# OS 2019

MIT ;)

https://pdos.csail.mit.edu/6.828/2019

# Alokacia pamate

# Struktura prednasky

- Motivacia
- Struktura programu v pamati
- Rozne implementacie alokatora pamate pre haldu

# Systemove programovanie

# Systemove programovanie

- Pouziva sa na tvorbu sluzieb pre aplikacie

# Systemove programovanie

- Pouziva sa na tvorbu sluzieb pre aplikacie
- Vyuziva priamo sluzby OS

# Systemove programovanie

- Pouziva sa na tvorbu sluzieb pre aplikacie
- Vyuziva priamo sluzby OS
- Priklady
  - Utility prikazoveho riadku
  - Rozne kniznice
  - Ssh, prihlasovaci program, ...

# Vyzvy pri SP

# Vyzvy pri SP

- Nizkourovnove programove prostredie
  - Napr. Obmedzenie velkosti cisel (max. 64 bit)
  - Nutna vlastna sprava pamate pre objekty (alokacia, uvolnovanie)

# Vyzvy pri SP

- Nizkourovnove programove prostredie
  - Napr. Obmedzenie velkosti cisel (max. 64 bit)
  - Nutna vlastna sprava pamate pre objekty (alokacia, uvolnovanie)
- Konkurencia
  - Problem zabezpecit sucasne vybavenie poziadaviek

# Vyzvy pri SP

- Nizkourovnove programove prostredie
  - Napr. Obmedzenie velkosti cisel (max. 64 bit)
  - Nutna vlastna sprava pamate pre objekty (alokacia, uvolnovanie)
- Konkurencia
  - Problem zabezpecit sucasne vybavenie poziadaviek
- Ladenie
  - Zvacsa ide o ladenie aplikacii v jazyku C

# Vyzvy pri SP

- Nizkourovnove programove prostredie
  - Napr. Obmedzenie velkosti cisel (max. 64 bit)
  - Nutna vlastna sprava pamate pre objekty (alokacia, uvolnovanie)
- Konkurencia
  - Problem zabezpecit sucasne vybavenie poziadaviek
- Ladenie
  - Zvacsa ide o ladenie aplikacii v jazyku C
- Vykon
  - Neumensovat vykon hw pri sw usmernovani

# Dynamicka alokacia pamate

# Dynamicka alokacia pamate

- Zakladna systemova sluzba

# Dynamicka alokacia pamate

- Zakladna systemova sluzba

- Absolutne nevyhnutna pri jazyku C
  - Pointerova aritmetika, pretypovanie...

# Dynamicka alokacia pamate

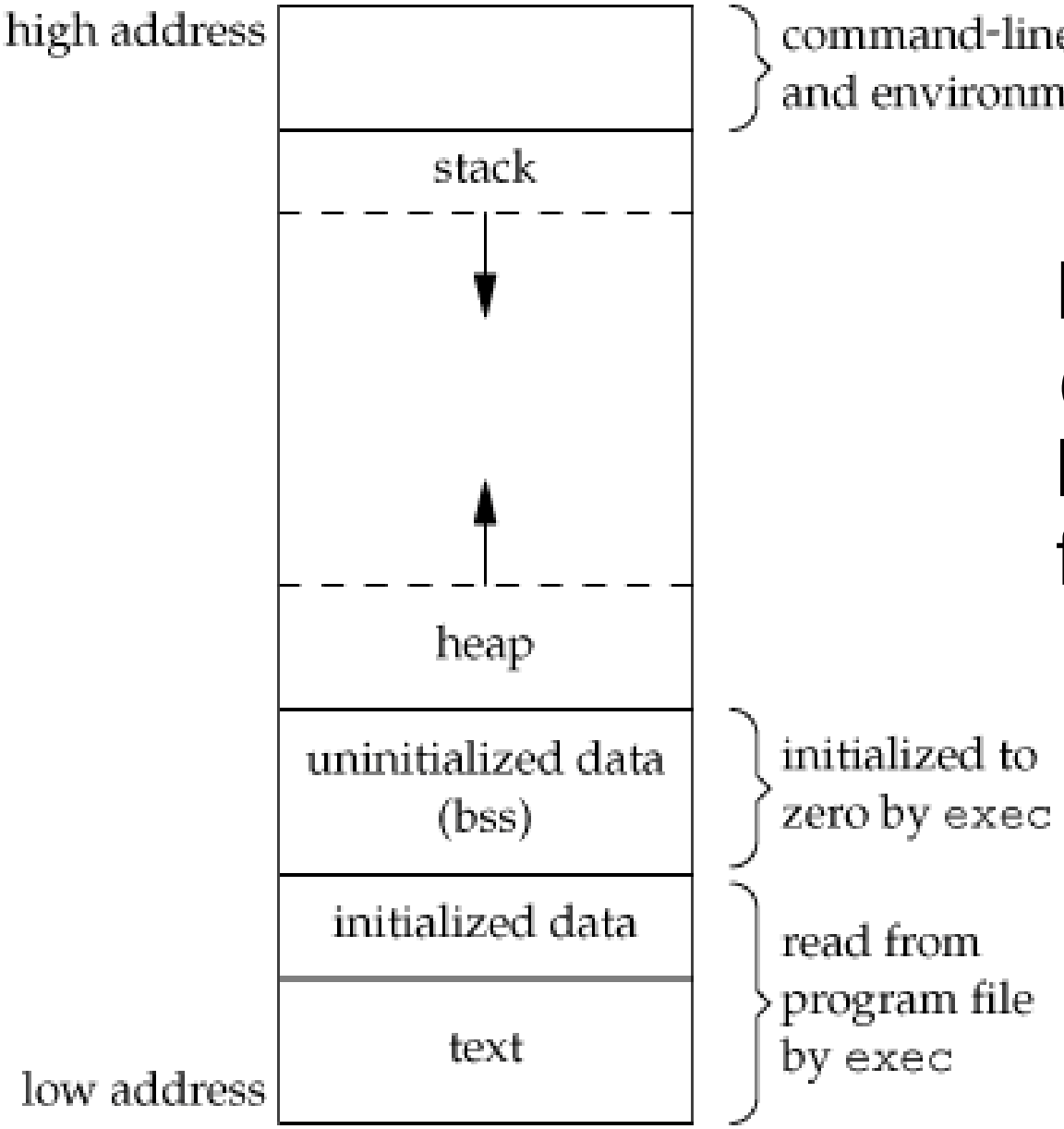
- Zakladna systemova sluzba
- Absolutne nevyhnutna pri jazyku C
  - Pointerova aritmetika, pretypovanie…
- Vykon alokacie je velmi dolezity

# Dynamicka alokacia pamate

- Zakladna systemova sluzba
- Absolutne nevyhnutna pri jazyku C
  - Pointerova aritmetika, pretypovanie…
- Vykon alokacie je velmi dolezity
  - Vacsinou byva uzkym hrdlom aplikacii
  - Velmi casto sa s alokatormi stretnete aj v praxi...

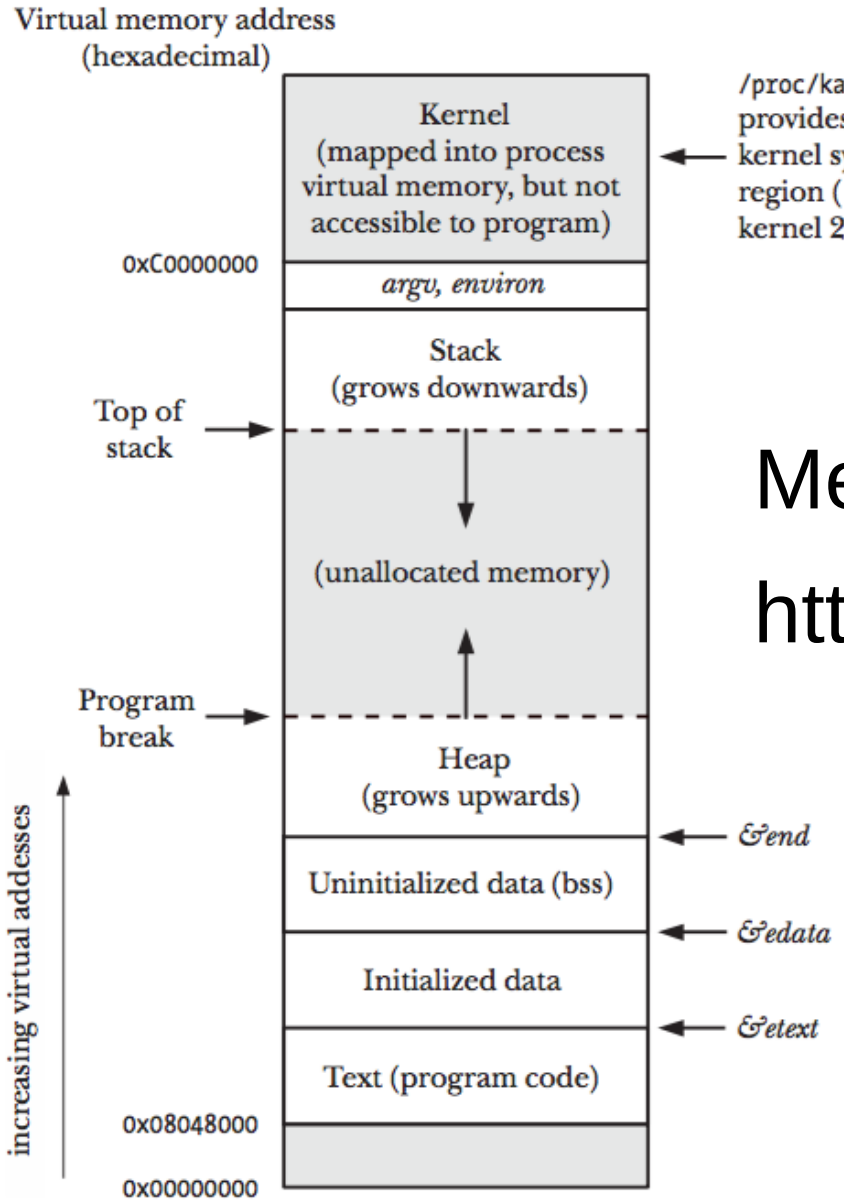
# Struktura programu

# Struktura programu




https://medium.com/@vikasv210/memory-layout-in-c-fe4dffdaeed6

# Struktura programu




Memory Layout of a Process

https://notes.shichao.io/tlpi/ch6/

# Pamat programu

# Pamat programu

- Datovy segment
  - 
- Zasobnik
  - 
  - 
- Halda
  - 
  - 

# Pamat programu

- Datovy segment
  - Staticka pamat, data pritomne pocas behu programu
- Zasobnik
  -
  -
- Halda
  -
  -

# Pamat programu

- Datovy segment
  - Staticka pamat, data pritomne pocas behu programu
- Zasobnik
  - Alokovanie pri vstupe do funkcie
  - Uvolnovanie pri vychode z funkcie
- Halda
  -
  -

# Pamat programu

- Datovy segment
  - Staticka pamat, data pritomne pocas behu programu; data instaluje zavadzac
- Zasobnik
  - Alokovanie pri vstupe do funkcie
  - Uvolnovanie pri vychode z funkcie
- Halda
  - Nutne explicitne rozhranie (malloc/free)
  - Viacere implementacie v tejto prednaske

# Poziadavky na alokator pre haldu

# Poziadavky na alokator pre haldu

- Rychlost alokacie / uvolnovania
- Nizke rezijne naklady na pamat
- Vyuzitie celej pamate
  - Co najviac sa vyhybat fragmentacii

# Jednoducha implementacia

# Jednoducha implementacia

- Priklad tejto implementacie vid kalloc() v xv6
- Reprezentuje haldu ako zoznam blokov usporiadanych podla adresy
- Priklad 200B halda

# Jednoducha implementacia

[ volne ]

0 200

malloc(100) → 0

malloc(50) → 100

[ AA | BB | volne ]

0 100 150 200

# Jednoducha implementacia

- Treba riesit API… preco?

# Jednoducha implementacia

- Treba riesit API… preco? free(100)

# Jednoducha implementacia

- Treba riesit API… preco? Free(100)
- Aka je velkost oblasti BB?

# Jednoducha implementacia

- Treba riesit API… preco? Free(100)
- Aka je velkost oblasti BB? Niekde musia byt ulozene informacie o velkosti bloku!

# Jednoducha implementacia

- Treba riesit API… preco? Free(100)
- Aka je velkost oblasti BB? Niekde musia byt ulozene informacie o velkosti bloku!
- Na to sa pouziva hlavicka bloku; staci nam velkost a pointer na dalsi blok

# Jednoducha implementacia

- Treba riesit API… preco? Free(100)
- Aka je velkost oblasti BB? Niekde musia byt ulozene informacie o velkosti bloku!
- Na to sa pouziva hlavicka bloku; staci nam velkost a pointer na dalsi blok (v priklade predpokladame 64-bit pre pointer aj velkost)

[H AA |H BB |H volne ]

0 16 100+16 100+32 200

# Jednoducha implementacia

- Po free(100) bude situacia nasledovna

  [H AA |H volne |H volne ]

  0 16 100+16 100+32 200

# Jednoducha implementacia

- Po free(100) bude situacia nasledovna

  [H AA |H volne |H volne ]

  0 16 100+16 100+32 200

- Kolko mame volnej pamate? Mozme urobit alloc(64)?

# Jednoducha implementacia

- Po free(100) bude situacia nasledovna

  [H AA |H volne |H volne ]

  0 16 100+16 100+32 200

- Kolko mame volnej pamate? Mozme urobit alloc(64)?
  - Volanie alloc() zlyha
  - Nemame volny blok, ktory by mal 64B

# Jednoducha implementacia

- Skusme volne bloky spojit

# Jednoducha implementacia

- Skusme volne bloky spojit

  [H AA |H volne ]

  0 16 100+16 200

# Jednoducha implementacia

- Skusme volne bloky spojit

  [H AA |H volne ]

  0 16 100+16 200

- Kolko mame volnej pamate? Mozme urobit alloc(64)?

# K&R malloc

# K&R malloc

- Pouziva ho xv6 shell, vid user/umalloc.c

# K&R malloc

- Pouziva ho xv6 shell, vid user/umalloc.c
- Algoritmus vyberu volneho bloku: first-fit

# K&R malloc

- Pouziva ho xv6 shell, vid user/umalloc.c
- Algoritmus vyberu volneho bloku: first-fit
  - Dalsie moznosti: worst-fit, best-fit

# K&R malloc

- Pouziva ho xv6 shell, vid user/umalloc.c
- Algoritmus vyberu volneho bloku: first-fit
  - Dalsie moznosti: worst-fit, best-fit
- Volne bloky spaja

# K&R malloc

- Pouziva ho xv6 shell, vid user/umalloc.c
- Algoritmus vyberu volneho bloku: first-fit
  - Dalsie moznosti: worst-fit, best-fit
- Volne bloky spaja
- Rezijne naklady pamate: ak by kazde volanie bolo malloc(16), tak 50%

# K&R malloc

- Pouziva ho xv6 shell, vid user/umalloc.c
- Algoritmus vyberu volneho bloku: first-fit
  - Dalsie moznosti: worst-fit, best-fit
- Volne bloky spaja
- Rezijne naklady pamate: ak by kazde volanie bolo malloc(16), tak 50%
- Zlozitost je variabilna
  - Bloky uvolnovane v poradi, zoznam ma 1 prvok
  - Bloky mimo poradia, zoznam velmi dlhy

# Useky pevnej dlzky

# Useky pevnej dlzky

- Pamat sa rozdeli na useky nemennej dlzky

# Useky pevnej dlzky

- Pamat sa rozdeli na useky nemennej dlzky
- Useky maju rovnaku alebo aj roznu velkost

# Useky pevnej dlzky

- Pamat sa rozdeli na useky nemennej dlzky
- Useky maju rovnaku alebo aj roznu velkost
- Kazdy usek moze vyuzivat v jednom cast iba jeden proces

# Useky pevnej dlzky

- Pamat sa rozdeli na useky nemennej dlzky
- Useky maju rovnaku alebo aj roznu velkost
- Kazdy usek moze vyuzivat v jednom cast iba jeden proces
  - Ak su useky rovnakej velkosti, hrozi riziko velkej vnutornej fragmentacie pamate
  - Ak su useky roznych velkosti, podla potreby procesu sa vyuzije najmensi volny usek; tym sa vnutorna fragmentacia minimalizuje

# Useky pevnej dlzky

- Ako sa zisti obsadenost usekov?

# Useky pevnej dlzky

- Ako sa zisti obsadenost usekov?
- Kedze pocet usekov pamate je nemenny, staci bitova mapa: 1 bit na 1 usek

# Useky pevnej dlzky

- Ako sa zisti obsadenost usekov?
- Kedze pocet usekov pamate je nemenny, staci bitova mapa: 1 bit na 1 usek

- Co ked proces pozaduje viac pamate ako je velkost najvacsieho useku?
- Co ak proces pocas svojho behu zvysuje postupne poziadavky na velkost pamate, a “vybehne” z oblasti pridelenej pamate?

# Useky premenlivej dlzky

# Useky premenlivej dlzky

- Zlepsenie systemu s fixnou dlzkou (minimalizacia internej fragmentacie)

# Useky premenlivej dlzky

- Zlepsenie systemu s fixnou dlzkou (minimalizacia internej fragmentacie)
- Ide o “jednoduchu implementaciu” spomenutu skor v prednaske

# Useky premenlivej dlzky

- Zlepsenie systemu s fixnou dlzkou (minimalizacia internej fragmentacie)
- Ide o “jednoduchu implementaciu” spomenutu skor v prednaske
- Ako sa postupne prideluju a uvolnuju useky pamate, vznikaju “diery” (volne miesta), ktore uz procesom po case nevyhovuju (prilis male useky) → externa fragmentacia

# Useky premenlivej dlzky

- Vyssie spomenute algoritmy na obsadenie bloku
  - First fit (prvy volny)
  - Next fit (nasledujuci volny)
  - Best fit (s najmensim zvyskom)
  - Worst fit (s najvacsim zvyskom)

# Useky premenlivej dlzky

- Vyssie spomenute algoritmy na obsadenie bloku
  - First fit (prvy volny) – hlada vzdy od zaciatku
  - Next fit (nasledujuci volny) – hlada od posledne najdeneho useku
  - Best fit (s najmensim zvyskom) – hlada blok, po obsadeni ktoreho ostane co najmenej nevyuzitej pamate
  - Worst fit (s najvacsim zvyskom) – hlada blok, po obsadeni ktoreho ostane co najviac nevyuzitej pamate

# Useky premenlivej dlzky

- Najrychlejsi byva “first fit”

- Problemom ostavaju zvysujuce sa poziadavky procesu na pamat (ked uz prideleny usek pamate nebude stacit)

# Useky premenlivej dlzky

- Najrychlejsi byva “first fit”
- Problemom ostavaju zvysujuce sa poziadavky procesu na pamat (ked uz prideleny usek pamate nebude stacit)
- Vid obrazok

# Buddy alokator

# Buddy alokator

- Prideluju sa useky velkosti mocniny 2
  - Proces ziada usek velkosti N
  - Dostane usek 2^i, pricom 2^(i-1) < N <= 2^i

# Buddy alokator

- Prideluju sa useky velkosti mocniny 2
  - Proces ziada usek velkosti N
  - Dostane usek 2^i, pricom 2^(i-1) < N <= 2^i
- Co ak nie je usek 2^i k dispozicii?

# Buddy alokator

- Prideluju sa useky velkosti mocniny 2
  - Proces ziada usek velkosti N
  - Dostane usek 2^i, pricom 2^(i-1) < N <= 2^i
- Co ak nie je usek 2^i k dispozicii? Rozdeli sa (rekurzivne):

  2^(i+d) = 2^(i+d-1)+2^(i+d-2)+...+2(i+1)+2^i+2^i

  napr.: 256 = 128 + 64 + 32 + 16 + 16

# Buddy alokator

- Prideluju sa useky velkosti mocniny 2
  - Proces ziada usek velkosti N
  - Dostane usek 2^i, pricom 2^(i-1) < N <= 2^i
- Co ak nie je usek 2^i k dispozicii? Rozdeli sa (rekurzivne):

  2^(i+d) = 2^(i+d-1)+2^(i+d-2)+...+2(i+1)+2^i+2^i

  napr.: 256 = 128 + 64 + 32 + 16 + 16
- Ak sa ocitnu vedla seba 2 useky 2^i (po uvolneni bloku), spoja sa (rekurzivne) do bloku 2^(i+1)

# Buddy alokator

- Algoritmus z V. Solcany, OS 2012/13, prednaska 7
- Kazda velkost ‘i’ ma vlastny zoznam usekov

```
void get_hole(int i) {
  if(i == MAX_I + 1) error();
  if(<i_list empty>) {
    get_hole(i+1);
    <split hole into buddies>;
    <put buddies on i_list>;
  }
  <take first hole on i_list>;
}
```

# Buddy alokator

- Triedy velkosti: 2^0, 2^1, 2^2, … 2^k
  - Velkost 0: najmensia velkost, napr. 16B
  - Velkost bloku = 2^i * min_size
  - min_size by mala byt dostatocna na uchovanie 2 ukazatelov pre zoznam volnych blokov

# Buddy alokator

- Triedy velkosti: 2^0, 2^1, 2^2, … 2^k
  - Velkost 0: najmensia velkost, napr. 16B
  - Velkost bloku = 2^i * min_size
  - min_size by mala byt dostatocna na uchovanie 2 ukazatelov pre zoznam volnych blokov
- Reprezentacia volnych blokov
  - Pre kazde ‘i’ jeden zoznam volnych blokov 2^i
  - Bitove vektory pre vsetky bloky vsetkych velkosti

# Buddy alokator

- Operacia malloc()
  - Alokuje prvy volny blok spravnej velkosti
  - Ak treba, rozdeli predtym blok vacsej velkosti

# Buddy alokator

- Operacia malloc()
  - Alokuje prvy volny blok spravnej velkosti
  - Ak treba, rozdeli predtym blok vacsej velkosti
- Operacia free()
  - Urci sa velkost bloku
  - Pokym jestvuje volny sused (‘buddy’) rovnakej velkosti, spajaj do vacsieho
  - Vloz blok do prislusneho zoznamu volnych blokov

# Buddy alokator

- Zaujimave odkazy

- https://en.wikipedia.org/wiki/Buddy_memory_allocation
- http://bitsquid.blogspot.com/2015/08/allocation-adventures-3-buddy-allocator.html

- http://gee.cs.oswego.edu/dl/html/malloc.html
- http://www.usenix.org/publications/library/proceedings/bos94/full_papers/bonwick.ps