;

  p = new int;          // Heap
  *p = 4;
  delete p;

  p = g(21);
  delete p;
}

void f(){
  int c = 2;            // Stack
  static int d = 3; // Static
}

int* g(int i){          // Stack
  int* p = new int; // Stack y Heap
  *p = i;
  return p;
}
```

- ¿Qué es el **stack overflow**?

¿Consultas?

Fin de la clase