Gestão de memória
Generalidades

José Pedro Oliveira
(jpo@di.uminho.pt)

Grupo de Sistemas Distribuídos
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Sistemas Operativos I
2006-2007

Conteúdo

Chamada ao sistema: brk

Synopsis

```
#include <unistd.h>
int brk(void *end_data_segment);
```

Sumário

Altera o topo do segmento de dados para *end_data_segment* (desde que o valor seja razoável).

Valor de retorno

sucesso - retorna o valor 0

insucesso - retorna o valor -1 e é atribuído o valor **ENOMEM** à variável **errno**.

Chamada ao sistema: sbrk

Synopsis

```
#include <unistd.h>
void *sbrk(intptr_t increment);
```

Sumário

Permite expandir o segmento de dados em *increment* bytes. Invocando **sbrk** com um incremento de 0 bytes, permite obter o endereço do topo do segmento de dados. Na realidade sbrk não é uma chamada ao sistema mas sim uma função *wrapper*.

Valor de retorno

sucesso - retorna o endereço base da nova área

insucesso - retorna o valor -1 e é atribuído o valor **ENOMEM** à variável **errno**.

Chamada ao sistema: sbrk - exemplo 1

Exemplo 1 - topo do segmento de dados

```
1  #include <unistd.h>
2  #include <stdio.h>
3
4  int main(void)
5  {
6      printf("Topo do segmento de dados: %p\n", sbrk(0));
7
8      return 0;
9  }
```

Chamada ao sistema: sbrk - exemplo 2

Exemplo 2 - expandir o segmento de dados

```
1  #include <unistd.h>
2  #include <stdio.h>
3
4  int main(void)
5  {
6      printf("Topo do segmento de dados : %p\n", sbrk(0));
7
8      printf("Endereco base da nova area: %p\n", sbrk(256));
9
10     printf("Topo do segmento de dados : %p\n", sbrk(0));
11
12     return 0;
13 }
```

Conteúdo

Funções da biblioteca C

Synopsis

```
#include <stdlib.h>

void *malloc(size_t size);

void *calloc(size_t nmemb, size_t size);

void *realloc(void *ptr, size_t size);

void free(void *ptr);
```

Funções da biblioteca C: calloc e malloc

malloc()

Aloca *size* bytes e retorna o endereço (*pointer*) da memória alocada. A memória alocada não é inicializada.

calloc()

Aloca um vector de *nmemb* elementos de *size* bytes cada e retorna o endereço (*pointer*) da memória alocada. O bloco de memória alocado é preenchido com o valor 0.

Valor de retorno

As funções **calloc** e **malloc** retornam NULL em caso de erro.

Funções da biblioteca C: free

free()

Liberta o bloco de memória cujo endereço é dado pelo parâmetro *ptr*. O endereço deve ter sido previamente obtido através da invocação de **malloc()**, **calloc()** ou **realloc()**.

Valor de retorno

A função **free** não retorna nada.

brk/sbrk e malloc

Exemplo

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(void)
6  {
7      int i;
8      printf("Topo : %p\n", sbrk(0));
9
10     for (i = 0; i < 10; i++) {
11         printf("Bloco: %p  Topo: %p\n",
12                malloc(32768), sbrk(0));
13     }
14     return 0;
15 }
```

Conteúdo

Exercício

Enunciado

Implementar um sistema de gestão de memória dinâmica que permita substituir o da biblioteca C.

Interface

- função **void * myalloc(unsigned int nbytes)**
- função **void myfree(void * ptr)**
- função **void mydump(void)** *(debug)*

Exercício - considerações

Considerações

- estrutura de controlo
  - campos (atenção ao espaço ocupado)
- lista de blocos ocupados
- lista de blocos livres
  - lista não-ordenada (tempo de pesquisa)
  - lista ordenada por dimensão do bloco
- fragmentação de memória
  - dimensão do bloco mínino
  - juntar blocos livres contíguos

Conteúdo

Referências

Referências

- **A Memory Allocator**
  Doug Lea
  `http://gee.cs.oswego.edu/dl/html/malloc.html`
- **Unix And C/C++ Runtime Memory Management For Programmers**
  `http://users.actcom.co.il/~choo/lupg/tutorials/unix-memory/unix-memory.html`
- **Inside memory management**
  `http://www-128.ibm.com/developerworks/library/l-memory/`
- **The C Programming Language**
  `http://en.wikipedia.org/wiki/C_programming_language`

Conteúdo

Linux e Glibc

Exemplo 1 - null pointer

```
1  #include <stdlib.h>
2
3  int main(void)
4  {
5      int *p = NULL;
6
7      *p = 1;
8
9      return 0;
10 }
```

Linux e Glibc

```
$ ./glibc_null_pointer
Segmentation fault
```

```
$ ulimit -c unlimited
$ ./glibc_null_pointer
Segmentation fault (core dumped)
```

```
$ gdb ./glibc_null_pointer core.12382

GNU gdb Red Hat Linux (6.3.0.0-1.134.fc5rh)
...
Core was generated by `./glibc_null_pointer'.
Program terminated with signal 11, Segmentation fault.
...
#0  0x0804836f in main () at glibc_null_pointer.c:7
7               *p = 1;
(gdb)
```

Linux e Glibc

Exemplo 2 - libertar o mesmo bloco duas vezes

```
1  #include <stdlib.h>
2
3  int main(void)
4  {
5      int *p = (int *) malloc(5 * sizeof(int));
6
7      free(p);
8      free(p);
9
10     return 0;
11 }
```

Linux e Glibc

```
$ ./glibc_double_free
```

```
*** glibc detected *** ./glibc_double_free: double free or corruption (fasttop): 0x09f47008 ***
======= Backtrace: =========
/lib/libc.so.6[0x43282a68]
/lib/libc.so.6(__libc_free+0x78)[0x43285f6f]
./glibc_double_free[0x80483ea]
/lib/libc.so.6(__libc_start_main+0xdc)[0x432344e4]
./glibc_double_free[0x8048331]
======= Memory map: ========
...
09f47000-09f68000 rwxp 09f47000 00:00 0          [heap]
43201000-43202000 r-xp 43201000 00:00 0          [vdso]
43202000-4321b000 r-xp 00000000 03:09 1102182    /lib/ld-2.4.so
...
b7f03000-b7f04000 rw-p b7f03000 00:00 0
bf98c000-bf9a1000 rw-p bf98c000 00:00 0          [stack]
Aborted (core dumped)
```

Linux e Glibc

Exemplo 3 - libertar um bloco não alocado

```
1  #include <stdlib.h>
2
3  int main(void)
4  {
5      int *p = (int *) malloc(5 * sizeof(int));
6
7      free(p + 1);
8
9      return 0;
10 }
```

Linux e Glibc

```
$ ./glibc_free_invalid_pointer
```

```
*** glibc detected *** ./glibc_free_invalid_pointer: free(): invalid pointer: 0x0961200c ***
======= Backtrace: =========
/lib/libc.so.6[0x43282a68]
/lib/libc.so.6(__libc_free+0x78)[0x43285f6f]
./glibc_free_invalid_pointer[0x80483e2]
/lib/libc.so.6(__libc_start_main+0xdc)[0x432344e4]
./glibc_free_invalid_pointer[0x8048331]
======= Memory map: ========
...
09612000-09633000 rwxp 09612000 00:00 0          [heap]
43201000-43202000 r-xp 43201000 00:00 0          [vdso]
43202000-4321b000 r-xp 00000000 03:09 1102182    /lib/ld-2.4.so
...
b7f38000-b7f39000 rw-p b7f38000 00:00 0
bffc0000-bffd6000 rw-p bffc0000 00:00 0          [stack]
Aborted (core dumped)
```

Linux e Glibc - mtrace

mtrace

Mecanismo para detectar perdas de memória:

- função **mtrace()** - activar o tracing
- função **muntrace()** - desactivar o tracing
- variável de ambiente **MALLOC_TRACE** - nome do ficheiro de trace a criar
- utilitário **mtrace** (RPM glibc-utils) - processar dados existentes no ficheiro de trace gerado durante a execução do programa

Referência adicional

```
$ info libc "Allocation Debugging"
```

Linux e Glibc - exemplo mtrace

Exemplo 4 - malloc_trace.c

```
1  #include <stdlib.h>
2  #include <mcheck.h>
3
4  int main(void)
5  {
6      int * p1, * p2;
7
8      mtrace();
9      p1 = (int *) malloc(10 * sizeof(int));
10     p2 = (int *) malloc(10 * sizeof(int));
11     free(p2);
12     muntrace();
13
14     return 0;
15 }
```

Apêndice - Biblioteca glibc

Linux e Glibc - exemplo mtrace

Compilar

```
$ gcc  -Wall  -Wextra  -g  malloc_trace.c
```

Executar programa

```
$ MALLOC_TRACE=trace.txt  ./a.out
```

Processar ficheiro gerado

```
$ mtrace  ./a.out  trace.txt

Memory not freed:
-----------------
   Address     Size     Caller
0x08aeb378     0x28  at .../memory/mtrace.c:9
```

José Pedro Oliveira Gestão de memória