Principy přidělování paměti překladačem

Thursday, May 30, 2013 8:43 AM

Přeložený program dostane od operačního počítače k dispozici blok paměti, který obecně může být rozdělen na následující části:

- Vygenerovaný cílový kód
- Statická data
- Řídící zásobník
- Hromada

Základní způsoby přidělování:

- **Statické** (přidělení paměti v čase překladu)
- **Dynamické** (přiděleno v run time) **- v zásobíku** nebo **na haldě**

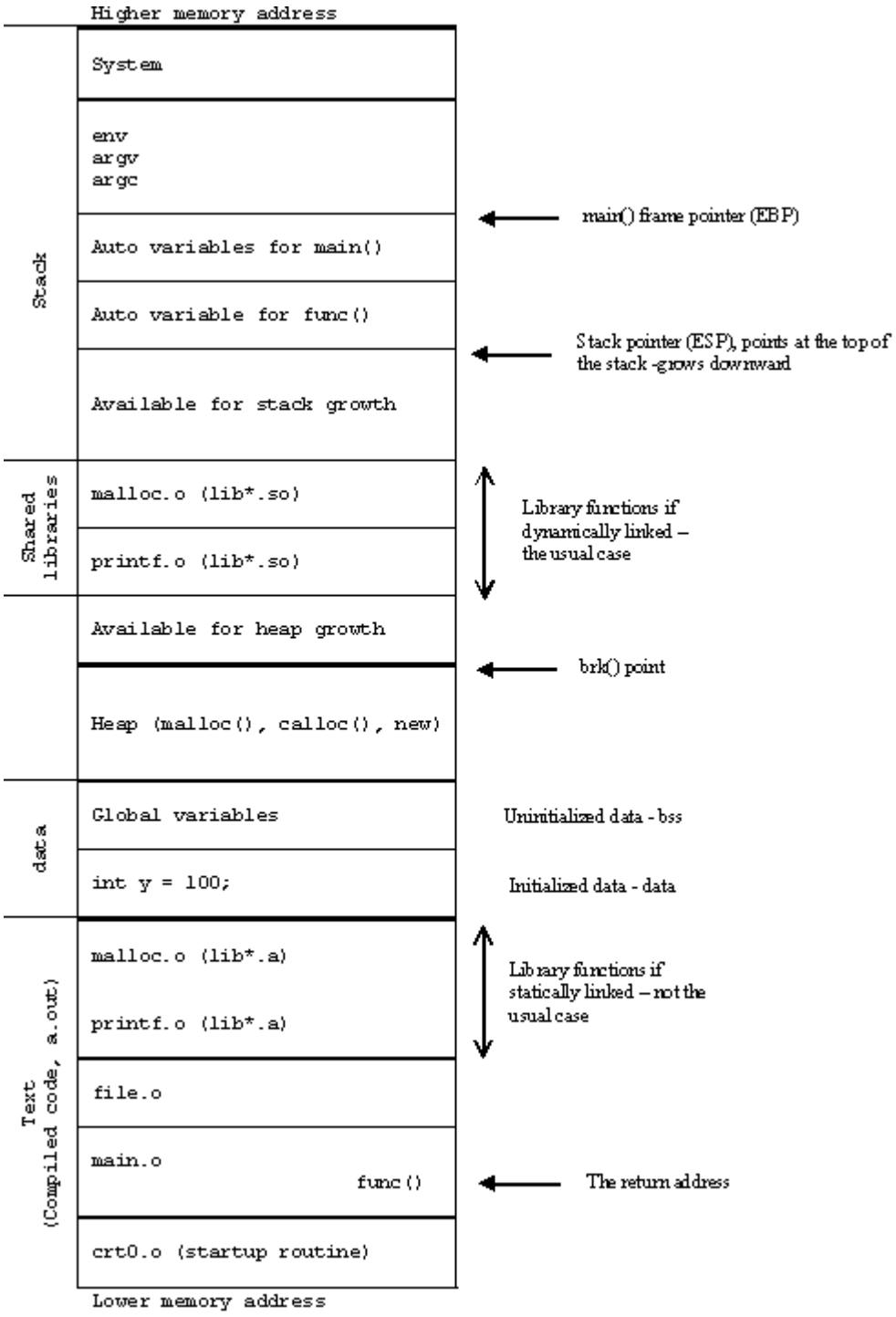
Velikost vygenerovaného kódu je známa již v době překladu, takže jej může překladač umístit **do staticky definované oblasti** (*Code/cílový kód programu*), obvykle na začátek přiděleného paměťového prostoru.

Rovněž velikost **statických datových objektů** může být známa již v době překladu a překladač je může **umístit za program** nebo uložit dokonce jako součást programu (to lze pouze u těch programovacích jazyků, které neumožňují rekurzivní volání procedur – Fortran).

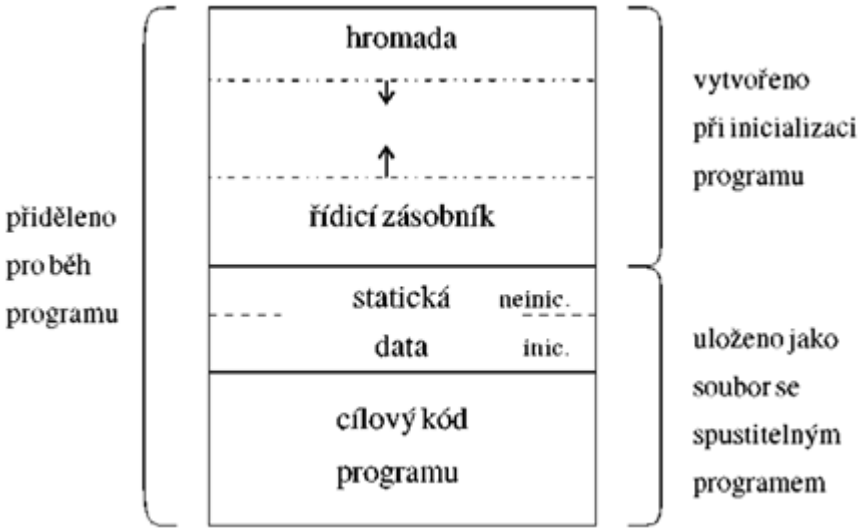
Jazyky umožňující rekurzi (Pascal, C, …) využívají pro aktivace podprogramů řídicího **zásobníku (stack)**, do kterého se ukládají jednotlivé ***aktivační záznamy*** (AZ jsou generovány při voláních procedur).

Pro účely **dynamického přidělování paměti** (explicitně vyžadovaného voláním příslušných funkcí nebo implicitně při přidělování paměti například pro pole s dynamickými rozměry) se používá zvláštní část paměti zvaná **hromada (heap)**.

Vzhledem k tomu, že se velikosti použité části paměti pro zásobník a hromadu v průběhu činnosti programu mohou značně měnit, je výhodné pro obě části využít opačné konce společné části paměti – viz obrázek. **Zásobník roste směrem k nižším adresám, hromada směrem k vyšším**. Nedostatek paměti se rozpozná tehdy, jestliže ukazatel konce některé oblasti překročí hodnotu ukazatele konce druhé oblasti.



From <http://www.tenouk.com/ModuleW_files/ccompilerlinker006.png>



Pro zmíněné datové oblasti se používají následující hlavní metody přidělování paměti:

- **Statické** přidělování paměti v době překladu
- **Dynamické** přidělování paměti za běhu programu:
  - Přidělování paměti na *zásobníku*
  - Přidělování paměti z *hromady*

Statické přidělování paměti v době překladu

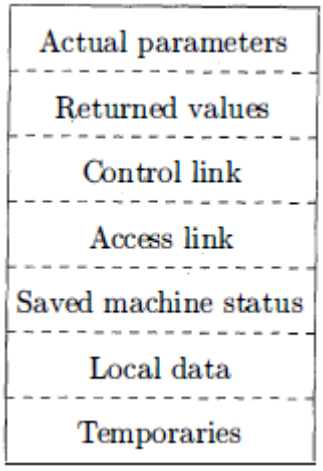
Při statickém přidělování paměti jsou všem objektům v programu **přiděleny adresy již v době překladu (globální data)**. Při kterémkoliv volání podprogramu jsou jeho lokální proměnné vždy na stejném místě, což umožňuje zachovávat hodnoty lokálních proměnných nezměněné mezi různými aktivacemi podprogramu. Statická alokace proměnných však klade na zdrojový jazyk určitá omezení. Údaje o velikosti a počtu všech datových objektů musejí být známy již v době překladu, **rekurzivní podprogramy mají velmi omezené možnosti**, neboť všechny aktivace podprogramu sdílejí tytéž proměnné, a konečně nelze vytvářet dynamické datové struktury.

Přidělování na zásobníku

Přidělování paměti pro aktivační záznamy na zásobníku se používá běžně u jazyků, které **umožňují rekurzivní volání podprogramů** nebo které **používají staticky do sebe zanořené podprogramy**. **Paměť pro lokální proměnné je přidělena při aktivaci podprogramu vždy na vrcholu zásobníku a při návratu je opět uvolněna**. To ale zároveň znamená, že hodnoty lokálních proměnných se **mezi dvěma aktivacemi podprogramu nezachovávají**.

Aktivace procedur při běhu programu jde znázornit ***aktivačním stromem***. Co uzel, to jedna aktivace, kořen je aktivací hlavní procedury, která je volána po spuštění programu; potomci uzlu *p* = volání procedur z procedury *p*. Kořen aktivačního stromu je na dně zásobníku, poslední aktivace má svůj záznam na vrcholu zásobníku.

Každá „živá“ aktivace má **aktivační záznam**. Obsah aktivačního záznamu se liší podle implementovaného jazyka.



1. ***dočasné hodnoty*** – vypadnou po vyhodnocení výrazů, jsou tu, pokud nemohou být udržovány v registrech
2. ***lokální data*** – patřící k dané proceduře s příslušným aktivačním záznamem
3. ***uložený strojový status*** – info o stavu stroje před voláním procedury. Typicky jde o:
   - *návratovou adresu* (= hodnota program counteru, kam se má pak procedura vrátit) a o
   - *obsah registrů* použitých procedurou (musí být obnoveny po návratu z procedury)
4. ***access link = statický ukazatel*** – pro lokaci dat, která procedura potřebuje, ale která se nachází v jiném aktivačním záznamu
5. ***control link*** – ukazuje na aktivační záznam volajícího (caller)
6. ***vrácené hodnoty*** – prostor pro návratovou hodnotu volané funkce (kvůli rychlosti lepší dávat do registru)
7. ***vlastní parametry*** – parametry použité volající procedurou; pokud je to možné, jsou umístěny radši v registrech kvůli výkonnosti.

Při implementaci přidělování paměti na zásobníku bývá **jeden registr vyhrazen jako ukazatel na začátek aktivačního záznamu na vrcholu zásobníku**. Relativně k tomuto registru se pak počítají všechny adresy datových objektů, které jsou umístěny v aktivačním záznamu. Naplnění registru a přidělení nového aktivačního záznamu je součástí volací posloupnosti, obnovení stavu před voláním se provádí během návratové posloupnosti.

Volací (a návratové) posloupnosti se od sebe v různých implementacích liší. Jejich činnost bývá rozdělena mezi volající a volaný program. Obvykle volající program určí adresu začátku nového aktivačního záznamu (k tomu potřebuje znát velikost záznamu vlastního), přesune do něj předávané argumenty a spustí volaný podprogram zároveň s uložením návratové adresy do určitého registru nebo na známé místo v paměti. Volaný podprogram nejprve uschová do svého aktivačního záznamu stavovou informaci (obsahy registrů, stavové slovo procesoru, návratovou adresu), inicializuje svá lokální data a pokračuje zpracováním svého těla. Při návratu opět volaný podprogram uloží hodnotu výsledku do registru nebo do paměti, obnoví uschovanou stavovou informaci a provede návrat do volajícího programu. Ten si převezme návratovou hodnotu a tím je volání podprogramu ukončeno.

přístup k nelokálním proměnným při statickém =lexikálním rozsahu platnosti jmen. To řeší tzv. **řetězec statických ukazatelů** (*access links*). Pro zrychlení přístupu k nelokálním proměnným se zavádí vektor ukazatelů – **displej**. Zamezí se tak průchod aktivačními záznamy pro hluboko zanořené podprogramy.

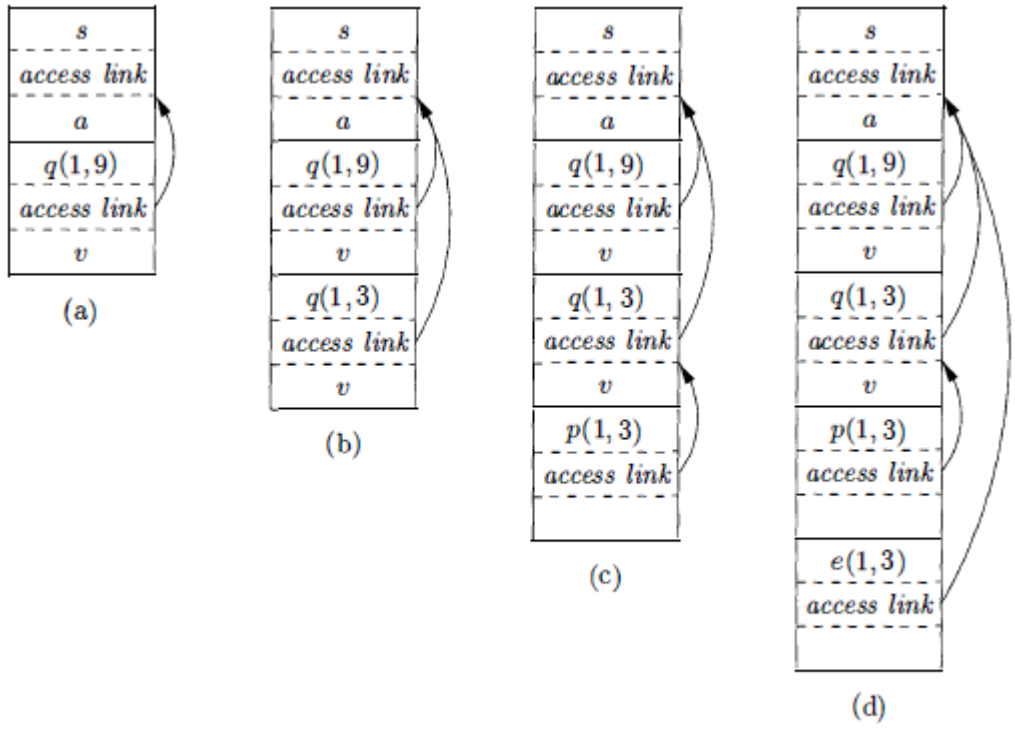


Figure 7.11: Access links for finding nonlocal data

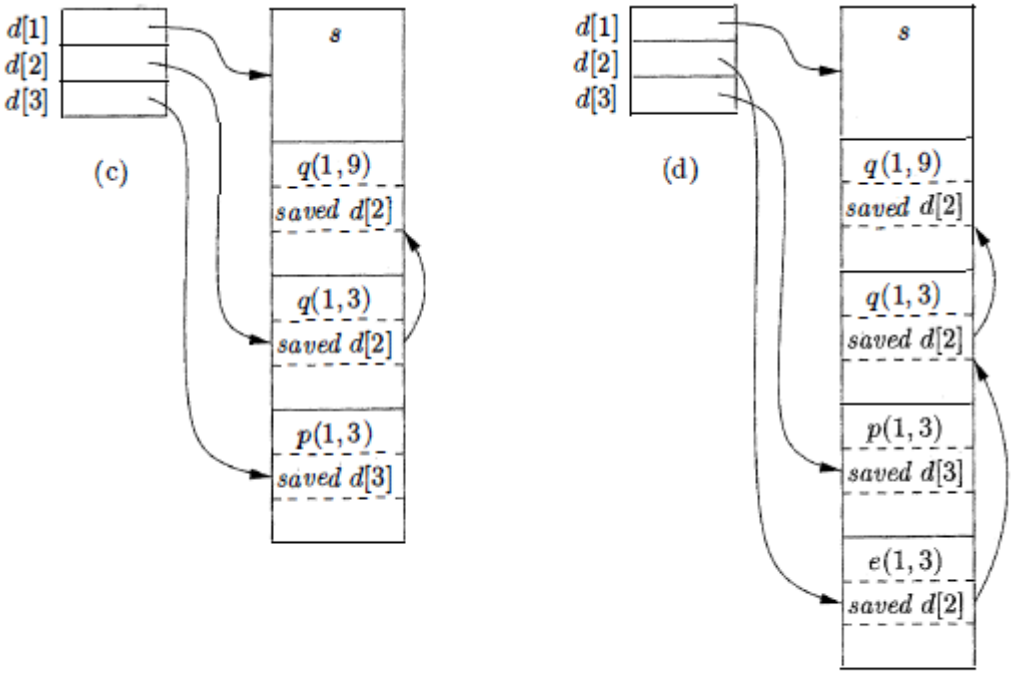


Figure 7.14: Maintaining the display

Kompilátor musí v čase kompilace určit rozvržení aktivačních záznamů a vygenerovat kód, který korektně přistupuje na místa v aktivačním záznamu. Proto *musí* být rozvržení AZ a generátor kódu navrhovány společně.

Přidělování z hromady

Strategie přidělování na zásobníku je nepoužitelná, pokud mohou hodnoty lokálních proměnných přetrvávat i po ukončení aktivace, případně pokud aktivace volaného podprogramu může přežít aktivaci volajícího. V těchto případech přidělování a uvolňování aktivačních záznamů se mohou překrývat, takže nemůžeme paměť organizovat jako zásobník. Aktivační záznamy se mohou v těchto nejobecnějších situacích přidělovat z volné oblasti paměti (hromady), která se jinak používá pro dynamické datové struktury vytvářené uživatelem. Přidělené aktivační záznamy se uvolňují až tehdy,

pokud se ukončí aktivace příslušného podprogramu nebo pokud už nejsou lokální data potřebná.

Při použití této strategie se pro vlastní přidělování a uvolňování paměti používají stejné techniky jako pro dynamické proměnné.

Správce paměti (memory manager) alokuje a dealokuje místo na heapu. V důsledku toho může dojít k fragmentaci heapu (vznik malých, nesouvislých míst = *děr*). Strategie *best fit* = alokuj nejmenší vhodnou a dostupnou díru.

Garbage collection hledá místo na heapu, které se už nepoužívá a může být proto realokované pro uchovávání dalších dat (Java, C#). Automaticky uvolňuje již nepoužívané objekty z paměti.

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>