

OS 2019

MIT ;)

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

Struktura prednasky

- 1. uvod
- 2. predmet
- 3. uvod do os
- 4. systemove volania UNIX

1. cast: UVOD

Ciele predmetu

- Detailnejšie pochopenie OS
- Ako:
 - Navrh
 - Implementacia
- malinkatY OS ;)

Naco je dobry OS

- Aplikacie (izolacia ↔ zdielanie)
- Sluzby (hw → app, spravodlivost)
- Hardver (vykon)

Co chcú aplikacie od OS

- Prístup k hardveru, čo vyzaduje:
 - Abstrakciu hardveru (problem ovladacov)
 - Multiplexovanie hw medzi aplikaciami
- Izolovanie poskodenyh aplikacii od zvyšku OS
- Komunikáciu medzi aplikaciami

Ake služby od OS očakávame

- Procesy
- Pamat
- Prístup ku suborom
- Adresarova struktura
- Bezpečnosť
- Siet
- Viaceri používatelia
- Komunikácia medzi aplikáciami (IPC)
- ...

Co znamena abstrakcia hw?

- Prístup ku hw spravovany jadrom na zaklade ziadosti aplikacie
- Jednotne rozhranie k roznyh zariadeniam toho isteho typu (zbernica)
- Programator ovladaca sa moze sustredit iba na jednu cast OS

Ako vyzerá abstrakcia OS?

- Aplikácie využívajú služby jadra cez tzv. systemové volania (syscalls)
- Příklad z Unix systemov:

```
fd = open("subor.txt",1);
```

```
write(fd, "ahoj svjete!\n", 13);
```

```
pid = fork();
```

Preco sa venovat takejto oblasti?

- Sklbenie vedomosti: AIPr, PT, OOP
- Cielom je vidiet realny program (jadro), ktory:
 - Musi byt extremne efektivny (rychlost), ale na druhej strane dostatočne abstraktny (prenositelny na ine platformy)
 - Musi byt vykonny (vela sluzieb a moznosti), ale pritom jednoduchy (zlozeny z jednoducho nahraditelnych blokov)

Preco sa venovat takejto oblasti?

- Naucit sa mysliet, spravne navrhovat a implementovat algoritmy, ktore medzi sebou spolupracuju
- Vyuzit vsetky doteraz ziskane vedomosti a sklbit ich
- OS je najkomplexnejši program vobec

Linux 4.8-rc7 19.9.2016

```
y@ellYah:/usr/src/linux-4.8-rc7$ cloc .  
55460 text files.  
54971 unique files.  
9923 files ignored.
```

```
http://cloc.sourceforge.net v 1.56 T=192.0 s (235.7 files/s, 106196.6 lines/s)
```

Language	files	blank	comment	code
C	23425	2258325	2074816	11503172
C/C++ Header	17771	433230	741250	2835111
Assembly	1433	48266	63102	294019
make	2217	8254	7915	34510
Perl	49	5183	3760	27188
Bourne Shell	201	2199	3513	11559
Python	44	1579	1917	9262
yacc	8	655	355	4327
HTML	3	512	0	4289
lex	8	299	289	1894
ASP.Net	19	133	0	1615
Bourne Again Shell	46	355	260	1585
C++	1	231	58	1581
awk	10	132	131	1138
NAnt scripts	2	128	0	475
Pascal	3	49	0	231
Lisp	1	63	0	218
Objective C++	1	55	0	189
m4	1	15	1	95
XSLT	6	13	27	71
vim script	1	3	12	27
CSS	1	12	22	25
sed	2	0	25	16
Teamcenter def	1	0	2	6
SUM:	45254	2759691	2897455	14732603

Linux 4.13.1 10.9.2017

```
y@ellYah:/usr/src/linux-4.13.1$ cloc .
```

```
60545 text files.
```

```
60002 unique files.
```

```
11247 files ignored.
```

```
http://cloc.sourceforge.net v 1.56 T=211.0 s (232.3 files/s, 108178.4 lines/s)
```

Language	files	blank	comment	code
C	25241	2468666	2227885	12531579
C/C++ Header	19482	484063	907454	3641969
Assembly	1434	49263	64821	298076
make	2362	8582	8151	36990
Perl	50	5061	3707	26293
Bourne Shell	246	3091	4180	15853
Python	72	2099	2426	12018
HTML	3	565	0	4730
yacc	9	682	357	4530
lex	8	302	301	1906
C++	7	287	71	1838
ASP.Net	32	158	0	1808
Bourne Again Shell	47	384	312	1713
awk	12	185	170	1510
NAnt scripts	2	158	0	588
Pascal	3	49	0	231
Objective C++	1	55	0	189
m4	1	15	1	95
XSLT	5	13	26	61
CSS	1	14	23	35
vim script	1	3	12	27
Teamcenter def	1	0	2	6
sed	1	2	5	5
SUM:	49021	3023697	3219904	16582050

Linux 4.19-rc3 10.9.2018

```
y@ellYah:/mnt/data1/skola/os/2018/___prednasky/01/linux-4.19-rc3$ cloc .  
61684 text files.  
61262 unique files.  
12218 files ignored.
```

```
http://cloc.sourceforge.net v 1.56 T=225.0 s (219.2 files/s, 104909.5 lines/s)
```

```
-----  
Language                files          blank          comment          code  
-----  
C                        26078         2586063        2268480        13128894  
C/C++ Header            18861         492135         903746         3659127  
Assembly                1323          47331          61174          278057  
make                    2386          8738           9443           37957  
Bourne Shell            376           6571           5894           28200  
Perl                    55            5426           4004           27407  
Python                  102           2775           2975           15979  
HTML                    5             670            0              5497  
yacc                    9             701            375            4648  
lex                     8             326            315            2006  
ASP.Net                 38            191            0              1981  
C++                     7             286            77             1844  
Bourne Again Shell     51            352            318            1722  
awk                     11            170            155            1386  
NAnt scripts            2             155            0              588  
Teamcenter def         2             14             2              100  
m4                      1             15             1              95  
XSLT                    5             13             26             61  
CSS                     1             18             27             44  
vim script              1             3              12            27  
Ruby                    1             4              0              25  
sed                     1             2              5              5  
-----  
SUM:                    49324         3151959        3257029        17195650  
-----
```

Linux 5.3 16.9.2019

```
y@e11Yah: /mnt/data1/skola/os/2019/prednasky/01/linux-5.3$ cloc .
```

```
65218 text files.  
64765 unique files.  
13172 files ignored.
```

```
github.com/AlDanial/cloc v 1.70 T=227.84 s (228.6 files/s, 110043.8 lines/s)
```

Language	files	blank	comment	code
C	27407	2706938	2262021	13772088
C/C++ Header	19656	523220	932408	4108616
Assembly	1327	47356	101761	230264
JSON	246	1	0	154515
Bourne Shell	543	11429	8786	45784
make	2508	9291	10602	40883
Perl	56	5590	4069	27896
Python	108	4374	4082	23575
YAML	115	2076	659	11118
HTML	5	656	0	5446
yacc	9	698	361	4627
lex	8	332	306	2013
C++	8	300	82	1871
Bourne Again Shell	51	354	296	1748
awk	11	171	148	1387
NAnt script	2	145	0	545
Markdown	2	133	0	423
Windows Module Definition	2	15	0	109
m4	1	15	1	95
CSS	1	27	28	72
XSLT	5	13	26	61
vim script	1	3	12	27
Ruby	1	4	0	25
INI	1	1	0	6
sed	1	2	5	5
SUM:	52075	3313144	3325653	18433199

2016 → 2017

- +5000 nových suborov
- +220 000 riadkov komentarov
- +1 000 000 riadkov kodu v C
- +800 000 riadkov v hlavickovych suboroch C
- +4000 riadkov kodu v ASM !!!

2017 → 2018

- +1 500 nových suborov
- +37 125 riadkov komentarov
- +600 000 riadkov kodu v C
- +17 000 riadkov v hlavickovych suboroch C
- +20 000 riadkov kodu v ASM !!! (5x viac ako vlani)

2018 → 2019

- +3 500 nových suborov
- +68 000 riadkov komentarov
- +640 000 riadkov kodu v C
- +450 000 riadkov v hlavickovych suboroch C !!!
(vlani iba 17 000)
- -50 000 riadkov kodu v ASM

JOS

- Jednoduchucke, minimalisticke
- Po nastartovani funguje jednoduchy shell, pomocou ktoreho je mozne interagovat s jadrom
- Ciel v predoslych rokoch: doplnat funkcionalitu
 - Sprava pamate (strankovanie)
 - Procesy
 - Subory
 - Siet
 - ...

JOS

```
y@ellYah:/mnt/data1/src/xv6/xv6-public.2016-09-06/lab$ cloc .  
  47 text files.  
  47 unique files.  
  70 files ignored.
```

```
http://cloc.sourceforge.net v 1.56 T=0.5 s (68.0 files/s, 10430.0 lines/s)
```

Language	files	blank	comment	code
C	12	289	288	2292
C/C++ Header	15	179	218	782
Python	1	91	76	380
make	1	54	46	218
Assembly	2	37	1	143
Perl	2	16	30	63
Lisp	1	0	0	12
SUM:	34	666	659	3890

2. cast: PREDMET

Struktura predmetu

- Stranka: do budúceho týždňa
- Kurz podľa MIT:
<https://pdos.csail.mit.edu/6.828>
- Praca na predmete:
 - Prace počas semestra vo dvojiciach
 - Hodnotenie každý samostatne

Struktura predmetu

- Prednasky:
 - Zakladna teoria OS
 - Pohlad na implementaciju systemu xv6
- Cvicenia:
 - Systemove (UNIX) programovanie
 - Zakladne veci OS (napr. vlakna)
 - Doplnanie funkcionality do xv6

Cvicenia a praca doma

- Praca na doma (DU) pozostava z 2 casti:
 - Priprava na dalsiu prednasku podla pokynov
 - Vypracovanie uloh pre dany tyzden
- Ohodnocovanie cinnosti:
 - Nahodne (napr. pisomkami alebo osobne)
 - Nepracovanie na ulohach pocas cvika → NZ (t.j. kto nechce pracovat na OS, nech ani na cviko radsej nejde)

Pracovne prostredie

- Ubuntu
- Qemu
- xv6
- (vid pokyny na stránke cviceni)

- Virtualky v cloude, vitane vlastne stroje
- Cvika v C-802

Hodnotenie (1)

- Akýkoľvek identifikovaný pokus o podvod:
 - Disciplinárna komisia FEI STU
 - FX hodnotenie z predmetu
- Nutná (nie postacujúca) podmienka získania hodnotenia lepšieho než FX
 - Vyplnenie evaluácie
 - Kontrola vyplnenia evaluácie individualne na konci semestra

Hodnotenie (2)

- Bonusy a malusy podľa ľubovôle prednasajúceho
- 2x 30b písomný test počas semestra
 - 6. týždeň (vo štvrtok a piatok je voľno! Cvička nebude ani v stredu, nebudeme nahrádzať)
 - 12. týždeň
- 1x 40b skuska

Obsah hodnotenia

- Cvicenia:
 - Pripravenost na cvicenie
 - Pripravenost zdrojakov
 - Cvicenia su KONZULTACNE!
- Testy:
 - Vedomosti
- Prednasky:
 - Pripravenost na prednasku podla pokynov

3. cast: UVOD DO OS

Ocakavania od OS

- Podpora viacerych veci SUCASNE
- Zdielanie a prerozdelenie zdrojov hw (CPU pre procesy, pamat, disk, tlaciaren, mys, monitor, sietova karta...)
- Izolacia procesov (aby nemohol jeden nicit druhy len tak, z cirej zloby)
- Komunikacia procesov je vsak tiez potrebna

Poziadavky na OS

- 1) Multiplex
- 2) Izolovanie
- 3) Interakcia

Preco multiplex?

- Co keby sme nemali OS, ale kazda aplikacia by pomocou nejakej kniznice priamo pristupovala k hw, ktory potrebuje?
- Vid niektore vnorene (embedded) aplikacie

Preco multiplex?

- Ak by bola aplikacia jedina, OK
- Ak je ich viac, musela by byt kazda “slusna”, museli by spolupracovat (kooperativny multitasking)
- Aplikacie su casto plne chyb... a nie su vzdy slusne

Preco izolacia?

- Aby sa zachovala “spravodlivosť” pri pridelení hw, oddelíme aplikácie
- Prístup nie k hw, ale k abstrakcii hw cez službu OS (príklad: prístup na disk cez sysvolania `open()`, `read()`, `write()`, `close()`)
- Vyhody:
 - Abstraktnejší prístup aplikácie (nemusi používať čísla sektorov, ale mená súborov)
 - Nepríde ku chybe pri práci s diskom

Preco interakcia?

- Ak máme izolované beh aplikácie, ako by medzi sebou mohli priamo komunikovať?
- Nepriamo cez služby OS:
 - Suborové popisovacie
 - Mapovanie pamäte
 - Siet

Uzivateľský priestor

- Silná izolácia si vyžaduje presne definované API (prístupové body ku službám jadra)
- Aplikácia by nemala byť schopná prístupovať k interným dátovým štruktúram a inštrukciám jadra
- CPU poskytuje hw podporu takejto ochrany

User / Kernel space

- Moderne CPU poskytujú minimalne 2 režimy činnosti:
 - Kernel mod (všetky instrukcie CPU povolené), napríklad priamy prístup k zariadeniam
 - User mod (nie všetky instrukcie CPU povolené); ak aplikácia v tomto mode skúsi vykonať privilegovanú instrukciu, vyvolá sa výnimka a OS má možnosť “prehovoriť do duše” tejto apke. Zväčša tým, že zneutráli jej existenciu ;)

User / Kernel space

- Aplikacia beziaca v rezime “user mod” bezi v tzv. Priestore uzivatela (“user space”)
- Program beziaci v rezime CPU “kernel mod” sa vykonava v tzv. Priestore jadra (“kernel space”)
- V pripade OS sa takemuto programu vravi “jadro” - angl. “kernel”

•Co vsetko ma bezat v jadre?

- Vsetky sys volania? → monoliticke jadro
 - Vsetko potrebne v jadre
 - Rozne casti jadra mozu priamo komunikovat (napr. spolocny buffer pre virtualnu pamat a suborovy system)
 - Problem je prave to, ze vsetko je v jadre a vsetko moze/chce komunikovat so vsetkym → komplexne rozhrania, vnasanie chyb
 - V pripade chyby hrozi pad celeho jadra a tym systemu, vyzaduje sa restart pocitaca

•Co vsetko ma bezat v jadre?

- Minimalna funkcionalita - mikrokernel
 - Sluzby OS bezia v priestore uzivatela, vtedy sa taketo aplikacie volaju servery
 - Na vyuzivanie sluzieb serverov je definovane rozhranie, tzv. posielanie sprav serverom
 - Funkcionalita mikrojadra – spustanie aplikacii, posielanie sprav, sprístupnovanie hw

Priklady z praxe

- Linux
 - Monoliticke jadro
 - Niektore sluzby v user space (graficky subsystem)

- Xv6
 - Monoliticke jadro
 - Tak malo sluzieb, ze jadro je mensie ako niektore mikrojadra ;)

4. cast: *Systemmove volania*

Systemove volania UNIX

- Priklady systemovych volani ukazane v xv6

Systemove volania UNIX

- Priklady systemovych volani ukazane v xv6
- Xv6 ma tradicny navrh podla UNIX

Systemove volania UNIX

- Príklady systemovych volani ukazane v xv6
- Xv6 ma tradicny navrh podla UNIX, ale uplne jednoduchy; nutne prestudovat a porozumiet knihe o xv6 (ako a preco)

Systemove volania UNIX

- Priklady systemovych volani ukazane v xv6
- Xv6 ma tradicny navrh podla UNIX, ale uplne jednoduchy; nutne prestudovat a porozumiet knihe o xv6 (ako a preco)
- Preco UNIX?

Systemove volania UNIX

- Priklady systemovych volani ukazane v xv6
- Xv6 ma tradicny navrh podla UNIX, ale uplne jednoduchy; nutne prestudovat a porozumiet knihe o xv6 (ako a preco)
- Preco UNIX?
 - Open source
 - Dobra dokumentacia
 - Cisty navrh
 - Siroke nasadenie

Systemové volania UNIX

- Xv6 bezi na architekture RISC
- Preto ho treba spustat pomocou virtualizacie nastrojom Qemu

SV read/write

- Prvy priklad systemovych volani 'copy.c'
- Kopiruje vstup na vystup ;)

SV read/write

- Prvy priklad systemovych volani 'copy.c'
- Kopiruje vstup na vystup ;)
- Napisany v jazyku C

SV read/write

- Prvy priklad systemovych volani 'copy.c'
- Kopiruje vstup na vystup ;)
- Napisany v jazyku C
- Vyuziva 2 systemove volania: read a write

SV read/write

- Prvy argument je FD (file deskriptor)

SV read/write

- Prvy argument je FD (file deskriptor)
 - Informuje jadro, o ktory subor sa jedna
 - Subor, na ktory odkazuje, musi byt otvoreny!!!

SV read/write

- Prvy argument je FD (file deskriptor)
 - Informuje jadro, o ktory subor sa jedna
 - Subor, na ktory odkazuje, musi byt otvoreny!!!
 - V systeme UNIX je “vsetko” subor (subor, znakove zariadenie, blokove zariadenie, soket...)

SV read/write

- Prvy argument je FD (file deskriptor)
 - Informuje jadro, o ktory subor sa jedna
 - Subor, na ktory odkazuje, musi byt otvoreny!!!
 - V systeme UNIX je “vsetko” subor (subor, znakove zariadenie, blokove zariadenie, soket...)
 - Proces (co to je?) moze mat otvorených vela suborov, pouzivat vela deskriptorov

SV read/write

- Prvy argument je FD (file deskriptor (?))
 - Informuje jadro, o ktory subor sa jedna
 - Subor, na ktory odkazuje, musi byt otvoreny!!!
 - V systeme UNIX je “vsetko” subor (subor, znakove zariadenie, blokove zariadenie, soket...)
 - Proces (?) moze mat otvorených vela suborov, pouzivat vela deskriptorov
- !!! **FD 0** “standard input”, **FD 1** “standard output”, **FD 2** “standard error output” !!!

SV read/write

- Druhý argument je pamätové miesto

SV read/write

- Druhý argument je pamätové miesto
 - Read(): kam sa uložia údaje načítané ZO suboru

SV read/write

- Druhy argument je pamatove miesto
 - Read(): kam sa uložia údaje načítane ZO suboru
 - Write(): skadiaľ sa zoberu údaje pri zápise DO suboru

SV read/write

- Druhý argument je pamätové miesto
 - Read(): kam sa uložia údaje načítané ZO suboru
 - Write(): skadiaľ sa zoberu údaje pri zápise DO suboru
 -
- Tretí argument je veľkosť (počet bajtov!!!)
 - Pri čítaní (read) možno načítať menej, ale nie viac!
 - Podobne pri zápise (write)

SV read/write

- Navratova hodnota systemoveho volania
read/write: pocet nacistanych/zapisanych bajtov

SV read/write

- Navratova hodnota systemoveho volania read/write: pocet nacistanych/zapisanych bajtov
- Read() a write() neriesia format udajov; UNIX I/O su v binarnej forme retazce bajtov (1 B 8 b)
- Interpretacia prudu bajtov ostava na aplikacii

SV open

- Skadial sa beru suborove popisovace (fd)?

SV open

- Příklad: open.c, vytvorenie suboru

SV open

- Příklad: open.c, vytvorenie suboru
 - ./open
 - ./cat output.txt

SV open

- Příklad: open.c, vytvorenie suboru
 - ./open
 - ./cat output.txt
- FD je male cele cislo, startuje sa od 0

SV open

- Příklad: open.c, vytvoření souboru
 - ./open
 - ./cat output.txt
- FD je malé celé číslo, startuje se od 0
- FD je indexem do tabulky procesu (?) (o tuto tabulku se stará jádro OS)

SV open

- Příklad: open.c, vytvoření souboru
 - ./open
 - ./cat output.txt
- FD je malé celé číslo, startuje se od 0
- FD je indexem do tabulky procesu (?) (o této tabulce se stará jádro OS)
- Každý proces má vlastní tabulku! (např. fd 3 ukazuje na různé soubory v různých procesech)

SV open

- Pozor na kontrolu chyb!!!
- Příkladiky na prednaske tuto kontrolu neobsahuju (pre jednoduchosť a názornosť jadra funkcionality)

SV open

- Pozor na kontrolu chyb!!!
- Příkladiky na prednaske tuto kontrolu neobsahuju (pre jednoduchost a nazornost jadra funkcionality)
- User/kernel diagram pre open
- Man 2 open
- Příkladiky open.c, cat.c

SV open

- Co vsetko sa udeje pri vyvolani systemoveho volania (v nasom pripade open)?

SV open

- Co vsetko sa udeje pri vyvolani systemoveho volania (v nasom pripade open)?
- Z pohladu uzivatela ide o akoby funkciu, ale v podstate sa jedna o specialnu instrukciu procesora s nejakymi parametrami

SV open

- Hw uchova registre procesora

SV open

- Hw uchova registre procesora
- Hw zvysi uroven opraveni procesora

SV open

- Hw uchova registre procesora
- Hw zvysi uroven opraveni procesora
- Hw zabezpeci vyvolanie vstupneho bodu do jadra OS

SV open

- Hw uchova registre procesora
- Hw zvysi uroven opraveni procesora
- Hw zabezpeci vyvolanie vstupneho bodu do jadra OS
- Vyvola sa funkcia na spracovanie open() (moze to chvilku trvat – praca s diskom, aktualizacia dat jadra ako tabulka FD, vyrovnavacia pamat)

SV open

- Hw uchova registre procesora
- Hw zvysi uroven opraveni procesora
- Hw zabezpeci vyvolanie vstupneho bodu do jadra OS
- Vyvola sa funkcia na spracovanie open() (moze to chvilku trvat – praca s diskom, aktualizacia dat jadra ako tabulka FD, vyrovnavacia pamat)
- Obnovia sa registre procesora pre proces

SV open

- Hw uchova registre procesora
- Hw zvysi uroven opraveni procesora
- Hw zabezpeci vyvolanie vstupneho bodu do jadra OS
- Vyvola sa funkcia na spracovanie open() (moze to chvilku trvat – praca s diskom, aktualizacia dat jadra ako tabulka FD, vyrovnavacia pamat)
- Obnovia sa registre procesora pre proces
- Znizi sa uroven opraveni procesora

SV open

- Hw uchova registre procesora
- Hw zvysi uroven opraveni procesora
- Hw zabezpeci vyvolanie vstupneho bodu do jadra OS
- Vyvola sa funkcia na spracovanie open() (moze to chvilku trvat – praca s diskom, aktualizacia dat jadra ako tabulka FD, vyrovnavacia pamat)
- Obnovia sa registre procesora pre proces
- Znizi sa uroven opraveni procesora
- Obnovi sa bez procesu v uzivatelskom rezime

SV fork

- Zrod noveho procesu

SV fork

- Zrod noveho procesu
- Co je to shell?

SV fork

- Zrod noveho procesu
- Co je to shell?
- Program (utilitka) prikazoveho riadku urcena na komunikaciu s OS
- Velmi vela systemovych volani je dosiahnutelna (vykonatelna) pomocou tohto programu
- Jedna sa o povodne rozhranie s OS pre system UNIX

SV fork

- Pre kazdy prikaz napisany v programe shell sa vytvori novy proces (napr. echo ahoj)

SV fork

- Pre kazdy prikaz napisany v programe shell sa vytvori novy proces (napr. echo ahoj)
- Ako? Pomocou systemoveho volania “vidlicka”

SV fork

- Pre kazdy prikaz napisany v programe shell sa vytvori novy proces (napr. echo ahoj)
- Ako? Pomocou systemoveho volania “vidlicka”
- Preto vidlicka?

SV fork

- Pre kazdy prikaz napisany v programe shell sa vytvori novy proces (napr. echo ahoj)
- Ako? Pomocou systemoveho volania “vidlicka”
- Preto vidlicka? Lebo sa proces rozdvoji!
Skopiruju sa instrukcie, udaje, registre, tabulka FD, ostatne interne udaje o procese...

SV fork

- Pre kazdy prikaz napisany v programe shell sa vytvori novy proces (napr. echo ahoj)
- Ako? Pomocou systemoveho volania “vidlicka”
- Preto vidlicka? Lebo sa proces rozdvoji!
Skopiruju sa instrukcie, udaje, registre, tabulka FD, ostatne interne udaje o procese...
- Bude jestvovat iba jedina odlisnost, a to PID; novo vytvoreny proces dostane nový identifikator v OS

SV fork

- Ako sa potom odlisia rodic a dieta, ako vobec zacne potomok “zit”? Ako sa “narodi”?

SV fork

- Ako sa potom odlisia rodic a dieta, ako vobec zacne potomok “zit”? Ako sa “narodi”?
- Zvlastnostou systemoveho volania fork je to, ze jeden krat je vyvolane (rodic zavola fork()), ale dva krat sa vykona navrat z jadra OS!!!!!!!!!!!!!!!

SV fork

- Ako sa potom odlisia rodic a dieta, ako vobec zacne potomok “zit”? Ako sa “narodi”?
- Zvlastnostou systemoveho volania fork je to, ze jeden krat je vyvolane (rodic zavola fork()), ale dva krat sa vykona navrat z jadra OS!!!!!!!!!!!!!!!!!!!!
- Raz pre rodica, druhy raz pre potomka

SV fork

- Ako sa potom odlisia rodic a dieta, ako vobec zacne potomok “zit”? Ako sa “narodi”?
- Zvlastnostou systemoveho volania fork je to, ze jeden krat je vyvolane (rodic zavola fork()), ale dva krat sa vykona navrat z jadra OS!!!!!!!!!!!!!!!!!!!!
- Raz pre rodica, druhy raz pre potomka
- Ako ich odlisime, ked vykonavaju rovnaky kod (kedze sa potomkovi skopiroval kod rodica?)

SV fork

- Odlisuje ich navratova hodnota volania fork

SV fork

- Odlisuje ich navratova hodnota volania fork
- <0 znamena chybu
- >0 znamena rodica
- $=0$ znamena potomka

SV fork

- Odlisuje ich navratova hodnota volania fork
- <0 znamena chybu
- >0 znamena rodica
- $=0$ znamena potomka

- If (pid == 0) → dieta
- If (pid > 0) → rodic

SV fork

- Fork.c

SV fork

- Fork.c
- Pomocou fork() vieme zduplicovat proces programu shell... ale ako spustime nejaky program?

SV exec

- Nahradenie aktualneho procesu inym zo suboru na disku

SV exec

- Ako shell spusti prikaz 'echo ahoj'?

SV exec

- Ako shell spusti prikaz 'echo ahoj'?
- Program echo je ulozeny niekde na disku (instrukcie programu spolu s inicializovanymi udajmi od prekladaca a linkera)

SV exec

- Volanie `exec()` nahradi v pamati (RAM) aktualny proces instrukciami noveho procesu; tieto instrukcie sa nacistaju z disku; ako sa to deje?

SV exec

- Volanie `exec()` nahradi v pamati (RAM) aktualny proces instrukciami noveho procesu; tieto instrukcie sa nacistaju z disku; ako sa to deje?
 - Zrusia sa instrukcie a data aktualneho procesu, ktory vyvolal `exec()`
 - Nahraju sa do pamate RAM instrukcie a data novo spustaneho procesu z disku
 - Zachovaju sa niektore interne udaje patriace k procesu, ktory vyvolal `exec()` (napr. Tabulka FD)

SV exec

- `exec(program_na_disku, argumenty_programu)`

SV exec

- `exec(program_na_disku, argumenty_programu)`
- `argumenty_programu` sa odovzdaju funkcii `main()`
- Vid `echo.c` ako sa manipuluje s argumentami

SV exec

- Príklad forkexec
 - Fork() vytvorí kópiu rodiča
 - Exec() sa spustí v potomkovi
 - Wait() v rodičovi čaká na ukončenie potomka

SV exec

- Príklad forkexec
 - Fork() vytvorí kópiu rodiča
 - Exec() sa spustí v potomkovi
 - Wait() v rodičovi čaká na ukončenie potomka
- Shell vykonáva sekvenciu fork-exec-wait pre každý zadávaný príkaz (wait sa môže vynechať, ak chceme spustiť príkaz 'na pozadí', t.j. asynchrónne)

Priklad redirect

- Presmerovanie vystupu programu do suboru na disku (miesto vypisania na monitore)

Priklad redirect

- Presmerovanie vystupu programu do suboru na disku (miesto vypisania na monitore)
- Co sa deje pri 'echo ahoj > output.txt'?

Priklad redirect

- Presmerovanie vystupu programu do suboru na disku (miesto vypisania na monitore)
- Co sa deje pri 'echo ahoj > output.txt'?
 - Fork v rodicovi (shell); wait v rodicovi
 - Zmena FD 1, exec 'echo' v potomkovi

Priklad redirect

- Opakovanie: fd 0, fd 1, fd 2 !!!!!

Priklad redirect

- Opakovanie: fd 0, fd 1, fd 2 !!!!!!!
- Poucenie: SV open() pouzije vzdy NAJMENSI volny index do tabulky FD !!!!!!!

Priklad redirect

- Opakovanie: fd 0, fd 1, fd 2 !!!!!!
- Poucenie: SV open() pouzije vzdy NAJMENSI volny index do tabulky FD !!!!!!!
- Dosledok 1:
 - Ak chceme nahradit fd 1 nasim suborom output.txt, musime vykonat operacie v nasledovnom poradi
 - close(1)
 - open('output.txt')

Priklad redirect

- Dosledok 2 – program echo nic ‘netusi’ o presmerovani!!!!!! vsetko sa deje na urovni programu shell
- Echo pracuje s deskriptormi 0 a 1:
 - Cita vzdy z fd 0
 - Zapisuje vzdy do fd 1
- Vďaka zachovaniu tabulky FD pri SV exec() vieme vymeniť to, kam ‘ukazuje’ FD 1; echo stále bude používať na výstup FD 1, ale v skutočnosti pôjde výstup do suboru

Komunikacia medzi procesmi

Komunikacia medzi procesmi

- Pomocou rury

Komunikacia medzi procesmi

- Pomocou rury: ma 2 konce; jeden na citanie, druhy na zapis

Komunikacia medzi procesmi

- Pomocou rury: ma 2 konce; jeden na citanie, druhy na zapis
- Ako je rura 'robena' v programe shell?
 - `$ ls | grep x`

Komunikacia medzi procesmi

- Pomocou systemoveho volania `pipe()`

Komunikacia medzi procesmi

- Pomocou systemoveho volania `pipe()`
- Vytvori sa par FD (pri `open` iba jeden FD, pri `pipe` dva FD) !!!!!
 - Cita sa z prveho
 - Zapisuje sa do druheho

Komunikacia medzi procesmi

- Pomocou systemoveho volania pipe()
- Vytvori sa par FD (pri open iba jeden FD, pri pipe dva FD) !!!!!
 - Cita sa z prveho
 - Zapisuje sa do druheho
- Priklad pipe1.c

Komunikacia medzi procesmi

- Ak chcem použiť ruru na komunikáciu medzi procesmi (na čo ine by sme ju použili, že?), musíme správne skombinovať `SV fork()` a `pipe()`

Komunikacia medzi procesmi

- Ak chcem pouzít ruru na komunikáciu medzi procesmi (na čo ine by sme ju použili, že?), musíme správne skombinovať SV `fork()` a `pipe()`
- Příklad 'ls | grep x':
 - Shell vytvorí ruru
 - Shell urobí fork (2x!!!!) (jeden pre ls, druhý pre grep)
 - Nahradí fd 1 vo forku pre ls za zapisovací koniec rury
 - Nahradí fd 0 vo forku pre grep za citací koniec rury
 - Vykoná exec pre ls a grep

Komunikacia medzi procesmi

- Příklad pipe2.c

Citanie obsahu priecinku

Citanie obsahu priecinku

- Ako Is ziska zoznam suborov (poloziek) adresara?
- Adresar je iba specialny typ suboru; vieme ho otvorit a citat
- Vid priklad ls.c

- Do budúcej prednasky PRECITAT kapitolu 1 z knizky:

<https://pdos.csail.mit.edu/6.828/2019/xv6/book-riscv-rev0.pdf>

MIT ;)