Dynamic Linking

Introduzione

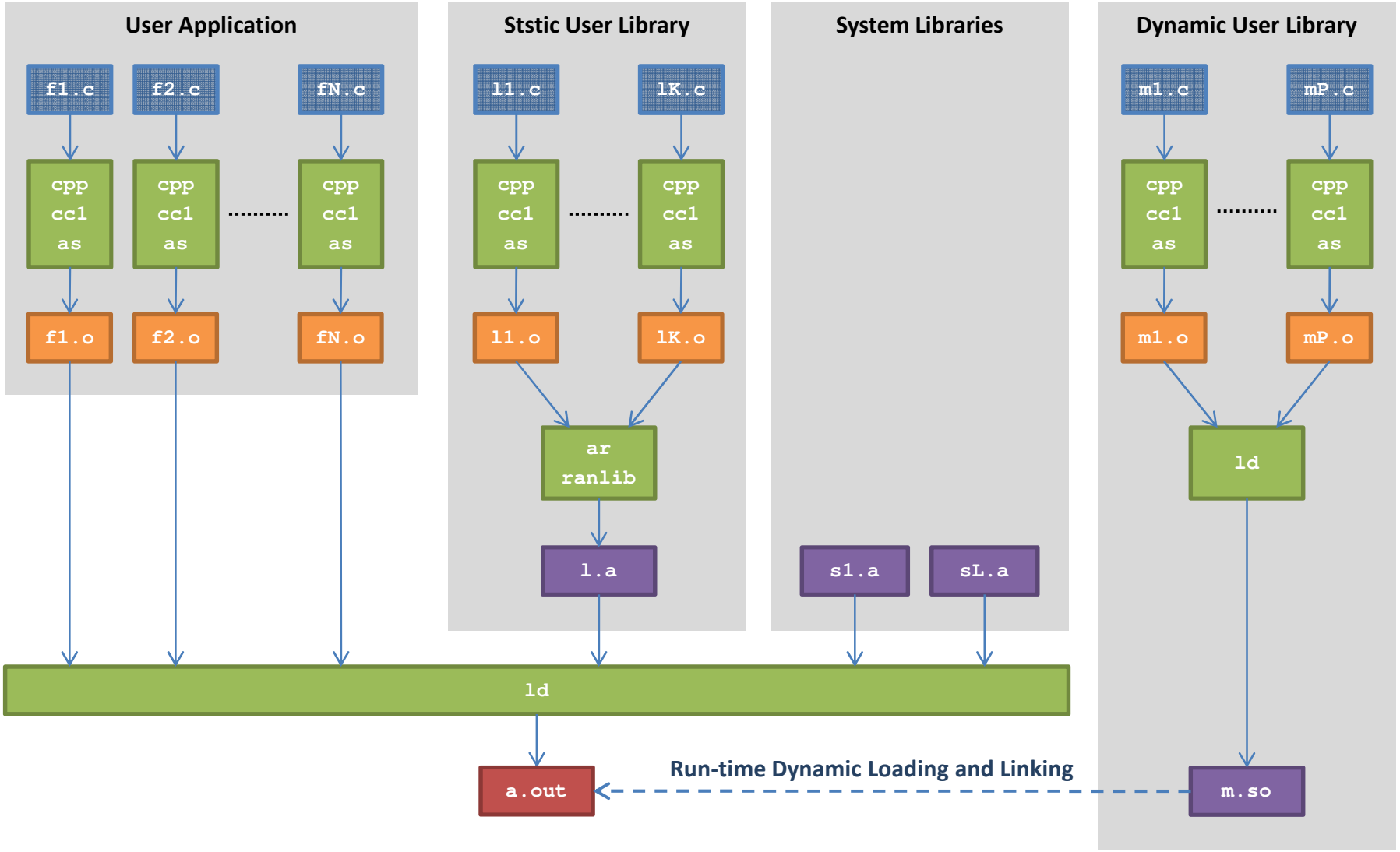
Creazione di una libreria dinamica

Uso di una libreria dinamica

Dynamic Linking

- **Il linking tra i moduli di un programma e le librerie da esso utilizzate può essere**
  - Statico
    - Avviene alla fine del processo di compilazione
    - E' il linker (ld) ad eseguire questo compito
    - Produce un file binario eseguibile che contiene il codice della libreria
  - Dinamico
    - Avviene durante l'esecuzione del programma
    - Deve essere gestito esplicitamente dal programmatore
    - Sfrutta un supporto offerto dal sistema operativo
    - Il file binario dell'applicazione non contiene il codice della libreria
    - Il codice della libreria viene caricato al momento in cui il programma ne fa richiesta
- **I vantaggi offerti dal linking dinamico sono i seguenti**
  - Flessibilità
    - Il codice dell'applicazione può utilizzare diverse librerie senza dover essere ricompilato
    - Questo, per esempio, è utilie quando una libreria subisce aggiornamenti
  - Utilizzo della memoria più contenuto
    - La libreria risiede in memoria solo per il tempo necessario

Dynamic Liniking



Dynamic Linking

- **Per poter utilizzare il meccanismo del dynamic linking è necessario**
  - Creare una libreria dinamica
  - Creare un'applicazione in grado di utilizzarla
- **Creazione di una libreria dinamica**
  - Il codice di una libreria dinamica è identico a quello di una libreria statica
  - Il processo di compilazione è differente:

```
$> gcc m1.c m2.c ... -shared -o mylib.so
```

- **Creazione di un'applicazione che usa una libreria dinamica**
  - Richiede alcune operazioni esplicite
    - Apertura e caricamento della libreria
    - Ricerca del "simbolo" (funzione o variabile) di interesse
    - Utilizzo del "simbolo" trovato
    - Chiusura e deallocazione della memoria
  - Per il resto l'applicazione è sviluppata in modo identico a quanto visto finora
  - All'applicazione è necessario linkare la libreria di supporto al dynamic linking

```
$> gcc s1.c s2.c ... -ldl -o myapp
```

Apertura

- **Per aprire e caricare una libreria dinamica si usa la funzione**

```
#include <dlfcn.h>
void *dlopen( const char *filename, int flag );
```

- **Argomenti**
  - filename
    - Specifica il nome del file binario della libreria.
    - Tipicamente ha estensione .so (Shared Object)
  - flag
    - Specifica la modalità con cui aprire la libreria
    - I valori di questo flag sono specificati mediante macro definite nel file dlfcn.h
    - Ai nostri fini questo parametro può essere fissato al valore RTLD_LAZY
- **Valore di ritorno**
  - In caso di successo viene restituito un "handler"
    - Cioè un puntatore di tipo void che indirizza la memoria in cui la libreria è stata caricata
  - In caso di fallimento ritorna il valore NULL
  - E' sempre bene verificare il valore di ritorno

Ricerca

- **Per cercare un simbolo nella libreria si usa la funzione**

```
#include <dlfcn.h>
void *dlsym( void *handle, const char *symbol );
```

- **Argomenti**
  - handle
    - Specifica l'handler della libreria, ottenuto all'atto dell'apertura
  - symbol
    - Specifica il nome del simbolo (funzione o variabile) da ricercare
- **Valore di ritorno**
  - In caso di successo viene restituito un puntatore al simbolo
    - Il tipo del puntatore è sempre void, per generalità
    - Tale puntatore deve essere convertito, mediante casting, nel tipo specifico del simbolo
    - Il tipo del simbolo deve essere noto all'utilizzatore
    - A questo punto il simbolo è utilizzabile
  - In caso di fallimento ritorna il valore NULL
  - E' sempre bene verificare il valore di ritorno

Chiusura e deallocazione

- **Per chiudere una libreria dinamica e deallocarne la memoria si usa la funzione**

```
#include <dlfcn.h>
int close( void *handle );
```

- **Argomenti**
  - handle
    - Specifica l'handler della libreria, ottenuto all'atto dell'apertura
- **Valore di ritorno**
  - In caso di successo viene restituito 0
  - In caso di errore ritorna un valore diverso da 0

Errori

- **Per avere informazioni su eventuali errori si usa la funzione**

```
#include <dlfcn.h>
char *dlerror(void);
```

- **Argomenti**
  - Nessuno
- **Valore di ritorno**
  - Una stringa che descrive in modo esplicativo la tipologia e la ragione dell'ultimo errore
    - Riguarda unicamente gli errori relativi al dynamic linking
  - Invocata senza che si sia verificato alcun errore
    - Riporta lo stato interno del gestore degli errori in una condizione nota

Esempio

- Si consideri una semplice libreria che dispone di due funzioni

mylib.c

```
int add( int a, int b ) {
  return a + b;
}

int sub( int a, int b ) {
  return a - b;
}
```

- Questo file sorgente deve essere compilato come segue

```
$> gcc mylib.c -shared -o mylib.so
```

- Il risultato è la libreria dinamica mylib.so

Esempio

- L'applicazione che volesse usare la funzione add() di tale libreria sarebbe

main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <dlfcn.h>

// Function-pointer type compatible
// with the add() function
typedef int (*fptr_t)(int,int);

int main() {
  void*  handle;  // Library handler
  fptr_t addptr;  // Symbol pointer

  // Opens & loads the library
  handle = dlopen( "./mylib.so", RTLD_LAZY );
  if( handle == NULL ) {
    puts( dlerror() );
    exit( 1 );
  }
```

main.c (continua)

```
  // Looks for the function "add"
  addptr = (fptr_t)dlsym( handle, "add" );
  if( addprt == NULL ) {
    puts( dlerror() );
    exit( 1 );
  }

  // Uses the function add() found in mylib.so
  // This is a call to add(3,5) made through
  // a function pointer
  printf( "Add: %d\n", (*addptr)(3,5) );

  dlclose( handle );

  exit( 0 );
}
```

- Questo file sorgente deve essere compilato come segue

```
$> gcc main.c -o myapp -ldl
```

Esempio

- **A questo punto, in una stessa directory si hanno**
  - Il file eseguibile myprog
  - La libreria dinamica mylib.so
- **Si noti che il programma myprog non contiene il codice della libreria**
- **All'atto dell'esecuzione di myprog**
  - Viene caricato il binario myprog in memoria
  - L'esecuzione di myprog inizia
  - Quindi myprog
    - Apre e carica la libreria dinamica mylib.so in memoria
    - Cerca il simbolo (funzione) "add", restituendone l'indirizzo in un puntatore
    - Invoca la funzione add() via puntatore
    - Chiude la libreria e dealloca la memoria
    - Quindi termina con statio di uscita 0