Systemy operacyjne

Lista zadań nr 5

Na zajęcia 12 i 20 listopada 2019

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- Tanenbaum (wydanie czwarte): 4.1, 4.2, 10.6
- APUE (wydanie trzecie): 3.12, 3.15, 4.14 - 4.18, 15.2

UWAGA! W trakcie prezentacji należy być gotowym do zdefiniowania pojęć oznaczonych **wytłuszczoną** czcionką.

Zadanie 1. Czym różnią się **ścieżka absolutna**, **relatywna** i **znormalizowana**? Względem którego katalogu obliczana jest ścieżka relatywna? Jakim wywołaniem systemowym zmienić ten katalog? Wyjaśnij czym są **punkty montażowe**, a następnie na podstawie `mount(8)` wyjaśnij znaczenie i zastosowanie następujących atrybutów punktów montażowych: «`noatime`», «`noexec`» i «`sync`».

Zadanie 2. Przywołując strukturę «`dirent`» i reprezentację katalogu z poprzednich ćwiczeń wyjaśnij krok po kroku jak działa `rename(2)`. Zauważ, że korzystając z «`rename`» można również przenieść **atomowo** plik do innego katalogu pod warunkiem, że ten znajduje się w obrębie tego samego systemu plików. Czemu «`rename`» zakończy się błędem «`EXDEV`» kiedy próbujemy przenieść plik do innego systemu plików?

Zadanie 3. Na podstawie slajdów do wykładu wyjaśnij różnice w sposobie implementacji **dowiązań twardych** (ang. *hard link*) i **symbolicznych** (ang. *symbolic link*). Jak za pomocą dowiązania symbolicznego stworzyć w systemie plików pętlę? Kiedy jądro systemu operacyjnego ją wykryje (błąd «`ELOOP`»)? Czemu pętli nie da się zrobić z użyciem dowiązania twardego? Skąd wynika liczba dowiązań do katalogów?

Zadanie 4 (P). Przeczytaj krytykę interfejsu plików przedstawioną w podrozdziale `ioctl and fcntl Are an Embarrassment`[1]. Do czego służy wywołanie systemowe `ioctl(2)`? Zauważ, że stosowane jest głównie do plików urządzeń znakowych lub blokowych. Na podstawie pliku `ioccom.h`[2] wyjaśnij znaczenie drugiego i trzeciego parametru wywołania `ioctl(2)`. Używając przeglądarki kodu[3] jądra NetBSD znajdź definicję identyfikatorów «`DIOCEJECT`», «`KIOCTYPE`» i «`SIOCGIFCONF`», a następnie krótko opisz co robią te polecenia.

Komentarz: Prowadzący przedmiot zgadza się z autorem krytyki. Czy i Ty widzisz brzydotę tego interfejsu?

Zadanie 5 (bonus). W bieżącej wersji biblioteki «`libcsapp`» znajduje się plik «`terminal.c`». Zapoznaj słuchaczy z działaniem procedury «`tty_curpos`» odczytującej pozycję kursora terminala. Do czego służy kod sterujący «`CPR`» opisany w Terminal output sequences[4]? Posiłkując się `ioctl_tty(2)` wytłumacz semantykę rozkazów «`TCGETS`» i «`TCSETSW`», wykorzystywanych odpowiednio przez `tcgetattr(2)` i `tcsetattr(2)`, oraz «`TIOCINQ`» i «`TIOCSTI`». Na podstawie `termios(4)` wyjaśnij jak flagi «`ECHO`», «`ICANON`», «`CREAD`» wpływają na działanie sterownika terminala.

Zadanie 6. Uruchamiamy w powłoce **potok** (ang. *pipeline*) «`ps -ef | grep zsh | wc -l > cnt`». Każde z poleceń używa wyłącznie standardowego wejścia i wyjścia. Dzięki `dup2(2)` i `pipe(2)` bez modyfikacji kodu źródłowego powyższych programów możemy połączyć je w potok i **przekierować** wyjście do pliku «`cnt`». Powłoka umieszcza wszystkie trzy procesy w nowej grupie procesów rozłącznej z grupą powłoki. Kiedy potok zakończy swe działanie, do powłoki zostanie przekazany kod wyjścia ostatniego polecenia w potoku. Uzasadnij kolejność tworzenia procesów potoku posługując się obrazem 9.10 z rozdziału „Shell Execution of Programs" (APUE). Następnie ustal, który z procesów powinien wołać `setpgrp(2)`, `creat(2)`, `dup2(2)`, `pipe(2)`, `close(2)` lub `waitpid(2)` i uzasadnij swój wybór.

[1] `http://www.catb.org/~esr/writings/taoup/html/ch20s03.html#id3016155`
[2] `https://grok.dragonflybsd.org/xref/netbsd/sys/sys/ioccom.h`
[3] `https://grok.dragonflybsd.org/xref/netbsd`
[4] `https://en.wikipedia.org/wiki/ANSI_escape_code#Terminal_output_sequences`

Ściągnij ze strony przedmiotu archiwum «so19_lista_5.tar.gz», następnie rozpakuj i zapoznaj się z dostarczonymi plikami.

UWAGA! Można modyfikować tylko te fragmenty programów, które zostały oznaczone w komentarzu napisem «TODO».

Zadanie 7 (P). (Pomysłodawcą zadania jest Tomasz Wierzbicki.)

Program «primes» używa Sita Eratostenesa[5] do obliczania liczb pierwszych z przedziału od 2 do 10000. Proces główny tworzy dwóch potomków wykonujących procedurę «generator» i «filter_chain», spiętych rurą «gen_pipe». Pierwszy podproces wpisuje do rury kolejne liczby z zadanego przedziału. Drugi podproces tworzy łańcuch procesów filtrów, z których każdy jest spięty rurą ze swoim poprzednikiem. Procesy w łańcuchu powstają w wyniku obliczania kolejnych liczb pierwszych. Każdy nowy filtr najpierw wczytuje liczbę pierwszą p od poprzednika, po czym drukuje ją, a następnie kopiuje kolejne liczby z poprzednika do następnika za wyjątkiem liczb podzielnych przez p.

Należy prawidłowo pochować dzieci i dbać o zamykanie nieużywanych końców rur. Program musi poprawnie działać dla argumentu 10000 – w tym przypadku powinno zostać utworzonych $1229 + 2$ podprocesów.

Zadanie 8 (P). Program «coro» wykonuje trzy współprogramy[6] połączone ze sobą w potok bez użycia rur. Pierwszy z nich czyta ze standardowego wejścia znaki, kompresuje białe znaki i zlicza słowa. Drugi usuwa wszystkie znaki niebędące literami. Trzeci zmienia wielkość liter i drukuje znaki na standardowe wyjście.

W wyniku wykonania procedury «coro_yield» współprogram przekazuje niezerową liczbę do następnego współprogramu, który otrzyma tę wartość w wyniku powrotu z «coro_yield». Efektywnie procedura ta implementuje **zmianę kontekstu**. Taką prymitywną formę **wielozadaniowości kooperacyjnej** (ang. *cooperative multitasking*) można zaprogramować za pomocą sigjmp(2) i longjmp(2). Przeczytaj ich odpowiedniki znajdujące się w pliku «libcsapp/Sigjmp.s» i wytłumacz co robią.

Uzupełnij procedurę «coro_add» tak, by po wznowieniu kontekstu przy pomocy «Longjmp» wykonała procedurę «fn», po czym zakończyła wykonanie współprogramu. Zaprogramuj procedurę «coro_switch» tak, by wybierała następny współprogram do uruchomienia i przełączała na niego kontekst. Jeśli współprogram przekazał wartość parametru «EOF», to należy go usunąć z listy aktywnych współprogramów.

Program używa listy dwukierunkowej «TAILQ» opisanej w queue(3). Zmienna «runqueue» przechowuje listę aktywnych współprogramów, «running» bieżąco wykonywany współprogram, a «dispatcher» kontekst programu, do którego należy wrócić, po zakończeniu wykonywania ostatniego aktywnego współprogramu.

[5] https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

[6] https://en.wikipedia.org/wiki/Coroutine