Zadanie 3

Otwarto: środa, 19 kwietnia 2023, 00:00
Wymagane do: piątek, 12 maja 2023, 23:59

System płatności w MINIX-ie

Celem zadania jest umożliwienie procesom w systemie MINIX posiadania pieniędzy i dokonywania przelewów wzajemnych.

Każdy proces otrzymuje na start `INIT_BALANCE` jednostek waluty. Następnie procesy mogą wzajemnie przelewać sobie pieniądze, tj. proces `P` może zlecić przelew `n` jednostek waluty procesowi `Q`. Aby przelew się udał, proces `P` musi mieć na swoim koncie przynajmniej `n` jednostek waluty (stan konta procesu nie może być ujemny), a proces `Q` musi mieć na swoim koncie nie więcej niż `MAX_BALANCE - n` jednostek waluty. Ponadto, jako podstawowe zabezpieczenie przed praniem brudnych pieniędzy, wymagamy, żeby procesy `P` i `Q` nie były w relacji potomek-przodek. Jeśli przelew się udaje, to stan konta procesu `P` zmniejsza się o `n` jednostek waluty, zaś stan konta procesu `Q` zwiększa się o `n` jednostek waluty.

Pieniądze procesów nie są dziedziczone - kiedy proces kończy działanie, zgromadzone przez niego jednostki waluty znikają.

Uwaga: przyznawanie każdemu procesowi na start nowych jednostek waluty nieuchronnie prowadzi do inflacji pieniądza, ale ten problem pozostawmy ekonomistom.

Nowe wywołanie systemowe

Zadanie polega na dodaniu wywołania systemowego `PM_TRANSFER_MONEY` oraz funkcji bibliotecznej `int transfermoney(pid_t recipient, int amount)`. Funkcja powinna być zadeklarowana w pliku `unistd.h`. Stałe `INIT_BALANCE = 100` i `MAX_BALANCE = 1000` powinny być zdefiniowane w pliku `minix/config.h`.

Wywołanie funkcji `int transfermoney(pid_t recipient, int amount)` powinno zrealizować przelew `amount` jednostek waluty z konta procesu wywołującego funkcję na konto procesu o identyfikatorze `recipient`. W przypadku powodzenia wykonania przelewu funkcja zwraca stan konta procesu, który ją wywołał, po wykonaniu tego przelewu.

Uwaga: proces może sprawdzić swój stan konta, np. przelewając sobie samemu `0` jednostek waluty.

W przypadku kiedy przelew nie udaje się, funkcja `transfermoney` zwraca w wyniku `-1` i ustawia `errno` na odpowiedni kod błędu:

- jeśli `recipient` nie jest identyfikatorem aktualnie działającego procesu, to `errno` zostaje ustawione na `ESRCH`;
- jeśli `recipient` jest identyfikatorem procesu będącego potomkiem lub przodkiem procesu wywołującego funkcję `transfermoney`, to `errno` zostaje ustawione na `EPERM`;
- jeśli wartość `amount` jest ujemna lub proces wywołujący funkcję `transfermoney` ma na koncie mniej niż `amount` jednostek waluty lub proces o identyfikatorze `recipient` ma więcej niż `MAX_BALANCE - amount` jednostek waluty, to `errno` zostaje ustawione na `EINVAL`.

Działanie funkcji `transfermoney()` powinno polegać na użyciu nowego wywołania systemowego `PM_TRANSFER_MONEY`, które należy dodać do serwera `PM`. Do przekazania parametrów należy zdefiniować własny typ komunikatu.

Format rozwiązania

Poniżej przyjmujemy, że `ab123456` oznacza identyfikator studentki rozwiązującej lub studenta rozwiązującego zadanie. Należy przygotować łatkę (ang. *patch*) ze zmianami w katalogu `/usr`. Plik zawierający łatkę o nazwie `ab123456.patch` uzyskujemy za pomocą polecenia

```
diff -rupNEZbB oryginalne-źródła/usr/ moje-rozwiązanie/usr/ > ab123456.patch
```

gdzie `oryginalne-źródła` to ścieżka do niezmienionych źródeł MINIX-a, natomiast `moje-rozwiązanie` to ścieżka do źródeł MINIX-a zawierających rozwiązanie. Tak użyte polecenie `diff` rekurencyjnie przeskanuje pliki ze ścieżki `oryginalne-źródła/usr`, porówna je z plikami ze ścieżki `moje-rozwiązanie/usr` i wygeneruje plik `ab123456.patch`, który podsumowuje różnice. Tego pliku będziemy używać, aby automatycznie nanieść zmiany na czystą kopię MINIX-a, gdzie będą przeprowadzane testy rozwiązania. Więcej o poleceniu `diff` można dowiedzieć się z podręcznika (`man diff`).

Umieszczenie łatki w katalogu `/` na czystej kopii MINIX-a i wykonanie polecenia

```
patch -p1 < ab123456.patch
```

powinno skutkować naniesieniem wszystkich oczekiwanych zmian wymaganych przez rozwiązanie. Należy zadbać, aby łatka zawierała tylko niezbędne różnice.

Po naniesieniu łatki zostaną wykonane polecenia:

- `make && make install` w katalogach `/usr/src/minix/fs/procfs`, `/usr/src/minix/servers/pm`, `/usr/src/minix/drivers/storage/ramdisk`, `/usr/src/minix/drivers/storage/memory` oraz `/usr/src/lib/libc`,
- `make do-hdboot` w katalogu `/usr/src/releasetools`,
- `reboot`.

Rozwiązanie w postaci łatki `ab123456.patch` należy umieścić w Moodle.

Uwagi

- Serwer PM przechowuje informacje o procesach w tablicy `mproc` zadeklarowanej w pliku `mproc.h`.
- Warto przeanalizować, jak PM realizuje wywołania systemowe. Więcej informacji o działaniu tego serwera będzie na laboratorium 8.
- W treści zadania mowa o wykonaniu polecenia `make` w katalogach `/usr/src/minix/fs/procfs`, `/usr/src/minix/drivers/storage/ramdisk` oraz `/usr/src/minix/drivers/storage/memory`, ponieważ są w nich pliki włączające plik `mproc.h`.
- Należy samodzielnie przetestować rozwiązanie. Jeden z podstawowych scenariuszy jest następujący: uruchamiamy proces A, który następnie uruchamia procesy B i C; procesy B i C zaczynają sobie wzajemnie przelewać pieniądze.
- Nie przyznajemy punktów za rozwiązanie, w którym łatka nie nakłada się poprawnie, które nie kompiluje się lub które powoduje *kernel panic* podczas uruchamiania systemu.

Pytania

Pytania dotyczące zadania można zadawać w przeznaczonym do tego wątku na Moodle.

Dodaj pracę

Status przesłanego zadania

Status przesłanego zadania	Nie przesłano jeszcze zadania
Stan oceniania	Nieocenione
Pozostały czas	Pozostało 21 dni 13 godzin
Ostatnio modyfikowane	-

Komentarz do przesłanego zadania	▸ Komentarze (0)