Otázky nesjou zdaleka všechny, jsou vytažené ze zveřejněných ukázkových otázek!!!

Mějme program:
static int pole[100]; main() { int pole1[200]; }
Tento program bude mít po zkompilování sekci DATA velkou přibližně (nepočítám to, co přidá standardní knihovna)

- 100 bajtů
- 100*sizeof(int) bajtů
- *0 bajtů
- 200*sizeof(int) bajtů
- 200 bajtů

Úspěšné volání lseek(fd, -10, SEEK_CUR) nastaví ukazovátko pozice v souboru

- deset bajtů před konec souboru
- na začátek souboru
- *o deset bajtů zpět
- do původní pozice (nezmění pozici)

Kontext v systému je

- okolí, ve kterém se proces nachází
- text programu, který je umístěn v jádře systému
- popisovač sdíleného modulu, získaný pomocí dlopen.
- *stav procesoru, příslušný běhu jednoho procesu nebo vlákna

Převody řetězců mezi různými znakovými sadami lze programově realizovat knihovní funkcí

- *iconv()
- strxfrm()
- setlocale()
- nl_langinfo()
- to_charset()

Má-li systém monolitické jádro, odpovídá ovladač klávesnice přibližně které části programu

- vláknu
- procesu
- hlavičkovému souboru
- *objektovému modulu

Hodnoty CHILD_MAX, LINK_MAX a další patří mezi

- systémové limity procesu
- *POSIX.1 compile-time limity
- run-time limity
- globální proměnné

Změnit čas poslední modifikace souboru je možno službou jádra

- getrusage()
- stat()
- link()
- time()
- *truncate()

Po chybě služby jádra je hodnota errno platná

- do přečtení této proměnné
- *do příští chyby služby jádra v tomtéž procesu/vlákně
- trvale -- chceme-li zjišťovat další chybu služby jádra, je třeba před voláním nastavit errno na 0
- do příští chyby služby jádra v celém systému

Knihovní funkce system() využívá těchto služeb jádra:
- jen fork() a exec()
- sysconf() a pathconf()
- sysctl() a setuid()
- *fork(), exec() a wait()

K zadanému deskriptoru nelze prostředky POSIXu zjistit
- typ souboru (běžný, roura, speciální, atd.)
- *jméno souboru
- aktuální pozici v souboru
- velikost souboru
- preferovanou velikost bloku pro I/O operace

Proces má následující UID:
- reálné, efektivní a seznam doplňkových
- reálné, uložené a seznam doplňkových
- *reálné, efektivní a uložené
- reálné, efektivní, uložené a seznam doplňkových

Konstruktory statických objektů v C++ se volají
- po návratu z funkce __main()
- v průběhu statického linkování programu
- *před vstupem do funkce main()
- po zpracování argumentů příkazové řádky například pomocí getopt()

Vytvořit novou verzi souboru s tím, že po případné havárii systému je garantováno, že soubor má buďto celý původní obsah nebo celý nový obsah lze sekvencí volání
- *open(), write(), fsync() a rename()
- open() s parametrem O_EXCL, write() a rename()
- open() s parametrem O_PONIES, write() a rename()
- open(), write() a rename() s parametrem FS_COMMIT
- open(), write() a rename() (žádné další potvrzování zde není nutné)

Alokátor paměti ve standardní knihovně (malloc() a další) získává paměť
- ze zásobníku procesu
- *od jádra po blocích velikosti násobku velikosti stránky
- z oblasti vytvořené staticky při startu procesu
- od jádra po blocích které specifikuje volající
- od jiných procesů, jejichž paměť je třeba nejprve odswapovat

Který ze systémových limitů definovaných normou není obvykle v systému implementován z důvodu přílišné režie a nejistého výsledku aplikace tohoto limitu?
- limit na počet procesů daného uživatele
- limit na velikost zásobníku
- limit na velikost virtuální paměti
- limit na strojový čas (problémem je zde více procesorů)
- *limit na resident set size

Které z následujících tvrzení je pravdivé?
- vedoucí proces skupiny je vždy zároveň vedoucím procesem session
- v rámci skupiny procesů existuje jedna nebo více sessions
- *v rámci session existuje jedna nebo více skupin procesů
- jedna session reprezentuje procesy na popředí na jednom terminálu
- proces může být obecně ve více než jedné session

Co nemají společného vlákna jednoho procesu?
- *zásobník
- text programu, nad kterým běží
- přístup k mutexům procesu
- virtuální paměť
- tabulku otevřených souborů

Pokud nepojmenovanou rouru nemá nikdo otevřenou pro čtení, pak
- proces při otevření roury pro zápis dostane signál SIGPIPE.
- proces při pokusu o zápis do roury dostane signál SIGPIPE.
- zápis do roury projde, ale pouze do velikosti bufferu roury.
- zápis do roury skončí s chybou ESPIPE.
- *se proces při otevření roury pro zápis zablokuje

Úspěšné volání open("file", O _WRONLY | O_CREAT | O_EXCL, 0640) vytvoří vždy soubor s právy
- r-x------
- rwxr-xr-x
- *žádná z ostatních možností obecně neplatí
- rw-r--r--
- rw-r-----

Použitím proměnné LD_PRELOAD lze předefinovat funkci původně definovanou v knihovně jen v případě, že
- se jedná o funkci s verzovanými symboly
- se jedná o knihovnu linkovanou v době kompilace (např. Linux a.out)
- se jedná o statickou knihovnu
- *se jedná o knihovnu linkovanou v době běhu (např. Linux ELF)

Nepodmíněné ukončení procesu je význam signálu
- SIGTERM
- SIGQUIT
- *SIGKILL
- SIGSTOP
- SIGHUP

Chceme-li kopírovat data z jednoho deskriptoru do druhého a naopak (bez zablokování přenosu jedním směrem čekáním na připravenost k přenosu druhým směrem), je nejlépe
- použít samostatný proces pro každý směr
- *službu select() nebo poll()
- neblokující čtení/zápis O_NDELAY
- samostatné vlákno pro každý směr

Jaká přístupová práva má proces běžící pod UID C a GID D k souboru vlastněnému uživatelem A a skupinou B s následujícím ACL: u::rwx,g::rw-,o:--x, u:C:rw-,g:D:-wx,m::r-x?
- čtení a zápis
- *jen čtení
- jen provádění
- čtení a provádění
- čtení, zápis, provádění

Proces se může prohlásit za vedoucí proces skupiny voláním služby jádra
- leader()
- initgroups()
- *setpgrp()
- setegid()
- setgid()

Při zablokování procesu se jádro se při výběru procesu kterému bude dále přidělen procesor řídí atributem

- velikost obsazené paměti procesu
- prioritou daného uživatele
- předpokládanou délkou běhu procesu
- *dispečerská priorita procesu
- uživatelská priorita procesu

Smazat soubor z adresáře smí vlastník souboru

- vždy
- jen má-li adresář nastavený sticky bit
- jen je-li zároveň vlastníkem adresáře
- jen má-li adresář nastavený set-gid bit
- *jen má-li i právo zápisu do adresáře

Volání rename(old, new) v případě, že soubory old a new neleží na stejném svazku

- funguje stejně, jako kdyby tyto soubory ležely na stejném svazku
- *vrátí chybu
- zaručuje atomický přesun dat, ne však metadat
- nezaručuje atomické přejmenování

Adresa, kterou používají instrukce procesoru pro čtení nebo zápis do paměti, se nazývá

- *virtuální adresa
- sběrnicová adresa
- fyzická adresa
- IP adresa

Advisory locking znamená, že

- *existence zámku nebrání operaci čtení nebo zápisu
- soubor může zamknout jen jeho vlastník
- lze zamykat vždy jen celý soubor
- dva procesy nemohou vytvořit zámek nad stejným souborem

Spinlocky se v jádře používají pro

- dlouhodobé vzájemné vyloučení procesů
- předávání paměťových bloků mezi procesy
- urychlení práce obsluhy přerušení
- *krátkodobé vzájemné vyloučení procesů

Při volání open("file", O_WRONLY|O_CREAT, 0666) se

- vrátí chyba, pokud soubor file neexistuje.
- vrátí chyba, pokud soubor file má již někdo otevřený pro zápis.
- vrátí chyba, pokud soubor file již existuje.
- *nepřepíše původní obsah souboru file.
- soubor file zarovná na nulovou délku.

Chce-li proces zablokovat doručování některých signálů během vykonávání ovladače určitého signálu, může to specifikovat pomocí služby

- *sigaction()
- signal()
- sigsetmask()
- sigsuspend()
- sigpending()

Velikost bufferu roury lze zjistit pomocí služby jádra
- pipe()
- sysconf()
- mknod()
- *pathconf()

Proměnné prostředí jsou uloženy
- *pro každý proces zvlášť v uživatelském prostoru procesu
- globálně pro celý systém v uživatelském prostoru procesu
- pro každého uživatele zvlášť v adresním prostoru jádra
- globálně pro celý systém v adresním prostoru jádra

Pokud proces právě ukončil vykonávání služby jádra, přejde do stavu
- přerušitelně čekající
- *běžící v uživatelském prostoru
- běžící v režimu jádra
- zombie
- nepřerušitelně čekající

Návratová hodnota funkce longjmp(jmp_buf env, int retval)
- je rovna proměnné env.
- *není; tato funkce nevrací žádnou hodnotu.
- je vždy 0.
- je 0 při prvním volání, retval při dalším.

Pokud chceme, aby se po pádu systému neobjevila v souborech data z jiných (dříve smazaných) souborů, je třeba
- žurnálovat změny v metadatech
- zapisovat metadata dříve než data
- volat fdatasync() po každém zápisu
- *zapisovat data dříve než metadata

Po vykonání služby fork() zdědí potomek od rodičovského procesu
- čekající signály
- číslo procesu
- *hodnotu ukazatele vrcholu zásobníku
- deskriptory kromě těch s flagem FD_CLOEXEC
- zámky na souborech

Mějme program:
```
static int pole[100];
main() { int pole1[200]; }
```
Tento program bude mít po zkompilování sekci BSS velkou přibližně
- 0 bajtů
- 200 bajtů
- 200*sizeof(int) bajtů
- 100 bajtů
- *100*sizeof(int) bajtů

Seznam sekcí souboru ve formátu ELF lze zjistit příkazem
- ranlib
- nm
- ar
- *objdump
- strip

Chceme-li, aby funkce readdir() vracela i jména souborů začínající tečkou,
- je třeba tento požadavek specifikovat jako parametr volání readdir()
- je třeba tento požadavek specifikovat nastavením proměnné prostředí
- *není třeba dělat nic zvláštního
- je třeba použít jiné nastavení u opendir()

Pro efektivní naprogramování lokalizovaného třídění je třeba tříděné řetězce předzpracovat funkcí
- *strxfrm()
- strcmp()
- strcoll()
- setlocale()
- bzero()

V případě modulárního jádra je komunikace mezi ovladači zařízení a zbytkem jádra zajištěna mechanismem
- carrier pigeon
- nepojmenovaná roura
- zasílání zpráv
- *volání funkcí
- zasílání přerušení

Pod pojmem stránkování na žádost (demand paging) rozumíme
- *načítání např. textu procesu do paměti až v případě prvního přístupu
- řešení křížových odkazů ve sdílených knihovnách až při prvním použití
- načítání celého souboru crt1.o při startu procesu
- načítání celého textu procesu do paměti hned při startu procesu

Služba jádra stat() neumožňuje zjistit
- počet odkazů na soubor
- *seznam bloků, ve kterých je soubor uložen
- číslo i-uzlu daného souboru.
- zařízení, na kterém je soubor uložen

UNIXové systémy obvykle procesům poskytují paměťový model
- *lineárně adresovaná paměť
- společná paměť pro všechny procesy
- segmentovaná paměť
- distribuovaná paměť

Spotřebovaný čas procesu a jeho potomků lze zjistit službou jádra
- sysconf()
- *times()
- time()
- wait()
- getresources()
- getrlimit()

Při statickém linkování objektových modulů s knihovnou se do výsledného programu přidá
- celý obsah linkované knihovny
- jen odkazy na pevné (statické) adresy symbolů z knihovny
- jen jména použitých symbolů z knihovny
- *část obsahu linkované knihovny

Volání rename(old, new) v případě, že soubory old a new neleží na stejném svazku
- funguje stejně, jako kdyby tyto soubory ležely na stejném svazku
- zaručuje atomický přesun dat, ne však metadat
- *vrátí chybu
- nezaručuje atomické přejmenování

Volání lseek(fd, 0, SEEK_SET) nastaví ukazovátko pozice v souboru
- *na začátek souboru
- do původní pozice (nezmění pozici)
- na konec souboru

Logical Volume Manager (lvm) umožňuje
- zmenšení/zvětšení logical volume, aniž by bylo nutno informovat o změně nadřazený souborový systém.
- *přesun dat z jednoho physical volume na jiný za běhu.
- přesun logical volume na jinou volume group za běhu.
- přesun physical extentu na jiný physical volume za běhu.

Změnit skupinu souboru lze službou jádra
- setgid()
- chmod()
- *chown()
- setegid()
- chgrp()

Nově vytvořené vlákno po pthread_create() začíná vykonávat instrukce od adresy
- 0x00000000
- stejné jako volající vlákno; dále se rozhoduje podle návratové hodnoty pthread_create()
- 0xdeadbeef
- *specifikované explicitně při vytvoření vlákna
- specifikované předem při deklaraci struktury vlákna

Program ld zpracovává
- zdrojový text v jazyce C
- hlavičkové soubory
- *objektové soubory
- programy v assembleru
- výstup preprocesoru cpp

Po sekvenci volání
umask(022);
open("file", O_WRONLY|O_CREAT, 0774);
vznikne soubor s právy
- rwxr-x-{-{}-}
- rw-rw-r-{-}
- *rwxr-xr-{-}
- rwxrwxr-{-}
- rw-r-{-r-{}-}

Má-li proces reálné a efektivní UID rovno 1 a uložené UID rovno 2, pak po spuštění (exec*()) programu bez set-UID bitu vlastněného uživatelem 3 bude mít proces tyto hodnoty reálného, efektivního a uloženého UID (v tomto pořadí):

- 1, 1, 2
- 1, 3, 3
- 1, 2, 2
- 1, 3, 2
- *1, 1, 1
- 3, 1, 2

Několik posledních hlášení jádra Linuxu (například bootovací zprávy a podobně) lze vypsat příkazem

- *dmesg
- kmessages
- mesg
- boot
- reboot

Funkce realloc()

- může přemístit původní data jen při požadovaném zvětšení alokovaného prostoru.
- *může přemístit původní data při požadovaném zmenšení i zvětšení alokovaného prostoru.
- nikdy nepřemisťuje původní data -- nemá li na příslušném místě dostatek prostoru, vrátí chybu.
- může přemístit původní data jen při požadovaném zmenšení alokovaného prostoru.

Souborový systém FFS/UFS neobsahuje

- tabulku i-uzlů
- volné datové bloky
- bitmapu volných datových bloků
- *bitmapu volných i-uzlů

Vykonání vlastního kódu v případě, že je v programu zavolána služba jádra _exit(), lze zajsitit pomocí funkce

- call_my_code()
- abort()
- atexit()
- signal()
- *žádné z uvedených

Proces může zabránit zápisu potenciálně citlivých dat na odkládací prostor pomocí operace

- fsync()
- mprotect()
- fcntl()
- flock()
- *mlock()

Jaká přístupová práva má proces běžící pod UID C a GID D k souboru vlastněnému uživatelem A a skupinou B s následujícím ACL: u::rwx,g::rw-,o:--x, u:C:rw-,g:D:-wx,m::r-x?

- *jen čtení
- čtení a zápis
- čtení a provádění
- jen provádění
- čtení, zápis, provádění

Pokud nepojmenovanou rouru nemá nikdo otevřenou pro čtení, pak
zápis do roury skončí s chybou ESPIPE.
zápis do roury projde, ale pouze do velikosti bufferu roury.
*proces při pokusu o zápis do roury dostane signál SIGPIPE.
se proces při otevření roury pro zápis zablokuje
proces při otevření roury pro zápis dostane signál SIGPIPE.

Má-li i-uzel 32-bitové ukazatele na blok, 10 přímých odkazů, 1 blok nepřímých odkazů, 1 blok nepřímých odkazů druhé úrovně a 1 blok nepřímých odkazů třetí úrovně (jak bylo popsáno na přednášce), pak souborový systém se 4 KB bloky může být velký maximálně
4 GB
4 TB
*16 TB
16 GB

Další otazky – nepřesné znění:

- Které signály vyvolávají vytvoření souboru core?
  - např: SIGQUIT, SIGABRT, SIGEMT, SIGKILL, SIGSEGV, SIGFPE...
- Máme 64-bitový systém, 4kB blok na disku, struktura i-node je definována stejně jako na přednášce. Jaká je maximální velikost souboru?
  - cca 550 GB
- Jak získat text chyby služy jádra pro vyspání do do např. modálního okna grafické aplikace
  - strerror()
- Co dělá program as?
  - Vytváří objekový soubor ze zdrojáku v assembleru.
- Jak zjistit velikost jednotlivých sekcí a další informace o obj.souboru / binárce?
  - Buď programem objdump nebo size
- Jak dosáhneme v paralelních systémech co největší propustnosti při použití zámků
  - Možnosti: spinlock, RCU, software TLB, semafor (nevím co je správně ☺)
- Co dělá démon při startu?
- Kořenový adresář je nastaven pro? – proces/thread/systém