Systemy operacyjne

Lista zadań nr 3

Na zajęcia 29 października 2020

Należy przygotować się do zajęć czytając następujące rozdziały książek:

- APUE (wydanie trzecie): 7.10 i 10.15, 9.4 – 9.8 i 10.21, 18.1 – 18.2 i 18.6

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

Zadanie 1. W artykule A `fork()` in the road[1] skrytykowano wywołanie `fork(2)`. Na podstawie sekcji 4 publikacji przedstaw pobieżnie argumentację autorów przeciwko «`fork`». Następnie opowiedz jak `posix_spawn(3)` pomagają zniwelować niektóre z wymienionych wad.

Zadanie 2. Zaprezentuj sytuację, w której proces zostanie **osierocony**. W terminalu uruchom dodatkową kopię powłoki «`bash`». Z jej poziomu wystartuj «`sleep 1000`» jako **zadanie drugoplanowe** i sprawdź, kto jest jego rodzicem. Poleceniem «`kill`» wyślij sygnał «`SIGKILL`» do uruchomionej wcześniej powłoki i sprawdź, kto stał się nowym rodzicem procesu «`sleep`». Wyjaśnij przebieg i wyniki powyższego eksperymentu. Wytłumacz zachowanie eksperymentu, gdy zamiast «`SIGKILL`» wyślemy powłoce sygnał «`SIGHUP`».

Wskazówka: Uruchom powłokę przy pomocy polecenia «`strace -e trace=signal`».

Zadanie 3. Wyświetl konfigurację **terminala** przy pomocy polecenia «`stty -a`» i wskaż znaki które (a) sterownik terminala zamienia na sygnały związane z **zarządzaniem zadaniami** (b) służą do **edycji wiersza**. Możliwości edycji wiersza zaprezentuj na przykładzie polecenia «`cat`». Następnie uruchom «`find /`» i obserwuj zachowanie programu po naciśnięciu kombinacji klawiszy «`CTRL+S`» i «`CTRL+Q`» – jakie sygnały wysyła sterownik terminala do **zadania pierwszoplanowego**? Następnie **wstrzymaj** zadanie pierwszoplanowe «`sleep 1000`» i przy pomocy wbudowanego polecenia powłoki «`bg`» przenieś to zadanie do wykonania w tle. Jaki sygnał został użyty do wstrzymania zadania? Na przykładzie programu «`vi`» wskaż kiedy program może być zainteresowany obsługą tego sygnału oraz «`SIGCONT`».

Zadanie 4. Uruchom w powłoce «`bash`» polecenie «`cat - &`». Czemu zadanie zostało od razu wstrzymane? Jaki sygnał otrzymało? Zakończ to zdanie **wbudowanym poleceniem powłoki** «`kill`». Następnie porównaj działanie polecenia «`cat /etc/shells &`» przed i po zmianie konfiguracji terminala poleceniem «`stty tostop`». Jaki efekt ma włączenie flagi «`tostop`» na zachowanie sterownika terminala?

Zauważ, że powłoka dostaje «`SIGCHLD`» w wyniku zmiany stanu procesu potomnego, a nie tylko jego zakończenia, i ewentualnie wybiera grupę pierwszoplanową. Jak przy pomocy `waitpid(2)` rozróżnić wstrzymanie i kontynuowanie od zakończenia procesu potomnego? Jaka funkcja biblioteki standardowej służy do wyboru grupy pierwszoplanowej?

[1] `https://www.microsoft.com/en-us/research/publication/a-fork-in-the-road/`

Ściągnij ze strony przedmiotu archiwum «so20_lista_3.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 5. Procedury setjmp(3) i longjmp(3) z biblioteki standardowej języka C służą do wykonywania nielokalnych skoków. Uproszczone odpowiedniki tych procedur znajdują się w pliku «libcsapp/Setjmp.s», a definicja «Jmpbuf» w pliku «include/csapp.h». Wyjaśnij co robią te procedury, a następnie przeprowadź uczestników zajęć przez ich kod. Dlaczego «Jmpbuf» nie przechowuje wszystkich rejestrów procesora? Czemu «Longjmp» zapisuje na stos wartość przed wykonaniem instrukcji «ret»?

Zadanie 6. Uzupełnij program «game» tj. prostą grę w szybkie obliczanie sumy dwóch liczb. Zadaniem procedury «readnum» jest wczytać od użytkownika liczbę. Jeśli w międzyczasie przyjdzie sygnał, to procedura ma natychmiast wrócić podając numer sygnału, który przerwał jej działanie. W przeciwnym przypadku zwraca zero i przekazuje wczytaną liczbę przez pamięć pod wskaźnikiem «num_p». Twoja implementacja procedury «readnum» musi wczytać całą linię w jednym kroku! Należy wykorzystać procedury siglongjmp(3), sigsetjmp(3) i alarm(2). Kiedy Twój program będzie zachowywać się poprawnie zamień procedury **nielokalnych skoków** na longjmp(3) i setjmp(3). Czemu program przestał działać?

UWAGA! We FreeBSD i macOS zamiast «longjmp» i «setjmp» należy użyć odpowiednio «_longjmp» i «_setjmp».

Zadanie 7 (2). Program «coro» wykonuje trzy współprogramy[2] połączone ze sobą w potok bez użycia pipe(2). 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 setjmp(3) i longjmp(3).

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.

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