Systemy operacyjne

Lista zadań nr 6

Na zajęcia 19 i 27 listopada 2019

Jeśli nie omówiliście przy tablicy zadań 5–8 z listy 5, to powinniście to zrobić na zajęciach w bieżącym tygodniu. Przed rozpoczęciem zajęć, zadania należy zadeklarować na kuponach z poprzedniego tygodnia, o ile prowadzący nie ustali inaczej.

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- APUE (wydanie trzecie): 3.13, 4.2-4.5, 4.16-4.17. 4.21, 4.24-4.24

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

Zadanie 1 (P). Przeczytaj krytykę kluczowej idei systemu UNIX, tj. A Unix File Is Just a Big Bag of Bytes[1]. Na podstawie Resource Fork[2] wyjaśnij czym były dodatkowe zasoby pliku w historycznych systemach MACOS.

Jaką postać mają **rozszerzone atrybuty pliku** `xattr(7)`? Gdzie są one składowane w systemie plików? Poleceniem `wget(1)` z opcją «`--xattr`» pobierz z Internetu plik, po czym wyświetl jego rozszerzone atrybuty przy pomocy polecenia `getfattr(1)`. Następnie policz sumę `md5` wybranego pliku i przypisz ją do atrybutu «`user.md5sum`» poleceniem `setfattr(1)`, po czym sprawdź czy operacja się powiodła.

Ściągnij ze strony przedmiotu archiwum «`so19_lista_6.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 2 (2pkt, P). Program «`writeperf`» służy do testowania wydajności operacji zapisu do pliku. Nasz microbenchmark[3] wczytuje z linii poleceń opcje i argumenty opisane dalej. Na standardowe wyjście drukuje t trójkątów (opcja «`-t`») prostokątnych o boku złożonym z l znaków gwiazdki «`*`» (opcja «`-l`»). Jeśli standardowe wyjście zostało przekierowane do pliku oraz została podana opcja «`-s`», to przed zakończeniem programu bufory pliku zostaną zsynchronizowane z dyskiem wywołaniem `fsync(2)`.

Program realizuje pięć wariantów zapisu do pliku:

- Każdą linię trójkąta zapisuje osobno wywołaniem `write(2)` (argument «`write`»).
- Używa strumienia biblioteki stdio bez buforowania (argument «`fwrite`»), z **buforowaniem liniami** (argument «`fwrite-line`») i **buforowaniem pełnym** (argument «`fwrite-full`»).
- Wykorzystuje wywołanie systemowe `writev(2)` do zapisania do «`IOV_MAX`» linii na raz.

Twoim zadaniem jest odpowiednie skonfigurowanie bufora strumienia «`stdout`» z użyciem procedury `setvbuf(3)` oraz zaimplementowanie metody zapisu z użyciem «`writev`».

Przy pomocy skryptu powłoki «`writeperf.sh`» porównaj wydajność wymienionych wcześniej metod zapisu. Uzasadnij przedstawione wyniki. Miej na uwadze liczbę wywołań systemowych (należy to zbadać posługując się narzędziem `strace(1)` z opcją «`-c`») oraz liczbę kopii danych wykonanych celem przesłania zawartości linii do buforów dysku.

[1] `http://www.catb.org/~esr/writings/taoup/html/ch20s03.html#id3015538`
[2] `https://en.wikipedia.org/wiki/Resource_fork`
[3] `https://en.wikipedia.org/wiki/Microbenchmark`

Zadanie 3 (P). Program «id» drukuje na standardowe wyjście **tożsamość**, z którą został utworzony, np.:

```
$ id
uid=1000(cahir) gid=1000(cahir) groups=1000(cahir),20(dialout),24(cdrom),25(floppy),
27(sudo),29(audio),30(dip),44(video),46(plugdev),108(netdev),123(vboxusers),999(docker)
```

Uzupełnij procedurę «getid» tak by zwracała identyfikator użytkownika `getuid(2)`, identyfikator grupy `getgid(2)` oraz tablicę identyfikatorów i liczbę grup dodatkowych `getgroups(2)`. Nie możesz z góry założyć liczby grup, do których należy użytkownik. Dlatego należy stopniowo zwiększać rozmiar tablicy «gids» przy pomocy `realloc(3)`, aż pomieści rezultat wywołania «getgroups». Należy również uzupełnić ciało procedur «uidname» i «gidname» korzystając odpowiednio z `getpwuid(3)` i `getgrgid(3)`.

Zadanie 4 (P). Program «listdir» drukuje zawartość katalogu w formacie podobnym do wyjścia polecenia «ls -l». Poniżej można znaleźć przykładowy wydruk, na którym widnieją odpowiednio: plik zwykły, dowiązanie symboliczne, urządzenie znakowe, plik wykonywalny z bitem set-uid, jeden katalog z ustawionym bitem set-gid i drugi z bitem sticky.

```
-rw-r--r--   1  cahir  cahir   2964  Fri Nov 15 14:36:59 2019  listdir.c
lrwxrwxrwx   1  cahir  cahir     17  Mon Nov  4 11:14:49 2019  libcsapp -> ../csapp/libcsapp
crw--w----   1  cahir    tty  4,  2  Tue Nov 12 08:42:33 2019  tty2
-rwsr-xr-x   1   root   root  63736  Fri Jul 27 10:07:37 2018  passwd
drwxrwsr-x  10   root  staff   4096  Mon Jan  9 13:49:40 2017  local
drwxrwxrwt  23   root   root  12288  Fri Nov 15 16:01:16 2019  tmp
```

Uzupełnij kod programu według wskazówek zawartych w komentarzach w kodzie źródłowym. Należy użyć:

- `fstatat(2)` do przeczytania metadanych pliku,
- `major(3)` i `minor(3)` do zdekodowania **numeru urządzenia**,
- `readlinkat(2)` to przeczytania ścieżki zawartej w dowiązaniu symbolicznym.

Implementacja iterowania zawartości katalogu będzie wymagała zapoznania się ze strukturą «linux_dirent» opisaną w podręczniku `getdents(2)`. Wywołanie systemowe «getdents» nie jest eksportowane przez bibliotekę standardową, zatem należało je wywołać pośrednio – zobacz plik «libcsapp/Getdents.c».

Zadanie 5 (2pkt, P). (Pomysłodawcą zadania jest Tomasz Wierzbicki.)

Program «mergesort» odczytuje ze standardowego wejście liczbę naturalną n, po czym czyta n liczb całkowitych. Program realizuje algorytm sortowania przez scalanie. Proces główny zajmuje się wczytywaniem danych wejściowych i drukowaniem posortowanego ciągu. Żeby posortować liczby, program uruchamia podproces, który wykonuje procedurę «Sort». Rozmawia z nim przy pomocy gniazda domeny uniksowej `unix(7)`, które tworzy z użyciem `socketpair(2)`, czyli **lokalnej dwukierunkowej** metody komunikacji międzyprocesowej. Jeśli proces sortujący otrzyma od rodzica pojedynczą liczbę, to natychmiast odsyła ją swojemu rodzicowi i kończy działanie. Jeśli dostanie więcej liczb, to startuje odpowiednio lewe i prawe dziecko, po czym za pomocą procedury «SendElem» przesyła im liczby do posortowania. Następnie wywołuje procedurę «Merge», która odbiera od potomków posortowane ciągi, scala je i wysyła do procesu nadrzędnego.

Twoim zadaniem jest uzupełnienie procedury «Sort» tak by wystartowała procesy potomne i uruchomiła procedury «SendElem» i «Merge». Należy odpowiednio połączyć procesy z użyciem gniazd oraz zamknąć niepotrzebne gniazda w poszczególnych procesach. Posługując się rysunkiem wyjaśnij strukturę programu. Kiedy tworzysz podprocesy i gniazda? Kiedy zamykasz niepotrzebne gniazda? Jak wygląda przepływ danych?

Skrypt «gen-nums.py» przyjmuje w linii poleceń n, czyli liczbę elementów do wygenerowania. Po uruchomieniu drukuje n na standardowe wyjście, po czym drukuje n losowych liczb całkowitych. Produkowane dane są w odpowiednim formacie do wprowadzenia do programu «mergesort».

UWAGA! Wszystkie procesy muszą działać w stałej pamięci!