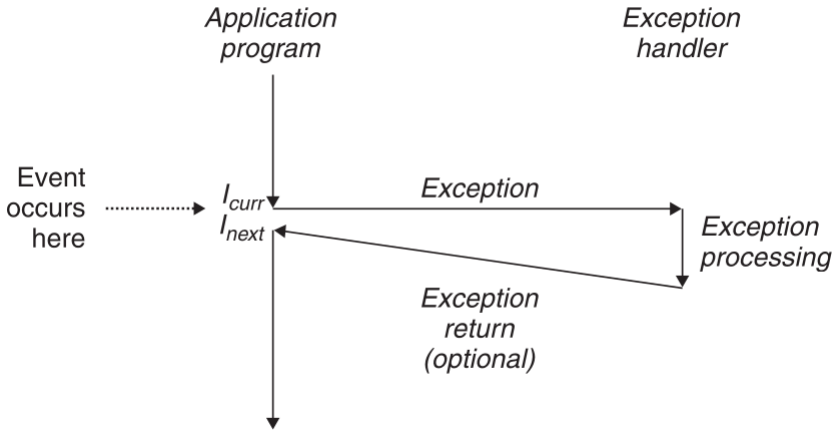
Lista 0

tags: SO

Zadanie 1

Zadanie 1. Opisz różnice między **przerwaniem sprzętowym** (ang. *hardware interr*
sora (ang. *exception*) i **pułapką** (ang. *trap*). Dla każdego z nich podaj co najmniej
które je wyzwalają. W jakim scenariuszu wyjątek procesora nie oznacza błędu czas
Kiedy pułapka jest generowana w wyniku prawidłowej pracy programu?

Wyjątek procesora - sposób kontroli przepływu sterowania, którego implementacja opiera się na bliskiej współpracy między sprzętem a systemem operacyjnym. Wyjątek jest zgłaszany w odpowiedzi na *zdarzenie*, może ono być związane z obecnie wykonywaną



instrukcją (np. dzielenie przez zero) lub też czymś bardziej zewnętrznym (np. sygnał z zegara)

Fault - wyjątek wywołany błędem wykonania instrukcji, który potencjalnie może zostać naprawiony. Jeśli naprawa jest pomyślna kontrola zostaje przekazana z powrotem do instrukcji, wpp. proces jest terminowany np. page fault (naprawialny), general protection fault, divide error.

Przerwanie sprzętowe - asynchroniczny wyjątek procesora, wywołany zewnętrznym zdarzeniem np. sygnałem z klawiatury, sygnałem z zegara lub obsługą karty sieciowej.

Pułapka - wyjątki stanowiące pewnego rodzaju interfejs jądra systemu używane w charakterze zleceń zadań, które wykonać może tylko kernel np. read, fork exit.

Class	Cause	Async/Sync	Return behavior
Interrupt	Signal from I/O device	Async	Always returns to next instruction
Trap	Intentional exception	Sync	Always returns to next instruction
Fault	Potentially recoverable error	Sync	Might return to current instruction
Abort	Nonrecoverable error	Sync	Never returns

W jakim scenariuszu wyjątek procesora nie oznacza błędu czasu wykonania programu?

Jeśli wyjątek jest naprawialny, przykładowo gdy dla wyjątku *page fault* można wczytać potrzebną stronę z dysku do pamięci.

Kiedy pułapka jest generowana w wyniku prawidłowej pracy programu?

Przy wywołaniach syscall

Zadanie 2

Zadanie 2. Opisz mechanizm **obsługi przerwań** bazujący na **wektorze przerwa**
table). Co robi procesor przed pobraniem pierwszej instrukcji **procedury obsługi prz**
handler) i po natrafieniu na instrukcję powrotu z przerwania? Czemu procedura obsł
być wykonywana w **trybie jądra** (ang. *kernel mode*) i używać stosu odrębnego od s

Figure 8.2
Exception table. The exception table is a jump table where entry k contains the address of the handler code for exception k.

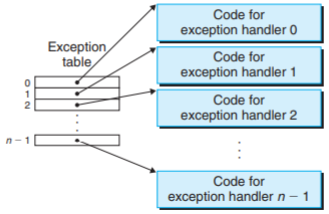
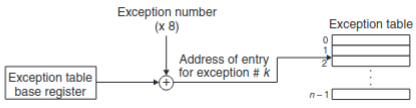


Figure 8.3
Generating the address of an exception handler. The exception number is an index into the exception table.



:::info 0. każdy wyjątek ma unikalny nieujemny numer

1. przy uruchomieniu systemu tworzona jest tablica adresów obsługujących wyjątki o odpowiednich numerach
2. w sytuacji wykrycia *zdarzenia* procesor zgłasza wyjątek o stosownym dla tego zdarzenia numerze. Adres w tablicy wyjątków jest wyliczany na podstawie wskaźnika na tą tablicę (exeception table base register) oraz numeru wyjątku. :::

Co robi procesor przed pobraniem pierwszej instrukcji procedury obsługi przerwania?

Procesor wrzuca na stos adres powrotu do obecnej (lub następnej w zależności od klasy wyjątku) instrukcji oraz dodatkowe informacje potrzebne do wznowienia przerwanego programu i przechodzi w tryb jądra.

Co robi procesor po natrafieniu na instrukcję powrotu z przerwania?

Przywraca odpowiedni stan procesora i rejestrów oraz przechodzi z powrotem w tryb użytkownika.

Czemu procedura obsługi przerwania powinna być wykonywana w trybie jądra?

Tryb jądra daje procedurze dostęp do całości zasobów komputera w szczególności do obszarów pamięci niedostępnych dla trybu użytkownika. Ten poziom uprzywilejowania może być przykładowo potrzebny do wykonania odczytu z dysku twardego.

Czemu procedura obsługi przerwania powinna używać stosu odrębnego od stosu użytkownika?

Użytkownik mógłby wówczas zmienić wartość %rsp w bezsensowny sposób.

Zadanie 3

Zadanie 3. Bazując na formacie ELF (ang. *Executable and Linkable Format*) opisz s
walnego. Czym różni się **sekcja** od **segmentu**? Co opisują **nagłówki programu**? S
wie, pod jakim adresem ma umieścić segmenty programu i gdzie położona jest pierwsz

Wskazówka: Skorzystaj z narzędzia «`readelf`».

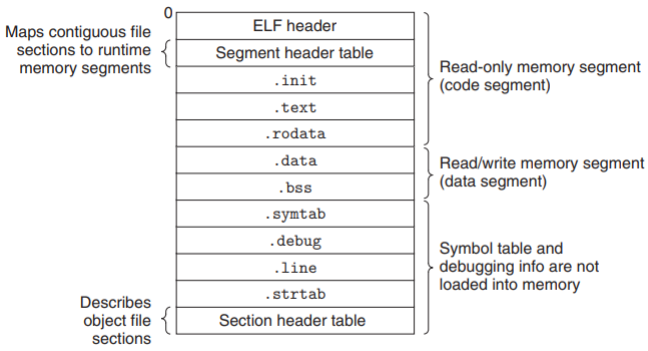


Figure 7.11 **Typical ELF executable object file.**

::: info **Sekcje:**

- .init -- funkcja wywoływana przez przy inicjalizacji programu
- .text -- kod programu
- .rodata -- read only data
- .data -- zmienne globalne
- .bss -- zmienne inicjowane zerami
- .symtab -- tablica symboli
- .debug -- informacje dla debuggera
- .line -- informacje o wierszach do debuggowania
- .strtab -- tablica stringów :::

Możemy zajrzeć do takiego pliku używając `readelf` z flagami odpowiednio: `--file-header`, `--segments`, `--sections`

- ELF header (adres pierwszej instrukcji programu i ogólny opis formatu pliku)

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x401100
  Start of program headers:          64 (bytes into file)
  Start of section headers:          29792 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         38
  Section header string table index: 37
```

- segment header (między innymi adresy segmentów tj.VirtAddr i PhysAddr)

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000400318 0x0000000000400318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x00000000000007f8 0x00000000000007f8  R      0x1000
  LOAD           0x0000000000001000 0x0000000000401000 0x0000000000401000
                 0x00000000000007d5 0x00000000000007d5  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000402000 0x0000000000402000
                 0x0000000000000290 0x0000000000000290  R      0x1000
  LOAD           0x0000000000002e10 0x0000000000403e10 0x0000000000403e10
                 0x0000000000000274 0x00000000000002c0  RW     0x1000
  DYNAMIC        0x0000000000002e20 0x0000000000403e20 0x0000000000403e20
                 0x00000000000001d0 0x00000000000001d0  RW     0x8
  NOTE           0x0000000000000338 0x0000000000400338 0x0000000000400338
                 0x0000000000000020 0x0000000000000020  R      0x8
  NOTE           0x0000000000000358 0x0000000000400358 0x0000000000400358
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_PROPERTY   0x0000000000000338 0x0000000000400338 0x0000000000400338
                 0x0000000000000020 0x0000000000000020  R      0x8
  GNU_EH_FRAME   0x000000000000203c 0x000000000040203c 0x000000000040203c
                 0x000000000000007c 0x000000000000007c  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000002e10 0x0000000000403e10 0x0000000000403e10
                 0x00000000000001f0 0x00000000000001f0  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt
   03     .init .plt .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .got.plt .data .bss
   06     .dynamic
   07     .note.gnu.property
   08     .note.gnu.build-id .note.ABI-tag
   09     .note.gnu.property
   10     .eh_frame_hdr
   11
   12     .init_array .fini_array .dynamic .got
```

- section header

```
Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000400318  00000318
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.pr[...] NOTE             0000000000400338  00000338
       0000000000000020  0000000000000000   A       0     0     8
```

Sekcje i segmenty

- Sekcje są wykorzystywane w procesie linkowania i relokacji
- Segmenty przechowują dane wykorzystywane w czasie wykonywania pliku

---

Adres pierwszej instrukcji jest w *ELF header* Adresy docelowe segmentów są w *segment header*

Zadanie 4

Zadanie 4. Zapoznaj się z rozdziałami 3.4 i A.2 dokumentu System V Application I
Architecture Processor Supplement[2] i odpowiedz na następujące pytania:

- W jaki sposób jądro musi przygotować **przestrzeń adresową** procesu? Co musi
w momencie wywołania procedury «_start»? Do czego służy auxiliary vecto
wydając przykładowe polecenie «LD_SHOW_AUXV=1 /bin/true».
- W jaki sposób wywołać funkcję jądra? W których rejestrach należy umieścić ar
spodziewać się wyników i jak jądro sygnalizuje niepowodzenie **wywołania sys**

Zadanie 5

Zadanie 5. Opisz znaczenie słowa kluczowego «volatile» w języku C. Wymień co n
w których użycie wskaźników do ulotnej zawartości pamięci jest niezbędne dla poprawn

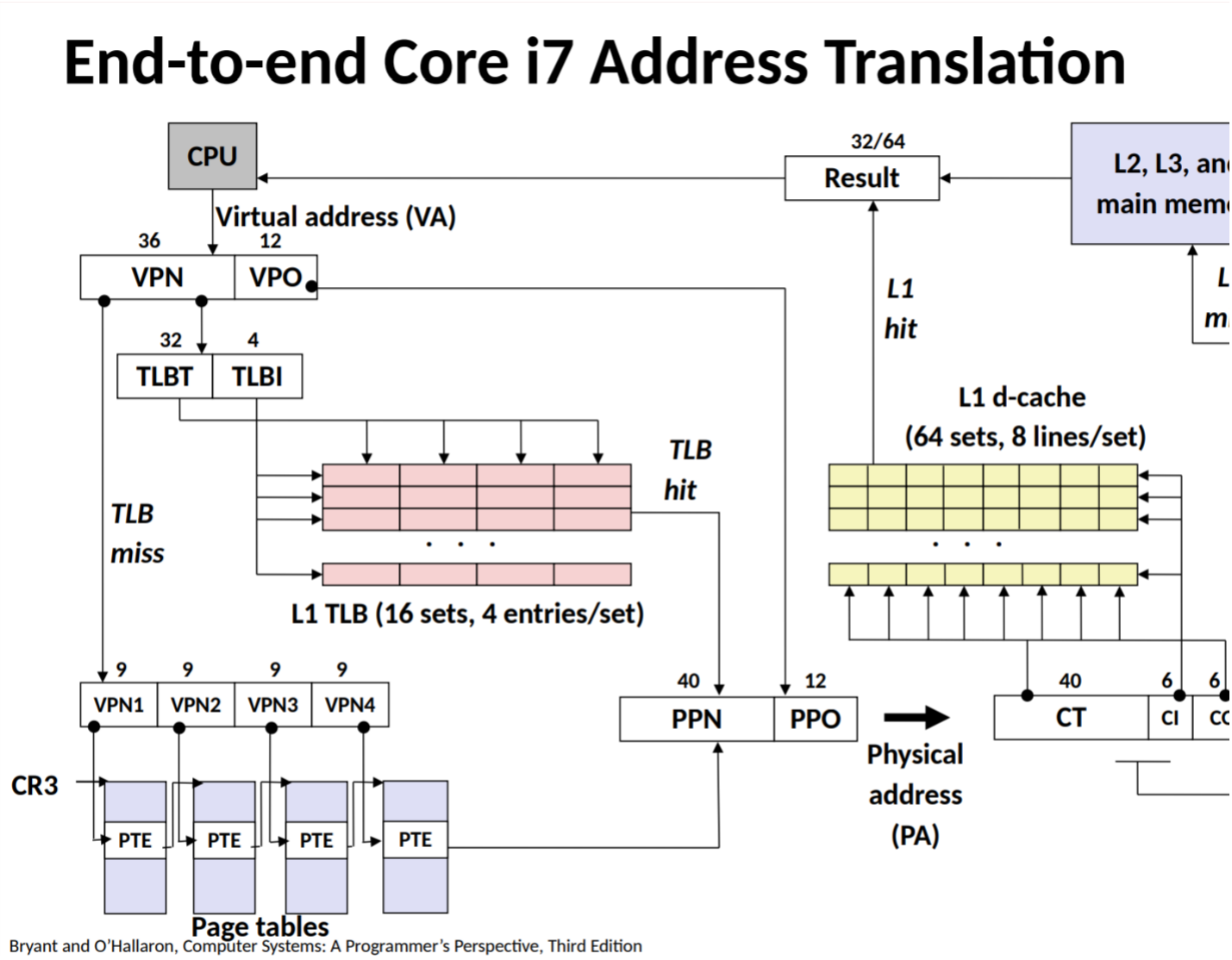
Uwaga: Zadaniem słowa kluczowego «volatile» nie jest wyłączenie optymalizacji!

Modyfikator typu *volatile* oznacza, że zmienna może zostać zmodyfikowana z zewnątrz, tj. pomiędzy różnymi odczytami, wartości zmiennej mogą być różne, nawet jeśli nie były modyfikowane w kodzie. Zastosowanie *volatile* powstrzymuje kompilator optymalizujący przed pomijaniem zapisów do pamięci. Zmienne *volatile* pojawiają się przede wszystkim w dostępie do sprzętu, gdzie pamięć jest wykorzystywana do komunikacji pomiędzy urządzeniami oraz w środowisku wielowątkowym, w którym różne wątki mogą korzystać z tej samej zmiennej.

Zadanie 6

Zadanie 6. Przypomnij jak wygląda mechanizm **tłumaczenia adresów** bazujący na
stron procesorów z rodziny x86–64. Przedstaw algorytm obliczania **adresu fizyczneg**
wirtualnego z uwzględnieniem uprawnień dostępu. Jaką rolę w procesie tłumaczenia

adres fizyczny –



Zadanie 7

Zadanie 7. Uruchom program «`1_ls`» pod kontrolą narzędzia «`ltrace -S`». Na po
programu zidentyfikuj, które z **wywołań systemowych** są używane przez procedury: «
«`printf`» i «`closedir`». Do czego służy wywołanie systemowe «`brk`»? Używając debu
«`catch syscall brk`» zidentyfikuj, która funkcja używa «`brk`».

wywołanie systemowe – interfejs pomiędzy programem użytkownika a jądrem systemu, pozwala na wykonanie tych zadań, do których proces użytkownika nie ma uprawnień.

Wydruk z `ltrace -S`:

```
(...)

opendir("." <unfinished ...>
openat@SYS(AT_FDCWD, ".", 0x90800, 00)                                  = 3
newfstatat@SYS(3, "", 0x7ffc2ac035b0, 0x1000)                           = 0
brk@SYS(nil)                                                            = 0x14ce000
brk@SYS(0x14ef000)
<... opendir resumed> )                                                 = { 3 }
readdir({ 3 } <unfinished ...>
getdents64@SYS(3, 0x14ce2d0, 0x8000, 0x7fe400ad0590)                    = 296
<... readdir resumed> )                                                 = { 708068, "." }
puts("." <unfinished ...>
newfstatat@SYS(1, "", 0x7ffc2ac03500, 0x1000)                           = 0
write@SYS(1, ".\n", 2.
)                                                                     = 2
<... puts resumed> )                                                    = 2
readdir({ 3 })                                                          = { 693072, ".." }
puts(".." <unfinished ...>
write@SYS(1, "..\n", 3..
)                                                                     = 3
<... puts resumed> )                                                    = 3
readdir({ 3 })                                                          = { 771331, "1_ls.c" }
puts("1_ls.c" <unfinished ...>
write@SYS(1, "1_ls.c\n", 71_ls.c
)                                                                   = 7
<... puts resumed> )                                                    = 7
readdir({ 3 })                                                          = { 771336, "2_cat.c" }
puts("2_cat.c" <unfinished ...>
write@SYS(1, "2_cat.c\n", 82_cat.c
)                                                                   = 8
<... puts resumed> )                                                    = 8
readdir({ 3 })                                                          = { 771334, "include" }
puts("include" <unfinished ...>
write@SYS(1, "include\n", 8include
)                                                                   = 8
<... puts resumed> )                                                    = 8
readdir({ 3 })                                                          = { 771332, "libapue" }
puts("libapue" <unfinished ...>
write@SYS(1, "libapue\n", 8libapue
)                                                                   = 8
<... puts resumed> )                                                    = 8
readdir({ 3 })                                                          = { 771337, "Makefile" }
puts("Makefile" <unfinished ...>
write@SYS(1, "Makefile\n", 9Makefile
)                                                                   = 9
<... puts resumed> )                                                    = 9
readdir({ 3 })                                                          = { 771361, "1_ls" }
puts("1_ls" <unfinished ...>
write@SYS(1, "1_ls\n", 51_ls
)                                                                     = 5
<... puts resumed> )                                                    = 5
readdir({ 3 })                                                          = { 771362, "2_cat" }
puts("2_cat" <unfinished ...>
write@SYS(1, "2_cat\n", 62_cat
)                                                                     = 6
<... puts resumed> )                                                    = 6
readdir({ 3 })                                                          = { 772407, ".gdb_history" }
puts(".gdb_history" <unfinished ...>
write@SYS(1, ".gdb_history\n", 13.gdb_history
)                                                                  = 13
<... puts resumed> )                                                    = 13
readdir({ 3 } <unfinished ...>
getdents64@SYS(3, 0x14ce2d0, 0x8000, 119)                               = 0
<... readdir resumed> )                                                 = nil
closedir({ 3 } <unfinished ...>
close@SYS(3)                                                            = 0
<... closedir resumed> )                                                = 0
exit(0 <unfinished ...>
exit_group@SYS(0 <no return ...>
+++ exited (status 0) +++
```

Wywołania "opendir": openat, newfstatat, brk

```
opendir("." <unfinished ...>
openat@SYS(AT_FDCWD, ".", 0x90800, 00)                                  = 3
newfstatat@SYS(3, "", 0x7ffc2ac035b0, 0x1000)                           = 0
brk@SYS(nil)                                                            = 0x14ce000
brk@SYS(0x14ef000)
<... opendir resumed> )
```

Wywołania "readdir": getdents64

```
readdir({ 3 } <unfinished ...>
getdents64@SYS(3, 0x14ce2d0, 0x8000, 0x7fe400ad0590)                    = 296
<... readdir resumed> )
```

Wywołania "printf": newfstatat, write

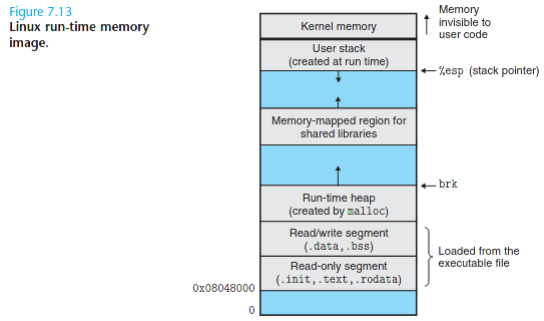
```
puts("." <unfinished ...>
newfstatat@SYS(1, "", 0x7ffc2ac03500, 0x1000)                           = 0
write@SYS(1, ".\n", 2.
)                                                                     = 2
<... puts resumed> )
```

Wywołania "closedir": close

```
closedir({ 3 } <unfinished ...>
close@SYS(3)                                                            = 0
<... closedir resumed> )
```

Do czego służy wywołanie systemowe brk?

Wywołanie systemowe brk rozszerza stertę




```
DESCRIPTION
brk()  and  sbrk()  change the location of the program break, which defines the end of the process's data segment (i.e., the program break is the first location after the end of the uninitialized data segment).  In
the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

brk() sets the end of the data segment to the value specified by addr, when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size (see setrlimit(2)).

sbrk() increments the program's data space by increment bytes.  Calling sbrk() with an increment of 0 can be used to find the current location of the program break.
```

GDB

```
gdb 1_ld
catch syscall brk
run .
continue
continue
```

```
Stack
[0] from 0x00007ffff7ed7fbb in brk
[1] from 0x00007ffff7ed803c in sbrk
[2] from 0x00007ffff7e6ed8d in __default_morecore
[3] from 0x00007ffff7e6a585 in sysmalloc
[4] from 0x00007ffff7e6c1ee in _int_malloc
[5] from 0x00007ffff7e6cd6b in tcache_init.part
[6] from 0x00007ffff7e6d1de in malloc
[7] from 0x00007ffff7ea9595 in __alloc_dir
[8] from 0x00007ffff7ea9613 in opendir_tail
[9] from 0x000000000040120c in main+38 at 1_ls.c:11
```

Jak widać z brk korzysta malloc

Zadanie 8

Zadanie 8. Pod kontrolą narzędzia «strace» uruchom program «2_cat» korzystaj
wołań systemowych do interakcji ze **standardowym wejściem i wyjściem**. Pokaż,
odczyt na deskryptorze pliku 0 i pisze do pliku o deskryptorze 1. Naciśnij kombina
kończąc wejściowy strumień danych – co zwróciło «read»? Zmodyfikuj program
podanego w linii poleceń. Co się stanie, jeśli przekażesz **ścieżkę** do katalogu zamias

deskryptor pliku – numer unikalnie identyfikujący otwarty w systemie plik. **standardowe wejście i wyjście** – kanał komunikacyjny między programem a jego środowiskiem poprzez domyślne deskryptory pliku (0 dla wejścia, 1 dla wyjścia). **ścieżka** – ciąg znaków unikalnie identyfikujący położenie pliku w systemie plików.

Po wprowadzeniu `strace ./2_cat` program zatrzymuje się czekając na input, widać tutaj również, że jest to deskryptor o numerze 0:

```
mprotect(0x7fdcf3d9b000, 8192, PROT_READ) = 0
munmap(0x7fdcf3d56000, 77508)           = 0
read(0,
```

Po wpisaniu `foo` i wciśnięciu kombinacji Ctrl+D program wypisuje dekryptorem o numerze 0 a następnie kończy działanie zwracając 0:

```
read(0, foo
"foo\n", 4096)                  = 4
write(1, "foo\n", 4foo
)                       = 4
read(0, "", 4096)                       = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Modyfikacja programu

```c
#include "apue.h"
#include <fcntl.h>

#define BUFFSIZE 4096

int main(int argc, char **argv) {
  int n;
  char buf[BUFFSIZE];


  int file = open(argv[1], O_RDONLY, 0);
  while ((n = read(file, buf, BUFFSIZE)) > 0)
    if (write(STDOUT_FILENO, buf, n) != n)
      err_sys("write error");

  if (n < 0)
    err_sys("read error");

  close(file);

  exit(0);
}
```

```
☁  programy  ./my_cat my_cat.c
#include "apue.h"
#include <fcntl.h>

#define BUFFSIZE 4096

int main(int argc, char **argv) {
  int n;
  char buf[BUFFSIZE];


  int file = open(argv[1], O_RDONLY, 0);
  while ((n = read(file, buf, BUFFSIZE)) > 0)
    if (write(STDOUT_FILENO, buf, n) != n)
      err_sys("write error");

  if (n < 0)
    err_sys("read error");

  close(file);

  exit(0);
}
```

Wynik dla pliku:

```
☁  programy  ./my_cat .
read error: Is a directory
```

Wynik dla ścieżki